Отчет по Домашнему заданию по курсу
"Разработка Интернет-Приложений"

Выполнила:
Студентка группы
ИУ5-55Б
Зубарева Антонина Михайловна

Москва, МГТУ – 2021

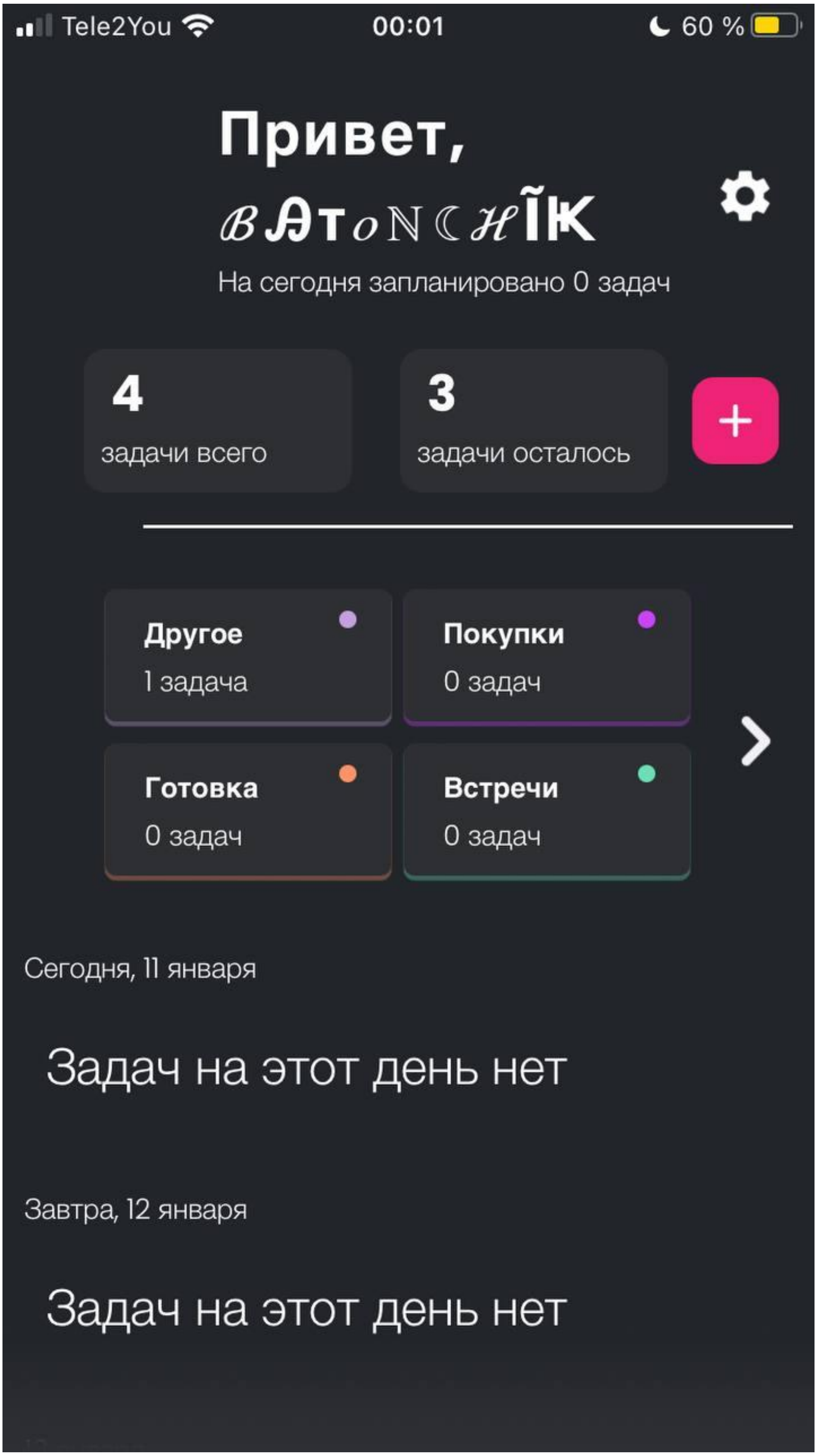
Задание:

На основе методических указаний разработайте мобильное приложение на языке Kotlin (платформа Android) или языке Swift (платформа iOS). Для создания приложения необходимо решить следующие задачи:

Создать стартовый проект мобильного приложения.
Добавить компонент таблицы в Ваше приложение.
Добавить подключение из мобильного приложения к веб-сервису, разработанному в лабораторной работе №6.
Наполнить таблицу элементами, полученными из вашего веб-сервиса: с помощью запросов к API получить из него данные.
Разработать дополнительный экран с UI компонентами для отображения детальной информации об элементе таблицы при касании по нему.

Указанное задание было частично изменено. Выполнено кроссплатформенное телефонное приложение с использованием React Native в качестве совместного проекта с Честновой Екатериной (ИУ5-52Б) и Мироновой Александрой (ИУ5-53Б). Приложение позволяет создавать категории задач с различными цветовыми индикаторами и создавать задачи с различными характеристиками (сроки выполнения и категории). Также есть отображение задач на ближайшую неделю на главном экране.

Результат работы программы



Привет,

ℬᎯᴛoNᑕℋĨK

На сегодня запланировано 0 задач

4
задачи всего

3
задачи осталось

Другое
1 задача

Покупки
0 задач

Готовка
0 задач

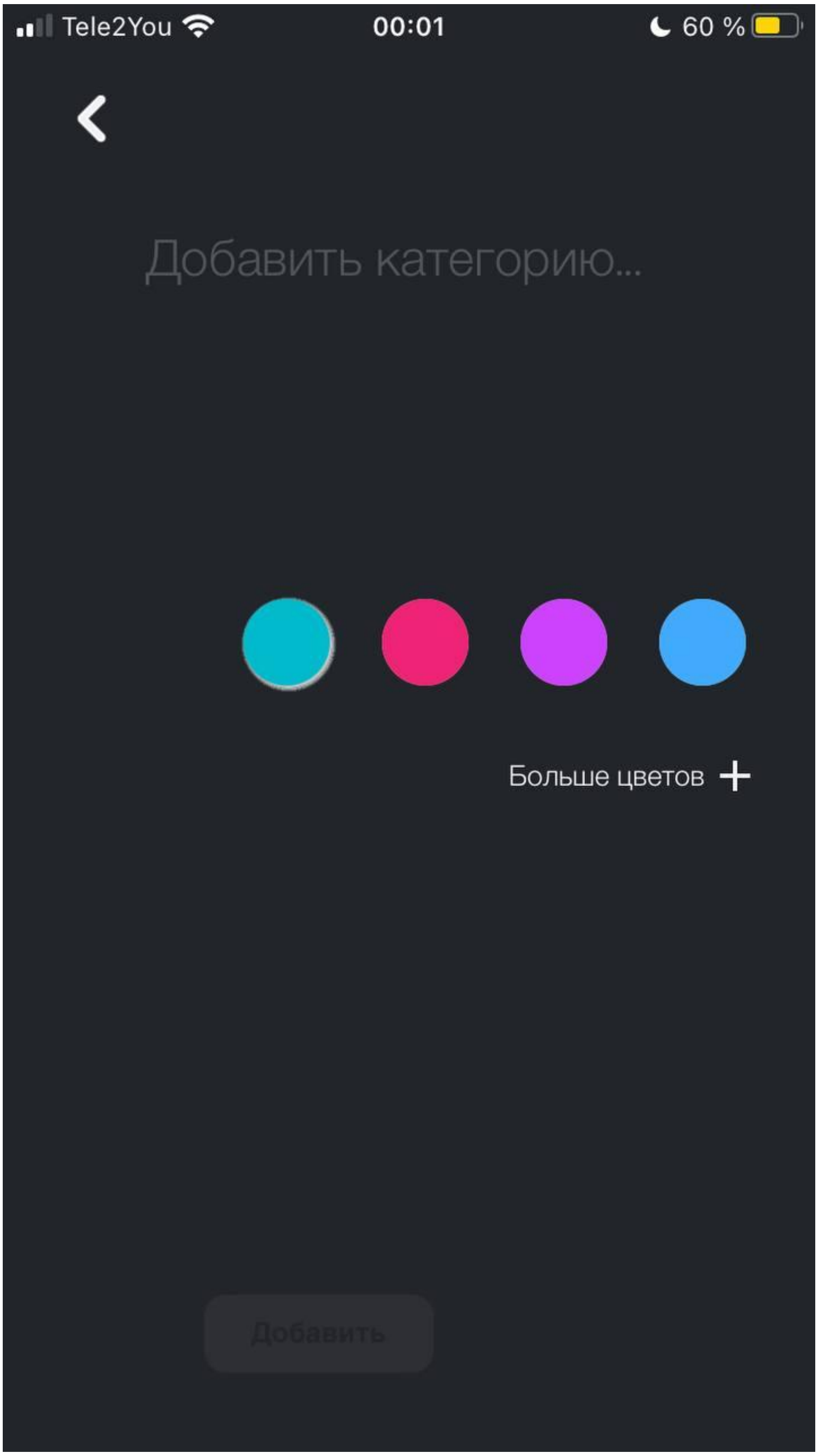
Встречи
0 задач

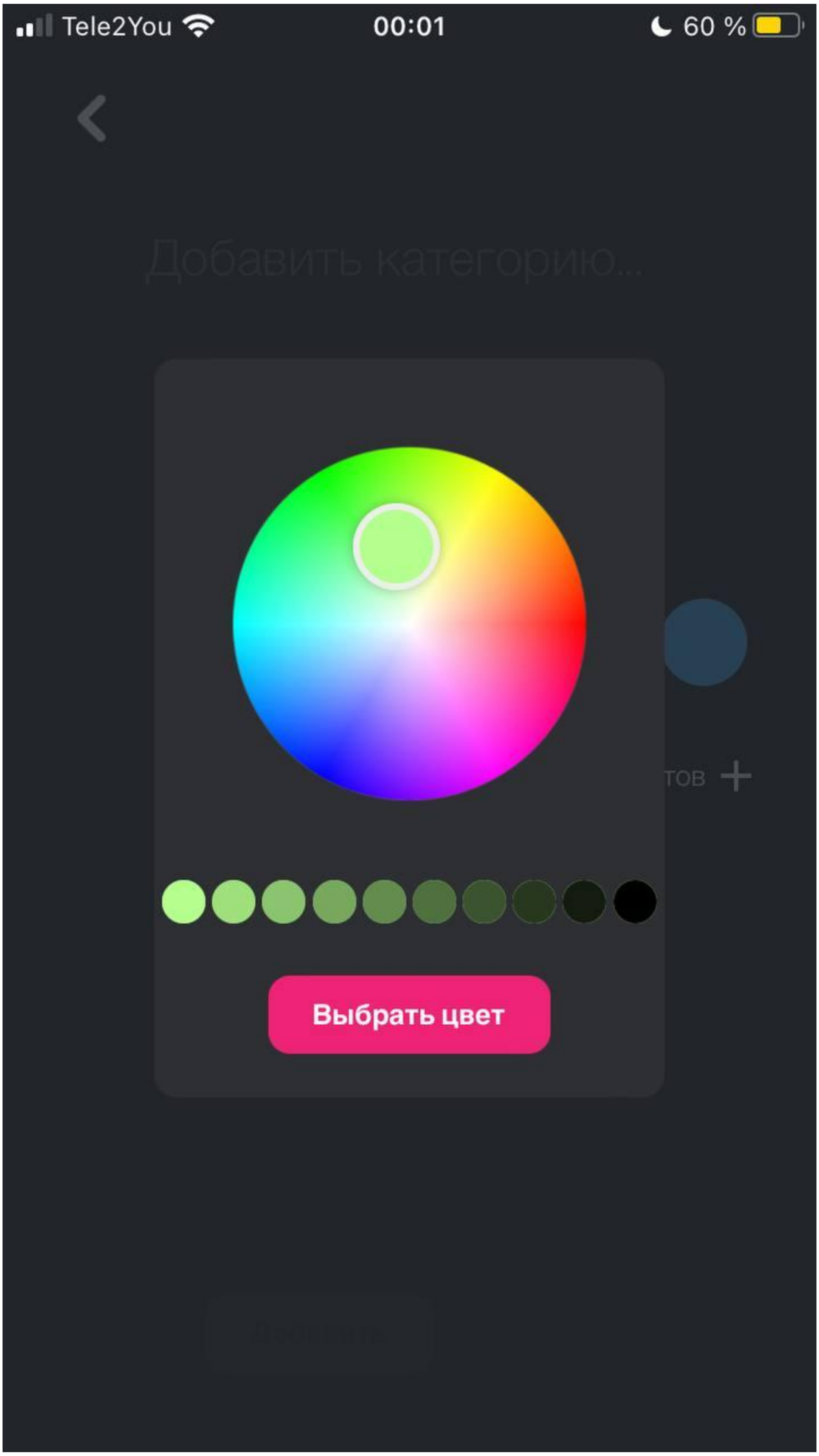
Сегодня, 11 января

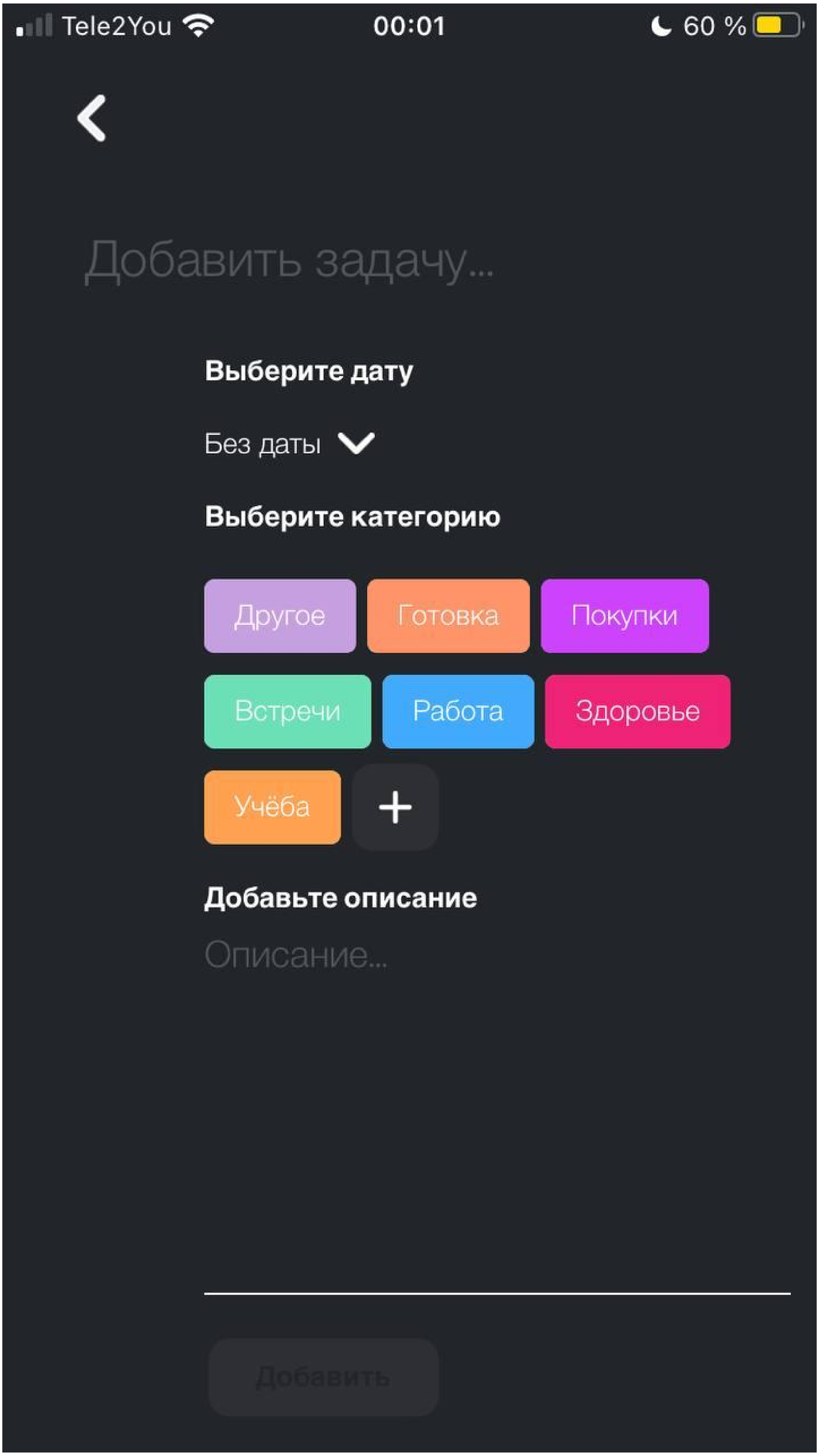
Задач на этот день нет

Завтра, 12 января

Задач на этот день нет


Tele2You
00:01
60 %
Добавить категорию...
Больше цветов +
Добавить


Tele2You
00:01
60 %
Добавить категорию...
тов +
Выбрать цвет
Добавить



Добавить задачу...

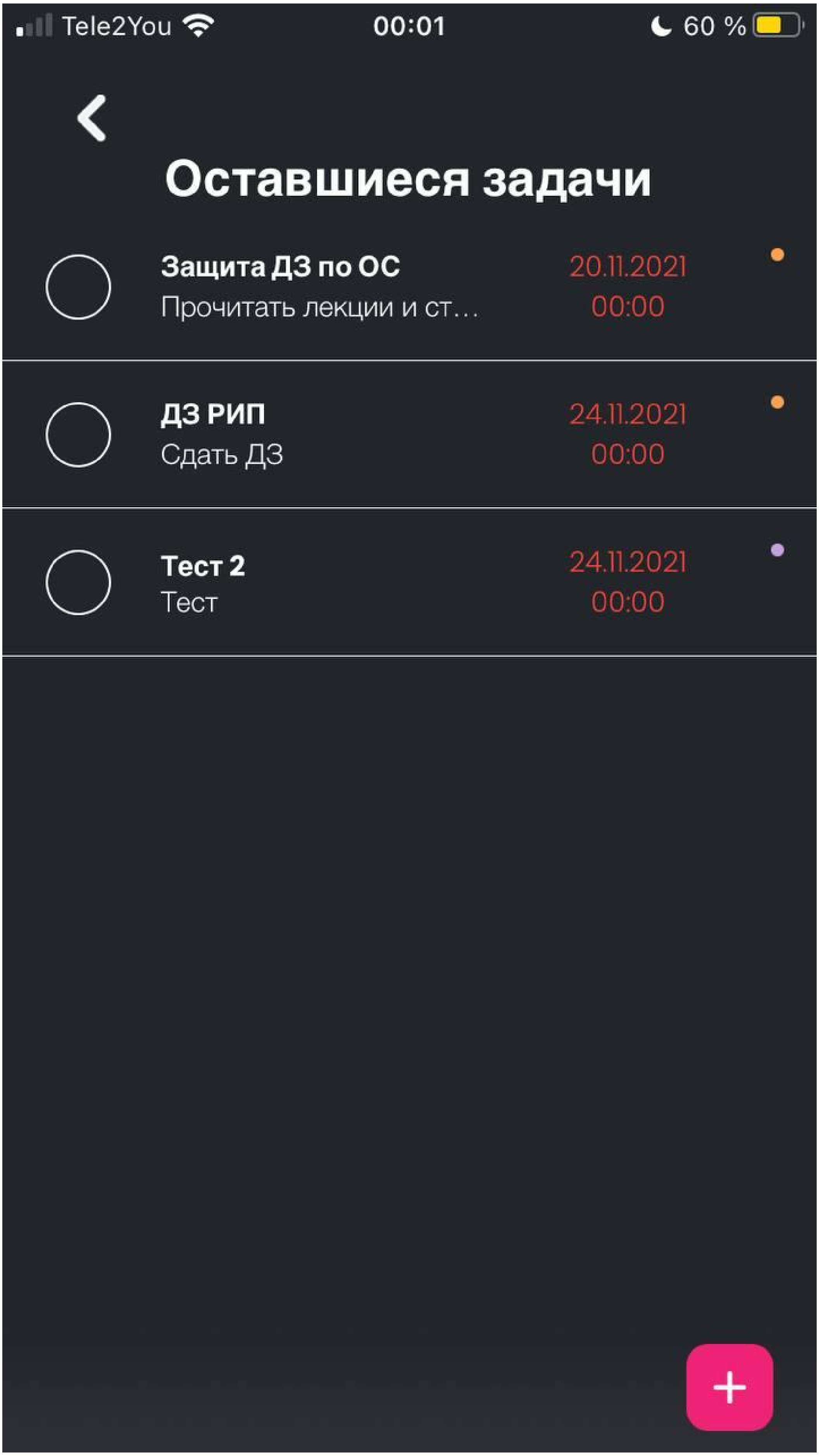
Выберите дату

Без даты

Выберите категорию

Добавьте описание

Описание...

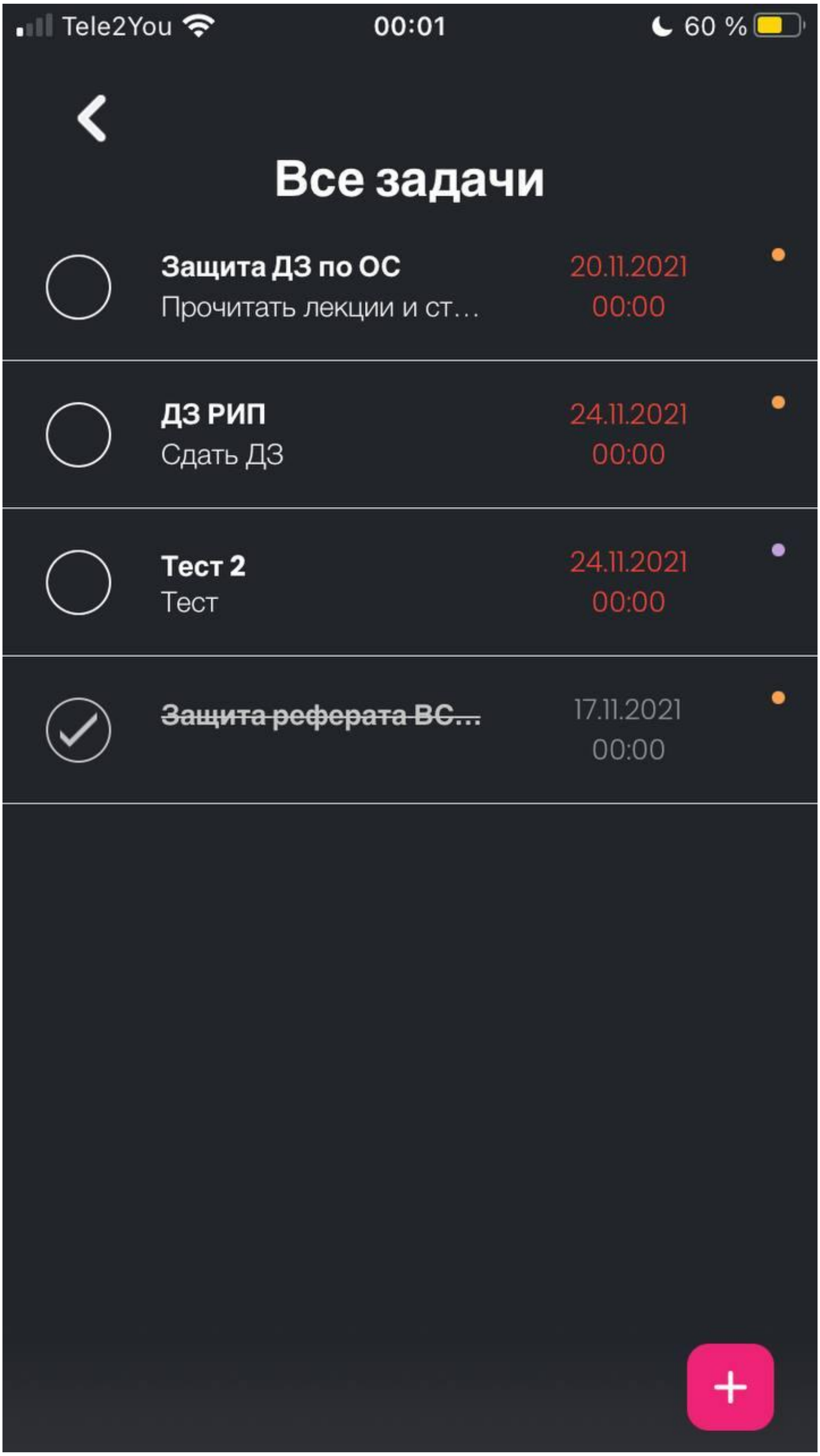


Оставшиеся задачи

○ **Защита ДЗ по ОС** 20.11.2021 00:00 ●
Прочитать лекции и ст…

○ **ДЗ РИП** 24.11.2021 00:00 ●
Сдать ДЗ

○ **Тест 2** 24.11.2021 00:00 ●
Тест

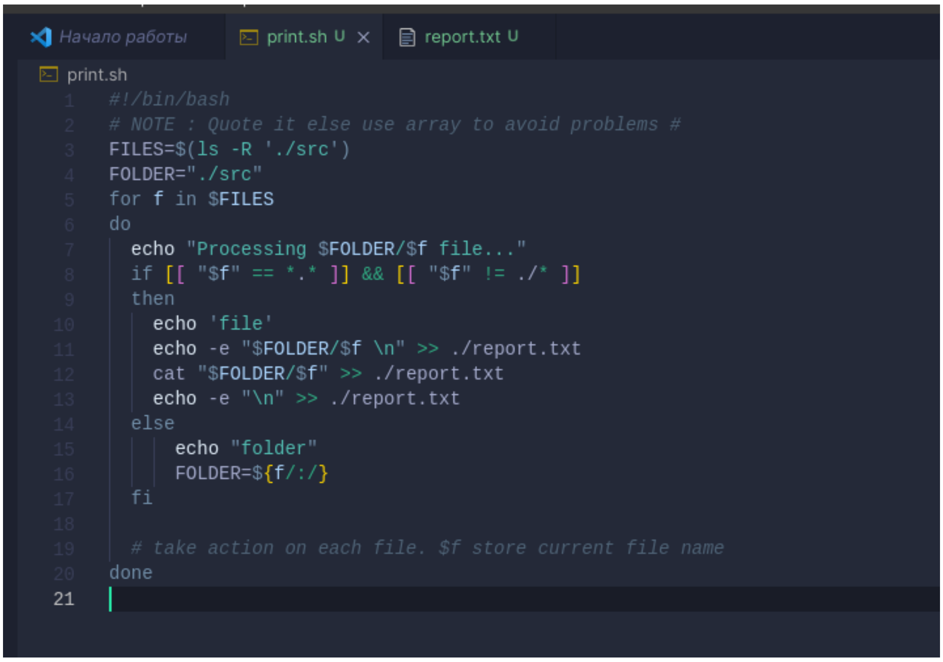


Все задачи

○	**Защита ДЗ по ОС** Прочитать лекции и ст…	20.11.2021 00:00	●
○	**ДЗ РИП** Сдать ДЗ	24.11.2021 00:00	●
○	**Тест 2** Тест	24.11.2021 00:00	●
✓	~~**Защита реферата ВС…**~~	17.11.2021 00:00	●

Текст программы

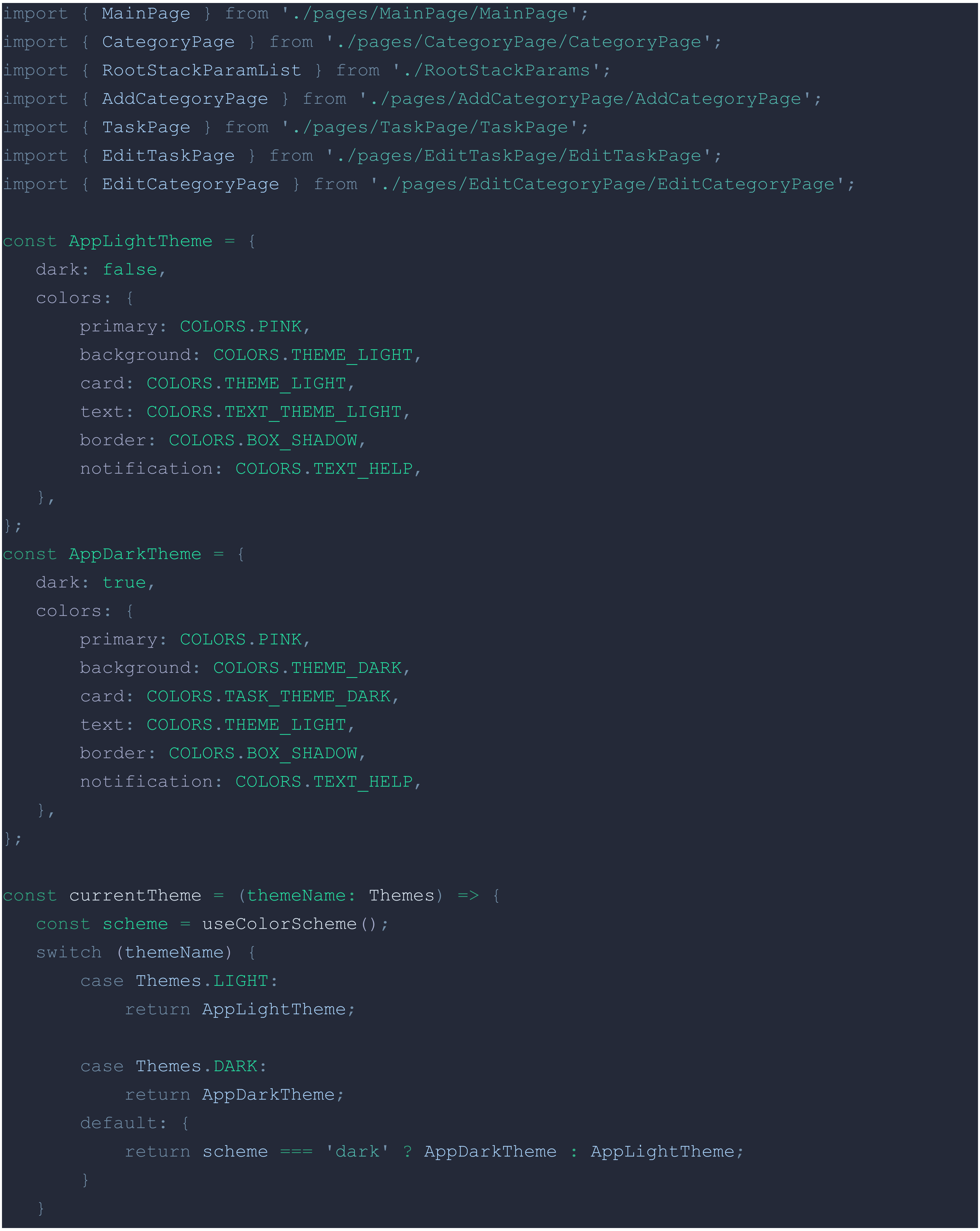
Всвязи с большим количеством файлов отчёт об их содержании сгенерирован автоматически bash скриптом



```bash
#!/bin/bash
# NOTE : Quote it else use array to avoid problems #
FILES=$(ls -R './src')
FOLDER="./src"
for f in $FILES
do
  echo "Processing $FOLDER/$f file..."
  if [[ "$f" == *.* ]] && [[ "$f" != ./* ]]
  then
    echo 'file'
    echo -e "$FOLDER/$f \n" >> ./report.txt
    cat "$FOLDER/$f" >> ./report.txt
    echo -e "\n" >> ./report.txt
  else
      echo "folder"
      FOLDER=${f/:/}
  fi

  # take action on each file. $f store current file name
done

```

Полученный код:

```
./src/components/AppNavigation.tsx

import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { NavigationContainer } from '@react-navigation/native';
import { useColorScheme, AppearanceProvider } from 'react-native-appearance';
import { StatusBar } from 'react-native';
import Constants from 'expo-constants';
import { ReduxDoToExample } from './development/ReduxDoToExample';
import { HomeScreen } from './development/HomeScreenTest';
import { ThemeButtonTest } from './development/ThemeButtonsTest';
import { AddTaskPage } from './pages/AddTaskPage/AddTaskPage';
import { useAppSelector } from '../store/hooks';
import { Themes } from '../store/types';
import { COLORS } from '../../assets/colors';
import { Tasks } from './development/Tasks';
import { TasksTonya } from './development/TasksTonya';
import { TasksSasha } from './development/TasksSasha';
import { TasksKate } from './development/TasksKate';
import { StartPage } from './pages/StartPage/StartPage';
import { SettingsPage } from './pages/SettingsPage/SettingsPage';
import { StorybookScreen } from './development/StorybookScreen';
```



```
import { MainPage } from './pages/MainPage/MainPage';
import { CategoryPage } from './pages/CategoryPage/CategoryPage';
import { RootStackParamList } from './RootStackParams';
import { AddCategoryPage } from './pages/AddCategoryPage/AddCategoryPage';
import { TaskPage } from './pages/TaskPage/TaskPage';
import { EditTaskPage } from './pages/EditTaskPage/EditTaskPage';
import { EditCategoryPage } from './pages/EditCategoryPage/EditCategoryPage';

const AppLightTheme = {
   dark: false,
   colors: {
       primary: COLORS.PINK,
       background: COLORS.THEME_LIGHT,
       card: COLORS.THEME_LIGHT,
       text: COLORS.TEXT_THEME_LIGHT,
       border: COLORS.BOX_SHADOW,
       notification: COLORS.TEXT_HELP,
   },
};
const AppDarkTheme = {
   dark: true,
   colors: {
       primary: COLORS.PINK,
       background: COLORS.THEME_DARK,
       card: COLORS.TASK_THEME_DARK,
       text: COLORS.THEME_LIGHT,
       border: COLORS.BOX_SHADOW,
       notification: COLORS.TEXT_HELP,
   },
};

const currentTheme = (themeName: Themes) => {
   const scheme = useColorScheme();
   switch (themeName) {
       case Themes.LIGHT:
           return AppLightTheme;

       case Themes.DARK:
           return AppDarkTheme;
       default: {
           return scheme === 'dark' ? AppDarkTheme : AppLightTheme;
       }
   }
};
const Stack = createStackNavigator<RootStackParamList>();
export const AppNavigation: React.FC = () => {
   const appThemeName = useAppSelector((state) => state.app.theme);
   const APP_DEV = Constants.manifest.extra && Constants.manifest.extra.APP_DEV;
   const appTheme = currentTheme(appThemeName);
   return (
       <AppearanceProvider>
           <NavigationContainer theme={appTheme}>
               <StatusBar barStyle={appTheme.dark ? 'light-content' : 'dark-content'}
/>
               <Stack.Navigator headerMode="none" initialRouteName={APP_DEV ? 'Tasks' :
'MainPage'}>
                   {APP_DEV && (
                       <>
                           <Stack.Screen name="Tasks" component={Tasks} />
                           <Stack.Screen name="TasksTonya" component={TasksTonya} />
                           <Stack.Screen name="TasksSasha" component={TasksSasha} />
                           <Stack.Screen name="TasksKate" component={TasksKate} />
                           <Stack.Screen name="StorybookScreen"
component={StorybookScreen} />
                           <Stack.Screen name="Home" component={HomeScreen} />
                           <Stack.Screen name="Example" component={ReduxDoToExample} />
                           <Stack.Screen name="ThemeButtonTest"
component={ThemeButtonTest} />
                       </>
                   )}
                   <Stack.Screen name="MainPage" component={MainPage} />
                   <Stack.Screen name="StartPage" component={StartPage} />
                   <Stack.Screen name="SettingsPage" component={SettingsPage} />
                   <Stack.Screen name="AddTaskPage" component={AddTaskPage} />
                   <Stack.Screen name="CategoryPage" component={CategoryPage} />
                   <Stack.Screen name="AddCategoryPage" component={AddCategoryPage} />
                   <Stack.Screen name="TaskPage" component={TaskPage} />
                   <Stack.Screen name="EditTaskPage" component={EditTaskPage} />
                   <Stack.Screen name="EditCategoryPage" component={EditCategoryPage}
/>
               </Stack.Navigator>
           </NavigationContainer>
       </AppearanceProvider>
   );
};
```

```
pages/RootStackParams.ts



ActionButton/AnimatedSquares.tsx



./src/components/atomic/ActionButton/ActionButton.tsx

import { useTheme } from '@react-navigation/native';
import React from 'react';
import { Text, Pressable, PressableProps, StyleProp, ViewStyle, View } from
'react-native';
import { COLORS } from '../../../../assets/colors';
import { styles } from './styles';

interface ActionButtonProps extends PressableProps {
   active: boolean;
   title: string;
   style?: StyleProp<ViewStyle>;
}

export const ActionButton: React.FC<ActionButtonProps> = ({ active, title, style,
...props }) => {
   const { onPress } = props;
   const { colors: themeColors, dark } = useTheme();
   const disabledColor = dark ? COLORS.TASK_THEME_DARK : COLORS.BUTTON_THEME_LIGHT;
   return (
       <View style={style}>
           <Pressable
               style={[{ backgroundColor: active ? COLORS.PINK : disabledColor },
styles.button]}
               onPress={active ? onPress : null}
           >
               <Text style={[{ color: active ? COLORS.THEME_LIGHT :
themeColors.background }, styles.text]}>
                   {title}
               </Text>
           </Pressable>
       </View>
   );
```

```
};


./src/components/atomic/ActionButton/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    button: {
        height: 36,
        borderRadius: 10,
        justifyContent: 'center',
        alignItems: 'center',
        paddingHorizontal: 20,
    },
    text: {
        fontSize: 14,
        fontFamily: 'bold-poppins',
    },
});


./src/components/atomic/BackArrowButton/BackArrowButton.tsx

import { useTheme } from '@react-navigation/native';
import React from 'react';
import { Pressable, PressableProps, Image, StyleProp, ViewStyle } from 'react-native';
import { ICONS } from '../../../../assets';

interface IconTransparentButtonProps extends PressableProps {
    style?: StyleProp<ViewStyle>;
}

export const BackArrowButton: React.FC<IconTransparentButtonProps> = ({ style, ...props
}) => {
    const { onPress } = props;
    const { dark } = useTheme();
    return (
        <Pressable onPress={onPress} style={style}>
            <Image source={dark ? ICONS.imgBackDarkTheme : ICONS.imgBackLightTheme}
style={{ width: 14, height: 22 }} />
        </Pressable>
    );
```

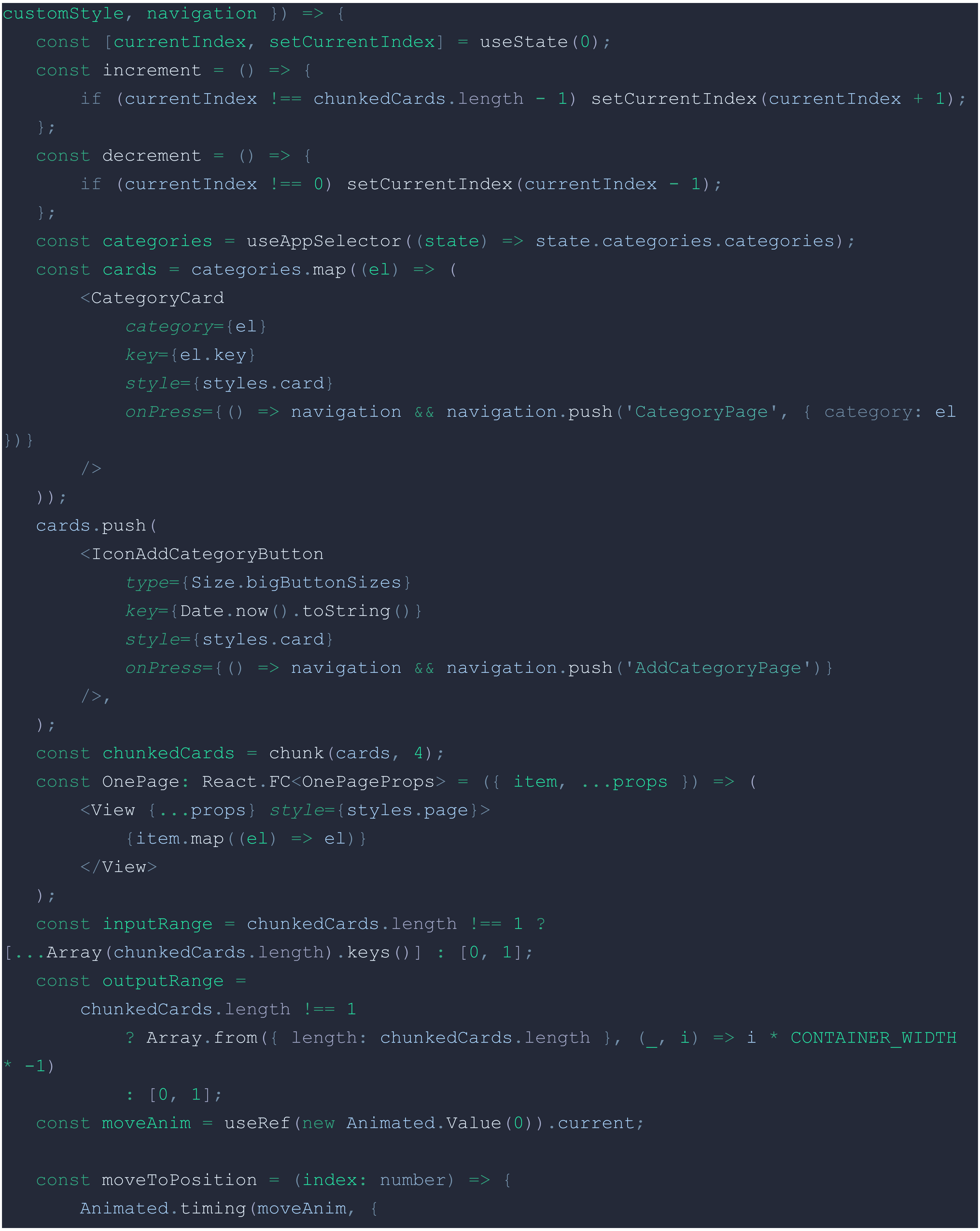
```
};



./src/components/atomic/CategoryCard/CategoryCard.tsx

import { useTheme } from '@react-navigation/native';
import React from 'react';
import { Text, View, StyleProp, ViewStyle } from 'react-native';
import { Shadow } from 'react-native-shadow-2';
import { TouchableOpacity } from 'react-native-gesture-handler';
import { GenericTouchableProps } from
'react-native-gesture-handler/lib/typescript/components/touchables/GenericTouchable';
import { useAppSelector } from '../../../store/hooks';
import { COLORS } from '../../../../assets/colors';
import { ICategory } from '../../../store/types';
import { styles } from './styles';
import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';

interface CategoryCardProps extends GenericTouchableProps {
   category: ICategory;
   style?: StyleProp<ViewStyle>;
}

export const CategoryCard: React.FC<CategoryCardProps> = ({ category, style, ...props
}) => {
   const { onPress } = props;
   const { colors: themeColors, dark } = useTheme();
   const todos = useAppSelector((state) => state.todos.todos);
   const count = todos.filter((item) => item.category.key === category.key).length;
   let word: string = 'задач';
   if (count <= 4 || count > 20) {
       if (count % 10 === 1) word = 'задача';
       else if (count % 10 > 1 && count % 10 < 5) word = 'задачи';
   }
   const colWithOp = convertHexToRGBA(category.color, 35);
   return (
       <View style={style}>
           <Shadow distance={1} startColor={colWithOp} radius={5} offset={[0, 1]}
size={[132, 62]}>
               <TouchableOpacity
                   style={[styles.card, { backgroundColor: dark ?
COLORS.TASK_THEME_DARK : themeColors.background }]}
                   onPress={onPress}
```

```
                >
                    <View style={styles.textField}>
                        <Text
                            numberOfLines={1}
                            style={[
                                { fontFamily: 'bold-poppins' },
                                { color: dark ? COLORS.THEME_LIGHT : themeColors.text },
                                styles.text,
                            ]}
                        >
                            {category.name}
                        </Text>
                        <Text
                            style={[
                                { fontFamily: 'light-poppins' },
                                { color: dark ? COLORS.THEME_LIGHT : themeColors.text },
                                styles.text,
                            ]}
                        >
                            {count.toString()} {word}
                        </Text>
                    </View>
                    <View style={[{ backgroundColor: category.color }, styles.circle]}
/>
                </TouchableOpacity>
            </Shadow>
        </View>
    );
};
```

./src/components/atomic/CategoryCard/styles.ts

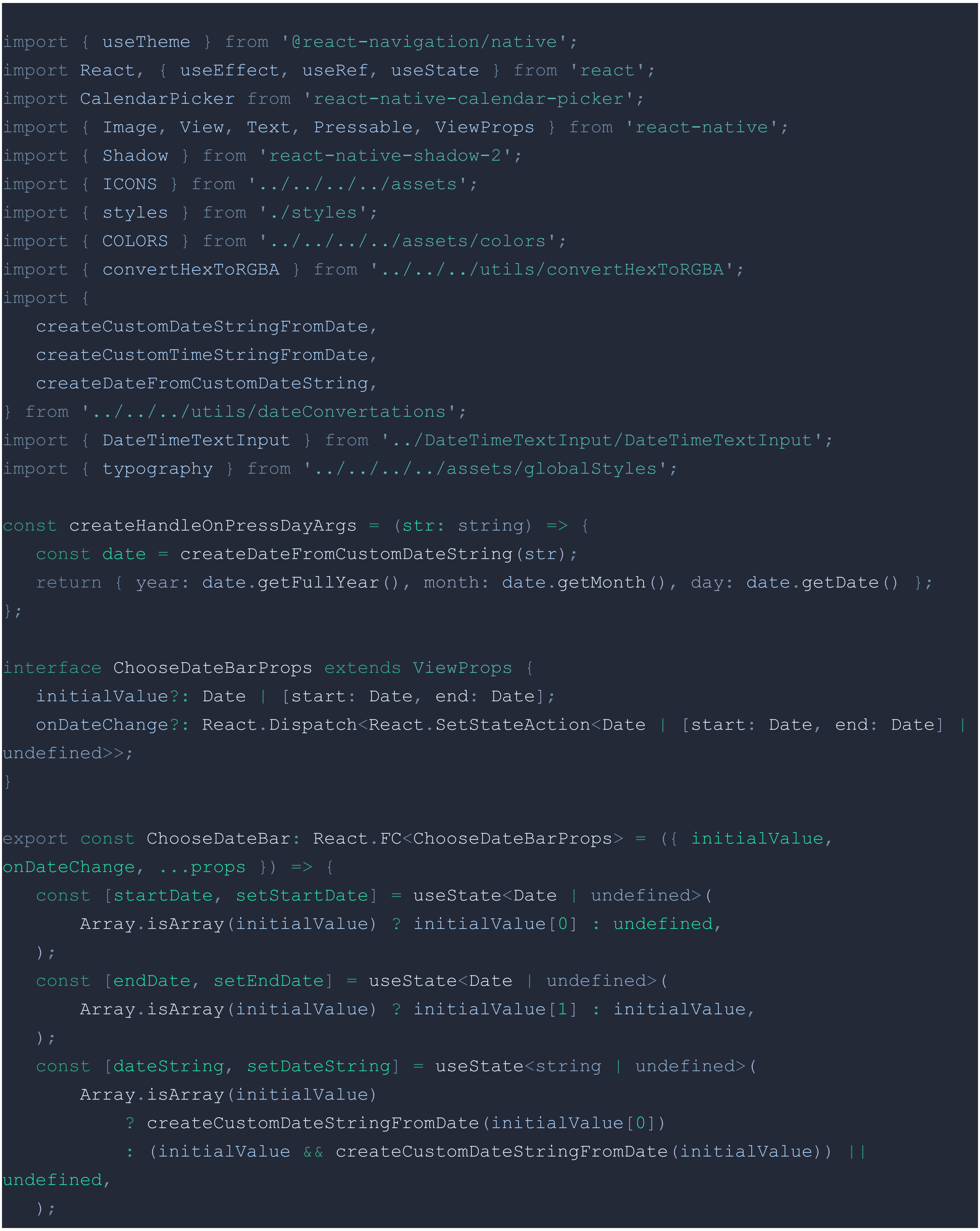
```
import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    card: {
        height: 61,
        width: 132,
        borderRadius: 5,
        flexDirection: 'row',
    },
    textField: {
        marginLeft: 18,
        marginVertical: 13,
        width: 90,
        padding: 0,
    },
    text: {
        fontSize: 14,
        height: 20,
    },
    circle: {
        marginTop: 10,
        marginRight: 6,
        height: 8,
        width: 8,
        borderRadius: 4,
    },
});
```

```
./src/components/atomic/ChooseCategoryBar/ChooseCategoryBar.tsx

import React, { useRef, useState } from 'react';
import { View, Pressable, Image, ViewProps, Animated } from 'react-native';
import GestureRecognizer from 'react-native-swipe-gestures';
import { StackNavigationProp } from '@react-navigation/stack';
import { IconAddCategoryButton } from '../IconAddCategoryButton/IconAddCategoryButton';
import { CategoryCard } from '../CategoryCard/CategoryCard';
import { useAppSelector } from '../../../store/hooks';
import { chunk } from '../../../utils/chunc';
import { Size } from '../IconAddCategoryButton/styles';
import { ICONS } from '../../../../assets';
import { CONTAINER_WIDTH, styles } from './styles';
import { RootStackParamList } from '../../RootStackParams';

interface OnePageProps extends ViewProps {
    item: Array<JSX.Element>;
}

interface ChooseCategoryBarProps extends ViewProps {
    navigation?: StackNavigationProp<RootStackParamList, 'MainPage'>;
}

export const ChooseCategoryBar: React.FC<ChooseCategoryBarProps> = ({ style:
```

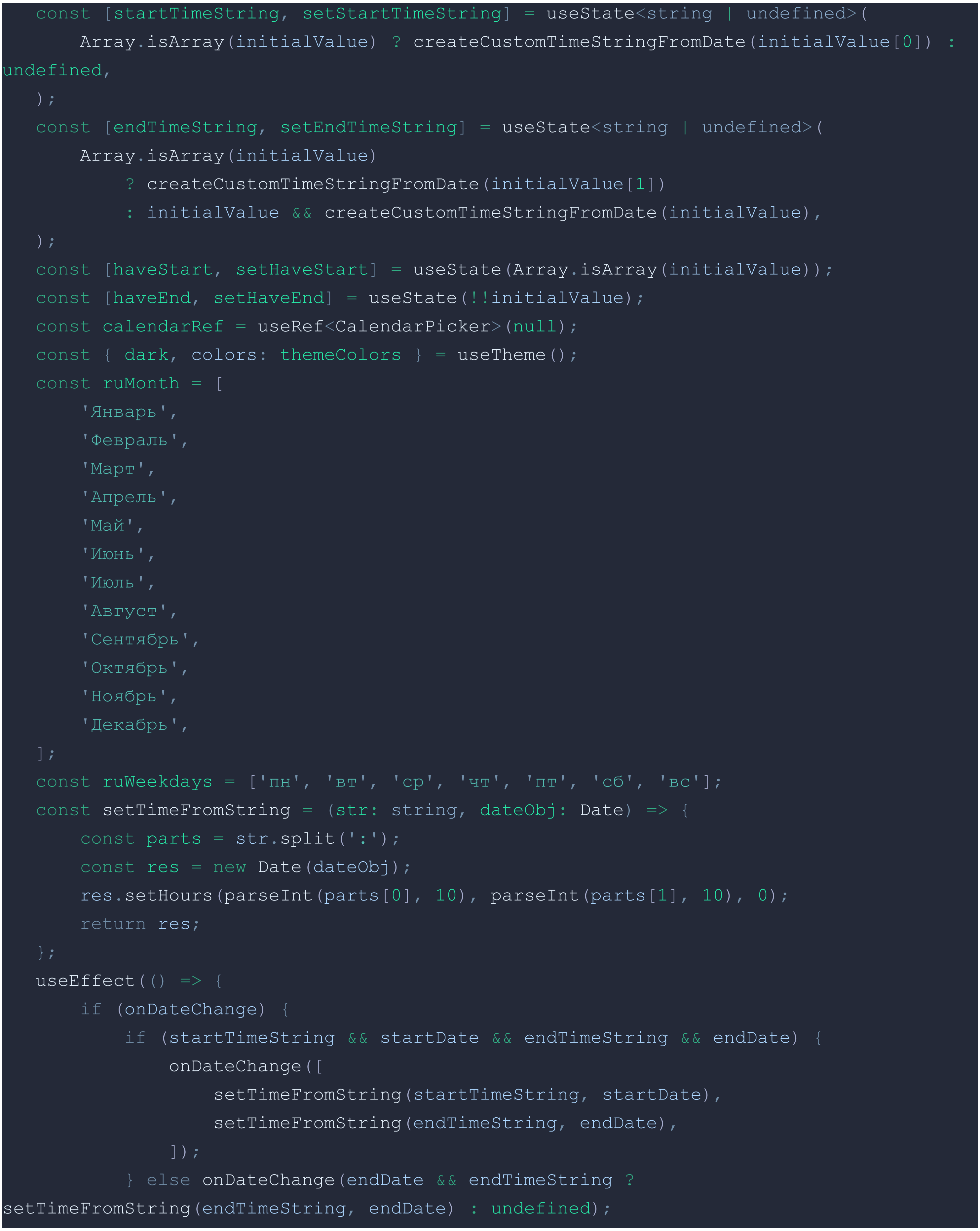


```
customStyle, navigation }) => {
    const [currentIndex, setCurrentIndex] = useState(0);
    const increment = () => {
        if (currentIndex !== chunkedCards.length - 1) setCurrentIndex(currentIndex + 1);
    };
    const decrement = () => {
        if (currentIndex !== 0) setCurrentIndex(currentIndex - 1);
    };
    const categories = useAppSelector((state) => state.categories.categories);
    const cards = categories.map((el) => (
        <CategoryCard
            category={el}
            key={el.key}
            style={styles.card}
            onPress={() => navigation && navigation.push('CategoryPage', { category: el
})}
        />
    ));
    cards.push(
        <IconAddCategoryButton
            type={Size.bigButtonSizes}
            key={Date.now().toString()}
            style={styles.card}
            onPress={() => navigation && navigation.push('AddCategoryPage')}
        />,
    );
    const chunkedCards = chunk(cards, 4);
    const OnePage: React.FC<OnePageProps> = ({ item, ...props }) => (
        <View {...props} style={styles.page}>
            {item.map((el) => el)}
        </View>
    );
    const inputRange = chunkedCards.length !== 1 ?
[...Array(chunkedCards.length).keys()] : [0, 1];
    const outputRange =
        chunkedCards.length !== 1
            ? Array.from({ length: chunkedCards.length }, (_, i) => i * CONTAINER_WIDTH
* -1)
            : [0, 1];
    const moveAnim = useRef(new Animated.Value(0)).current;

    const moveToPosition = (index: number) => {
        Animated.timing(moveAnim, {
            useNativeDriver: true,
            toValue: index,
            duration: 500,
        }).start();
    };

    const translationX = moveAnim.interpolate({
        inputRange,
        outputRange,
    });

    return (
        <View style={[customStyle, styles.wrapper]}>
            {currentIndex === 0 ? (
                <View style={{ width: 15 }} />
            ) : (
                <Pressable onPress={decrement} style={{ padding: 10 }}>
                    <Image source={ICONS.imgLeftArrowGrey} style={styles.arrow} />
                </Pressable>
            )}
            <GestureRecognizer onSwipeLeft={increment} onSwipeRight={decrement}>
                <View style={styles.visible}>
                    <Animated.View style={[{ transform: [{ translateX: translationX }],
flexDirection: 'row' }]}>
                        {chunkedCards.map((el) => (
                            <OnePage
                                item={el}
                                onLayout={() => moveToPosition(currentIndex)}
                                key={Math.random().toString()}
                            />
                        ))}
                    </Animated.View>
                </View>
            </GestureRecognizer>
            {currentIndex === chunkedCards.length - 1 ? (
                <View style={{ width: 15 }} />
            ) : (
                <Pressable onPress={increment} style={{ padding: 10 }}>
                    <Image source={ICONS.imgRightArrowGrey} style={styles.arrow} />
                </Pressable>
            )}
        </View>
    );
```

```
};



./src/components/atomic/ChooseCategoryBar/styles.ts

import { StyleSheet } from 'react-native';

const CARD_WIDTH = 132;
const SINGLE_MARGIN = 3;
export const CONTAINER_WIDTH = (CARD_WIDTH + SINGLE_MARGIN * 2) * 2;

export const styles = StyleSheet.create({
    wrapper: {
        flexDirection: 'row',
        justifyContent: 'center',
        alignItems: 'center',
    },
    arrow: {
        width: 15,
        height: 22,
        resizeMode: 'contain',
        paddingVertical: 12,
        paddingHorizontal: 5,
    },
    card: {
        marginHorizontal: 3,
        marginVertical: 5,
    },
    page: {
        width: CONTAINER_WIDTH,
        flexDirection: 'column',
        flexWrap: 'wrap',
        height: 150,
    },
    visible: {
        marginHorizontal: 10,
        width: CONTAINER_WIDTH,
        overflow: 'hidden',
    },
});



./src/components/atomic/ChooseDateBar/ChooseDateBar.tsx
```



```
import { useTheme } from '@react-navigation/native';
import React, { useEffect, useRef, useState } from 'react';
import CalendarPicker from 'react-native-calendar-picker';
import { Image, View, Text, Pressable, ViewProps } from 'react-native';
import { Shadow } from 'react-native-shadow-2';
import { ICONS } from '../../../../assets';
import { styles } from './styles';
import { COLORS } from '../../../../assets/colors';
import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';
import {
   createCustomDateStringFromDate,
   createCustomTimeStringFromDate,
   createDateFromCustomDateString,
} from '../../../utils/dateConvertations';
import { DateTimeTextInput } from '../DateTimeTextInput/DateTimeTextInput';
import { typography } from '../../../../assets/globalStyles';

const createHandleOnPressDayArgs = (str: string) => {
   const date = createDateFromCustomDateString(str);
   return { year: date.getFullYear(), month: date.getMonth(), day: date.getDate() };
};

interface ChooseDateBarProps extends ViewProps {
   initialValue?: Date | [start: Date, end: Date];
   onDateChange?: React.Dispatch<React.SetStateAction<Date | [start: Date, end: Date] |
undefined>>;
}

export const ChooseDateBar: React.FC<ChooseDateBarProps> = ({ initialValue,
onDateChange, ...props }) => {
   const [startDate, setStartDate] = useState<Date | undefined>(
       Array.isArray(initialValue) ? initialValue[0] : undefined,
   );
   const [endDate, setEndDate] = useState<Date | undefined>(
       Array.isArray(initialValue) ? initialValue[1] : initialValue,
   );
   const [dateString, setDateString] = useState<string | undefined>(
       Array.isArray(initialValue)
           ? createCustomDateStringFromDate(initialValue[0])
           : (initialValue && createCustomDateStringFromDate(initialValue)) ||
undefined,
   );
```



```
    const [startTimeString, setStartTimeString] = useState<string | undefined>(
        Array.isArray(initialValue) ? createCustomTimeStringFromDate(initialValue[0]) : undefined,
    );
    const [endTimeString, setEndTimeString] = useState<string | undefined>(
        Array.isArray(initialValue)
            ? createCustomTimeStringFromDate(initialValue[1])
            : initialValue && createCustomTimeStringFromDate(initialValue),
    );
    const [haveStart, setHaveStart] = useState(Array.isArray(initialValue));
    const [haveEnd, setHaveEnd] = useState(!!initialValue);
    const calendarRef = useRef<CalendarPicker>(null);
    const { dark, colors: themeColors } = useTheme();
    const ruMonth = [
        'Январь',
        'Февраль',
        'Март',
        'Апрель',
        'Май',
        'Июнь',
        'Июль',
        'Август',
        'Сентябрь',
        'Октябрь',
        'Ноябрь',
        'Декабрь',
    ];
    const ruWeekdays = ['пн', 'вт', 'ср', 'чт', 'пт', 'сб', 'вс'];
    const setTimeFromString = (str: string, dateObj: Date) => {
        const parts = str.split(':');
        const res = new Date(dateObj);
        res.setHours(parseInt(parts[0], 10), parseInt(parts[1], 10), 0);
        return res;
    };
    useEffect(() => {
        if (onDateChange) {
            if (startTimeString && startDate && endTimeString && endDate) {
                onDateChange([
                    setTimeFromString(startTimeString, startDate),
                    setTimeFromString(endTimeString, endDate),
                ]);
            } else onDateChange(endDate && endTimeString ? setTimeFromString(endTimeString, endDate) : undefined);
```

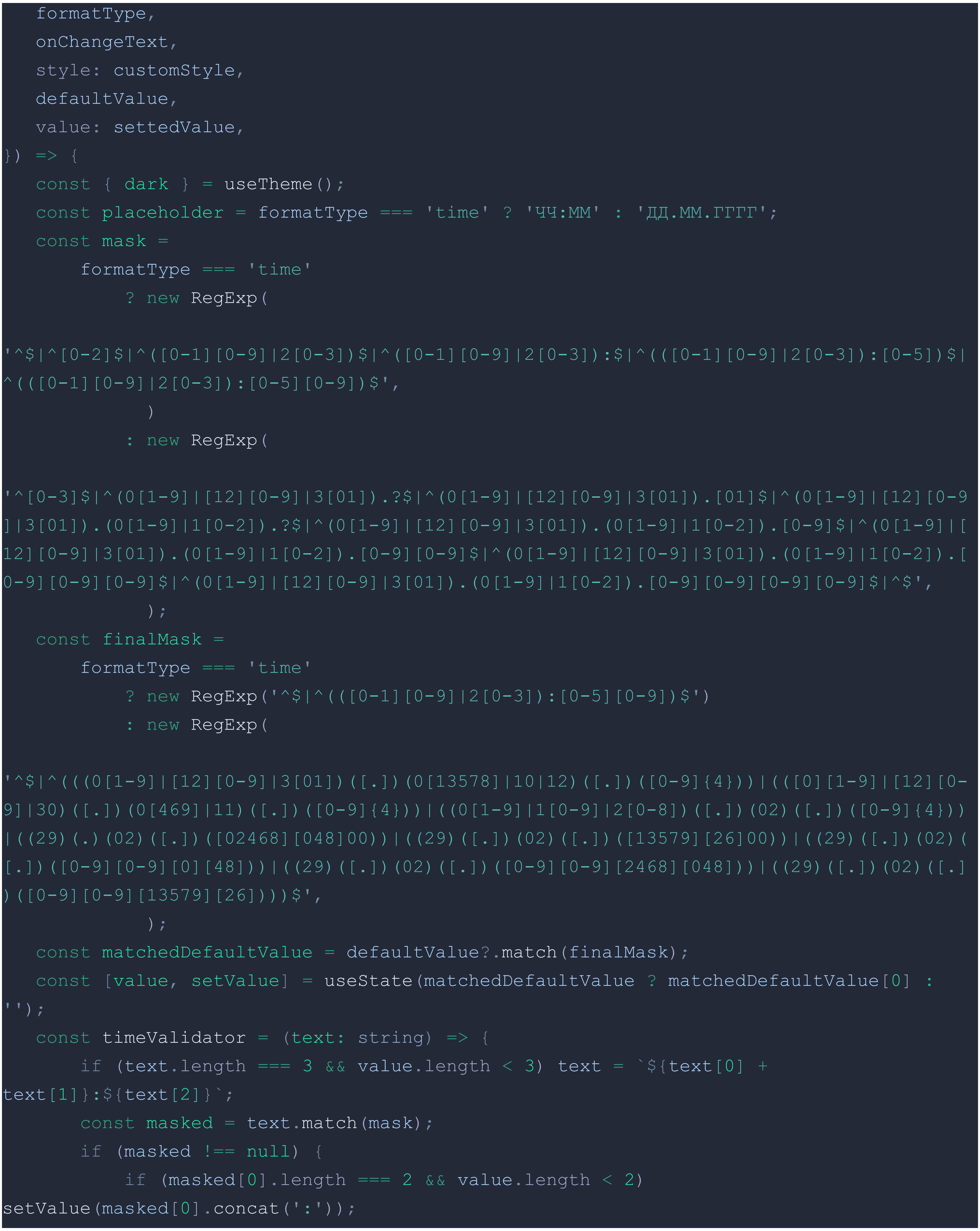
```
       }
   }, [startDate, startTimeString, endTimeString, endDate]);
   return (
       <View {...props} style={[{ width: 282 }, props.style]}>
           <Shadow distance={5} startColor={convertHexToRGBA(themeColors.text, 25)}
radius={10} offset={[1, 1]}>
               <View style={[styles.main, { backgroundColor: themeColors.card }]}>
                   <View
                       style={[
                           styles.calendarWrapper,
                           { borderColor: dark ? themeColors.background :
COLORS.BUTTON_THEME_LIGHT },
                       ]}
                   >
                       <CalendarPicker
                           ref={calendarRef}
                           initialDate={dateString ?
createDateFromCustomDateString(dateString) : undefined}
                           onDateChange={(date) => {

setDateString(createCustomDateStringFromDate(date.toDate()));
                               if (haveEnd || (!haveStart && !haveEnd)) {
                                   setEndDate(date.toDate());
                               }
                               if (haveStart || (!haveStart && !haveEnd)) {
                                   setStartDate(date.toDate());
                               }
                               if (!haveStart && !haveEnd) {
                                   setStartTimeString('00:00');
                                   setEndTimeString('00:00');
                                   setHaveStart(true);
                                   setHaveEnd(true);
                               }
                           }}
                           dayLabelsWrapper={{ borderColor: 'transparent' }}
                           selectYearTitle="Выберите год"
                           selectMonthTitle="Выберите месяц "
                           width={250}
                           textStyle={[styles.daysFont, { color: themeColors.text }]}
                           months={ruMonth}
                           weekdays={ruWeekdays}
                           startFromMonday
                           todayBackgroundColor={dark ? COLORS.THEME_DARK :
```

```
COLORS.BUTTON_THEME_LIGHT}
                           todayTextStyle={{ color: themeColors.text }}
                           selectedDayStyle={[
                               styles.selectedDay,
                               {
                                   backgroundColor: dark
                                       ? COLORS.BUTTON_THEME_LIGHT
                                       : convertHexToRGBA(COLORS.TEXT_THEME_LIGHT, 30),
                               },
                           ]}
                           previousComponent={
                               <Image
                                   style={styles.icon}
                                   source={
                                       dark
                                           ? ICONS.imgCalendarBackArrowDarkTheme
                                           : ICONS.imgCalendarBackArrowLightTheme
                                   }
                               />
                           }
                           nextComponent={
                               <Image
                                   source={
                                       dark
                                           ? ICONS.imgCalendarNextArrowDarkTheme
                                           : ICONS.imgCalendarNextArrowLightTheme
                                   }
                                   style={styles.icon}
                               />
                           }
                           monthTitleStyle={styles.header}
                           yearTitleStyle={styles.header}
                       />
                   </View>
                   <Text style={[dark ? typography.textDarkTheme :
typography.textLightTheme, { marginVertical: 6 }]}>
                       Начало
                   </Text>
                   <View style={{ flexDirection: 'row', alignItems: 'center' }}>
                       <Pressable
                           style={[
                               styles.selectionBox,
                               {
                                   backgroundColor: haveStart
                                       ? (dark &&
convertHexToRGBA(COLORS.BUTTON_THEME_LIGHT, 80)) ||
                                         convertHexToRGBA(COLORS.TEXT_THEME_LIGHT, 50)
                                       : undefined,
                                   borderColor: dark ? COLORS.THEME_LIGHT :
COLORS.THEME_DARK,
                               },
                           ]}
                           onPress={() => {
                               if (haveEnd && endDate) {
                                   setStartDate(endDate);
                                   setStartTimeString('00:00');
                               }
                               if (haveStart) {
                                   setDateString(undefined);
                                   calendarRef.current?.resetSelections();
                                   setHaveEnd(false);
                                   setStartDate(undefined);
                                   setEndDate(undefined);
                                   setStartTimeString(undefined);
                                   setEndTimeString(undefined);
                               } else if (!haveEnd) {
                                   setHaveEnd(true);
                                   setEndTimeString('00:00');
                                   if (!startTimeString) setStartTimeString('00:00');
                               }
                               setHaveStart(!haveStart);
                           }}
                       />
                       {haveStart && (
                           <View style={{ flexDirection: 'row' }}>
                               <DateTimeTextInput
                                   formatType="date"
                                   onChangeText={(text) => {
                                       setDateString(text);
                                       setStartDate(text ?
createDateFromCustomDateString(text) : undefined);
                                       setEndDate(text ?
createDateFromCustomDateString(text) : undefined);
                                       if
(!Number.isNaN(createHandleOnPressDayArgs(text).year))
calendarRef.current?.handleOnPressDay(createHandleOnPressDayArgs(text));
                                   }}
                                   value={dateString}
                                   style={{ marginRight: 20 }}
                               />
                               <DateTimeTextInput
                                   formatType="time"
                                   defaultValue={startTimeString || '00:00'}
                                   onChangeText={(text) => {
                                       setStartTimeString(text);
                                   }}
                               />
                           </View>
                       )}
                   </View>
                   <Text style={[dark ? typography.textDarkTheme :
typography.textLightTheme, { marginVertical: 6 }]}>
                       Окончание
                   </Text>
                   <View style={{ flexDirection: 'row' }}>
                       <Pressable
                           style={[
                               styles.selectionBox,
                               {
                                   backgroundColor: haveEnd
                                       ? (dark &&
convertHexToRGBA(COLORS.BUTTON_THEME_LIGHT, 80)) ||
                                         convertHexToRGBA(COLORS.TEXT_THEME_LIGHT, 50)
                                       : undefined,
                                   borderColor: dark ? COLORS.THEME_LIGHT :
COLORS.THEME_DARK,
                               },
                           ]}
                           onPress={() => {
                               if (haveEnd && !haveStart) {
                                   setDateString(undefined);
                                   setStartDate(undefined);
                                   setEndDate(undefined);
                                   setEndTimeString(undefined);
                                   calendarRef.current?.resetSelections();
                               }
                               if (!haveEnd) setEndTimeString('00:00');
                               if (!haveStart) setHaveEnd(!haveEnd);
                           }}
                       />
                       {haveEnd && (
                           <View style={{ flexDirection: 'row', alignItems: 'center'
}}>
                               {!haveStart && (
                                   <DateTimeTextInput
                                       formatType="date"
                                       onChangeText={(text) => {
                                           setDateString(text);
                                           setEndDate(text ?
createDateFromCustomDateString(text) : undefined);
                                           if
(!Number.isNaN(createHandleOnPressDayArgs(text).year))

calendarRef.current?.handleOnPressDay(createHandleOnPressDayArgs(text));
                                       }}
                                       style={{ marginRight: 20 }}
                                       value={dateString}
                                   />
                               )}
                               <DateTimeTextInput
                                   formatType="time"
                                   defaultValue={endTimeString || '00:00'}
                                   onChangeText={(text) => {
                                       setEndTimeString(text);
                                   }}
                               />
                           </View>
                       )}
                   </View>
               </View>
           </Shadow>
       </View>
   );
};


./src/components/atomic/ChooseDateBar/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
   main: { width: 282, paddingTop: 6, paddingHorizontal: 9, paddingBottom: 21,
borderRadius: 10 },
   icon: {
       width: 10,
       height: 15,
       resizeMode: 'contain',
   },
   selectedDay: {
       width: 24,
       height: 24,
       borderRadius: 5,
   },
   calendarWrapper: {
       width: 253,
       borderRadius: 10,
       borderWidth: 0.5,
       paddingTop: 10,
       alignSelf: 'center',
   },
   daysFont: {
       fontSize: 14,
   },
   header: {
       fontWeight: 'bold',
   },
   selectionBox: {
       width: 20,
       height: 20,
       borderWidth: 1,
       borderRadius: 5,
       marginVertical: 6,
       marginRight: 20,
   },
});


./src/components/atomic/CountCard/CountCard.tsx

import { useTheme } from '@react-navigation/native';
import React from 'react';
import { PressableProps, Text, Pressable, StyleProp, ViewStyle } from 'react-native';
import { useAppSelector } from '../../../store/hooks';
import { styles } from './styles';
import { COLORS } from '../../../../assets/colors';

interface CountCardProps extends PressableProps {
   ending: string;
   style?: StyleProp<ViewStyle>;
}

export enum COUNTWORDS {
   LEFT = 'осталось',
   TOTAL = 'всего',
}

export const CountCard: React.FC<CountCardProps> = ({ ending, ...props }) => {
   const { colors: themeColors, dark } = useTheme();
   const { style, onPress } = props;
   const textColor = dark ? COLORS.THEME_LIGHT : themeColors.text;
   const tasks = useAppSelector((state) => state.todos.todos);
   const count = ending === 'всего' ? tasks.length : tasks.filter((item) =>
item.completed === false).length;
   let word: string = 'задач';
   if (count <= 4 || count > 20) {
       if (count % 10 === 1) word = 'задача';
       else if (count % 10 > 1 && count % 10 < 5) word = 'задачи';
   }
   return (
       <Pressable
           style={[style, styles.card, { backgroundColor: dark ? COLORS.TASK_THEME_DARK
: themeColors.background }]}
           onPress={onPress}
       >
           <Text style={[styles.number, { color: textColor }]}> {count.toString()}
</Text>
           <Text style={[styles.text, { color: textColor }]}>
               {word} {ending === 'осталось' && word === 'задача' ? 'осталась' :
ending}
           </Text>
       </Pressable>
   );
};



./src/components/atomic/CountCard/styles.ts
```

```
import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    card: {
        width: 124,
        height: 66,
        backgroundColor: COLORS.BUTTON_THEME_LIGHT,
        paddingLeft: 8,
        paddingTop: 4,
        borderRadius: 10,
    },
    text: {
        height: 20,
        fontFamily: 'light-poppins',
        fontSize: 14,
    },
    number: {
        fontFamily: 'bold-poppins',
        justifyContent: 'flex-start',
        fontSize: 24,
        textAlign: 'left',
        height: 34,
        width: 34,
    },
});
```

./src/components/atomic/DateTimeTextInput/DateTimeTextInput.tsx

```
import React, { useState } from 'react';
import { useTheme } from '@react-navigation/native';
import { TextInput, TextInputProps } from 'react-native';
import { COLORS } from '../../../../assets/colors';
import { typography } from '../../../../assets/globalStyles';
import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';
import { styles } from './style';

interface DateTimeTextInputProps extends TextInputProps {
    formatType: 'time' | 'date';
}

export const DateTimeTextInput: React.FC<DateTimeTextInputProps> = ({
```



```
   formatType,
   onChangeText,
   style: customStyle,
   defaultValue,
   value: settedValue,
}) => {
   const { dark } = useTheme();
   const placeholder = formatType === 'time' ? 'ЧЧ:ММ' : 'ДД.ММ.ГГГГ';
   const mask =
       formatType === 'time'
           ? new RegExp(

'^$|^[0-2]$|^([0-1][0-9]|2[0-3])$|^([0-1][0-9]|2[0-3]):$|^(([0-1][0-9]|2[0-3]):[0-5])$|
^(([0-1][0-9]|2[0-3]):[0-5][0-9])$',
             )
           : new RegExp(

'^[0-3]$|^(0[1-9]|[12][0-9]|3[01]).?$|^(0[1-9]|[12][0-9]|3[01]).[01]$|^(0[1-9]|[12][0-9
]|3[01]).(0[1-9]|1[0-2]).?$|^(0[1-9]|[12][0-9]|3[01]).(0[1-9]|1[0-2]).[0-9]$|^(0[1-9]|[
12][0-9]|3[01]).(0[1-9]|1[0-2]).[0-9][0-9]$|^(0[1-9]|[12][0-9]|3[01]).(0[1-9]|1[0-2]).[
0-9][0-9][0-9]$|^(0[1-9]|[12][0-9]|3[01]).(0[1-9]|1[0-2]).[0-9][0-9][0-9][0-9]$|^$',
             );
   const finalMask =
       formatType === 'time'
           ? new RegExp('^$|^(([0-1][0-9]|2[0-3]):[0-5][0-9])$')
           : new RegExp(

'^$|^(((0[1-9]|[12][0-9]|3[01])([.])(0[13578]|10|12)([.])([0-9]{4}))|(([0][1-9]|[12][0-
9]|30)([.])(0[469]|11)([.])([0-9]{4}))|((0[1-9]|1[0-9]|2[0-8])([.])(02)([.])([0-9]{4}))
|((29)(.)(02)([.])([02468][048]00))|((29)([.])(02)([.])([13579][26]00))|((29)([.])(02)(
[.])([0-9][0-9][0][48]))|((29)([.])(02)([.])([0-9][0-9][2468][048]))|((29)([.])(02)([.]
)([0-9][0-9][13579][26])))$',
             );
   const matchedDefaultValue = defaultValue?.match(finalMask);
   const [value, setValue] = useState(matchedDefaultValue ? matchedDefaultValue[0] :
'');
   const timeValidator = (text: string) => {
       if (text.length === 3 && value.length < 3) text = `${text[0] +
text[1]}:${text[2]}`;
       const masked = text.match(mask);
       if (masked !== null) {
           if (masked[0].length === 2 && value.length < 2)
setValue(masked[0].concat(':'));
```

```
            else setValue(masked[0]);
        }
    };
    const dateValidator = (text: string) => {
        if (text.length === 3 && value.length < 3) text = `${text[0] +
text[1]}.${text[2]}`;
        if (text.length === 6 && value.length < 6) text = `${text[0] +
text[1]}.${text[3] + text[4]}.${text[5]}`;
        const masked = text.match(mask);
        if (masked !== null) {
            if ((masked[0].length === 2 || masked[0].length === 5) && value.length <
masked[0].length)
                setValue(masked[0].concat('.'));
            else setValue(masked[0]);
        }
    };
    const validator = formatType === 'time' ? timeValidator : dateValidator;
    return (
        <TextInput
            defaultValue={defaultValue}
            keyboardType="numeric"
            style={[
                customStyle,
                styles.default,
                {
                    borderColor: dark ? COLORS.THEME_LIGHT : COLORS.BUTTON_THEME_LIGHT,
                    width: formatType === 'time' ? 50 : 80,
                    backgroundColor: value.match(finalMask) ? undefined :
convertHexToRGBA(COLORS.EXPIRED, 30),
                },
                dark ? typography.textDarkTheme : typography.textLightTheme,
            ]}
            onChangeText={(text) => {
                if (text.match(finalMask) && onChangeText) onChangeText(text);
                else if (onChangeText && !(text.length === 6 || text.length === 11))
onChangeText('');
            }}
            onChange={(e) => {
                const { text } = e.nativeEvent;
                validator(text);
            }}
            value={settedValue || value}
            placeholder={placeholder}
            placeholderTextColor={dark ? convertHexToRGBA(COLORS.TEXT_HELP, 30) :
COLORS.TEXT_HELP}
        />
    );
};


./src/components/atomic/DateTimeTextInput/style.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    default: {
        textAlign: 'center',
        borderWidth: 0.5,
        borderRadius: 5,
        height: 20,
    },
});


./src/components/atomic/DescriptionTextInput/DescriptionTextInput.tsx

import React from 'react';
import { useTheme } from '@react-navigation/native';
import { TextInput, TextInputProps } from 'react-native';
import { COLORS } from '../../../../assets/colors';
import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';
import { styles } from './styles';

export interface InputProps extends TextInputProps {
    ref: HTMLElement | undefined;
}

export const DescriptionTextInput = React.forwardRef<TextInput, InputProps>(({ ...props
}, ref) => {
    const { defaultValue, onChangeText, style } = props;
    const { dark } = useTheme();
    return (
        <TextInput
            ref={ref}
            multiline={!false}
            onChangeText={onChangeText}
            defaultValue={defaultValue === undefined || defaultValue === '' ? undefined : defaultValue}
            placeholder={defaultValue === undefined || defaultValue === '' ? 'Описание...' : undefined}
            placeholderTextColor={dark ? convertHexToRGBA(COLORS.TEXT_HELP, 30) : COLORS.TEXT_HELP}
            style={[style, styles.wrapper, { color: dark ? COLORS.THEME_LIGHT : COLORS.TEXT_THEME_LIGHT }]}
        />
    );
});


./src/components/atomic/DescriptionTextInput/styles.ts

import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    wrapper: {
        height: 174,
        borderBottomWidth: 1,
        borderBottomColor: COLORS.BUTTON_THEME_LIGHT,
        textAlignVertical: 'top',
        fontFamily: 'light-poppins',
        fontSize: 18,
        alignItems: 'stretch',
    },
});


./src/components/atomic/DropdownButton/DropdownButton.tsx

import { useTheme } from '@react-navigation/native';
import React, { useRef, useState } from 'react';
import {
    Pressable,
    Image,
    Text,
    View,
    StyleProp,
    ViewStyle,
    ImageStyle,
    PressableProps,
    Animated,
    Easing,
    Platform,
} from 'react-native';
import { ICONS } from '../../../../assets';
import { typography } from '../../../../assets/globalStyles';

interface DropdownBarProps extends PressableProps {
    title: string;
    component: React.ReactNode;
    style?: StyleProp<ViewStyle>;
    arrowStyle?: StyleProp<ImageStyle>;
    titleStyle?: StyleProp<ViewStyle>;
}

export const DropdownButton: React.FC<DropdownBarProps> = ({
    title,
    component,
    style: customStyle,
    arrowStyle,
    titleStyle,
    ...props
}) => {
    const [platform] = useState(Platform.OS);
    const { onPress } = props;
    const [open, setOpen] = useState(false);
    const { dark } = useTheme();
    let icon: ICONS;
    if (dark) {
        if (open) icon = ICONS.imgUpArrowDarkTheme;
        else icon = ICONS.imgDownArrowDarkTheme;
    } else if (open) icon = ICONS.imgUpArrowLightTheme;
    else icon = ICONS.imgDownArrowLightTheme;
    const dropAnim = useRef(new Animated.Value(0)).current;
    const dropScale = dropAnim.interpolate({
        inputRange: [0, 0.5, 1],
        outputRange: [0, 0.5, 1],
    });

    const fadeAnim = useRef(new Animated.Value(0)).current;
    const fadeScale = fadeAnim.interpolate({
        inputRange: [0, 1],
        outputRange: [0, 1],
    });

    return (
        <View style={[customStyle]}>
            <Pressable
                onPress={(e) => {
                    if (onPress) onPress(e);
                    Animated.timing(fadeAnim, {
                        toValue: open ? 0 : 1,
                        duration: platform === 'ios' ? 500 : 1000,
                        useNativeDriver: true,
                        easing: Easing.linear,
                    }).start();

                    Animated.timing(dropAnim, {
                        toValue: open ? 0 : 1,
                        duration: 200,
                        easing: Easing.linear,
                        useNativeDriver: true,
                    }).start();
                    setOpen(!open);
                }}
                style={[{ flexDirection: 'row', alignItems: 'center' }, titleStyle]}
            >
                <Text style={[dark ? typography.textDarkTheme :
 typography.textLightTheme, { fontSize: 14 }]}>
                    {title}
                </Text>
                <Image
                    source={icon}
                    style={[{ width: 20, height: 10, resizeMode: 'contain', marginLeft:
 6 }, arrowStyle]}
                />
            </Pressable>
            <Animated.View
                style={{
                    transform: platform === 'ios' ? [{ scaleY: dropScale }] : [{ scaleY:
 !open ? 0 : 1 }],
                    opacity: fadeScale,
                    justifyContent: 'flex-start',
                    height: !open ? 0 : undefined,
                }}
```

```
            >
                {component}
            </Animated.View>
        </View>
    );
};


./src/components/atomic/GradientFooter/GradientFooter.tsx

import React from 'react';
import { View, ViewProps } from 'react-native';
import { LinearGradient } from 'expo-linear-gradient';
import { useTheme } from '@react-navigation/native';
import { COLORS } from '../../../../assets/colors';
import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';

export const GradientFooter: React.FC<ViewProps> = ({ ...props }) => {
    const { dark } = useTheme();
    return (
        <View {...props}>
            <LinearGradient
                colors={[
                    convertHexToRGBA(dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT, 1),
                    convertHexToRGBA(dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT, 5),
                    convertHexToRGBA(dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT, 20),
                    convertHexToRGBA(dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT, 40),
                    convertHexToRGBA(dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT, 60),
                    convertHexToRGBA(dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT, 80),
                    convertHexToRGBA(dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT, 90),
                    convertHexToRGBA(dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT, 100),
                ]}
                style={{ height: 60 }}
            />
        </View>
    );
};


./src/components/atomic/IconActionButton/IconActionButton.tsx

import React from 'react';
import { Pressable, PressableProps, Image, ImageSourcePropType, ViewStyle, StyleProp,
View } from 'react-native';
import { styles } from './styles';

interface IconActionButtonProps extends PressableProps {
    icon: ImageSourcePropType;
    style?: StyleProp<ViewStyle>;
    iconSize?: number;
}
export const IconActionButton: React.FC<IconActionButtonProps> = ({ icon, style,
iconSize, ...props }) => {
    const { onPress } = props;
    return (
        <View style={[{ width: 40 }, style]}>
            <Pressable style={styles.iconButton} onPress={onPress}>
                <Image style={(iconSize && { width: iconSize, height: iconSize }) ||
styles.icon} source={icon} />
            </Pressable>
        </View>
    );
};


./src/components/atomic/IconActionButton/styles.ts

import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    iconButton: {
        backgroundColor: COLORS.PINK,
        height: 40,
        width: '100%',
        borderRadius: 10,
        justifyContent: 'center',
        alignItems: 'center',
    },
    icon: {
        width: 15,
        height: 15,
    },
});
```

./src/components/atomic/IconAddCategoryButton/IconAddCategoryButton.tsx

```
import React from 'react';
import { Image, StyleProp, ViewStyle, SafeAreaView } from 'react-native';
import { useTheme } from '@react-navigation/native';
import { TouchableOpacity } from 'react-native-gesture-handler';
import { GenericTouchableProps } from
'react-native-gesture-handler/lib/typescript/components/touchables/GenericTouchable';
import { COLORS } from '../../../../assets/colors';
import { ICONS } from '../../../../assets/index';
import { styles, Size } from './styles';

interface IconAddCategoryButtonProps extends GenericTouchableProps {
    style?: StyleProp<ViewStyle>;
    type: Size;
}

export const IconAddCategoryButton: React.FC<IconAddCategoryButtonProps> = ({ style,
type, ...props }) => {
    const { onPress } = props;
    const { dark } = useTheme();
    const choosenSize = type === 'small' ? styles.iconAddCategoryButtonSmall :
styles.iconAddCategoryButtonBig;
    return (
        <SafeAreaView>
            <TouchableOpacity
                style={[
                    choosenSize,
                    style,
                    { backgroundColor: dark ? COLORS.TASK_THEME_DARK :
COLORS.BUTTON_THEME_LIGHT },
                ]}
                onPress={onPress}
            >
                <Image style={styles.icon} source={ICONS.imgAddIconWhite} />
            </TouchableOpacity>
        </SafeAreaView>
    );
};


./src/components/atomic/IconAddCategoryButton/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    iconAddCategoryButtonSmall: {
        height: 40,
        width: 40,
        borderRadius: 10,
        justifyContent: 'center',
        alignItems: 'center',
    },
    iconAddCategoryButtonBig: {
        height: 61,
        width: 132,
        borderRadius: 5,
        justifyContent: 'center',
        alignItems: 'center',
    },
    icon: {
        width: 15,
        height: 15,
    },
    margins: {
        marginTop: 10,
        marginLeft: 10,
    },
});

export enum Size {
    smallButtonSizes = 'small',
    bigButtonSizes = 'big',
}


./src/components/atomic/IconSquareActionButton/IconSquareActionButton.tsx
```

```
import React from 'react';
import { Pressable, Image, PressableProps, ViewStyle, StyleProp } from 'react-native';
import { ICONS } from '../../../../assets';
import { COLORS } from '../../../../assets/colors';
import { styles } from './styles';

interface IconSquareActionButtonProps extends PressableProps {
   type: 'delete' | 'edit';
   style?: StyleProp<ViewStyle>;
}

export const IconSquareActionButton: React.FC<IconSquareActionButtonProps> = ({ type,
...props }) => {
   const icon = type === 'delete' ? ICONS.imgTrashIconWhite : ICONS.imgEditIconWhite;
   const color = type === 'delete' ? COLORS.PINK : COLORS.BLUE;
   return (
       <Pressable {...props} style={[styles.button, props.style, { backgroundColor:
color }]}>
           <Image source={icon} style={styles.icon} />
       </Pressable>
   );
};
```

./src/components/atomic/IconSquareActionButton/styles.ts

```
import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
   button: { width: 65, height: 61, alignItems: 'center', justifyContent: 'center' },
   icon: { width: 26, height: 26, resizeMode: 'contain' },
});
```

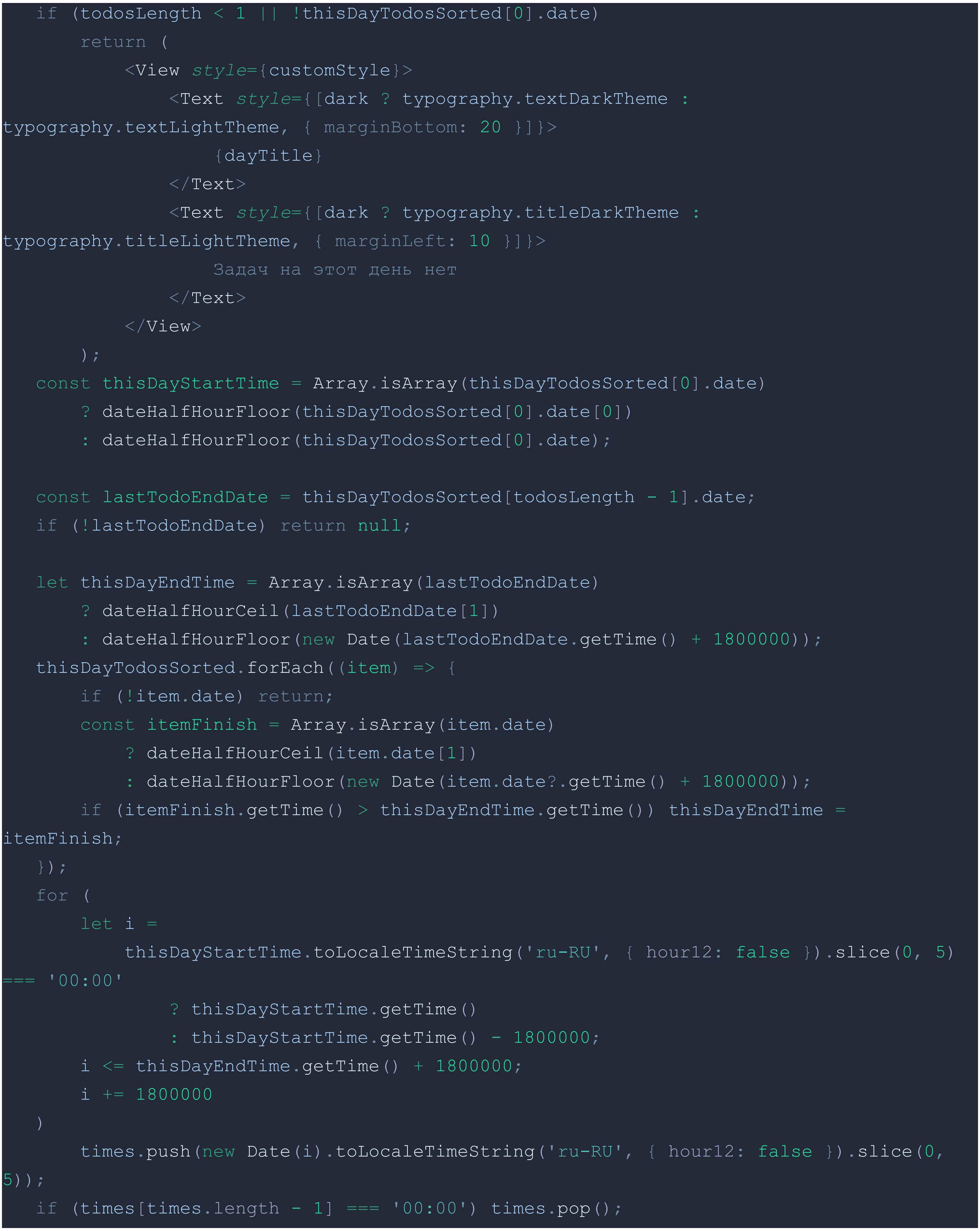
./src/components/atomic/IconTransparentButton/IconTransparentButton.tsx

```
import React from 'react';
import { Pressable, PressableProps, Image, ImageSourcePropType } from 'react-native';

interface IconTransparentButtonProps extends PressableProps {
   icon: ImageSourcePropType;
   iconWidth?: number;
   iconHeight?: number;
```

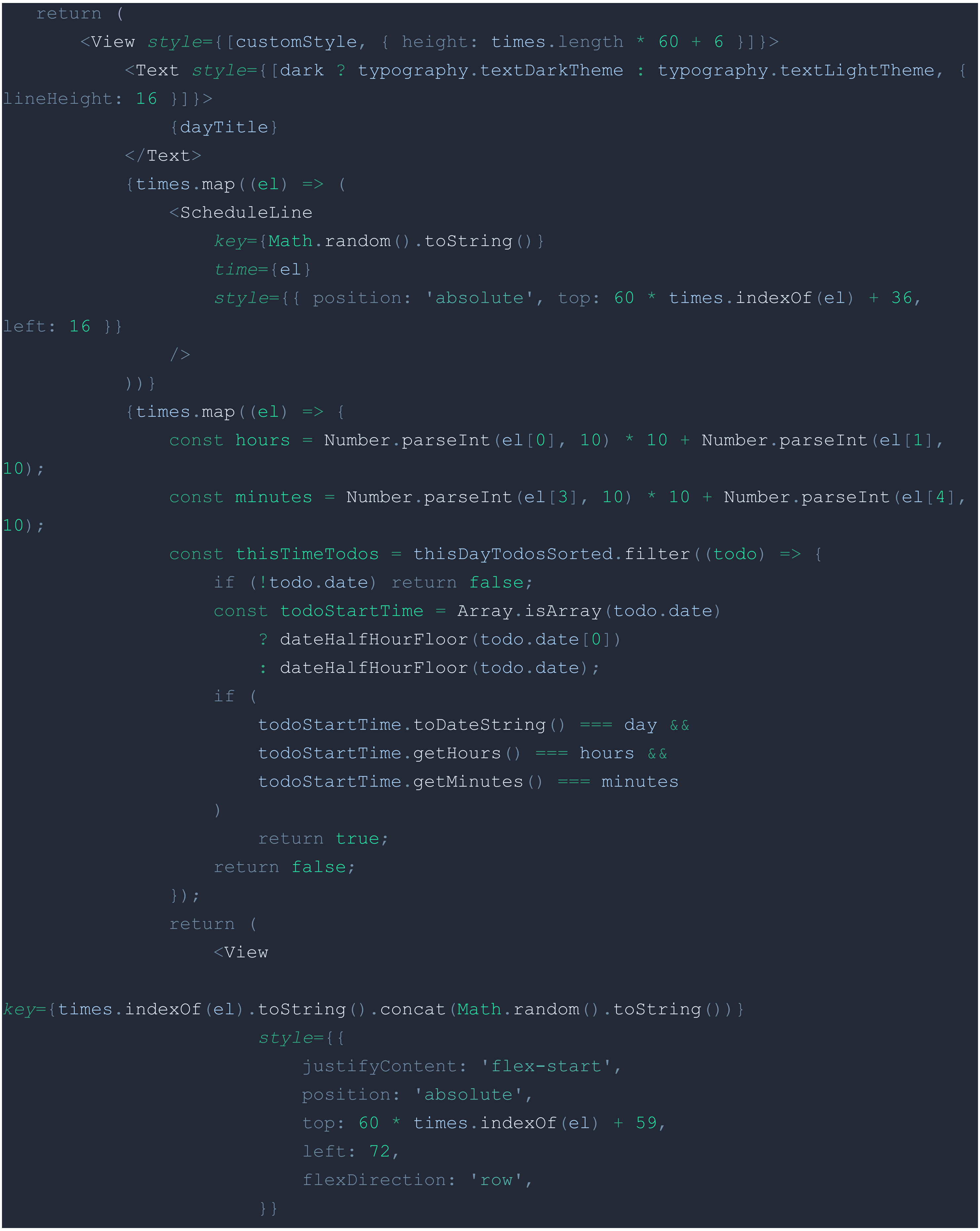
```
}

export const IconTransparentButton: React.FC<IconTransparentButtonProps> = ({
   icon,
   iconHeight = 24,
   iconWidth = 24,
   style: customStyle,
   onPress,
}) => (
   <Pressable style={customStyle} onPress={onPress}>
       <Image source={icon} style={{ width: iconWidth, height: iconHeight }} />
   </Pressable>
);


./src/components/atomic/ModalColorPicker/ModalColorPicker.tsx

import { useTheme } from '@react-navigation/native';
import React, { useEffect, useMemo, useState } from 'react';
import { Dimensions, LogBox, Modal, ModalProps, Pressable, View } from 'react-native';
import ColorPicker from 'react-native-color-picker-wheel';
import { COLORS } from '../../../../assets/colors';
import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';
import { ActionButton } from '../ActionButton/ActionButton';
import { styles } from './styles';

interface ModalColorPickerProps extends ModalProps {
   onColorSelected?: (color: string) => void;
   initialColor?: string;
   onMissClick?: () => void;
}

export const ModalColorPicker: React.FC<ModalColorPickerProps> = ({
   onColorSelected,
   initialColor,
   onMissClick,
   ...props
}) => {
   const { colors: themeColors, dark } = useTheme();
   const blurColor = useMemo(() => (dark ? COLORS.THEME_DARK :
COLORS.BUTTON_THEME_LIGHT), [dark]);
   const [color, setColor] = useState<string>(initialColor || COLORS.THEME_LIGHT);
   const [visible, setVisible] = useState(false);
```

```
    useEffect(() => {
        setVisible(props.visible || false);
    }, [props.visible]);
    useEffect(() => {
        LogBox.ignoreLogs(['Animated: `useNativeDriver`']);
    }, []);
    return (
        <Modal {...props} transparent visible={visible} animationType="fade">
            <View
                style={{
                    width: Dimensions.get('screen').width,
                    height: Dimensions.get('screen').height,
                }}
            >
                <Pressable
                    style={{
                        backgroundColor: convertHexToRGBA(blurColor, 80),
                        height: '100%',
                        width: '100%',
                        justifyContent: 'center',
                        alignItems: 'center',
                    }}
                    onPress={() => {
                        setVisible(false);
                        if (onMissClick) onMissClick();
                        if (onColorSelected && initialColor)
onColorSelected(initialColor);
                    }}
                >
                    <View style={[styles.main, { backgroundColor: themeColors.card }]}>
                        <View style={styles.colorPickerWrapper}>
                            <ColorPicker
                                row={false}
                                noSnap
                                color={color}
                                thumbSize={40}
                                sliderSize={20}
                                discrete
                                onColorChange={(newColor: string) => {
                                    setColor(newColor);
                                }}
                                onColorChangeComplete={(newColor: string) => {
                                    setColor(newColor);
                                }}
                                swatchesLast
                                swatchesOnly={false}
                                shadeSliderThumb={false}
                                shadeWheelThumb={false}
                                swatches={false}
                                autoResetSlider={false}
                            />
                        </View>
                        <ActionButton
                            title="Выбрать цвет"
                            onPress={() => {
                                if (onColorSelected) onColorSelected(color);
                                setVisible(false);
                            }}
                            active
                        />
                    </View>
                </Pressable>
            </View>
        </Modal>
    );
};


./src/components/atomic/ModalColorPicker/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    main: {
        width: 235,
        height: 340,
        borderRadius: 10,
        justifyContent: 'center',
        alignItems: 'center',
        padding: 5,
    },
    colorPickerWrapper: {
        height: 240,
        marginBottom: 24,
    },
});
```

```
./src/components/atomic/MoreColorsButton/MoreColorsButton.tsx

import { useTheme } from '@react-navigation/native';
import React from 'react';
import { Text, Pressable, PressableProps, Image, View, StyleProp, ViewStyle } from 'react-native';
import { ICONS } from '../../../../assets';
import { styles } from './styles';

interface MoreColorsButtonProps extends PressableProps {
   style?: StyleProp<ViewStyle>;
}

export const MoreColorsButton: React.FC<MoreColorsButtonProps> = ({ style, ...props }) => {
   const { onPress } = props;
   const { colors: themeColors, dark } = useTheme();
   const icon = dark ? ICONS.imgColorsAddWhiteIconDarkTheme : ICONS.imgColorsAddBlackIconLightTheme;
   return (
       <View style={style}>
           <Pressable style={[styles.button, { backgroundColor: themeColors.background }]} onPress={onPress}>
               <Text style={[styles.text, { color: themeColors.text }]}>Больше цветов</Text>
               <Image source={icon} style={styles.icon} />
           </Pressable>
       </View>
   );
};


./src/components/atomic/MoreColorsButton/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
   button: { width: 130, alignItems: 'center', justifyContent: 'center', flexDirection: 'row' },
   icon: { width: 16, height: 16 },
   text: { fontFamily: 'light-poppins', fontSize: 14, marginRight: 5 },
});



./src/components/atomic/OneDaySchedule/OneDaySchedule.tsx

import React from 'react';
import { useTheme } from '@react-navigation/native';
import { View, Text, ViewProps } from 'react-native';
import { StackNavigationProp } from '@react-navigation/stack';
import { useAppSelector } from '../../../store/hooks';
import { todosSortByDateAscending } from '../../../utils/todosSortByDate';
import { dateHalfHourCeil, dateHalfHourFloor } from '../../../utils/dateHalfHourRound';
import { ScheduleLine } from '../ScheduleLine/ScheduleLine';
import { TodoScheduleCard } from '../TodoScheduleCard/TodoScheduleCard';
import { typography } from '../../../../assets/globalStyles';
import { getRUMonthFromDate } from '../../../utils/getRUMonthFromDate';
import { RootStackParamList } from '../../RootStackParams';

interface OneDayScheduleProps extends ViewProps {
   date: Date;
   navigation?: StackNavigationProp<RootStackParamList, 'MainPage'>;
}
export const OneDaySchedule: React.FC<OneDayScheduleProps> = ({ date, style:
customStyle, navigation }) => {
   const { dark } = useTheme();
   const times: string[] = [];
   const day = date.toDateString();
   let dayTitle = `${date.getDate()} ${getRUMonthFromDate(date)}`;
   if (day === new Date(Date.now()).toDateString()) dayTitle = `Сегодня, ${dayTitle}`;
   else if (day === new Date(Date.now() + 86400000).toDateString()) dayTitle = `Завтра,
${dayTitle}`;
   const thisDayTodosSorted = todosSortByDateAscending(
       useAppSelector((state) =>
           state.todos.todos.filter((todo) => {
               if (!todo.date || todo.dateType === 'WITHOUT') return false;
               return Array.isArray(todo.date)
                   ? todo.date[0].toDateString() === day
                   : todo.date.toDateString() === day;
           }),
       ),
   );

   const todosLength = thisDayTodosSorted.length;
```



```
    if (todosLength < 1 || !thisDayTodosSorted[0].date)
        return (
            <View style={customStyle}>
                <Text style={[dark ? typography.textDarkTheme :
typography.textLightTheme, { marginBottom: 20 }]}>
                    {dayTitle}
                </Text>
                <Text style={[dark ? typography.titleDarkTheme :
typography.titleLightTheme, { marginLeft: 10 }]}>
                    Задач на этот день нет
                </Text>
            </View>
        );
    const thisDayStartTime = Array.isArray(thisDayTodosSorted[0].date)
        ? dateHalfHourFloor(thisDayTodosSorted[0].date[0])
        : dateHalfHourFloor(thisDayTodosSorted[0].date);

    const lastTodoEndDate = thisDayTodosSorted[todosLength - 1].date;
    if (!lastTodoEndDate) return null;

    let thisDayEndTime = Array.isArray(lastTodoEndDate)
        ? dateHalfHourCeil(lastTodoEndDate[1])
        : dateHalfHourFloor(new Date(lastTodoEndDate.getTime() + 1800000));
    thisDayTodosSorted.forEach((item) => {
        if (!item.date) return;
        const itemFinish = Array.isArray(item.date)
            ? dateHalfHourCeil(item.date[1])
            : dateHalfHourFloor(new Date(item.date?.getTime() + 1800000));
        if (itemFinish.getTime() > thisDayEndTime.getTime()) thisDayEndTime =
itemFinish;
    });
    for (
        let i =
            thisDayStartTime.toLocaleTimeString('ru-RU', { hour12: false }).slice(0, 5)
=== '00:00'
                ? thisDayStartTime.getTime()
                : thisDayStartTime.getTime() - 1800000;
        i <= thisDayEndTime.getTime() + 1800000;
        i += 1800000
    )
        times.push(new Date(i).toLocaleTimeString('ru-RU', { hour12: false }).slice(0,
5));
    if (times[times.length - 1] === '00:00') times.pop();
```



```
return (
    <View style={[customStyle, { height: times.length * 60 + 6 }]}>
        <Text style={[dark ? typography.textDarkTheme : typography.textLightTheme, { lineHeight: 16 }]}>
            {dayTitle}
        </Text>
        {times.map((el) => (
            <ScheduleLine
                key={Math.random().toString()}
                time={el}
                style={{ position: 'absolute', top: 60 * times.indexOf(el) + 36, left: 16 }}
            />
        ))}
        {times.map((el) => {
            const hours = Number.parseInt(el[0], 10) * 10 + Number.parseInt(el[1], 10);
            const minutes = Number.parseInt(el[3], 10) * 10 + Number.parseInt(el[4], 10);
            const thisTimeTodos = thisDayTodosSorted.filter((todo) => {
                if (!todo.date) return false;
                const todoStartTime = Array.isArray(todo.date)
                    ? dateHalfHourFloor(todo.date[0])
                    : dateHalfHourFloor(todo.date);
                if (
                    todoStartTime.toDateString() === day &&
                    todoStartTime.getHours() === hours &&
                    todoStartTime.getMinutes() === minutes
                )
                    return true;
                return false;
            });
            return (
                <View
                    key={times.indexOf(el).toString().concat(Math.random().toString())}
                    style={{
                        justifyContent: 'flex-start',
                        position: 'absolute',
                        top: 60 * times.indexOf(el) + 59,
                        left: 72,
                        flexDirection: 'row',
                    }}
                    >
                        {thisTimeTodos.map((ttt) => (
                            <TodoScheduleCard
                                todo={ttt}
                                key={ttt.key}
                                onPress={() => navigation?.push('TaskPage', { todo: ttt
})}
                            />
                        ))}
                    </View>
                );
            })}
        </View>
    );
};


./src/components/atomic/PaletteBar/PaletteBar.tsx

import React, { useEffect, useState } from 'react';
import { View, StyleProp, ViewStyle, Pressable, PressableProps } from 'react-native';
import { COLORS } from '../../../../assets/colors';
import { SelectCategoryColor } from '../SelectCategoryColor/SelectCategoryColor';
import { styles } from './styles';

interface PaletteBarProps extends PressableProps {
    colors: string[];
    style?: StyleProp<ViewStyle>;
    initColor?: string;
    onColorSelected?: (selectedColor: string) => void;
}

export const PaletteBar: React.FC<PaletteBarProps> = ({ colors, style, initColor,
onColorSelected, ...props }) => {
    const [color, setColor] = useState<string | undefined>(initColor || undefined);
    useEffect(() => {
        setColor(initColor);
    }, [initColor]);
    let selectedIndex = -1;
    colors.map((item, index) => {
        if (item === initColor) selectedIndex = index;
        return item;
    });
```

```
   if (selectedIndex !== -1) colors.unshift(...colors.splice(selectedIndex, 1));
   const { onPress } = props;
   useEffect(() => {
       if (onColorSelected && color) {
           onColorSelected(color);
       }
   }, [color]);
   return (
       <View style={style}>
           <Pressable style={styles.card} onPress={onPress}>
               {colors.map((item) => (
                   <SelectCategoryColor
                       color={item}
                       selected={item === color}
                       key={Math.random().toString().slice(2, 9)}
                       style={{ marginHorizontal: 12 }}
                       onPress={() => setColor(item)}
                   />
               ))}
           </Pressable>
       </View>
   );
};

export const categoryColors: string[] = [COLORS.PINK, COLORS.PURPLE, COLORS.TIFFANY,
COLORS.BLUE];


./src/components/atomic/PaletteBar/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
   card: {
       flexDirection: 'row',
       height: 50,
       alignItems: 'center',
       justifyContent: 'center',
   },
});


./src/components/atomic/ScheduleLine/ScheduleLine.tsx
```

```
import { useTheme } from '@react-navigation/native';
import React from 'react';
import { View, ViewProps, Text } from 'react-native';
import { styles } from './styles';

interface ScheduleLineProps extends ViewProps {
   time: string;
}

export const ScheduleLine: React.FC<ScheduleLineProps> = ({ time, ...props }) => {
   const { colors: themeColors, dark } = useTheme();
   const { style } = props;
   return (
       <View style={[styles.wrapper, style]}>
           <View>
               <Text style={[styles.time, { color: dark ? themeColors.text :
themeColors.text }]}>{time}</Text>
           </View>
           <View style={styles.line} />
       </View>
   );
};


./src/components/atomic/ScheduleLine/styles.ts

import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
   line: {
       width: '100%',
       height: 1.5,
       backgroundColor: COLORS.BUTTON_THEME_LIGHT,
   },
   wrapper: {
       height: 46,
       flexDirection: 'row',
       alignItems: 'center',
   },
   time: {
       fontFamily: 'light-poppins',
        fontSize: 14,
        width: 40,
        textAlign: 'center',
        marginRight: 20,
    },
});


./src/components/atomic/SelectCategoryCard/SelectCategoryCard.tsx

import { useTheme } from '@react-navigation/native';
import React from 'react';
import { Text, View, StyleProp, ViewStyle } from 'react-native';
import { Shadow } from 'react-native-shadow-2';
import { GenericTouchableProps } from
'react-native-gesture-handler/lib/typescript/components/touchables/GenericTouchable';
import { TouchableOpacity } from 'react-native-gesture-handler';
import { COLORS } from '../../../../assets/colors';
import { ICategory } from '../../../store/types';
import { styles } from './styles';
import { typography } from '../../../../assets/globalStyles';

interface CategoryCardProps extends GenericTouchableProps {
    category: ICategory;
    style?: StyleProp<ViewStyle>;
    selected?: boolean;
}

export const SelectCategoryCard: React.FC<CategoryCardProps> = ({ category, style,
selected, ...props }) => {
    const { onPress } = props;
    const { dark } = useTheme();
    selected = selected === undefined ? false : selected;
    const categoryColor = category.color;
    return (
        <View style={[style, styles.card]}>
            <Shadow
                distance={selected ? 1 : 0}
                startColor={dark ? COLORS.TEXT_HELP : COLORS.TEXT_THEME_LIGHT}
                radius={5}
                offset={selected ? [1, 1] : [0, 0]}
            >
                <TouchableOpacity style={[styles.card, { backgroundColor: categoryColor
}]} onPress={onPress}>
                    <Text numberOfLines={1} style={[styles.text,
typography.textDarkTheme]}>
                        {category.name}
                    </Text>
                </TouchableOpacity>
            </Shadow>
        </View>
    );
};
```

./src/components/atomic/SelectCategoryCard/styles.ts

```
import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    card: {
        height: 34,
        borderRadius: 5,
        alignSelf: 'flex-start',
        justifyContent: 'center',
    },
    text: {
        alignSelf: 'center',
        paddingHorizontal: 14,
        color: COLORS.THEME_LIGHT,
        maxWidth: 100,
    },
});
```

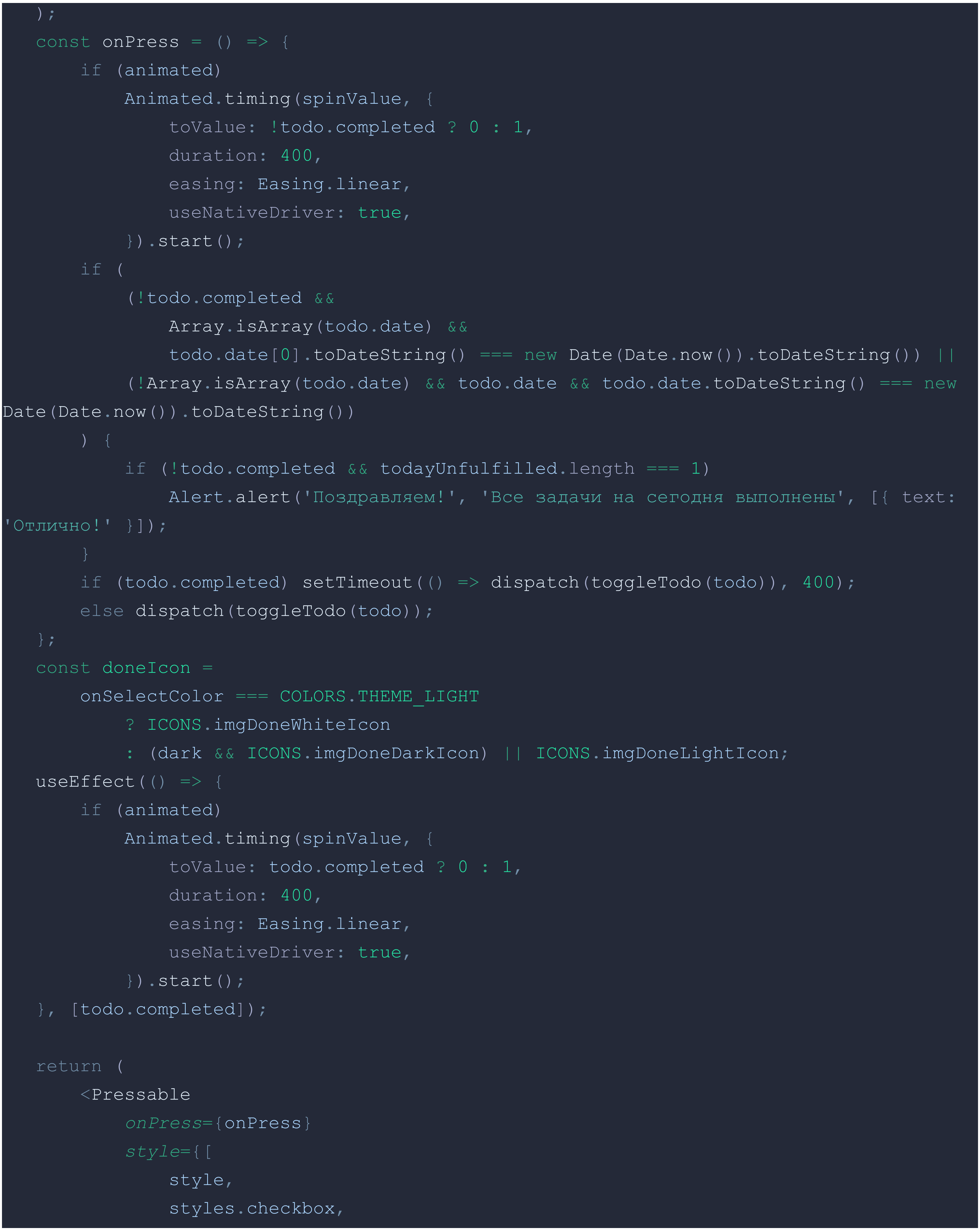
./src/components/atomic/SelectCategoryColor/SelectCategoryColor.tsx

```
import React from 'react';
import { useTheme } from '@react-navigation/native';
import { Shadow } from 'react-native-shadow-2';
import { View, StyleProp, ViewStyle, Pressable, PressableProps } from 'react-native';
import { COLORS } from '../../../../assets/colors';
import { styles } from './styles';

interface SelectCategoryColorProps extends PressableProps {
   style?: StyleProp<ViewStyle>;
   color?: string;
   selected?: boolean;
}

export const SelectCategoryColor: React.FC<SelectCategoryColorProps> = ({ style, color,
selected, ...props }) => {
   const { onPress } = props;
   const { dark } = useTheme();
   selected = selected === undefined ? false : selected;
   return (
       <View style={style}>
           <Shadow
               startColor={dark ? COLORS.TEXT_HELP : COLORS.TEXT_THEME_LIGHT}
               finalColor={dark ? COLORS.THEME_DARK : COLORS.THEME_LIGHT}
               radius={20}
               offset={selected ? [1, 1] : [0, 0]}
               distance={selected ? 2 : 0}
               containerViewStyle={styles.colorCategoryRound}
           >
               <Pressable style={[{ backgroundColor: color },
styles.colorCategoryRound]} onPress={onPress} />
           </Shadow>
       </View>
   );
};


./src/components/atomic/SelectCategoryColor/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
   colorCategoryRound: {
       justifyContent: 'center',
       height: 40,
       width: 40,
       borderRadius: 50,
   },
   margins: {
       marginTop: 10,
       marginLeft: 10,
   },
});


./src/components/atomic/ThemeButton/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
   button: {
       borderWidth: 1,
       width: 110,
       height: 30,
       borderRadius: 10,
       justifyContent: 'center',
       alignItems: 'center',
       marginVertical: 8,
   },
   selected: {
       width: 110,
       height: 30,
       borderRadius: 10,
       justifyContent: 'center',
       alignItems: 'center',
       marginVertical: 8,
   },
   text: {
       fontSize: 14,
       fontFamily: 'light-poppins',
   },
});


./src/components/atomic/ThemeButton/ThemeButton.tsx

import { useTheme } from '@react-navigation/native';
import React from 'react';
import { Text, ButtonProps, Pressable } from 'react-native';
import { COLORS } from '../../../../assets/colors';
import { styles } from './styles';

interface ThemeButtonProps extends ButtonProps {
   selected?: boolean;
}
```

```
export const ThemeButton: React.FC<ThemeButtonProps> = ({ selected, ...props }) => {
   const { colors: themeColors, dark: darkTheme } = useTheme();
   const { title, onPress } = props;
   return (
       <Pressable
           style={
               selected
                   ? [styles.selected, { backgroundColor: darkTheme ?
COLORS.TASK_THEME_DARK : COLORS.TEXT_HELP }]
                   : [
                         styles.button,
                         {
                             borderColor: darkTheme ? COLORS.TASK_THEME_DARK :
COLORS.TEXT_THEME_LIGHT,
                             backgroundColor: themeColors.background,
                         },
                     ]
           }
           onPress={onPress}
       >
           <Text style={[styles.text, { color: selected ? COLORS.THEME_LIGHT :
themeColors.text }]}>{title}</Text>
       </Pressable>
   );
};


./src/components/atomic/TitleTextInput/TitleTextInput.tsx

import React from 'react';
import { useTheme } from '@react-navigation/native';
import { TextInput, TextInputProps } from 'react-native';
import { COLORS } from '../../../../assets/colors';
import { typography } from '../../../../assets/globalStyles';
import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';

export const TitleTextInput: React.FC<TextInputProps> = ({ ...props }) => {
   const { dark } = useTheme();
   const { style: customStyle } = props;
   return (
       <TextInput
           {...props}
```

```
      placeholderTextColor={dark ? convertHexToRGBA(COLORS.TEXT_HELP, 30) : COLORS.TEXT_HELP}
      style={[customStyle, dark ? typography.titleDarkTheme : typography.titleLightTheme]}
    />
  );
};


./src/components/atomic/TodoCheckbox/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
  checkbox: {
    borderWidth: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  image: {
    alignItems: 'center',
    justifyContent: 'center',
    width: 17,
    height: 17,
  },
});


./src/components/atomic/TodoCheckbox/TodoCheckbox.tsx

import { useTheme } from '@react-navigation/native';
import React, { useEffect, useRef } from 'react';
import { Pressable, ViewProps, View, Alert, Animated, Easing, Image } from 'react-native';
import { ICONS } from '../../../../assets';
import { COLORS } from '../../../../assets/colors';
import { useAppDispatch, useAppSelector } from '../../../store/hooks';
import { toggleTodo } from '../../../store/todoSlice';
import { ITodo } from '../../../store/types';
import { styles } from './styles';

interface TodoCheckboxProps extends ViewProps {
  todo: ITodo;
```

```
   size?: number;
   onSelectColor?: string;
   defaultBorderColor?: string;
   animated?: boolean;
}

export const TodoCheckbox: React.FC<TodoCheckboxProps> = ({
   todo,
   size = 30,
   style,
   onSelectColor,
   defaultBorderColor,
   animated = true,
}) => {
   const spinValue = useRef(new Animated.Value(todo.completed ? 0 : 1)).current;

   const spinZ = spinValue.interpolate({
       inputRange: [0, 1],
       outputRange: ['0deg', '90deg'],
   });

   const { colors: themeColors, dark } = useTheme();
   const completedColor = onSelectColor || COLORS.TODO_COMPLETED;
   const dispatch = useAppDispatch();
   const todayUnfulfilled = useAppSelector((state) =>
       (Array.isArray(todo.date) && todo.date[0].toDateString() === new
Date(Date.now()).toDateString()) ||
       (!Array.isArray(todo.date) && todo.date && todo.date.toDateString() === new
Date(Date.now()).toDateString())
           ? state.todos.todos.filter((el) => {
                 if (el.completed) return false;
                 if (Array.isArray(el.date) && el.date[0].toDateString() === new
Date(Date.now()).toDateString())
                     return true;
                 if (
                     el.date &&
                     !Array.isArray(el.date) &&
                     el.date.toDateString() === new Date(Date.now()).toDateString()
                 )
                     return true;
                 return false;
             })
           : [],
```



```
    );
    const onPress = () => {
        if (animated)
            Animated.timing(spinValue, {
                toValue: !todo.completed ? 0 : 1,
                duration: 400,
                easing: Easing.linear,
                useNativeDriver: true,
            }).start();
        if (
            (!todo.completed &&
                Array.isArray(todo.date) &&
                todo.date[0].toDateString() === new Date(Date.now()).toDateString()) ||
            (!Array.isArray(todo.date) && todo.date && todo.date.toDateString() === new
Date(Date.now()).toDateString())
        ) {
            if (!todo.completed && todayUnfulfilled.length === 1)
                Alert.alert('Поздравляем!', 'Все задачи на сегодня выполнены', [{ text:
'Отлично!' }]);
        }
        if (todo.completed) setTimeout(() => dispatch(toggleTodo(todo)), 400);
        else dispatch(toggleTodo(todo));
    };
    const doneIcon =
        onSelectColor === COLORS.THEME_LIGHT
            ? ICONS.imgDoneWhiteIcon
            : (dark && ICONS.imgDoneDarkIcon) || ICONS.imgDoneLightIcon;
    useEffect(() => {
        if (animated)
            Animated.timing(spinValue, {
                toValue: todo.completed ? 0 : 1,
                duration: 400,
                easing: Easing.linear,
                useNativeDriver: true,
            }).start();
    }, [todo.completed]);

    return (
        <Pressable
            onPress={onPress}
            style={[
                style,
                styles.checkbox,
                {
                    width: size,
                    height: size,
                    borderRadius: size / 2,
                    borderColor: todo.completed ? completedColor : defaultBorderColor ||
themeColors.text,
                },
            ]}
        >
            <Animated.View style={{ transform: [{ rotateZ: spinZ }] }}>
                {todo.completed ? <Image source={doneIcon} style={styles.image} /> :
<View />}
            </Animated.View>
        </Pressable>
    );
};
```

./src/components/atomic/TodoScheduleCard/styles.ts

```
import { Platform, StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    default: {
        borderRadius: 5,
        zIndex: 1,
        justifyContent: 'flex-start',
        alignItems: 'flex-start',
        paddingTop: 5,
        paddingLeft: 10,
    },
    defaultLight: { color: COLORS.TEXT_THEME_LIGHT, backgroundColor: COLORS.THEME_LIGHT
},
    defaultDark: { color: COLORS.THEME_LIGHT, backgroundColor: COLORS.TASK_THEME_DARK },
    todoContainer: { flexDirection: 'row', justifyContent: 'center' },
    textDeadline: {
        fontFamily: 'light-poppins',
        fontSize: Platform.OS === 'ios' ? 12 : 10,
        alignSelf: 'flex-end',
        marginRight: 5,
        lineHeight: 14,
    },
    circle: { width: 6, height: 6, borderRadius: 3, marginRight: 5 },
    icon: { width: 13, height: 13, marginRight: 5 },
    checkBox: { marginRight: 5, marginTop: 3 },
});
```

./src/components/atomic/TodoScheduleCard/TodoScheduleCard.tsx

```
import React from 'react';
import { useTheme } from '@react-navigation/native';
import {
   View,
   Text,
   Pressable,
   PressableProps,
   StyleProp,
   ViewStyle,
   Image,
   TextStyle,
   Dimensions,
} from 'react-native';
import { Shadow } from 'react-native-shadow-2';
import { ICONS } from '../../../../assets';
import { COLORS } from '../../../../assets/colors';
import { typography } from '../../../../assets/globalStyles';
import { useAppSelector } from '../../../store/hooks';
import { DateTypes, ITodo } from '../../../store/types';
import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';
import { dateHalfHourCeil, dateHalfHourFloor } from '../../../utils/dateHalfHourRound';
import { TodoCheckbox } from '../TodoCheckbox/TodoCheckbox';
import { styles } from './styles';
import { todosSortByDateAscending } from '../../../utils/todosSortByDate';

const DEFAULT_WIDTH = 280;
const UNIT_HEIGHT = 60;
const DEFAULT_TEXT_WIDTH = 215;
const DEFAULT_WINDOW_WIDTH = 375;

interface TodoScheduleCardProps extends PressableProps {
   todo: ITodo;
   style?: StyleProp<ViewStyle>;
}
```

```
const useCrossingIndex = (todo: ITodo) => {
  if (!todo.date) return 0;
  const todos = todosSortByDateAscending(useAppSelector((state) =>
state.todos.todos));
  const todoStartTime = Array.isArray(todo.date) ? dateHalfHourFloor(todo.date[0]) :
dateHalfHourFloor(todo.date);
  const todoEndTime = Array.isArray(todo.date)
    ? dateHalfHourCeil(todo.date[1])
    : dateHalfHourFloor(new Date(todo.date.getTime() + 1800000));
  if (!todo.date || todo.dateType === DateTypes.WITHOUT) return 1;
  const beforeTodos: ITodo[] = [];
  let lastTodoStart: null | Date = null;
  todos.forEach((el) => {
    if (!el.date) return;
    const elEnd = Array.isArray(el.date)
      ? dateHalfHourCeil(el.date[1])
      : dateHalfHourFloor(new Date(el.date.getTime() + 1800000));
    const elStart = Array.isArray(el.date) ? dateHalfHourFloor(el.date[0]) :
dateHalfHourFloor(el.date);
    if (
      (elStart.getTime() - todoStartTime.getTime() !== 0 &&
        elEnd.getTime() - todoEndTime.getTime() <= 0 &&
        elEnd.getTime() - todoStartTime.getTime() > 0) ||
      (elEnd.getTime() - todoEndTime.getTime() >= 0 && elStart.getTime() -
todoStartTime.getTime() < 0)
    ) {
      if (
        elStart.getTime() - todoStartTime.getTime() < 0 &&
        (!lastTodoStart || elStart.getTime() - lastTodoStart.getTime())
      ) {
        lastTodoStart = elStart;
        beforeTodos.push(el);
      }
    }
  });
  return beforeTodos.length + 1;
};

const useSameTimeCount = (todo: ITodo) => {
  if (!todo.date) return 0;
  const todoStartTime = Array.isArray(todo.date) ? dateHalfHourFloor(todo.date[0]) :
dateHalfHourFloor(todo.date);
  const todos = useAppSelector((state) => state.todos);
```

```
    if (!todo.date || todo.dateType === DateTypes.WITHOUT) return 0;
    return todos.todos.filter((el) => {
        if (!el.date) return false;
        const elStart = Array.isArray(el.date) ? dateHalfHourFloor(el.date[0]) :
dateHalfHourFloor(el.date);
        return elStart.getTime() === todoStartTime.getTime();
    }).length;
};

export const TodoScheduleCard: React.FC<TodoScheduleCardProps> = ({ todo, onPress,
style: customStyle }) => {
    if (!todo.date || todo.dateType === DateTypes.WITHOUT) return null;
    const { category, dateType, title, description } = todo;
    const { color: todoColor } = category;
    const { colors: themeColors, dark } = useTheme();
    const wrapperStyles: Array<StyleProp<ViewStyle>> = [styles.default];
    const fontStyle: Array<StyleProp<TextStyle>> = [{ lineHeight: 18, letterSpacing:
0.33 }];
    const todoStartTime = Array.isArray(todo.date) ? dateHalfHourFloor(todo.date[0]) :
dateHalfHourFloor(todo.date);
    const todoEndTime = Array.isArray(todo.date)
        ? dateHalfHourCeil(todo.date[1])
        : dateHalfHourFloor(new Date(todo.date.getTime() + 1800000));
    const index = useCrossingIndex(todo);
    const count = useSameTimeCount(todo);
    const windowWidth = Dimensions.get('screen').width;
    const ratio = windowWidth / DEFAULT_WINDOW_WIDTH;
    const cardWidth = count !== 0 ? Math.floor((DEFAULT_WIDTH * ratio) / count) :
DEFAULT_WIDTH;
    const textWidth = count !== 0 ? Math.floor((DEFAULT_TEXT_WIDTH * ratio) / count) :
DEFAULT_TEXT_WIDTH;
    const cardHeight = (UNIT_HEIGHT * (todoEndTime.getTime() - todoStartTime.getTime()))
/ 1800000;
    const positionSettings = { width: cardWidth, height: cardHeight + 1, zIndex: index
};
    wrapperStyles.push(positionSettings);
    let alarmIcon = ICONS.imgAlarmOn;
    if (index % 2 === 0) {
        wrapperStyles.push({ backgroundColor: todoColor });
        fontStyle.push({ color: COLORS.THEME_LIGHT });
        alarmIcon = dark ? ICONS.imgAlarmOnDark : ICONS.imgAlarmOnLight;
    } else {
        fontStyle.push({ color: themeColors.text });
```

```
   }
   const deadline =
       count === 1 ? (
           <Text style={[styles.textDeadline, { color: index % 2 === 0 ?
themeColors.card : COLORS.TEXT_HELP }]}>
               Контрольный срок
           </Text>
       ) : (
           cardWidth > 50 && <Image source={alarmIcon} style={styles.icon} />
       );
   const checkboxComponent =
       count === 1 || count === 0 ? (
           <TodoCheckbox
               animated={false}
               todo={todo}
               style={styles.checkBox}
               onSelectColor={index % 2 === 0 ? COLORS.THEME_LIGHT : undefined}
               defaultBorderColor={index % 2 === 0 ? COLORS.THEME_LIGHT : undefined}
           />
       ) : null;
   return (
       <Pressable style={[customStyle, positionSettings]} onPress={onPress}>
           <Shadow distance={3} startColor={convertHexToRGBA(todoColor, 35)} radius={5}
offset={[0, 1]}>
               <View style={[{ backgroundColor: themeColors.card }, ...wrapperStyles]}>
                   <View style={{ flexDirection: 'row', alignSelf: 'flex-end',
justifyContent: 'flex-start' }}>
                       {dateType === DateTypes.DEADLINE ? deadline : undefined}
                       <View
                           style={[styles.circle, { backgroundColor: index % 2 === 1 ?
todoColor : themeColors.card }]}
                       />
                   </View>
                   <View style={styles.todoContainer}>
                       {checkboxComponent}
                       <View style={{ width: textWidth }}>
                           <Text numberOfLines={1} style={[typography.subtitle,
...fontStyle]}>
                               {title}
                           </Text>
                           <Text numberOfLines={1} style={[typography.text,
...fontStyle]}>
                               {description}
```

```
                            </Text>
                        </View>
                    </View>
                </View>
            </Shadow>
        </Pressable>
    );
};
```

./src/components/atomic/TodoSwipeableCard/styles.ts

```
import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    card: {
        height: 68,
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignItems: 'center',
        width: '100%',
        borderColor: COLORS.BUTTON_THEME_LIGHT,
    },
    point: { width: 6, height: 6, borderRadius: 3, marginBottom: 30, marginRight: 15 },
    swipedTextWrapper: {
        justifyContent: 'center',
        height: 68,
        marginLeft: 15,
        width: 210,
    },
    buttonsWrapper: {
        flexDirection: 'row',
        justifyContent: 'center',
        alignItems: 'center',
        height: 68,
    },
    swipedWrapper: {
        flexDirection: 'row',
        height: 68,
        justifyContent: 'space-between',
        borderTopLeftRadius: 5,
        borderBottomLeftRadius: 5,
    },
    textWrapper: {
        width: 150,
    },
    dateContainer: {
        width: 85,
        justifyContent: 'center',
        alignItems: 'center',
    },
 });


 ./src/components/atomic/TodoSwipeableCard/TodoSwipeableCard.tsx

 import { useNavigation, useTheme } from '@react-navigation/native';
 import React, { useMemo } from 'react';
 import { View, Text, Animated, Alert, ViewProps, Pressable } from 'react-native';
 import Swipeable from 'react-native-gesture-handler/Swipeable';
 import { COLORS } from '../../../../assets/colors';
 import { typography } from '../../../../assets/globalStyles';
 import { useAppDispatch } from '../../../store/hooks';
 import { deleteTodo } from '../../../store/todoSlice';
 import { ITodo } from '../../../store/types';
 import { convertHexToRGBA } from '../../../utils/convertHexToRGBA';
 import { createCustomDateStringFromDate, createCustomTimeStringFromDate } from
 '../../../utils/dateConvertations';
 import { IconSquareActionButton } from
 '../IconSquareActionButton/IconSquareActionButton';
 import { TodoCheckbox } from '../TodoCheckbox/TodoCheckbox';
 import { styles } from './styles';

 interface TodoSwipeableCardProps extends ViewProps {
    todo: ITodo;
 }

 export const TodoSwipeableCard: React.FC<TodoSwipeableCardProps> = ({ todo, ...props })
 => {
    const { dark, colors: themeColors } = useTheme();
    const navigation = useNavigation();
    const dispatch = useAppDispatch();
    const fonts = useMemo(
        () =>
            dark
```

```
               ? {
                     subtitle: typography.subtitleDarkTheme,
                     text: typography.textDarkTheme,
                     expired: typography.textExpired,
                 }
               : {
                     subtitle: typography.subtitleLightTheme,
                     text: typography.textLightTheme,
                     expired: typography.textExpired,
                 },
       [dark],
   );
   const renderLeftActions = (progress: Animated.AnimatedInterpolation) => {
       const trans = progress.interpolate({
           inputRange: [0, 1],
           outputRange: [50, 0],
       });
       return (
           <Animated.View
               style={[
                   {
                       justifyContent: 'center',
                       transform: [{ translateX: trans }],
                       width: '100%',
                       backgroundColor: themeColors.background,
                   },
               ]}
           >
               <View
                   style={[
                       {
                           backgroundColor: convertHexToRGBA(todo.category.color, 30),
                       },
                       styles.swipedWrapper,
                   ]}
               >
                   <View style={styles.swipedTextWrapper}>
                       <Text numberOfLines={1} style={[fonts.subtitle, { color:
todo.category.color }]}>
                           {todo.title}
                       </Text>
                       <Text numberOfLines={1} style={[fonts.text, { color:
todo.category.color }]}>
```

```
                            {todo.description}
                        </Text>
                    </View>
                    <View style={styles.buttonsWrapper}>
                        <IconSquareActionButton
                            style={{ height: '100%' }}
                            type="edit"
                            onPress={() => navigation.navigate('EditTaskPage', { todo
})}
                        />
                        <IconSquareActionButton
                            style={{ height: '100%' }}
                            type="delete"
                            onPress={() =>
                                Alert.alert(
                                    'Подтвердите действие',
                                    'Вы уверены, что хотите безвозвратно удалить
задачу?',
                                    [
                                        { text: 'Удалить', onPress: () =>
dispatch(deleteTodo(todo)) },
                                        { text: 'Отмена' },
                                    ],
                                )
                            }
                        />
                    </View>
                </View>
            </Animated.View>
        );
    };
    return (
        <Swipeable renderRightActions={renderLeftActions}>
            <View
                {...props}
                style={[
                    props.style,
                    {
                        backgroundColor: themeColors.background,
                        borderBottomWidth: dark ? 0.5 : 1.5,
                    },
                    styles.card,
                ]}
            >
                <TodoCheckbox todo={todo} style={{ marginLeft: 20 }} />
                <Pressable onPress={() => navigation.navigate('TaskPage', { todo })}>
                    <Text
                        numberOfLines={1}
                        style={[
                            fonts.subtitle,
                            styles.textWrapper,
                            todo.completed && { color: COLORS.TODO_COMPLETED,
textDecorationLine: 'line-through' },
                        ]}
                    >
                        {todo.title}
                    </Text>
                    <Text
                        numberOfLines={1}
                        style={[fonts.text, styles.textWrapper, todo.completed && {
color: COLORS.TODO_COMPLETED }]}
                    >
                        {todo.description}
                    </Text>
                </Pressable>
                <View style={styles.dateContainer}>
                    {Array.isArray(todo.date) ? (
                        <View style={styles.dateContainer}>
                            <Text
                                style={
                                    !todo.completed && todo.date[1].getTime() <
Date.now()
                                        ? fonts.expired
                                        : [fonts.text, { color:
convertHexToRGBA(themeColors.text, 50) }]
                                }
                            >
                                {createCustomDateStringFromDate(todo.date[0])}
                            </Text>
                            <Text
                                style={
                                    !todo.completed && todo.date[1].getTime() <
Date.now()
                                        ? fonts.expired
                                        : [fonts.text, { color:
convertHexToRGBA(themeColors.text, 50) }]
```

```
                               }
                           >
                               {`${createCustomTimeStringFromDate(todo.date[0])} -
${createCustomTimeStringFromDate(
                                   todo.date[1],
                               )}`}
                           </Text>
                       </View>
                   ) : (
                       todo.date && (
                           <View style={styles.dateContainer}>
                               <Text
                                   style={
                                       !todo.completed && todo.date.getTime() <
Date.now()
                                           ? fonts.expired
                                           : [fonts.text, { color:
convertHexToRGBA(themeColors.text, 50) }]
                                   }
                               >
                                   {createCustomDateStringFromDate(todo.date)}
                               </Text>
                               <Text
                                   style={
                                       !todo.completed && todo.date.getTime() <
Date.now()
                                           ? fonts.expired
                                           : [fonts.text, { color:
convertHexToRGBA(themeColors.text, 50) }]
                                   }
                               >
                                   {createCustomTimeStringFromDate(todo.date)}
                               </Text>
                           </View>
                       )
                   )}
               </View>
               <View style={[{ backgroundColor: todo.category.color }, styles.point]}
/>
           </View>
       </Swipeable>
   );
};
```

```
./src/components/development/HomeScreenTest.tsx

import React from 'react';
import { Button, View, Text } from 'react-native';
import { StackNavigationProp } from '@react-navigation/stack';
import { useNavigation, useTheme } from '@react-navigation/native';
import { RootStackParamList } from '../RootStackParams';
import { useAppSelector } from '../../store/hooks';

type HomeScreenProp = StackNavigationProp<RootStackParamList, 'Home'>;

export function HomeScreen() {
   const { colors } = useTheme();
   const username = useAppSelector((state) => state.app.username);
   const navigation = useNavigation<HomeScreenProp>();
   return (
       <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center',
backgroundColor: colors.background }}>
           <Text style={{ color: colors.text }}>{`You are at the Homescreen,
${username}`}</Text>
           <Button title="Go to Example" onPress={() => navigation.push('Example')} />
           <Button title="Go to ThemeButtons test" onPress={() =>
navigation.push('ThemeButtonTest')} />
       </View>
   );
}


./src/components/development/ReduxDoToExample.tsx

import React from 'react';
import { StyleSheet, Text, View, Button, ScrollView } from 'react-native';
import { useNavigation, useTheme } from '@react-navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import { useAppSelector, useAppDispatch } from '../../store/hooks';
import { changeTheme, changeUsername } from '../../store/appSlice';
import { addCategory, deleteCategory, editCategory } from '../../store/categorySlice';
import { COLORS } from '../../../assets/colors';
import { Themes } from '../../store/types';
import { addTodo, deleteTodo, editTodo, toggleTodo } from '../../store/todoSlice';
import { RootStackParamList } from '../RootStackParams';
import { AnimatedSquaresOne, AnimatedSquaresTwo, AnimatedSquaresThree } from
'../atomic/AnimatedSquares';

type ExampleScreenProp = StackNavigationProp<RootStackParamList, 'Example'>;

export function ReduxDoToExample(): JSX.Element {
   const { colors } = useTheme();
   const navigation = useNavigation<ExampleScreenProp>();
   const theme = useAppSelector((state) => state.app.theme);
   const username = useAppSelector((state) => state.app.username);
   const categories = useAppSelector((state) => state.categories.categories);
   const todos = useAppSelector((state) => state.todos.todos);
   const dispatch = useAppDispatch();

   return (
       <ScrollView>
           <View style={{ alignItems: 'center' }}>
               <View style={styles.animationBlock}>
                   <AnimatedSquaresOne />
                   <AnimatedSquaresTwo />
                   <AnimatedSquaresThree />
               </View>
           </View>
           <Button title="Go to Homepage" onPress={() => navigation.push('Home')} />
           <View style={{ backgroundColor: colors.card }}>
               <Text style={{ color: colors.text }}>{`theme = ${theme}`}</Text>
               <Text style={{ color: colors.text }}>{`username = ${username}`}</Text>
               <Button
                   title="Toggle theme"
                   onPress={() => dispatch(changeTheme(theme === Themes.DARK ?
Themes.LIGHT : Themes.DARK))}
               />
               <Button title="Add heart to username" onPress={() =>
dispatch(changeUsername(`${username}♥`))} />
           </View>
           <View>
               <Button
                   disabled={!categories[0]}
                   title="Add category"
                   onPress={() => dispatch(addCategory({ ...categories[0], key:
Date.now().toString() }))}
               />
               {categories.map((item) => (
                   <View key={item.key} style={{ borderColor: item.color, borderWidth:
2 }}>
                       <Text style={{ fontSize: 24, color: colors.text
}}>{item.key}</Text>
                       <Text style={{ fontSize: 24, color: colors.text, fontFamily:
'light-poppins' }}>
                           {item.name}
                       </Text>
                       <Button
                           title="Add heart to categories name"
                           onPress={() => dispatch(editCategory({ ...item, name:
`${item.name}♥` }))}
                       />
                       <Button title="DELETE" onPress={() =>
dispatch(deleteCategory(item))} />
                   </View>
               ))}
           </View>
           <View>
               <Text style={(styles.header, { color: colors.text })}>Задачи</Text>
               <Text style={(styles.header, { color: colors.text })}>
                   {categories ? categories[0].name : 'no categories'}
               </Text>
               {todos
                   .filter((item) => item.category.key === categories[0].key)
                   .map((todo) => (
                       <View key={todo.key}>
                           <Text style={(styles.text, { color: colors.text
})}>{todo.title}</Text>
                           <Text style={(styles.text, { color: colors.notification })}>
                               {todo.completed.toString()}
                           </Text>
                           <Text style={(styles.text, { color: colors.text
})}>{todo.description}</Text>
                           <Button color="red" title="delete todo" onPress={() =>
dispatch(deleteTodo(todo))} />
                           <Button title="toggle" onPress={() =>
dispatch(toggleTodo(todo))} />
                           <Button
                               title="add todo"
                               onPress={() => dispatch(addTodo({ ...todo, key:
Date.now().toString() }))}
                           />
```

```
                            <Button
                                title="edit"
                                onPress={() => dispatch(editTodo({ ...todo, title:
`${todo.title}♥` }))}
                            />
                        </View>
                    ))}
            </View>
        </ScrollView>
    );
}

const styles = StyleSheet.create({
    example: {
        alignItems: 'center',
        justifyContent: 'center',
        borderColor: COLORS.VIOLET,
        borderWidth: 10,
    },
    header: {
        fontSize: 24,
        fontWeight: '500',
        fontFamily: 'light-poppins',
    },
    text: {
        fontWeight: '200',
        fontFamily: 'light-poppins',
    },
    animationBlock: {
        width: 375,
        height: 200,
        borderColor: COLORS.TEXT_HELP,
        borderWidth: 1,
    },
});


./src/components/development/StorybookScreen.tsx

import * as React from 'react';
import StorybookUIRoot from '../../../storybook';

export function StorybookScreen() {
    return <StorybookUIRoot />;
}



./src/components/development/TasksKate.tsx

import { useNavigation } from '@react-navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import React, { useRef, useState } from 'react';
import { SafeAreaView, Button, View, StyleSheet, ScrollView, TextInput, Pressable }
from 'react-native';
import { useAppSelector } from '../../store/hooks';
import { RootStackParamList } from '../RootStackParams';
import { CategoryCard } from '../atomic/CategoryCard/CategoryCard';
import { ScheduleLine } from '../atomic/ScheduleLine/ScheduleLine';
import { ActionButton } from '../atomic/ActionButton/ActionButton';
import { BackArrowButton } from '../atomic/BackArrowButton/BackArrowButton';
import { SelectCategoryCard } from '../atomic/SelectCategoryCard/SelectCategoryCard';
import { CountCard, COUNTWORDS } from '../atomic/CountCard/CountCard';
import { DescriptionTextInput } from
'../atomic/DescriptionTextInput/DescriptionTextInput';
import { MoreColorsButton } from '../atomic/MoreColorsButton/MoreColorsButton';
import { ITodo } from '../../store/types';
import { COLORS } from '../../../assets/colors';
import { categoryColors, PaletteBar } from '../atomic/PaletteBar/PaletteBar';
import { ModalColorPicker } from '../atomic/ModalColorPicker/ModalColorPicker';

export type TasksKateScreenProp = StackNavigationProp<RootStackParamList, 'TasksKate'>;
export const TasksKate: React.FC = () => {
    const navigation = useNavigation<TasksKateScreenProp>();
    const categories = useAppSelector((state) => state.categories.categories);
    const todos = useAppSelector((state) => state.todos.todos);
    const exmpl: ITodo = todos[0];
    const inputRef = useRef<TextInput>(null);
    const [inputValueFirst, setInputValueFirst] = useState(todos[0] ?
todos[0].description : 'Нет задач');
    const [inputValueSecond, setInputValueSecond] = useState('');
    const [customColor, setCustomColor] = useState<string>();
    const [color, setColor] = useState<string | undefined>();
    const [modalVisible, setModalVisible] = useState<boolean>(false);

    const styles = StyleSheet.create({
        card: {
            marginVertical: 5,
            marginHorizontal: 20,
        },
    });
    return (
        <SafeAreaView>
            <Pressable onPress={() => (inputRef?.current ? inputRef.current.blur() :
undefined)}>
                <ScrollView>
                    <Button title="Back to tasks" onPress={() =>
navigation.push('Tasks')} />
                    <View>
                        {categories.map((item) => (
                            <CategoryCard
                                category={item}
                                style={styles.card}
                                onPress={() => navigation.push('Example')}
                                key={item.key}
                            />
                        ))}
                    </View>
                    <View style={{ flexDirection: 'row', flexWrap: 'wrap' }}>
                        {categories.map((item) => (
                            <SelectCategoryCard
                                category={item}
                                key={item.key}
                                style={{ margin: 5 }}
                                selected={item.name[0] === 'П' ? !false : false}
                            />
                        ))}
                    </View>
                    <ScheduleLine time="08:00" />
                    <ScheduleLine time="08:30" />
                    <ScheduleLine time="09:00" />
                    <View style={{ flexDirection: 'row' }}>
                        <CountCard ending={COUNTWORDS.TOTAL} style={{ marginHorizontal:
24 }} />
                        <CountCard ending={COUNTWORDS.LEFT} />
                    </View>
                    <View>
                        <BackArrowButton onPress={() => navigation.goBack()} style={{
margin: 20 }} />
                    </View>
                    <DescriptionTextInput
                        ref={inputRef}
                        style={{ marginLeft: 94, marginTop: 20 }}
                        defaultValue={inputValueFirst}
                        onChangeText={(text) => setInputValueFirst(text)}
                    />
                    <DescriptionTextInput
                        style={{ marginLeft: 94, marginVertical: 20 }}
                        defaultValue={inputValueSecond}
                        onChangeText={(text) => setInputValueSecond(text)}
                    />
                    <View style={{ width: 30, height: 30, borderRadius: 15,
backgroundColor: color, margin: 20 }} />
                    <MoreColorsButton onPress={() => setModalVisible(true)} />
                    <ModalColorPicker
                        onColorSelected={(newColor) => {
                            setCustomColor(newColor);
                            setModalVisible(false);
                        }}
                        visible={modalVisible}
                    />
                    <PaletteBar
                        colors={customColor ? categoryColors.concat(customColor) :
categoryColors}
                        initColor={COLORS.TIFFANY}
                        style={{ margin: 10, alignSelf: 'flex-end' }}
                        onColorSelected={(pickedColor) => {
                            setColor(pickedColor);
                        }}
                    />
                    <ActionButton active={!false} title="Settings" onPress={() =>
navigation.push('SettingsPage')} />
                    <ActionButton
                        active={!!exmpl}
                        title="EditTaskPage"
                        onPress={() => navigation.push('EditTaskPage', { todo: exmpl })}
                    />
                    <ActionButton
                        active={!!categories[0]}
                        title="EditCategoryPage"
                        onPress={() => navigation.push('EditCategoryPage', { category:
categories[0] })}
                    />
```

```
                </ScrollView>
            </Pressable>
        </SafeAreaView>
    );
};
```

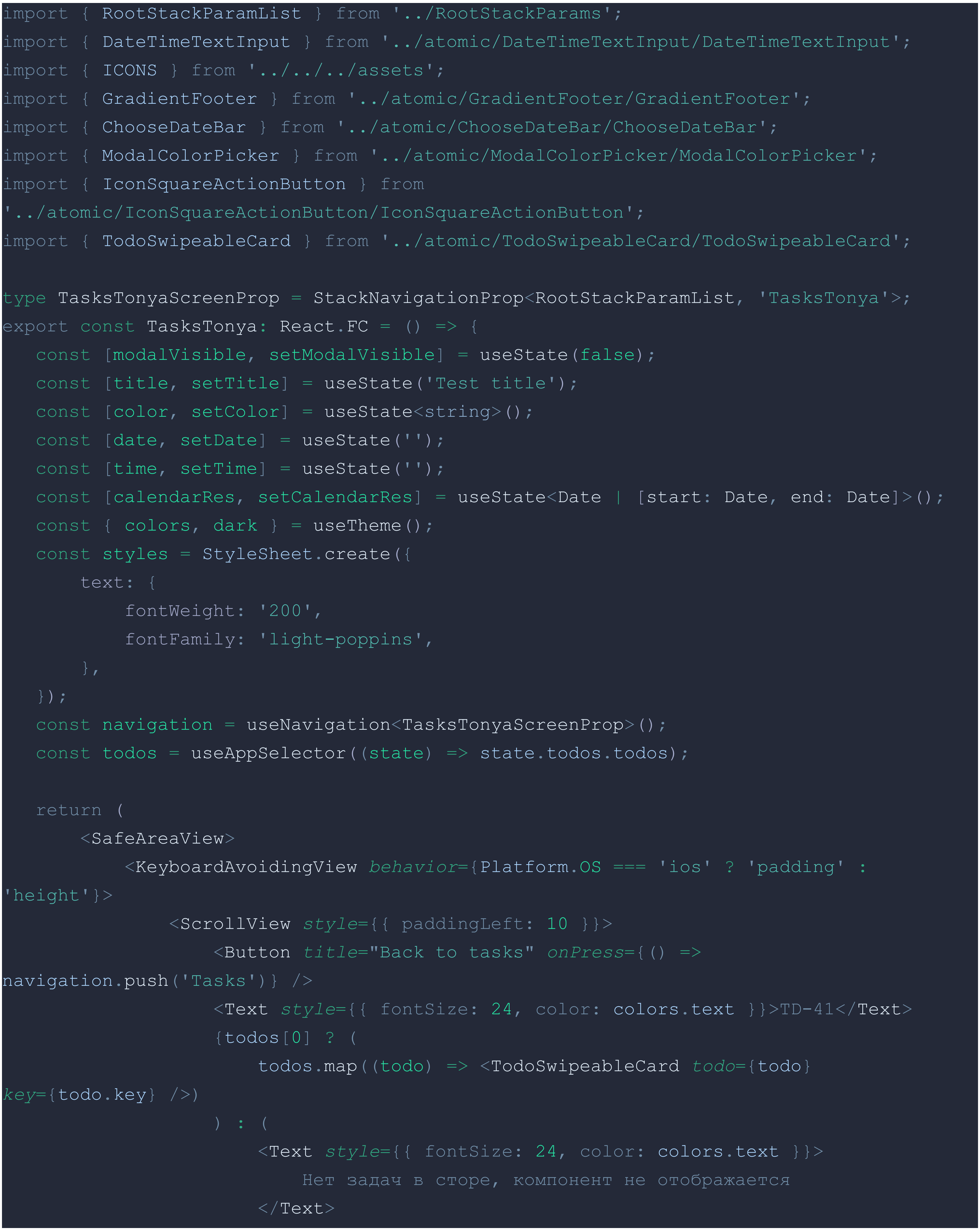
./src/components/development/TasksSasha.tsx

```
import React from 'react';
import { useNavigation } from '@react-navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import { SafeAreaView, Button, View } from 'react-native';
import { RootStackParamList } from '../RootStackParams';
import { IconAddCategoryButton } from
'../atomic/IconAddCategoryButton/IconAddCategoryButton';
import { SelectCategoryColor } from
'../atomic/SelectCategoryColor/SelectCategoryColor';
import { styles, Size } from '../atomic/IconAddCategoryButton/styles';
import { useAppDispatch } from '../../store/hooks';
import { addCategory } from '../../store/categorySlice';
import { ICategory } from '../../store/types';
import { COLORS } from '../../../assets/colors';
import { AnimatedSquaresOne, AnimatedSquaresThree, AnimatedSquaresTwo } from
'../atomic/AnimatedSquares';

type TasksSashaScreenProp = StackNavigationProp<RootStackParamList, 'TasksSasha'>;
export const TasksSasha: React.FC = () => {
    const navigation = useNavigation<TasksSashaScreenProp>();
    const newCategory: ICategory = {
        key: Date.now().toString(),
        name: 'Новая категория',
        color: COLORS.BLUE,
    };
    const dispatch = useAppDispatch();
    const submitCategorySave = () => {
        dispatch(addCategory({ ...newCategory, key: Date.now().toString() }));
        navigation.push('Home');
    };
    return (
        <SafeAreaView>
            <Button title="Back to tasks" onPress={() => navigation.push('Tasks')} />
            <View style={{ width: 375, alignSelf: 'center', height: 100 }}>
```

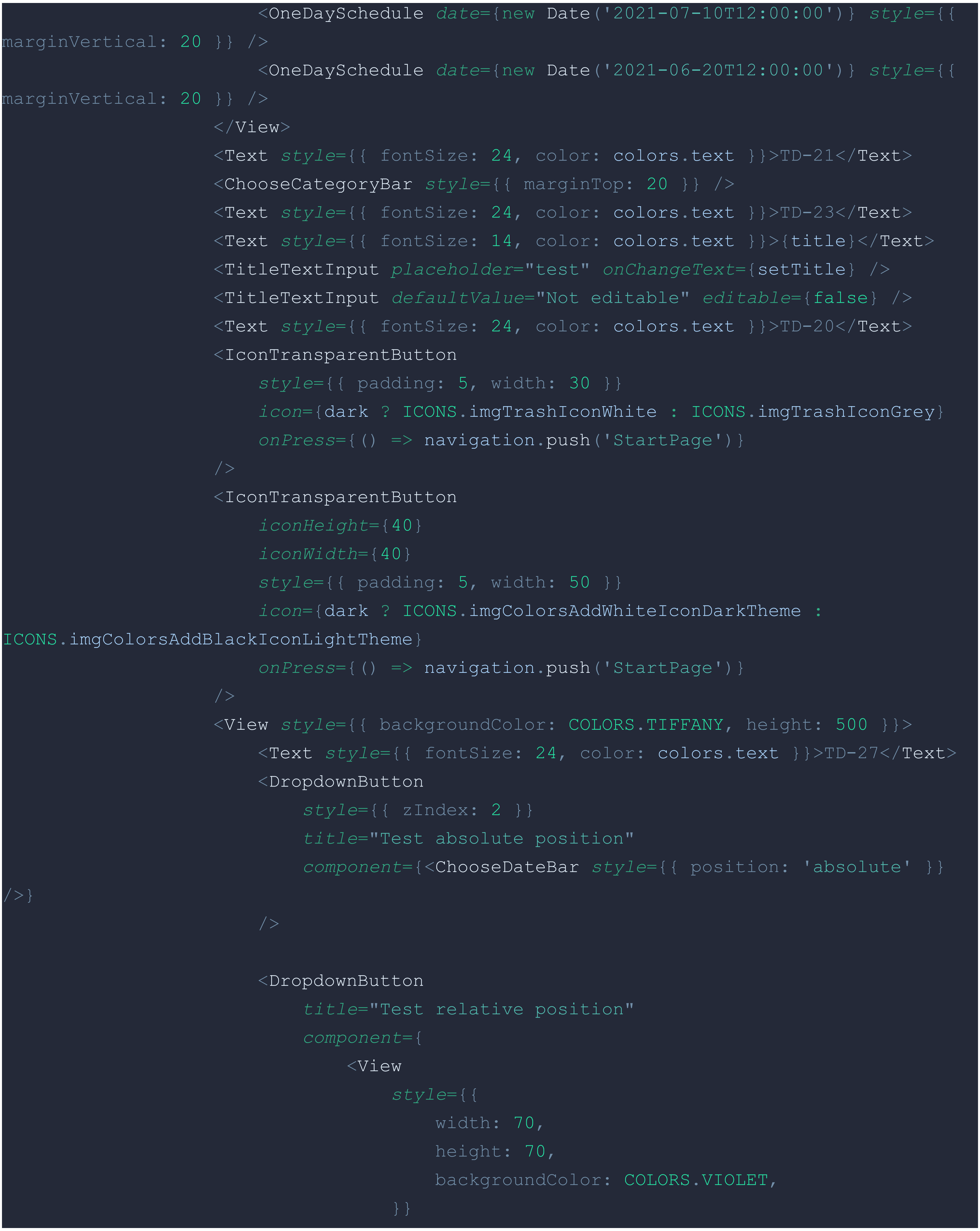
```
               <AnimatedSquaresOne />
               <AnimatedSquaresTwo />
               <AnimatedSquaresThree />
           </View>
           <IconAddCategoryButton type={Size.smallButtonSizes} style={styles.margins}
onPress={submitCategorySave} />
           <IconAddCategoryButton type={Size.bigButtonSizes} style={styles.margins}
onPress={submitCategorySave} />
           <SelectCategoryColor style={styles.margins} color={COLORS.BLUE} />
           <SelectCategoryColor style={styles.margins} color={COLORS.BLUE} selected />
           <Button title="AddCategoryPage" onPress={() =>
navigation.push('AddCategoryPage')} />
       </SafeAreaView>
   );
};
```

./src/components/development/TasksTonya.tsx

```
import React, { useState } from 'react';
import { useNavigation, useTheme } from '@react-navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import {
   Button,
   SafeAreaView,
   View,
   Text,
   StyleSheet,
   ScrollView,
   KeyboardAvoidingView,
   Platform,
   Alert,
} from 'react-native';
import { COLORS } from '../../../assets/colors';
import { useAppSelector } from '../../store/hooks';
import { ChooseCategoryBar } from '../atomic/ChooseCategoryBar/ChooseCategoryBar';
import { DropdownButton } from '../atomic/DropdownButton/DropdownButton';
import { IconTransparentButton } from
'../atomic/IconTransparentButton/IconTransparentButton';
import { OneDaySchedule } from '../atomic/OneDaySchedule/OneDaySchedule';
import { TitleTextInput } from '../atomic/TitleTextInput/TitleTextInput';
import { TodoCheckbox } from '../atomic/TodoCheckbox/TodoCheckbox';
import { TodoScheduleCard } from '../atomic/TodoScheduleCard/TodoScheduleCard';
```



```
import { RootStackParamList } from '../RootStackParams';
import { DateTimeTextInput } from '../atomic/DateTimeTextInput/DateTimeTextInput';
import { ICONS } from '../../../assets';
import { GradientFooter } from '../atomic/GradientFooter/GradientFooter';
import { ChooseDateBar } from '../atomic/ChooseDateBar/ChooseDateBar';
import { ModalColorPicker } from '../atomic/ModalColorPicker/ModalColorPicker';
import { IconSquareActionButton } from
'../atomic/IconSquareActionButton/IconSquareActionButton';
import { TodoSwipeableCard } from '../atomic/TodoSwipeableCard/TodoSwipeableCard';

type TasksTonyaScreenProp = StackNavigationProp<RootStackParamList, 'TasksTonya'>;
export const TasksTonya: React.FC = () => {
    const [modalVisible, setModalVisible] = useState(false);
    const [title, setTitle] = useState('Test title');
    const [color, setColor] = useState<string>();
    const [date, setDate] = useState('');
    const [time, setTime] = useState('');
    const [calendarRes, setCalendarRes] = useState<Date | [start: Date, end: Date]>();
    const { colors, dark } = useTheme();
    const styles = StyleSheet.create({
        text: {
            fontWeight: '200',
            fontFamily: 'light-poppins',
        },
    });
    const navigation = useNavigation<TasksTonyaScreenProp>();
    const todos = useAppSelector((state) => state.todos.todos);

    return (
        <SafeAreaView>
            <KeyboardAvoidingView behavior={Platform.OS === 'ios' ? 'padding' :
'height'}>
                <ScrollView style={{ paddingLeft: 10 }}>
                    <Button title="Back to tasks" onPress={() =>
navigation.push('Tasks')} />
                    <Text style={{ fontSize: 24, color: colors.text }}>TD-41</Text>
                    {todos[0] ? (
                        todos.map((todo) => <TodoSwipeableCard todo={todo}
key={todo.key} />)
                    ) : (
                        <Text style={{ fontSize: 24, color: colors.text }}>
                            Нет задач в сторе, компонент не отображается
                        </Text>
                    )}
                    <Text style={{ fontSize: 24, color: colors.text }}>TD-35</Text>
                    <Button title="Open modal" onPress={() =>
setModalVisible(!modalVisible)} />
                    <Text style={{ fontSize: 24, color: colors.text }}>{color}</Text>
                    <View style={{ width: 50, height: 50, borderRadius: 25,
backgroundColor: color }} />
                    <ModalColorPicker
                        onMissClick={() => setModalVisible(false)}
                        onColorSelected={(newColor) => {
                            setColor(newColor);
                            setModalVisible(false);
                        }}
                        visible={modalVisible}
                        initialColor={color}
                    />
                    <Text style={{ fontSize: 24, color: colors.text }}>TD-33</Text>
                    <IconSquareActionButton
                        type="delete"
                        onPress={() =>
                            Alert.alert('SquareActionButton clicked', "You clicked
button with type='delete'")
                        }
                    />
                    <IconSquareActionButton
                        type="edit"
                        onPress={() => Alert.alert('SquareActionButton clicked', "You
clicked button with type='edit'")}
                    />
                    <Text style={{ fontSize: 24, color: colors.text }}>TD-29</Text>
                    <View style={{ padding: 10 }}>
                        <Text>
                            {Array.isArray(calendarRes)
                                ? `FROM ${calendarRes[0].toString()} TO
${calendarRes[1].toString()}`
                                : calendarRes?.toString()}
                        </Text>
                        <ChooseDateBar
                            initialValue={[new Date(Date.now()), new Date(Date.now() +
60000)]}
                            onDateChange={setCalendarRes}
                        />
                    </View>
                    <View>
                        <Text style={{ fontSize: 24, color: colors.text }}>TD-12</Text>
                        <View>
                            {todos[0] ? (
                                <View>
                                    <Text style={(styles.text, { color: colors.text
})}>{todos[0].title}</Text>
                                    <Text style={(styles.text, { color:
colors.notification })}>
                                        {todos[0].completed.toString()}
                                    </Text>
                                    <TodoCheckbox todo={todos[0]} size={30} style={{
marginLeft: 10 }} />
                                </View>
                            ) : (
                                <Text style={{ fontSize: 24, color: colors.text }}>
                                    Нет задач в сторе, компонент не отображается
                                </Text>
                            )}
                        </View>
                    </View>
                    <View>
                        <Text style={{ fontSize: 24, color: colors.text }}>TD-15</Text>

                        {todos[0] ? (
                            <TodoScheduleCard
                                todo={todos[0]}
                                key={todos[0].key}
                                onPress={() => {
                                    navigation.push('Example');
                                }}
                                style={{ margin: 10 }}
                            />
                        ) : (
                            <Text style={{ fontSize: 24, color: colors.text }}>
                                Нет задач в сторе, компонент не отображается
                            </Text>
                        )}
                    </View>
                    <View>
                        <Text style={{ fontSize: 24, color: colors.text }}>TD-17</Text>
                        <OneDaySchedule date={new Date('2021-07-14T12:00:00')} style={{
marginVertical: 20 }} />
```



```
                        <OneDaySchedule date={new Date('2021-07-10T12:00:00')} style={{
marginVertical: 20 }} />
                        <OneDaySchedule date={new Date('2021-06-20T12:00:00')} style={{
marginVertical: 20 }} />
                    </View>
                    <Text style={{ fontSize: 24, color: colors.text }}>TD-21</Text>
                    <ChooseCategoryBar style={{ marginTop: 20 }} />
                    <Text style={{ fontSize: 24, color: colors.text }}>TD-23</Text>
                    <Text style={{ fontSize: 14, color: colors.text }}>{title}</Text>
                    <TitleTextInput placeholder="test" onChangeText={setTitle} />
                    <TitleTextInput defaultValue="Not editable" editable={false} />
                    <Text style={{ fontSize: 24, color: colors.text }}>TD-20</Text>
                    <IconTransparentButton
                        style={{ padding: 5, width: 30 }}
                        icon={dark ? ICONS.imgTrashIconWhite : ICONS.imgTrashIconGrey}
                        onPress={() => navigation.push('StartPage')}
                    />
                    <IconTransparentButton
                        iconHeight={40}
                        iconWidth={40}
                        style={{ padding: 5, width: 50 }}
                        icon={dark ? ICONS.imgColorsAddWhiteIconDarkTheme :
ICONS.imgColorsAddBlackIconLightTheme}
                        onPress={() => navigation.push('StartPage')}
                    />
                    <View style={{ backgroundColor: COLORS.TIFFANY, height: 500 }}>
                        <Text style={{ fontSize: 24, color: colors.text }}>TD-27</Text>
                        <DropdownButton
                            style={{ zIndex: 2 }}
                            title="Test absolute position"
                            component={<ChooseDateBar style={{ position: 'absolute' }}
/>}
                        />

                        <DropdownButton
                            title="Test relative position"
                            component={
                                <View
                                    style={{
                                        width: 70,
                                        height: 70,
                                        backgroundColor: COLORS.VIOLET,
                                    }}
```

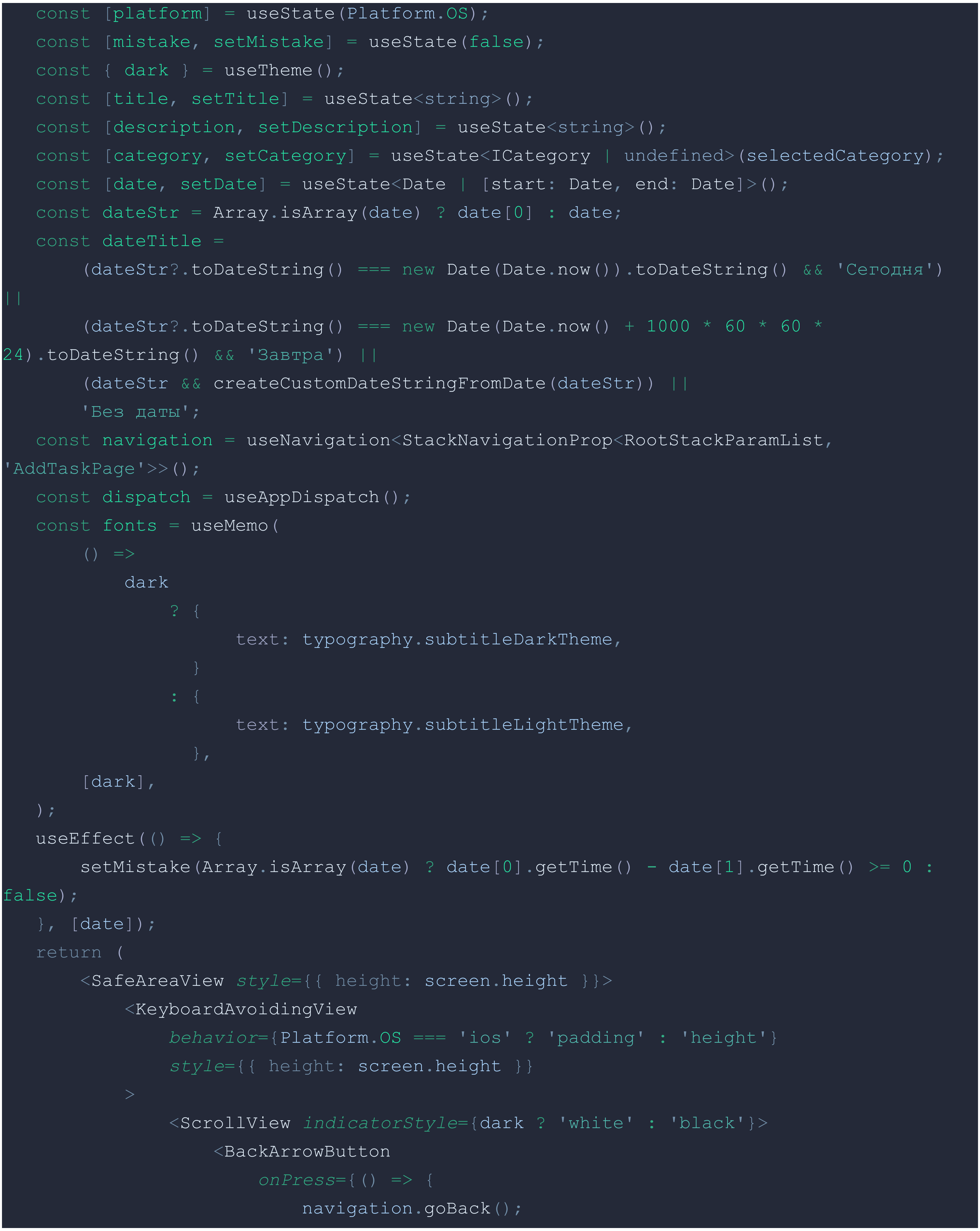
```
                               />
                           }
                           arrowStyle={{ height: 50 }}
                       />
                   </View>
                   <View style={{ backgroundColor: colors.card, alignItems: 'center'
}}>
                       <Text style={{ fontSize: 24, color: colors.text }}>TD-28</Text>
                       <Button onPress={() => setTime('08:30')} title="Set first input
08:30" />
                       <DateTimeTextInput formatType="time" onChangeText={setTime}
defaultValue="08:30" value={time} />
                       <DateTimeTextInput formatType="date" onChangeText={setDate}
style={{ marginVertical: 10 }} />
                       <Text style={{ fontSize: 14, color: colors.text }}>{date}</Text>
                       <DateTimeTextInput formatType="time" onChangeText={setDate}
style={{ marginBottom: 70 }} />
                   </View>
               </ScrollView>
               <GradientFooter style={{ position: 'absolute', bottom: 0, zIndex: 2,
height: 60, width: '100%' }} />
           </KeyboardAvoidingView>
       </SafeAreaView>
   );
};


./src/components/development/Tasks.tsx

import { useNavigation } from '@react-navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import React from 'react';
import { SafeAreaView, Button } from 'react-native';
import { RootStackParamList } from '../RootStackParams';

type TasksScreenProp = StackNavigationProp<RootStackParamList, 'Tasks'>;
export const Tasks: React.FC = () => {
   const navigation = useNavigation<TasksScreenProp>();
   return (
       <SafeAreaView>
           <Button title="Tonya's tasks" onPress={() => navigation.push('TasksTonya')}
/>
           <Button title="Sasha's tasks" onPress={() => navigation.push('TasksSasha')}
```

```
/>
           <Button title="Kate's tasks" onPress={() => navigation.push('TasksKate')} />
           <Button title="Go to app" onPress={() => navigation.push('MainPage')} />
           <Button title="Go to Storybook" onPress={() =>
navigation.push('StorybookScreen')} />
       </SafeAreaView>
   );
};


./src/components/development/ThemeButtonsTest.tsx

import { useTheme, useNavigation } from '@react-navigation/native';
import React from 'react';
import { View, Text } from 'react-native';
import { StackNavigationProp } from '@react-navigation/stack';
import { changeTheme } from '../../store/appSlice';
import { useAppDispatch, useAppSelector } from '../../store/hooks';
import { Themes, ITodo, DateTypes } from '../../store/types';
import { ThemeButton } from '../atomic/ThemeButton/ThemeButton';
import { ActionButton } from '../atomic/ActionButton/ActionButton';
import { IconActionButton } from '../atomic/IconActionButton/IconActionButton';
import { addTodo, editTodo } from '../../store/todoSlice';
import { COLORS } from '../../../assets/colors';
import { RootStackParamList } from '../RootStackParams';
import { ICONS } from '../../../assets/index';

type ThemeButtonsTestProp = StackNavigationProp<RootStackParamList, 'ThemeButtonTest'>;

export const ThemeButtonTest = () => {
   const todos = useAppSelector((state) => state.todos.todos);
   const newToDo: ITodo = {
       key: Date.now().toString(),
       title: 'Новая задачка',
       dateType: DateTypes.WITHOUT,
       description: 'Самая новая задачка',
       completed: false,
       category: {
           key: '1',
           name: 'Работа',
           color: COLORS.BLUE,
       },
   };
```

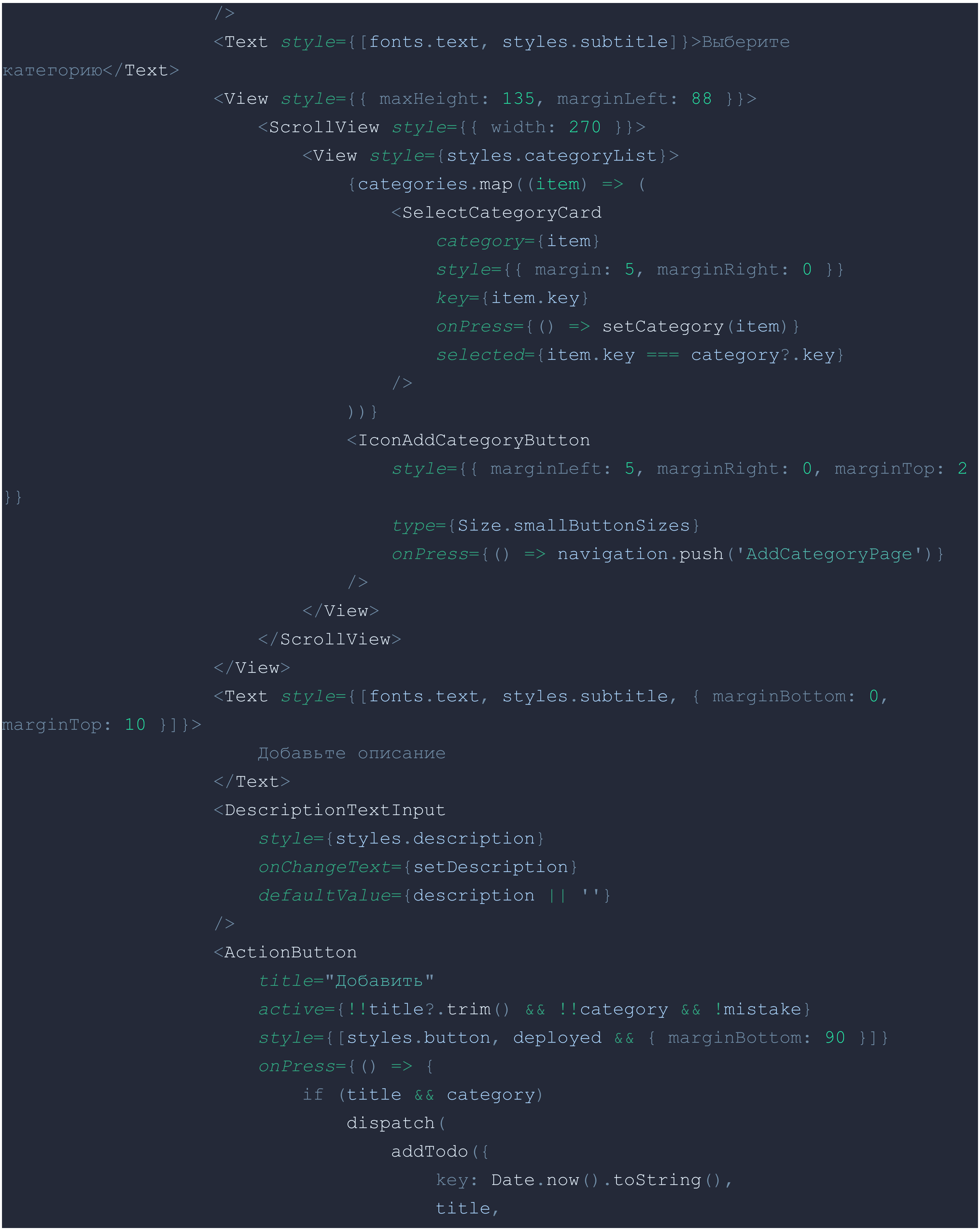
```
    const dispatch = useAppDispatch();
    const setDarkTheme = () => {
        dispatch(changeTheme(Themes.DARK));
    };
    const setLightTheme = () => {
        dispatch(changeTheme(Themes.LIGHT));
    };
    const setSystemTheme = () => {
        dispatch(changeTheme(Themes.SYSTEMIC));
    };
    const submitTodoSave = () => {
        dispatch(addTodo({ ...newToDo, key: Date.now().toString() }));
    };
    const submitTodoEdit = () => {
        if (todos[0]) dispatch(editTodo({ ...todos[0], title: `${todos[0].title} edited`
}));
    };

    const theme = useAppSelector((state) => state.app.theme);
    const { colors } = useTheme();
    const navigation = useNavigation<ThemeButtonsTestProp>();
    return (
        <View style={{ alignItems: 'center' }}>
            <Text style={{ color: colors.text, fontSize: 20 }}>Current theme is
{theme}</Text>
            <ThemeButton title="Тёмная" onPress={setDarkTheme} selected={theme ===
Themes.DARK} />
            <ThemeButton title="Светлая" onPress={setLightTheme} selected={theme ===
Themes.LIGHT} />
            <ThemeButton title="Системная" onPress={setSystemTheme} selected={theme ===
Themes.SYSTEMIC} />
            <ActionButton title="Сохранить" onPress={submitTodoEdit} active={!!todos[0]}
/>
            <ActionButton title="Сохранить" onPress={submitTodoEdit} active={!!todos[0]}
/>
            <ActionButton title="Начать" onPress={() => navigation.push('Example')}
active={!false} />
            <ActionButton title="Добавить" onPress={submitTodoSave} active={!false} />
            <IconActionButton
                onPress={() => navigation.push('AddTaskPage', { category: undefined })}
                icon={ICONS.imgColorsAddWhiteIconDarkTheme}
            />
            <ActionButton title="Добавить" onPress={submitTodoSave} active={false} />
        </View>
    );
};
```

./src/components/pages/AddCategoryPage/AddCategoryPage.tsx

```
import React, { useEffect, useState } from 'react';
import { SafeAreaView, KeyboardAvoidingView, Platform, Dimensions, View } from
'react-native';
import { useNavigation } from '@react-navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import { ScrollView } from 'react-native-gesture-handler';
import { useAppDispatch } from '../../../store/hooks';
import { BackArrowButton } from '../../atomic/BackArrowButton/BackArrowButton';
import { MoreColorsButton } from '../../atomic/MoreColorsButton/MoreColorsButton';
import { RootStackParamList } from '../../RootStackParams';
import { TitleTextInput } from '../../atomic/TitleTextInput/TitleTextInput';
import { ActionButton } from '../../atomic/ActionButton/ActionButton';
import { PaletteBar, categoryColors } from '../../atomic/PaletteBar/PaletteBar';
import { ModalColorPicker } from '../../atomic/ModalColorPicker/ModalColorPicker';
import { addCategory } from '../../../store/categorySlice';
import { styles } from './styles';
import { COLORS } from '../../../../assets/colors';

export const AddCategoryPage = () => {
    const screen = Dimensions.get('screen');
    const navigation = useNavigation<StackNavigationProp<RootStackParamList,
'AddCategoryPage'>>();
    const [category, setCategory] = useState<string>();
    const [color, setColor] = useState<string>(COLORS.TIFFANY);
    const [initialColor, setInitialColor] = useState<string>(COLORS.TIFFANY);
    const [customColor, setCustomColor] = useState<string>();
    const [modalVisible, setModalVisible] = useState<boolean>(false);
    const dispatch = useAppDispatch();
    useEffect(() => {
        if (!initialColor.trim()) setInitialColor(COLORS.TIFFANY);
    }, [initialColor]);
    return (
        <SafeAreaView style={{ height: screen.height }}>
            <KeyboardAvoidingView
                behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
                style={{ height: screen.height }}
            >
                <ScrollView>
                    <BackArrowButton
                        onPress={() => {
                            navigation.goBack();
                        }}
                        style={styles.backArrow}
                    />
                    <TitleTextInput
                        placeholder="Добавить категорию..."
                        onChangeText={(text) => {
                            setCategory(text.trim());
                        }}
                        style={styles.titleInput}
                        defaultValue={category || ''}
                    />
                    <View style={{ marginRight: 20, flexDirection: 'column' }}>
                        <PaletteBar
                            colors={customColor ? categoryColors.concat(customColor) :
categoryColors}
                            initColor={initialColor}
                            style={{ alignSelf: 'flex-end' }}
                            onColorSelected={(pickedColor) => {
                                setColor(pickedColor);
                            }}
                        />
                        <MoreColorsButton
                            style={{ marginTop: 27, alignSelf: 'flex-end' }}
                            onPress={() => setModalVisible(true)}
                        />
                    </View>
                    <ModalColorPicker
                        onColorSelected={(newColor: string) => {
                            setCustomColor(newColor);
                            setColor(newColor);
                            setInitialColor(newColor);
                            setModalVisible(false);
                        }}
                        visible={modalVisible}
                        onMissClick={() => setModalVisible(false)}
                    />
                    <ActionButton
                        title="Добавить"
                        active={!!category?.trim() && !!color}
                        style={styles.button}
                        onPress={() => {
                            setCategory('');
                            setColor(COLORS.TIFFANY);
                            setInitialColor('');
                            setCustomColor(undefined);
                            if (color && category)
                                dispatch(
                                    addCategory({
                                        key: Date.now().toString(),
                                        name: category,
                                        color,
                                    }),
                                );
                        }}
                    />
                </ScrollView>
            </KeyboardAvoidingView>
        </SafeAreaView>
    );
};


./src/components/pages/AddCategoryPage/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    backArrow: {
        marginLeft: 34,
        marginTop: 22,
    },
    titleInput: {
        height: 34,
        marginLeft: 66,
        marginTop: 36,
        marginBottom: 135,
        maxWidth: 280,
    },
    button: {
        width: 106,
        marginLeft: 93,
        marginTop: 230,
        marginBottom: 45,
    },
 });


 ./src/components/pages/AddTaskPage/AddTaskPage.tsx

 import { RouteProp, useNavigation, useTheme } from '@react-navigation/native';
 import { StackNavigationProp } from '@react-navigation/stack';
 import React, { useEffect, useMemo, useState } from 'react';
 import { Text, SafeAreaView, KeyboardAvoidingView, View, Platform, Dimensions } from
 'react-native';
 import { ScrollView } from 'react-native-gesture-handler';
 import { typography } from '../../../../assets/globalStyles';
 import { useAppDispatch, useAppSelector } from '../../../store/hooks';
 import { addTodo } from '../../../store/todoSlice';
 import { DateTypes, ICategory } from '../../../store/types';
 import { createCustomDateStringFromDate, createCustomTimeStringFromDate } from
 '../../../utils/dateConvertations';
 import { ActionButton } from '../../atomic/ActionButton/ActionButton';
 import { BackArrowButton } from '../../atomic/BackArrowButton/BackArrowButton';
 import { ChooseDateBar } from '../../atomic/ChooseDateBar/ChooseDateBar';
 import { DescriptionTextInput } from
 '../../atomic/DescriptionTextInput/DescriptionTextInput';
 import { DropdownButton } from '../../atomic/DropdownButton/DropdownButton';
 import { IconAddCategoryButton } from
 '../../atomic/IconAddCategoryButton/IconAddCategoryButton';
 import { Size } from '../../atomic/IconAddCategoryButton/styles';
 import { SelectCategoryCard } from
 '../../atomic/SelectCategoryCard/SelectCategoryCard';
 import { TitleTextInput } from '../../atomic/TitleTextInput/TitleTextInput';
 import { RootStackParamList } from '../../RootStackParams';
 import { styles } from './styles';

 type AddTaskPageRouteProp = RouteProp<RootStackParamList, 'AddTaskPage'>;
 type Props = { route: AddTaskPageRouteProp };

 export const AddTaskPage: React.FC<Props> = ({ route }) => {
    const { category: selectedCategory } = route.params;
    const screen = Dimensions.get('screen');
    const categories = useAppSelector((state) => state.categories.categories);
    const [deployed, setDeployed] = useState(false);
```



```
    const [platform] = useState(Platform.OS);
    const [mistake, setMistake] = useState(false);
    const { dark } = useTheme();
    const [title, setTitle] = useState<string>();
    const [description, setDescription] = useState<string>();
    const [category, setCategory] = useState<ICategory | undefined>(selectedCategory);
    const [date, setDate] = useState<Date | [start: Date, end: Date]>();
    const dateStr = Array.isArray(date) ? date[0] : date;
    const dateTitle =
        (dateStr?.toDateString() === new Date(Date.now()).toDateString() && 'Сегодня')
||
        (dateStr?.toDateString() === new Date(Date.now() + 1000 * 60 * 60 *
24).toDateString() && 'Завтра') ||
        (dateStr && createCustomDateStringFromDate(dateStr)) ||
        'Без даты';
    const navigation = useNavigation<StackNavigationProp<RootStackParamList,
'AddTaskPage'>>();
    const dispatch = useAppDispatch();
    const fonts = useMemo(
        () =>
            dark
                ? {
                      text: typography.subtitleDarkTheme,
                  }
                : {
                      text: typography.subtitleLightTheme,
                  },
        [dark],
    );
    useEffect(() => {
        setMistake(Array.isArray(date) ? date[0].getTime() - date[1].getTime() >= 0 :
false);
    }, [date]);
    return (
        <SafeAreaView style={{ height: screen.height }}>
            <KeyboardAvoidingView
                behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
                style={{ height: screen.height }}
            >
                <ScrollView indicatorStyle={dark ? 'white' : 'black'}>
                    <BackArrowButton
                        onPress={() => {
                            navigation.goBack();
                        }}
                        style={styles.backArrow}
                    />
                    <TitleTextInput
                        placeholder="Добавить задачу..."
                        onChangeText={setTitle}
                        style={styles.titleInput}
                        defaultValue={title || ''}
                    />
                    <Text style={[fonts.text, styles.subtitle]}>Выберите дату</Text>
                    <DropdownButton
                        onPress={platform !== 'ios' ? () => setDeployed(!deployed) :
undefined}
                        style={[
                            {
                                zIndex: platform !== 'ios' ? 1 : 2,
                                marginLeft: 93,
                                marginBottom: platform === 'ios' ? 15 : 10,
                                height: (deployed && 400) || undefined,
                            },
                        ]}
                        title={
                            Array.isArray(date)
                                ? `${dateTitle}, ${createCustomTimeStringFromDate(
                                      date[0],
                                  )} - ${createCustomTimeStringFromDate(date[1])}`
                                : (date && `${dateTitle},
${createCustomTimeStringFromDate(date)}`) || 'Без даты'
                        }
                        titleStyle={mistake ? styles.mistake : undefined}
                        component={
                            <ChooseDateBar
                                initialValue={date}
                                style={{
                                    position: platform === 'ios' ? 'absolute' :
'relative',
                                    marginTop: platform === 'ios' ? 10 : 1,
                                    marginLeft: -15,
                                    zIndex: 1,
                                }}
                                onDateChange={setDate}
                            />
                        }
```

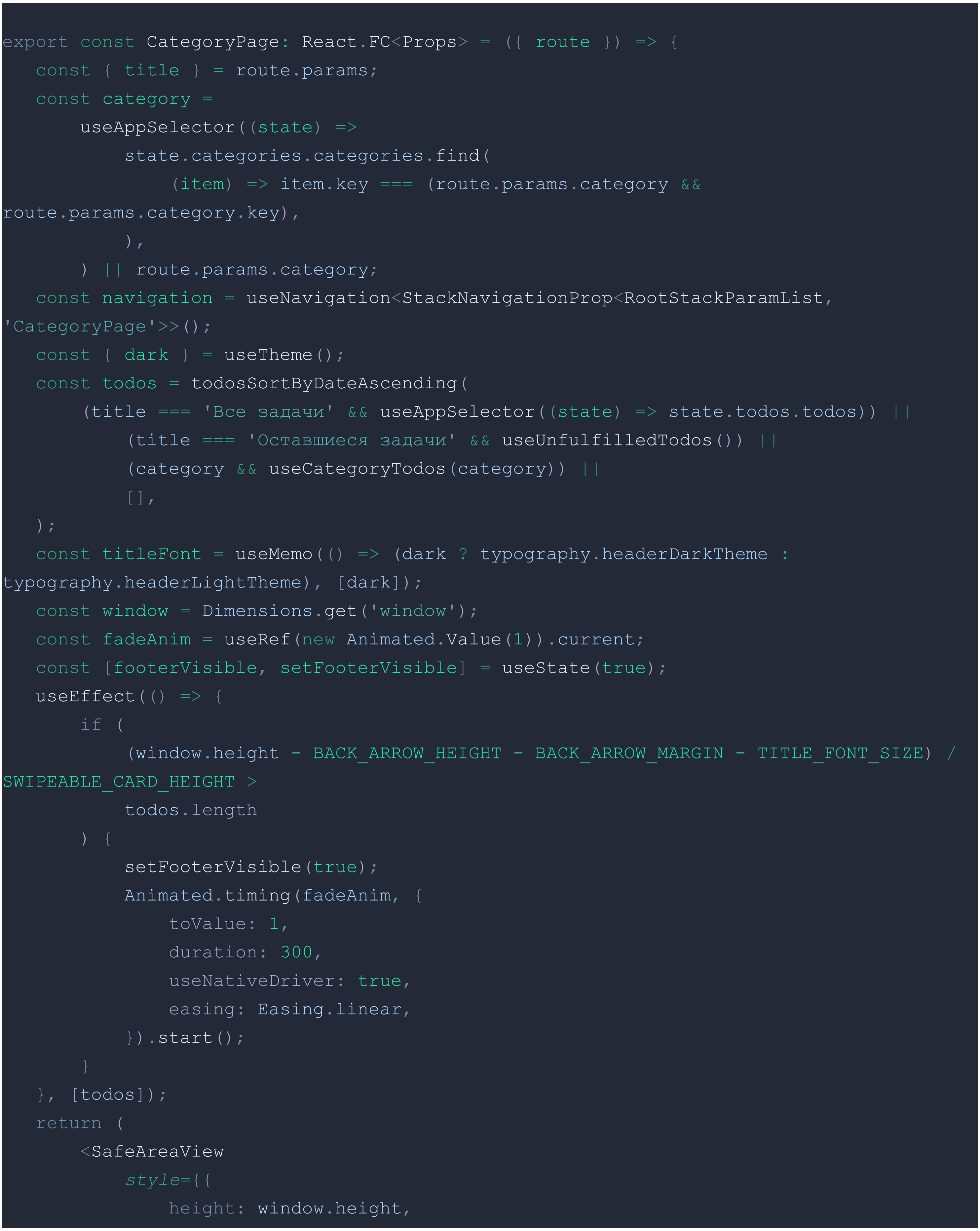


```
                    />
                    <Text style={[fonts.text, styles.subtitle]}>Выберите
категорию</Text>
                    <View style={{ maxHeight: 135, marginLeft: 88 }}>
                        <ScrollView style={{ width: 270 }}>
                            <View style={styles.categoryList}>
                                {categories.map((item) => (
                                    <SelectCategoryCard
                                        category={item}
                                        style={{ margin: 5, marginRight: 0 }}
                                        key={item.key}
                                        onPress={() => setCategory(item)}
                                        selected={item.key === category?.key}
                                    />
                                ))}
                                <IconAddCategoryButton
                                    style={{ marginLeft: 5, marginRight: 0, marginTop: 2
}}
                                    type={Size.smallButtonSizes}
                                    onPress={() => navigation.push('AddCategoryPage')}
                                />
                            </View>
                        </ScrollView>
                    </View>
                    <Text style={[fonts.text, styles.subtitle, { marginBottom: 0,
marginTop: 10 }]}>
                        Добавьте описание
                    </Text>
                    <DescriptionTextInput
                        style={styles.description}
                        onChangeText={setDescription}
                        defaultValue={description || ''}
                    />
                    <ActionButton
                        title="Добавить"
                        active={!!title?.trim() && !!category && !mistake}
                        style={[styles.button, deployed && { marginBottom: 90 }]}
                        onPress={() => {
                            if (title && category)
                                dispatch(
                                    addTodo({
                                        key: Date.now().toString(),
                                        title,
                                        description: description || '',
                                        completed: false,
                                        category,
                                        date,
                                        dateType: Array.isArray(date)
                                            ? DateTypes.INTERVAL
                                            : (date && DateTypes.DEADLINE) ||
DateTypes.WITHOUT,
                                    }),
                                );
                            setTitle('');
                            setDescription('');
                            setCategory(selectedCategory);
                            setDate(undefined);
                        }}
                    />
                </ScrollView>
            </KeyboardAvoidingView>
        </SafeAreaView>
    );
};
```

```
./src/components/pages/AddTaskPage/styles.ts

import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    backArrow: { marginLeft: 34, marginTop: 22 },
    categoryList: { width: 270, flexDirection: 'row', flexWrap: 'wrap' },
    titleInput: { marginLeft: 38, marginTop: 36, marginBottom: 25, maxWidth: 280 },
    subtitle: { marginLeft: 93, marginBottom: 15, fontSize: 14 },
    description: {
        height: 155,
        maxWidth: 270,
        marginLeft: 93,
        marginRight: 5,
    },
    button: { marginLeft: 95, width: 106, marginTop: 20, marginBottom: 45 },
    mistake: {
        backgroundColor: COLORS.EXPIRED,
        height: 20,
        borderRadius: 5,
        maxWidth: 210,
        justifyContent: 'center',
        alignItems: 'center',
    },
 });
```

./src/components/pages/CategoryPage/CategoryPage.tsx

```
import { RouteProp, useNavigation, useTheme } from '@react-navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import React, { useEffect, useMemo, useRef, useState } from 'react';
import { Text, SafeAreaView, ScrollView, Dimensions, Platform, Animated, Easing } from
'react-native';
import { ICONS } from '../../../../assets';
import { typography } from '../../../../assets/globalStyles';
import { useAppSelector } from '../../../store/hooks';
import { ICategory } from '../../../store/types';
import { isCloseToBottom } from '../../../utils/isCloseToBottom';
import { todosSortByDateAscending } from '../../../utils/todosSortByDate';
import { BackArrowButton } from '../../atomic/BackArrowButton/BackArrowButton';
import { GradientFooter } from '../../atomic/GradientFooter/GradientFooter';
import { IconActionButton } from '../../atomic/IconActionButton/IconActionButton';
import { IconTransparentButton } from
'../../atomic/IconTransparentButton/IconTransparentButton';
import { TodoSwipeableCard } from '../../atomic/TodoSwipeableCard/TodoSwipeableCard';
import { RootStackParamList } from '../../RootStackParams';
import { styles } from './styles';

type CategoryPageRouteProp = RouteProp<RootStackParamList, 'CategoryPage'>;
type Props = { route: CategoryPageRouteProp };

const useUnfulfilledTodos = () => useAppSelector((state) =>
state.todos.todos.filter((todo) => !todo.completed));
const useCategoryTodos = (category: ICategory) =>
   useAppSelector((state) => state.todos.todos.filter((todo) => todo.category.key ===
category.key));

const SWIPEABLE_CARD_HEIGHT = 68;
const BACK_ARROW_MARGIN = 22;
const BACK_ARROW_HEIGHT = 22;
const TITLE_FONT_SIZE = 24;
```



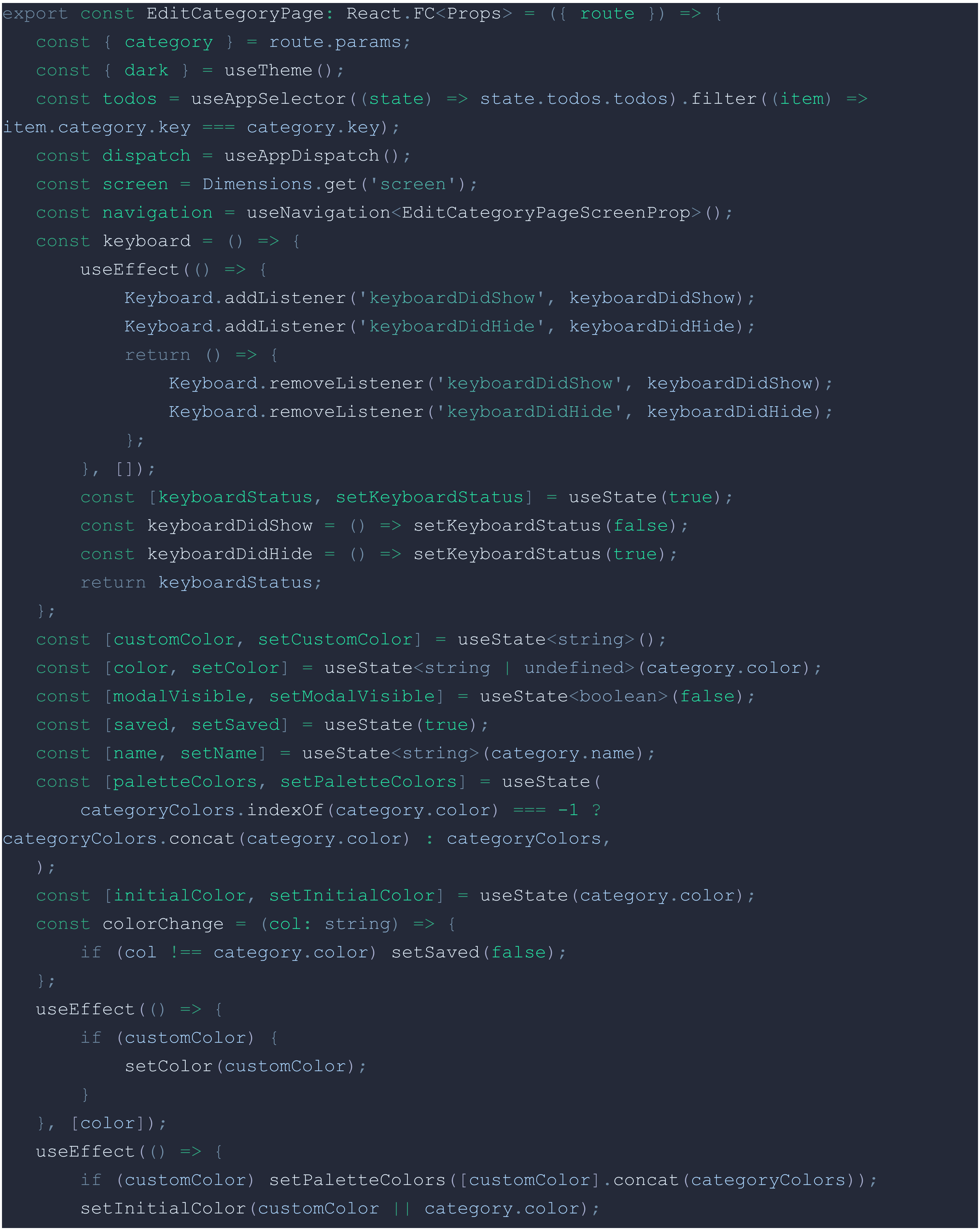
```
export const CategoryPage: React.FC<Props> = ({ route }) => {
    const { title } = route.params;
    const category =
        useAppSelector((state) =>
            state.categories.categories.find(
                (item) => item.key === (route.params.category &&
route.params.category.key),
            ),
        ) || route.params.category;
    const navigation = useNavigation<StackNavigationProp<RootStackParamList,
'CategoryPage'>>();
    const { dark } = useTheme();
    const todos = todosSortByDateAscending(
        (title === 'Все задачи' && useAppSelector((state) => state.todos.todos)) ||
            (title === 'Оставшиеся задачи' && useUnfulfilledTodos()) ||
            (category && useCategoryTodos(category)) ||
            [],
    );
    const titleFont = useMemo(() => (dark ? typography.headerDarkTheme :
typography.headerLightTheme), [dark]);
    const window = Dimensions.get('window');
    const fadeAnim = useRef(new Animated.Value(1)).current;
    const [footerVisible, setFooterVisible] = useState(true);
    useEffect(() => {
        if (
            (window.height - BACK_ARROW_HEIGHT - BACK_ARROW_MARGIN - TITLE_FONT_SIZE) /
SWIPEABLE_CARD_HEIGHT >
            todos.length
        ) {
            setFooterVisible(true);
            Animated.timing(fadeAnim, {
                toValue: 1,
                duration: 300,
                useNativeDriver: true,
                easing: Easing.linear,
            }).start();
        }
    }, [todos]);
    return (
        <SafeAreaView
            style={{
                height: window.height,
                paddingBottom: Platform.OS !== 'ios' ? 30 : undefined,
            }}
        >
            <ScrollView
                indicatorStyle={dark ? 'white' : 'black'}
                style={[{ width: window.width, paddingBottom: footerVisible ? 50 :
undefined }]}
                onScroll={({ nativeEvent }) => {
                    if (
                        (window.height - BACK_ARROW_HEIGHT - BACK_ARROW_MARGIN -
TITLE_FONT_SIZE) /
                            SWIPEABLE_CARD_HEIGHT <
                        todos.length
                    ) {
                        setTimeout(() =>
setFooterVisible(!isCloseToBottom(nativeEvent)), footerVisible ? 200 : 0);
                        Animated.timing(fadeAnim, {
                            toValue: isCloseToBottom(nativeEvent) ? 0 : 1,
                            duration: 300,
                            useNativeDriver: true,
                            easing: Easing.linear,
                        }).start();
                    }
                }}
            >
                <BackArrowButton onPress={() => navigation.goBack()}
style={styles.backArrow} />
                <Text style={[titleFont, { alignSelf: 'center', textAlign: 'center',
maxWidth: 240 }]}>
                    {title || (category && category.name)}
                </Text>
                {category && (
                    <IconTransparentButton
                        icon={dark ? ICONS.imgSettingsDarkTheme :
ICONS.imgSettingsLightTheme}
                        style={styles.settings}
                        onPress={() => navigation.push('EditCategoryPage', { category
})}
                    />
                )}
                {todos.length === 0 && (
                    <Text style={[titleFont, { textAlign: 'center', marginTop: 30 }]}>
                        В данной категории задач нет
                    </Text>
                )}
                {todos
                    .slice()
                    .sort((a, b) => {
                        if ((a.completed && b.completed) || (!a.completed &&
!b.completed)) return 0;
                        return a.completed ? 1 : -1;
                    })
                    .map((el) => (
                        <TodoSwipeableCard todo={el} key={el.key} />
                    ))}
            </ScrollView>
            {footerVisible && (
                <Animated.View
                    style={{
                        opacity: fadeAnim,
                        position: 'absolute',
                        top: window.height - 60,
                        width: '100%',
                    }}
                >
                    <GradientFooter
                        style={{
                            position: 'absolute',
                            height: 60,
                            width: '100%',
                        }}
                    />
                    <IconActionButton
                        icon={ICONS.imgAddIconWhite}
                        style={{ left: window.width - 60, top: 10 }}
                        onPress={() => navigation.push('AddTaskPage', { category })}
                    />
                </Animated.View>
            )}
        </SafeAreaView>
    );
};
```

```
./src/components/pages/CategoryPage/styles.ts
```

```
import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
   backArrow: { marginLeft: 34, marginTop: 22 },
   settings: { position: 'absolute', alignSelf: 'flex-end', right: 20, marginTop: 42 },
});
```

./src/components/pages/EditCategoryPage/EditCategoryPage.tsx

```
import { useNavigation, RouteProp, useTheme } from '@react-navigation/native';
import React, { useState, useEffect } from 'react';
import {
   SafeAreaView,
   ScrollView,
   Keyboard,
   View,
   Dimensions,
   KeyboardAvoidingView,
   Platform,
   Alert,
} from 'react-native';
import { StackNavigationProp } from '@react-navigation/stack';
import { RootStackParamList } from '../../RootStackParams';
import { useAppDispatch, useAppSelector } from '../../../store/hooks';
import { BackArrowButton } from '../../atomic/BackArrowButton/BackArrowButton';
import { ActionButton } from '../../atomic/ActionButton/ActionButton';
import { categoryColors, PaletteBar } from '../../atomic/PaletteBar/PaletteBar';
import { TitleTextInput } from '../../atomic/TitleTextInput/TitleTextInput';
import { deleteCategory, editCategory } from '../../../store/categorySlice';
import { MoreColorsButton } from '../../atomic/MoreColorsButton/MoreColorsButton';
import { ModalColorPicker } from '../../atomic/ModalColorPicker/ModalColorPicker';
import { styles } from './styles';
import { IconTransparentButton } from
'../../atomic/IconTransparentButton/IconTransparentButton';
import { ICONS } from '../../../../assets';
import { deleteTodo, editTodo } from '../../../store/todoSlice';

type EditCategoryPageRouteProp = RouteProp<RootStackParamList, 'EditCategoryPage'>;
type EditCategoryPageScreenProp = StackNavigationProp<RootStackParamList,
'EditCategoryPage'>;
type Props = { route: EditCategoryPageRouteProp };
```



```
export const EditCategoryPage: React.FC<Props> = ({ route }) => {
    const { category } = route.params;
    const { dark } = useTheme();
    const todos = useAppSelector((state) => state.todos.todos).filter((item) =>
item.category.key === category.key);
    const dispatch = useAppDispatch();
    const screen = Dimensions.get('screen');
    const navigation = useNavigation<EditCategoryPageScreenProp>();
    const keyboard = () => {
        useEffect(() => {
            Keyboard.addListener('keyboardDidShow', keyboardDidShow);
            Keyboard.addListener('keyboardDidHide', keyboardDidHide);
            return () => {
                Keyboard.removeListener('keyboardDidShow', keyboardDidShow);
                Keyboard.removeListener('keyboardDidHide', keyboardDidHide);
            };
        }, []);
        const [keyboardStatus, setKeyboardStatus] = useState(true);
        const keyboardDidShow = () => setKeyboardStatus(false);
        const keyboardDidHide = () => setKeyboardStatus(true);
        return keyboardStatus;
    };
    const [customColor, setCustomColor] = useState<string>();
    const [color, setColor] = useState<string | undefined>(category.color);
    const [modalVisible, setModalVisible] = useState<boolean>(false);
    const [saved, setSaved] = useState(true);
    const [name, setName] = useState<string>(category.name);
    const [paletteColors, setPaletteColors] = useState(
        categoryColors.indexOf(category.color) === -1 ?
categoryColors.concat(category.color) : categoryColors,
    );
    const [initialColor, setInitialColor] = useState(category.color);
    const colorChange = (col: string) => {
        if (col !== category.color) setSaved(false);
    };
    useEffect(() => {
        if (customColor) {
            setColor(customColor);
        }
    }, [color]);
    useEffect(() => {
        if (customColor) setPaletteColors([customColor].concat(categoryColors));
        setInitialColor(customColor || category.color);
    }, [customColor]);

    return (
        <SafeAreaView accessible>
            <KeyboardAvoidingView
                behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
                style={{ height: screen.height }}
            >
                <ScrollView style={{ height: screen.height }}>
                    <View style={{ position: 'absolute', width: 375 }}>
                        <BackArrowButton
                            onPress={() => {
                                navigation.goBack();
                            }}
                            style={styles.arrow}
                        />
                        <View style={styles.headline}>
                            <TitleTextInput
                                style={styles.titleInput}
                                defaultValue={name}
                                onChangeText={(text) => {
                                    setName(text.trim());
                                    setSaved(false);
                                }}
                                placeholder="Название категории..."
                            />
                            <IconTransparentButton
                                icon={dark ? ICONS.imgTrashIconWhite :
ICONS.imgTrashIconGrey}
                                style={styles.icon}
                                onPress={() =>
                                    Alert.alert(
                                        'Подтвердите действие',
                                        'Вы уверены, что хотите безвозвратно удалить
категорию? Все задачи этой категории будут удалены.',
                                        [
                                            {
                                                text: 'Удалить',
                                                onPress: () => {
                                                    todos.map((todo) =>
dispatch(deleteTodo(todo)));
                                                    dispatch(deleteCategory(category));
                                                    navigation.navigate('MainPage');
                                                },
                                            },
                                            { text: 'Отмена' },
                                        ],
                                    )
                                }
                            />
                        </View>
                    </View>
                    <View style={styles.main}>
                        <PaletteBar
                            style={{ marginRight: -12 }}
                            colors={paletteColors}
                            initColor={initialColor}
                            onColorSelected={(pickedColor) => {
                                setColor(pickedColor);
                                colorChange(pickedColor);
                            }}
                        />
                        <MoreColorsButton style={{ marginTop: 28 }} onPress={() =>
setModalVisible(true)} />
                        <ModalColorPicker
                            onColorSelected={(newColor: string) => {
                                setCustomColor(newColor);
                                setColor(newColor);
                                setModalVisible(false);
                            }}
                            visible={modalVisible}
                            onMissClick={() => setModalVisible(false)}
                        />
                    </View>
                    {keyboard() ? (
                        <ActionButton
                            title="Сохранить"
                            active={!saved && !!name.trim()}
                            style={styles.button}
                            onPress={() => {
                                dispatch(
                                    editCategory({
                                        ...category,
                                        name,
                                        color: color || category.color,
                                    }),
                                );
                                todos.map((todo) =>
                                    dispatch(
                                        editTodo({
                                            ...todo,
                                            category: { ...category, name, color: color || category.color },
                                        }),
                                    ),
                                );
                                setSaved(true);
                            }}
                        />
                    ) : null}
                </ScrollView>
            </KeyboardAvoidingView>
        </SafeAreaView>
    );
};
```

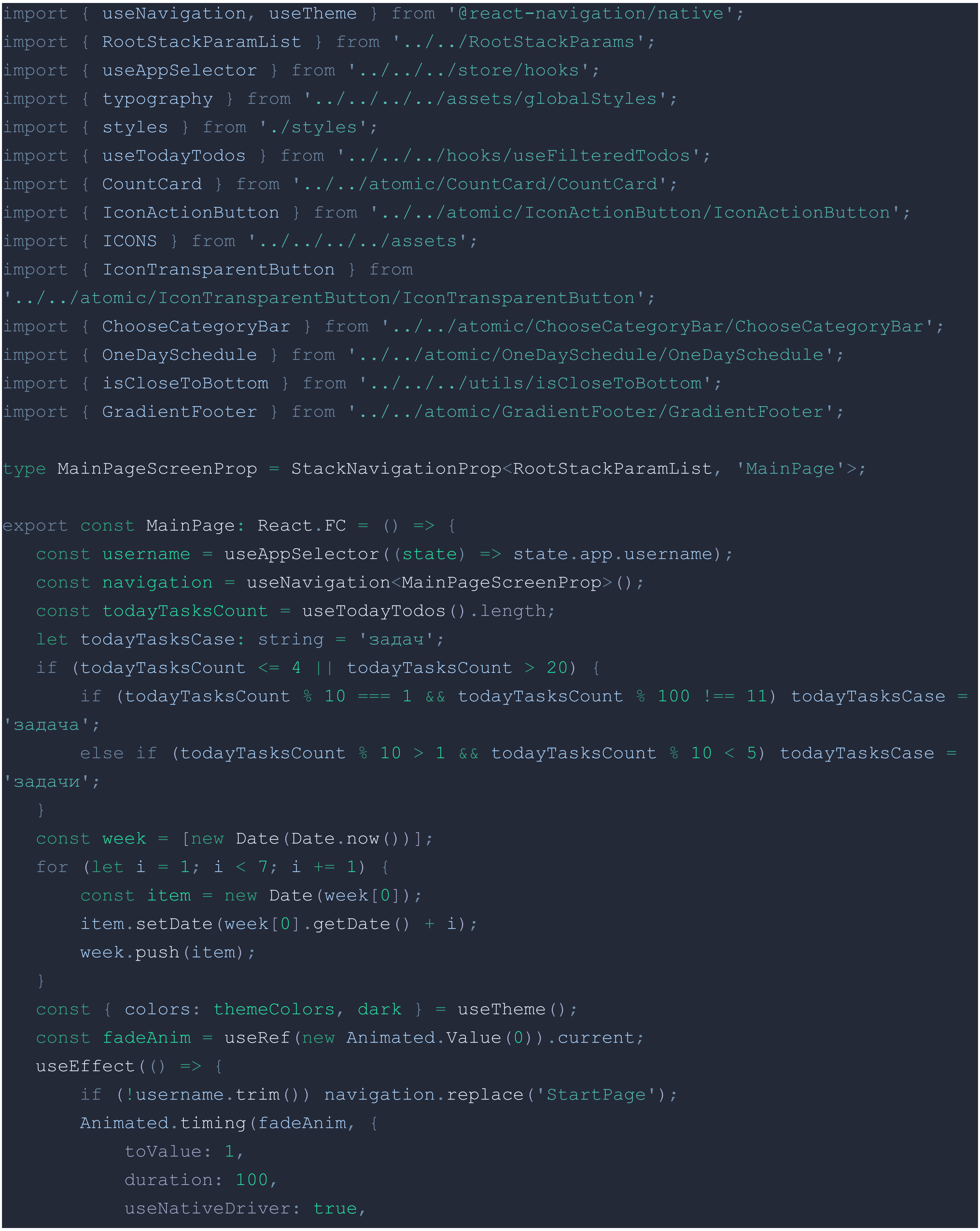
./src/components/pages/EditCategoryPage/styles.ts

```
import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    titleInput: {
        maxWidth: 242,
        marginLeft: 66,
        height: 34,
    },
    button: {
        marginBottom: 40,
        width: 112,
        marginLeft: 93,
        marginTop: 230,
    },
    arrow: {
        marginLeft: 34,
        marginTop: 22,
    },
    main: {
        marginTop: 248,
        marginRight: 20,
        alignItems: 'flex-end',
    },
    headline: {
        flexDirection: 'row',
        marginTop: 36,
        justifyContent: 'space-between',
    },
    icon: {
        height: 26,
        width: 26,
        marginRight: 26,
        alignSelf: 'baseline',
    },
});


./src/components/pages/EditTaskPage/EditTaskPage.tsx

import React, { useState, useRef, useEffect } from 'react';
import {
    Text,
    SafeAreaView,
    ScrollView,
    Keyboard,
    View,
    TextInput,
    Dimensions,
    KeyboardAvoidingView,
    Platform,
} from 'react-native';
import { StackNavigationProp } from '@react-navigation/stack';
import { useNavigation, RouteProp, useTheme } from '@react-navigation/native';
import { RootStackParamList } from '../../RootStackParams';
import { useAppDispatch, useAppSelector } from '../../../store/hooks';
import { BackArrowButton } from '../../atomic/BackArrowButton/BackArrowButton';
import { ActionButton } from '../../atomic/ActionButton/ActionButton';
import { TitleTextInput } from '../../atomic/TitleTextInput/TitleTextInput';
import { SelectCategoryCard } from
'../../atomic/SelectCategoryCard/SelectCategoryCard';
import { DropdownButton } from '../../atomic/DropdownButton/DropdownButton';
import { DescriptionTextInput } from
'../../atomic/DescriptionTextInput/DescriptionTextInput';
```

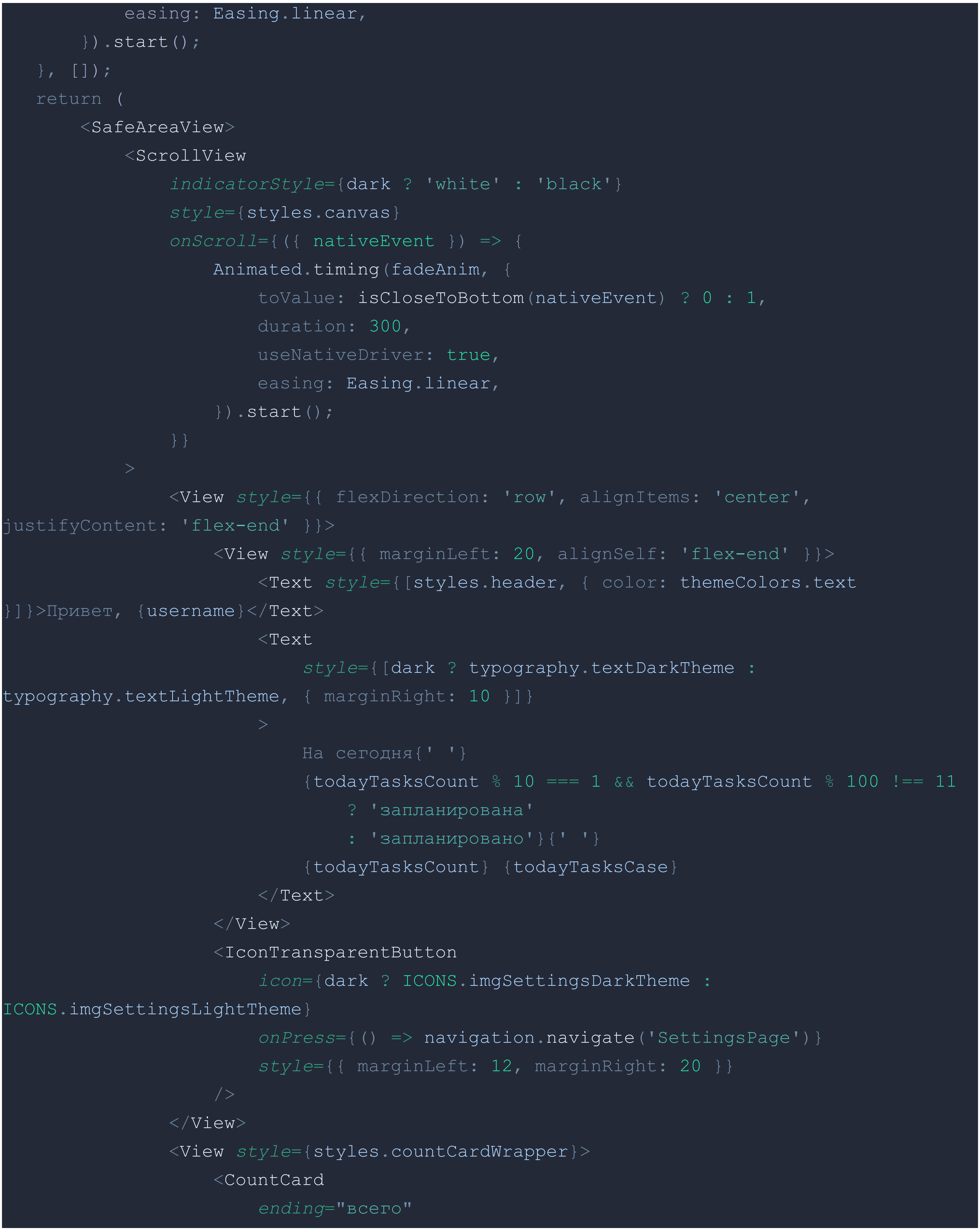
```
import { styles } from './styles';
import { createCustomDateStringFromDate, createCustomTimeStringFromDate } from
'../../../utils/dateConvertations';
import { ChooseDateBar } from '../../atomic/ChooseDateBar/ChooseDateBar';
import { DateTypes, ICategory } from '../../../store/types';
import { IconAddCategoryButton } from
'../../atomic/IconAddCategoryButton/IconAddCategoryButton';
import { Size } from '../../atomic/IconAddCategoryButton/styles';
import { typography } from '../../../../assets/globalStyles';
import { editTodo } from '../../../store/todoSlice';

type EditTaskPageRouteProp = RouteProp<RootStackParamList, 'EditTaskPage'>;
type EditTaskPageScreenProp = StackNavigationProp<RootStackParamList, 'EditTaskPage'>;
type Props = { route: EditTaskPageRouteProp };

export const EditTaskPage: React.FC<Props> = ({ route }) => {
   const { dark } = useTheme();
   const dispatch = useAppDispatch();
   const screen = Dimensions.get('screen');
   const categories = useAppSelector((state) => state.categories.categories);
   const { todo } = route.params;
   const inputRef = useRef<TextInput>(null);
   const [deployed, setDeployed] = useState(false);
   const [saved, setSaved] = useState(true);
   const [platform] = useState(Platform.OS);
   const [mistake, setMistake] = useState(false);
   const [category, setCategory] = useState<ICategory>(todo.category);
   const [title, setTitle] = useState<string>(todo.title);
   const [description, setDescription] = useState<string>(todo.description);
   const [date, setDate] = useState<Date | [start: Date, end: Date] |
undefined>(todo.date);
   const dateStr = Array.isArray(date) ? date[0] : date;
   const dateTitle =
       (dateStr?.toDateString() === new Date(Date.now()).toDateString() && 'Сегодня')
||
       (dateStr?.toDateString() === new Date(Date.now() + 1000 * 60 * 60 *
24).toDateString() && 'Завтра') ||
       (dateStr && createCustomDateStringFromDate(dateStr)) ||
       'Без даты';
   const navigation = useNavigation<EditTaskPageScreenProp>();
   const keyboard = () => {
       useEffect(() => {
           Keyboard.addListener('keyboardDidShow', keyboardDidShow);
```

```
            Keyboard.addListener('keyboardDidHide', keyboardDidHide);
            return () => {
                Keyboard.removeListener('keyboardDidShow', keyboardDidShow);
                Keyboard.removeListener('keyboardDidHide', keyboardDidHide);
            };
        }, []);
        const [keyboardStatus, setKeyboardStatus] = useState(true);
        const keyboardDidShow = () => setKeyboardStatus(false);
        const keyboardDidHide = () => setKeyboardStatus(true);
        return keyboardStatus;
    };
    useEffect(() => {
        setMistake(Array.isArray(date) ? date[0].getTime() - date[1].getTime() >= 0 :
false);
        if (saved && Array.isArray(date) && Array.isArray(todo.date))
            setSaved(date[0].getTime() - todo.date[0].getTime() === 0);
        else if (saved && !Array.isArray(todo.date) && !Array.isArray(date) && date &&
todo.date)
            setSaved(date.getTime() - todo.date.getTime() === 0);
        else if (saved && !todo.date && !date) setSaved(true);
        else setSaved(false);
    }, [date]);
    return (
        <SafeAreaView accessible>
            <KeyboardAvoidingView
                behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
                style={{ height: screen.height }}
            >
                <ScrollView style={{ height: screen.height }} indicatorStyle={dark ?
'white' : 'black'}>
                    <View style={{ position: 'absolute', width: 375 }}>
                        <BackArrowButton
                            onPress={() => {
                                navigation.goBack();
                            }}
                            style={styles.arrow}
                        />
                        <TitleTextInput
                            style={styles.titleInput}
                            defaultValue={title}
                            onChangeText={(text) => {
                                setTitle(text.trim());
                                setSaved(false);
                            }}
                            placeholder="Название задачи..."
                        />
                    </View>
                    <View style={styles.main}>
                        <Text
                            style={[
                                dark ? typography.subtitleDarkTheme :
typography.subtitleLightTheme,
                                { fontSize: 14 },
                            ]}
                        >
                            Дата
                        </Text>
                        <DropdownButton
                            onPress={platform !== 'ios' ? () => setDeployed(!deployed) :
undefined}
                            style={[
                                {
                                    zIndex: platform !== 'ios' ? 1 : 2,
                                    marginBottom: platform === 'ios' ? 15 : (deployed &&
-25) || 1,
                                    marginTop: 10,
                                    height: (deployed && 400) || undefined,
                                },
                            ]}
                            title={
                                Array.isArray(date)
                                    ? `${dateTitle}, ${createCustomTimeStringFromDate(
                                          date[0],
                                      )} - ${createCustomTimeStringFromDate(date[1])}`
                                    : (date && `${dateTitle},
${createCustomTimeStringFromDate(date)}`) || 'Без даты'
                            }
                            titleStyle={mistake ? styles.mistake : undefined}
                            component={
                                <ChooseDateBar
                                    initialValue={date}
                                    style={{
                                        position: platform === 'ios' ? 'absolute' :
'relative',
                                        marginTop: platform === 'ios' ? 10 : 1,
                                        marginLeft: -15,
                                        zIndex: 1,
                                    }}
                                    onDateChange={setDate}
                                />
                            }
                        />
                        <Text
                            style={[
                                dark ? typography.subtitleDarkTheme :
typography.subtitleLightTheme,
                                { marginTop: 36, fontSize: 14 },
                            ]}
                        >
                            Категория
                        </Text>
                        <View style={styles.categoryBar}>
                            <View style={{ width: 76 }}>
                                <SelectCategoryCard category={category} />
                            </View>
                            <DropdownButton
                                title="Изменить категорию"
                                style={{
                                    paddingLeft: -70,
                                    zIndex: platform !== 'ios' ? 1 : 2,
                                }}
                                titleStyle={{ height: 30, marginBottom: 10 }}
                                component={
                                    <ScrollView style={styles.chooseCategoryBar}>
                                        <View style={styles.categoryList}>
                                            {categories.map((item) => (
                                                <SelectCategoryCard
                                                    category={item}
                                                    style={{ marginTop: 5, marginRight:
5 }}

                                                    key={item.key}
                                                    onPress={() => {
                                                        setCategory(item);
                                                        setSaved(saved && item.key ===
todo.category.key);

                                                    }}
                                                    selected={item.key ===
category?.key}
                                                />
                                            ))}
                                            <IconAddCategoryButton
                                                style={styles.addIcon}
                                                type={Size.smallButtonSizes}
                                                onPress={() =>
navigation.push('AddCategoryPage')}
                                            />
                                        </View>
                                    </ScrollView>
                                }
                            />
                        </View>
                        <Text
                            style={[
                                dark ? typography.subtitleDarkTheme :
typography.subtitleLightTheme,
                                { marginTop: 36, fontSize: 14 },
                            ]}
                        >
                            Описание
                        </Text>
                        <DescriptionTextInput
                            style={styles.descriptionInput}
                            ref={inputRef}
                            defaultValue={description}
                            onChangeText={(text) => {
                                setDescription(text.trim());
                                setSaved(false);
                            }}
                        />
                    </View>

                    {keyboard() ? (
                        <ActionButton
                            title="Сохранить"
                            active={!mistake && !saved && !!title.trim()}
                            style={[
                                styles.button,
                                deployed && { marginBottom: 90 },
                                platform === 'ios' && { zIndex: -1 },
                            ]}
                            onPress={() => {
                                dispatch(
                                    editTodo({
                                        ...todo,
                                        title,
                                        description,
                                        date,
                                        category,
                                        dateType: Array.isArray(date)
                                            ? DateTypes.INTERVAL
                                            : (date && DateTypes.DEADLINE) ||
DateTypes.WITHOUT,
                                    }),
                                );
                                setSaved(true);
                            }}
                        />
                    ) : null}
                </ScrollView>
            </KeyboardAvoidingView>
        </SafeAreaView>
    );
};
```

```
./src/components/pages/EditTaskPage/styles.ts

import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    titleInput: {
        marginLeft: 72,
        maxWidth: 280,
        marginTop: 36,
    },
    descriptionInput: {
        maxWidth: 270,
        marginTop: 10,
    },
    button: {
        marginBottom: 45,
        width: 112,
        marginLeft: 93,
        marginTop: 20,
    },
    arrow: {
        marginLeft: 34,
        marginTop: 22,
    },
    categoryList: {
        width: 270,
        flexDirection: 'row',
        flexWrap: 'wrap',
    },
    main: {
        marginTop: 142,
        marginLeft: 93,
    },
    categoryBar: {
        marginTop: 15,
        flexDirection: 'row',
    },
    chooseCategoryBar: {
        maxHeight: 140,
        marginLeft: -75,
        overflow: 'scroll',
        width: 270,
        zIndex: 1,
    },
    mistake: {
        backgroundColor: COLORS.EXPIRED,
        height: 20,
        borderRadius: 5,
        maxWidth: 210,
        justifyContent: 'center',
        alignItems: 'center',
    },
    addIcon: { marginLeft: 5, marginRight: 0, marginTop: 4 },
});


./src/components/pages/MainPage/MainPage.tsx

import React, { useEffect, useRef } from 'react';
import { Text, View, SafeAreaView, ScrollView, Animated, Easing, Platform } from
'react-native';
import { StackNavigationProp } from '@react-navigation/stack';
```



```
import { useNavigation, useTheme } from '@react-navigation/native';
import { RootStackParamList } from '../../RootStackParams';
import { useAppSelector } from '../../../store/hooks';
import { typography } from '../../../../assets/globalStyles';
import { styles } from './styles';
import { useTodayTodos } from '../../../hooks/useFilteredTodos';
import { CountCard } from '../../atomic/CountCard/CountCard';
import { IconActionButton } from '../../atomic/IconActionButton/IconActionButton';
import { ICONS } from '../../../../assets';
import { IconTransparentButton } from
'../../atomic/IconTransparentButton/IconTransparentButton';
import { ChooseCategoryBar } from '../../atomic/ChooseCategoryBar/ChooseCategoryBar';
import { OneDaySchedule } from '../../atomic/OneDaySchedule/OneDaySchedule';
import { isCloseToBottom } from '../../../utils/isCloseToBottom';
import { GradientFooter } from '../../atomic/GradientFooter/GradientFooter';

type MainPageScreenProp = StackNavigationProp<RootStackParamList, 'MainPage'>;

export const MainPage: React.FC = () => {
    const username = useAppSelector((state) => state.app.username);
    const navigation = useNavigation<MainPageScreenProp>();
    const todayTasksCount = useTodayTodos().length;
    let todayTasksCase: string = 'задач';
    if (todayTasksCount <= 4 || todayTasksCount > 20) {
        if (todayTasksCount % 10 === 1 && todayTasksCount % 100 !== 11) todayTasksCase =
'задача';
        else if (todayTasksCount % 10 > 1 && todayTasksCount % 10 < 5) todayTasksCase =
'задачи';
    }
    const week = [new Date(Date.now())];
    for (let i = 1; i < 7; i += 1) {
        const item = new Date(week[0]);
        item.setDate(week[0].getDate() + i);
        week.push(item);
    }
    const { colors: themeColors, dark } = useTheme();
    const fadeAnim = useRef(new Animated.Value(0)).current;
    useEffect(() => {
        if (!username.trim()) navigation.replace('StartPage');
        Animated.timing(fadeAnim, {
            toValue: 1,
            duration: 100,
            useNativeDriver: true,
```

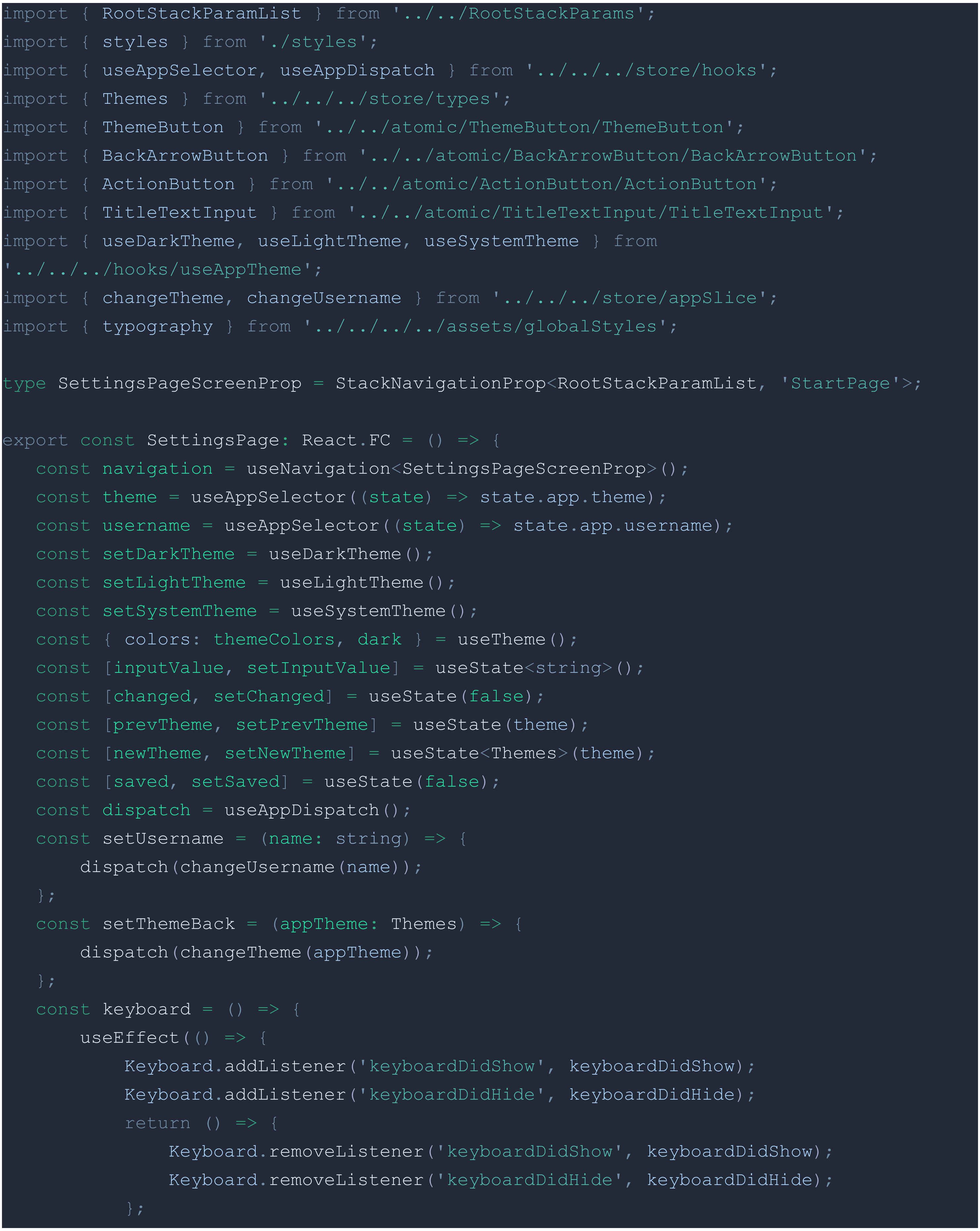


```
            easing: Easing.linear,
        }).start();
    }, []);
    return (
        <SafeAreaView>
            <ScrollView
                indicatorStyle={dark ? 'white' : 'black'}
                style={styles.canvas}
                onScroll={({ nativeEvent }) => {
                    Animated.timing(fadeAnim, {
                        toValue: isCloseToBottom(nativeEvent) ? 0 : 1,
                        duration: 300,
                        useNativeDriver: true,
                        easing: Easing.linear,
                    }).start();
                }}
            >
                <View style={{ flexDirection: 'row', alignItems: 'center',
justifyContent: 'flex-end' }}>
                    <View style={{ marginLeft: 20, alignSelf: 'flex-end' }}>
                        <Text style={[styles.header, { color: themeColors.text
}]}>Привет, {username}</Text>
                        <Text
                            style={[dark ? typography.textDarkTheme :
typography.textLightTheme, { marginRight: 10 }]}
                        >
                            На сегодня{' '}
                            {todayTasksCount % 10 === 1 && todayTasksCount % 100 !== 11
                                ? 'запланирована'
                                : 'запланировано'}{' '}
                            {todayTasksCount} {todayTasksCase}
                        </Text>
                    </View>
                    <IconTransparentButton
                        icon={dark ? ICONS.imgSettingsDarkTheme :
ICONS.imgSettingsLightTheme}
                        onPress={() => navigation.navigate('SettingsPage')}
                        style={{ marginLeft: 12, marginRight: 20 }}
                    />
                </View>
                <View style={styles.countCardWrapper}>
                    <CountCard
                        ending="всего"
                        style={{ marginHorizontal: 11 }}
                        onPress={() => navigation.navigate('CategoryPage', { title: 'Все
задачи' })}
                    />
                    <CountCard
                        ending="осталось"
                        style={{ marginHorizontal: 11 }}
                        onPress={() => navigation.navigate('CategoryPage', { title:
'Оставшиеся задачи' })}
                    />
                    <IconActionButton
                        icon={ICONS.imgAddIconWhite}
                        onPress={() => navigation.push('AddTaskPage', { category:
undefined })}
                    />
                </View>
                <View style={styles.line} />
                <ChooseCategoryBar navigation={navigation} />
                {week.map((day) => (
                    <OneDaySchedule
                        key={day.getTime().toString()}
                        navigation={navigation}
                        date={day}
                        style={
                            week.indexOf(day) !== 6
                                ? { marginVertical: 20 }
                                : { marginTop: 20, marginBottom: Platform.OS === 'ios' ?
40 : 50 }
                        }
                    />
                ))}
            </ScrollView>
            <Animated.View style={{ opacity: fadeAnim }}>
                <GradientFooter style={{ position: 'absolute', bottom: 0, zIndex: 2,
height: 60, width: '100%' }} />
            </Animated.View>
        </SafeAreaView>
    );
};



./src/components/pages/MainPage/styles.ts
```

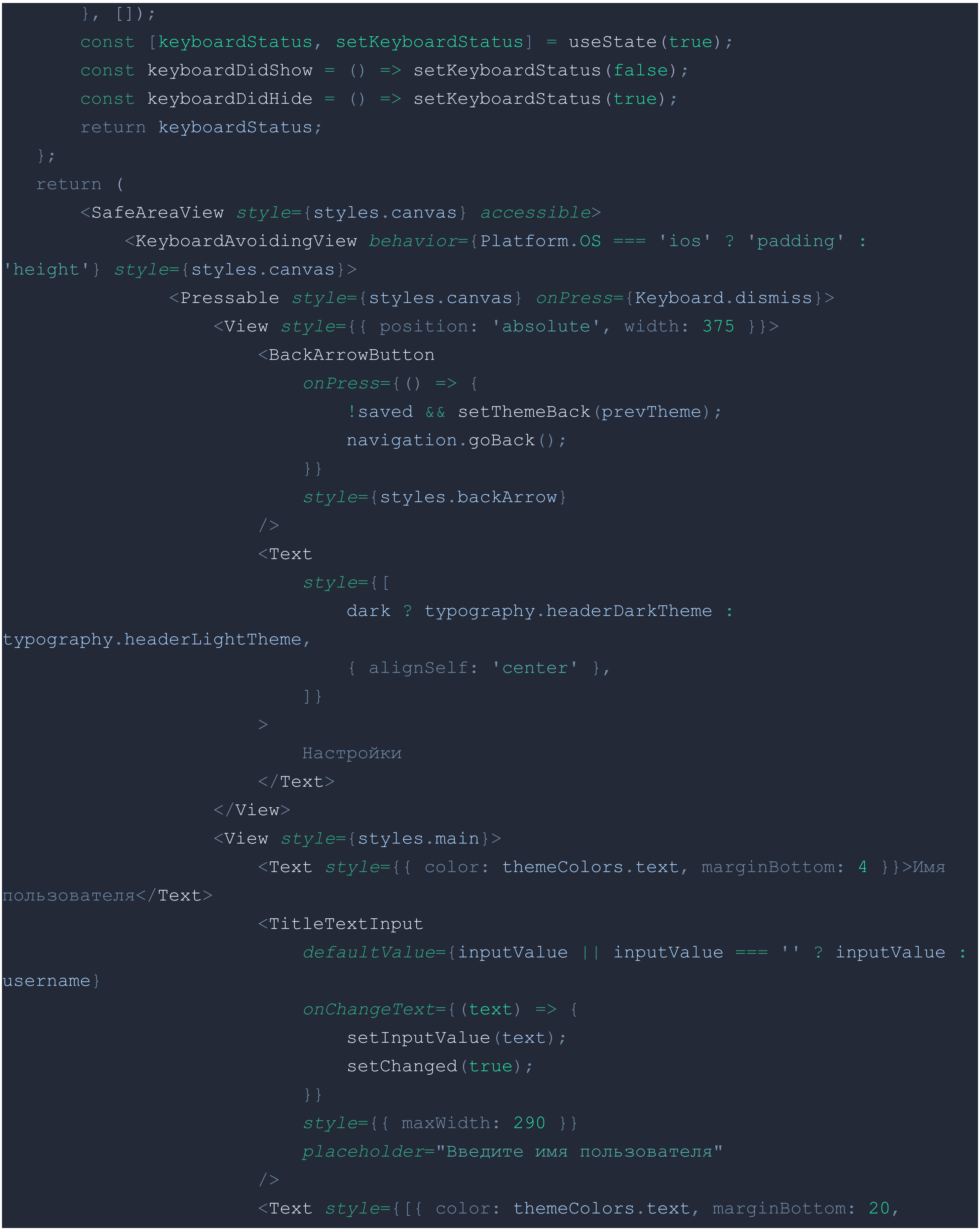
```
import { StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
   canvas: {
       paddingTop: 20,
       paddingLeft: 10,
   },
   header: {
       fontFamily: 'bold-poppins',
       fontSize: 30,
       fontWeight: '800',
       letterSpacing: 1,
       alignSelf: 'flex-start',
       justifyContent: 'flex-start',
       maxWidth: 290,
   },
   countCardWrapper: {
       flexDirection: 'row',
       alignSelf: 'center',
       alignItems: 'center',
       marginTop: 22,
       marginBottom: 15,
   },
   line: {
       height: 1.5,
       backgroundColor: COLORS.BUTTON_THEME_LIGHT,
       width: 300,
       marginRight: 10,
       alignSelf: 'flex-end',
       marginBottom: 23,
   },
});
```

./src/components/pages/SettingsPage/SettingsPage.tsx

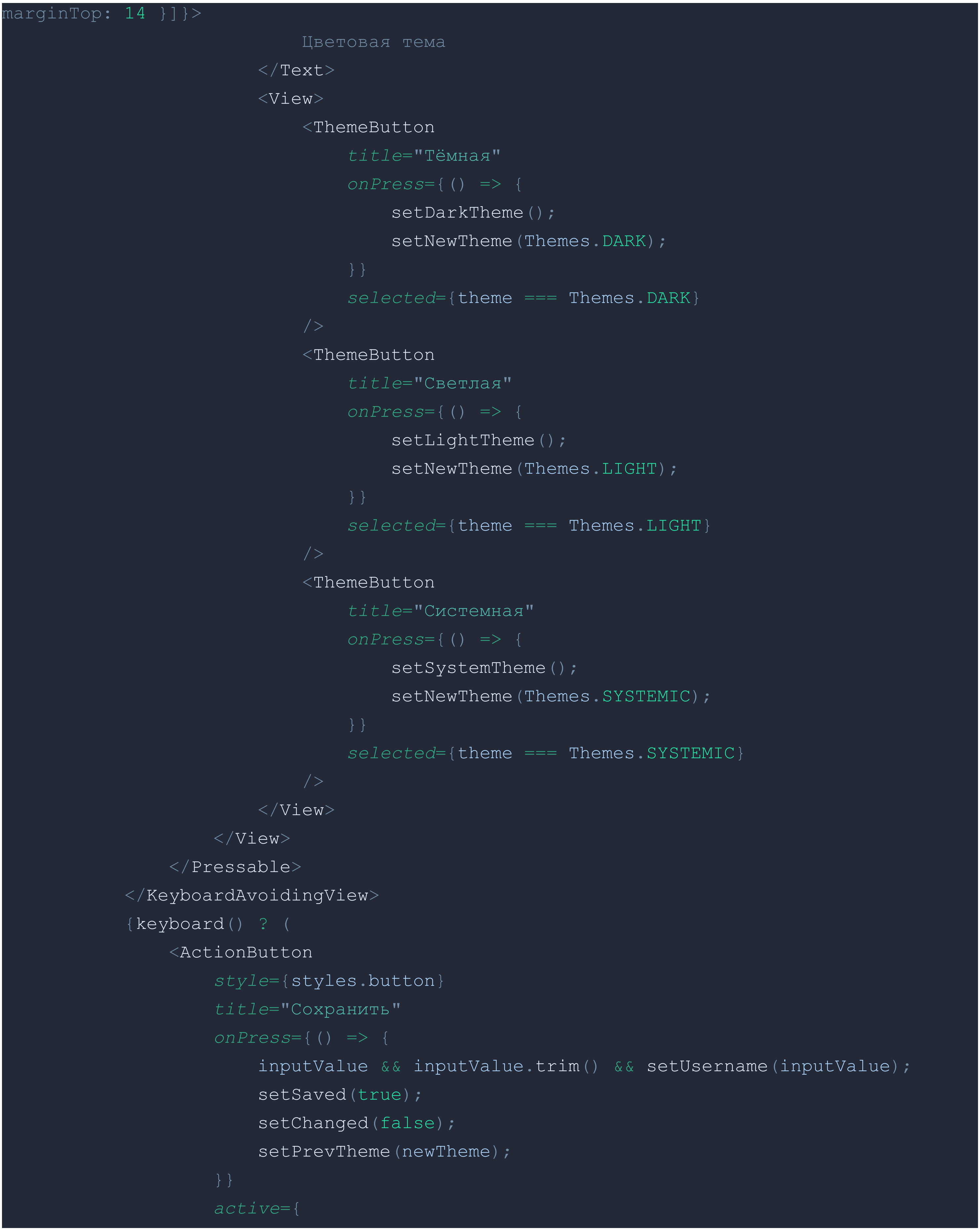
```
/* eslint-disable @typescript-eslint/no-unused-expressions */
import React, { useState, useEffect } from 'react';
import { Text, View, SafeAreaView, Pressable, KeyboardAvoidingView, Platform, Keyboard
} from 'react-native';
import { StackNavigationProp } from '@react-navigation/stack';
import { useNavigation, useTheme } from '@react-navigation/native';
```



```
import { RootStackParamList } from '../../RootStackParams';
import { styles } from './styles';
import { useAppSelector, useAppDispatch } from '../../../store/hooks';
import { Themes } from '../../../store/types';
import { ThemeButton } from '../../atomic/ThemeButton/ThemeButton';
import { BackArrowButton } from '../../atomic/BackArrowButton/BackArrowButton';
import { ActionButton } from '../../atomic/ActionButton/ActionButton';
import { TitleTextInput } from '../../atomic/TitleTextInput/TitleTextInput';
import { useDarkTheme, useLightTheme, useSystemTheme } from
'../../../hooks/useAppTheme';
import { changeTheme, changeUsername } from '../../../store/appSlice';
import { typography } from '../../../../assets/globalStyles';

type SettingsPageScreenProp = StackNavigationProp<RootStackParamList, 'StartPage'>;

export const SettingsPage: React.FC = () => {
    const navigation = useNavigation<SettingsPageScreenProp>();
    const theme = useAppSelector((state) => state.app.theme);
    const username = useAppSelector((state) => state.app.username);
    const setDarkTheme = useDarkTheme();
    const setLightTheme = useLightTheme();
    const setSystemTheme = useSystemTheme();
    const { colors: themeColors, dark } = useTheme();
    const [inputValue, setInputValue] = useState<string>();
    const [changed, setChanged] = useState(false);
    const [prevTheme, setPrevTheme] = useState(theme);
    const [newTheme, setNewTheme] = useState<Themes>(theme);
    const [saved, setSaved] = useState(false);
    const dispatch = useAppDispatch();
    const setUsername = (name: string) => {
        dispatch(changeUsername(name));
    };
    const setThemeBack = (appTheme: Themes) => {
        dispatch(changeTheme(appTheme));
    };
    const keyboard = () => {
        useEffect(() => {
            Keyboard.addListener('keyboardDidShow', keyboardDidShow);
            Keyboard.addListener('keyboardDidHide', keyboardDidHide);
            return () => {
                Keyboard.removeListener('keyboardDidShow', keyboardDidShow);
                Keyboard.removeListener('keyboardDidHide', keyboardDidHide);
            };
```



```
        }, []);
        const [keyboardStatus, setKeyboardStatus] = useState(true);
        const keyboardDidShow = () => setKeyboardStatus(false);
        const keyboardDidHide = () => setKeyboardStatus(true);
        return keyboardStatus;
    };
    return (
        <SafeAreaView style={styles.canvas} accessible>
            <KeyboardAvoidingView behavior={Platform.OS === 'ios' ? 'padding' :
'height'} style={styles.canvas}>
                <Pressable style={styles.canvas} onPress={Keyboard.dismiss}>
                    <View style={{ position: 'absolute', width: 375 }}>
                        <BackArrowButton
                            onPress={() => {
                                !saved && setThemeBack(prevTheme);
                                navigation.goBack();
                            }}
                            style={styles.backArrow}
                        />
                        <Text
                            style={[
                                dark ? typography.headerDarkTheme :
typography.headerLightTheme,
                                { alignSelf: 'center' },
                            ]}
                        >
                            Настройки
                        </Text>
                    </View>
                    <View style={styles.main}>
                        <Text style={{ color: themeColors.text, marginBottom: 4 }}>Имя
пользователя</Text>
                        <TitleTextInput
                            defaultValue={inputValue || inputValue === '' ? inputValue :
username}

                            onChangeText={(text) => {
                                setInputValue(text);
                                setChanged(true);
                            }}
                            style={{ maxWidth: 290 }}
                            placeholder="Введите имя пользователя"
                        />
                        <Text style={[{ color: themeColors.text, marginBottom: 20,
```



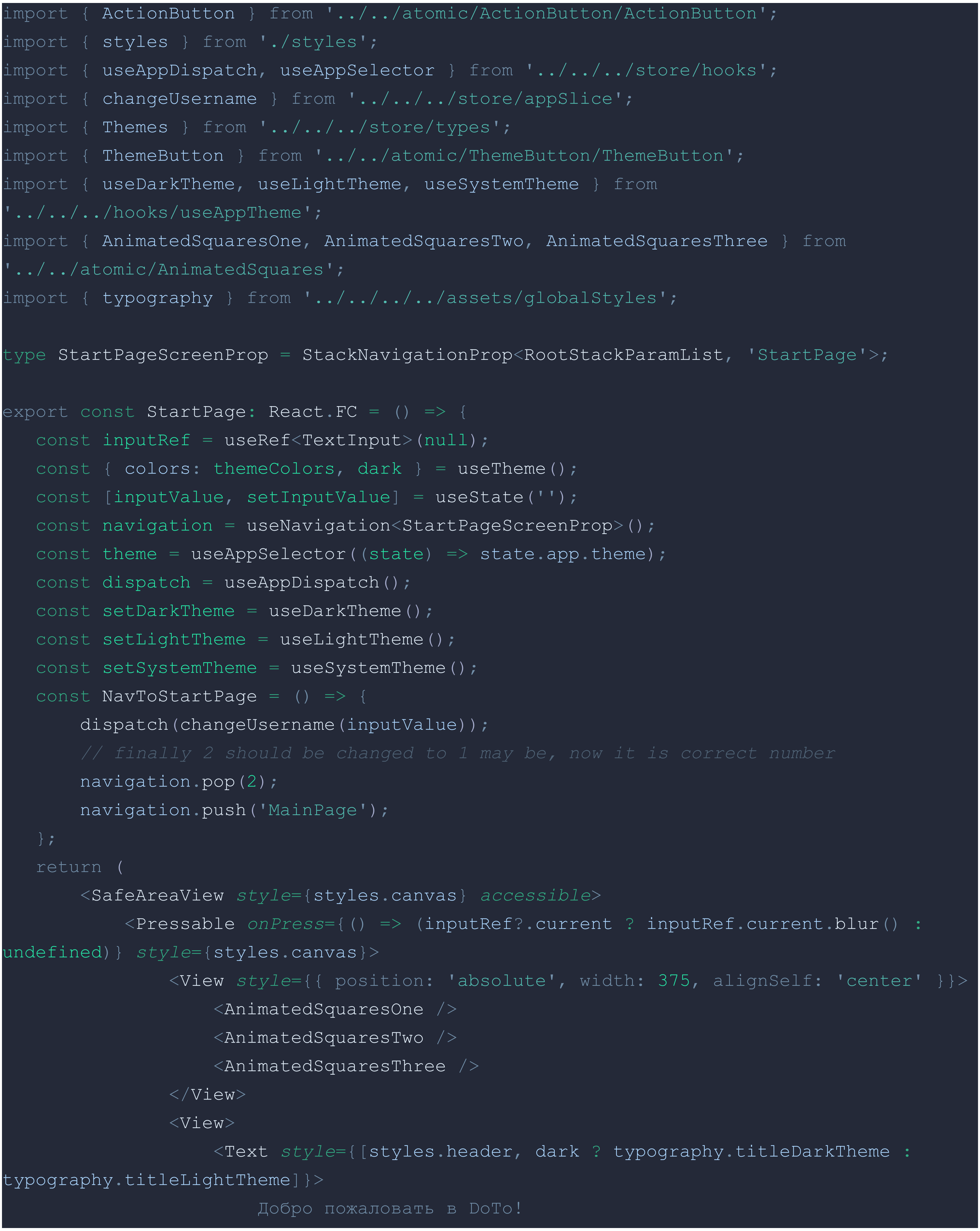
```
marginTop: 14 }]}>
                            Цветовая тема
                        </Text>
                        <View>
                            <ThemeButton
                                title="Тёмная"
                                onPress={() => {
                                    setDarkTheme();
                                    setNewTheme(Themes.DARK);
                                }}
                                selected={theme === Themes.DARK}
                            />
                            <ThemeButton
                                title="Светлая"
                                onPress={() => {
                                    setLightTheme();
                                    setNewTheme(Themes.LIGHT);
                                }}
                                selected={theme === Themes.LIGHT}
                            />
                            <ThemeButton
                                title="Системная"
                                onPress={() => {
                                    setSystemTheme();
                                    setNewTheme(Themes.SYSTEMIC);
                                }}
                                selected={theme === Themes.SYSTEMIC}
                            />
                        </View>
                    </View>
                </Pressable>
            </KeyboardAvoidingView>
            {keyboard() ? (
                <ActionButton
                    style={styles.button}
                    title="Сохранить"
                    onPress={() => {
                        inputValue && inputValue.trim() && setUsername(inputValue);
                        setSaved(true);
                        setChanged(false);
                        setPrevTheme(newTheme);
                    }}
                    active={
                        ((!inputValue && inputValue !== '') || !!inputValue?.trim()) &&
                        (changed || newTheme !== prevTheme)
                    }
                />
            ) : null}
        </SafeAreaView>
    );
};
```

./src/components/pages/SettingsPage/styles.ts

```
import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    canvas: {
        flex: 1,
        justifyContent: 'space-between',
    },
    backArrow: { marginLeft: 34, marginTop: 22 },
    header: {
        alignSelf: 'center',
    },
    main: {
        marginTop: 130,
        marginLeft: 54,
        justifyContent: 'center',
    },
    button: {
        bottom: 45,
        alignSelf: 'flex-start',
        marginLeft: 95,
    },
});
```

./src/components/pages/StartPage/StartPage.tsx

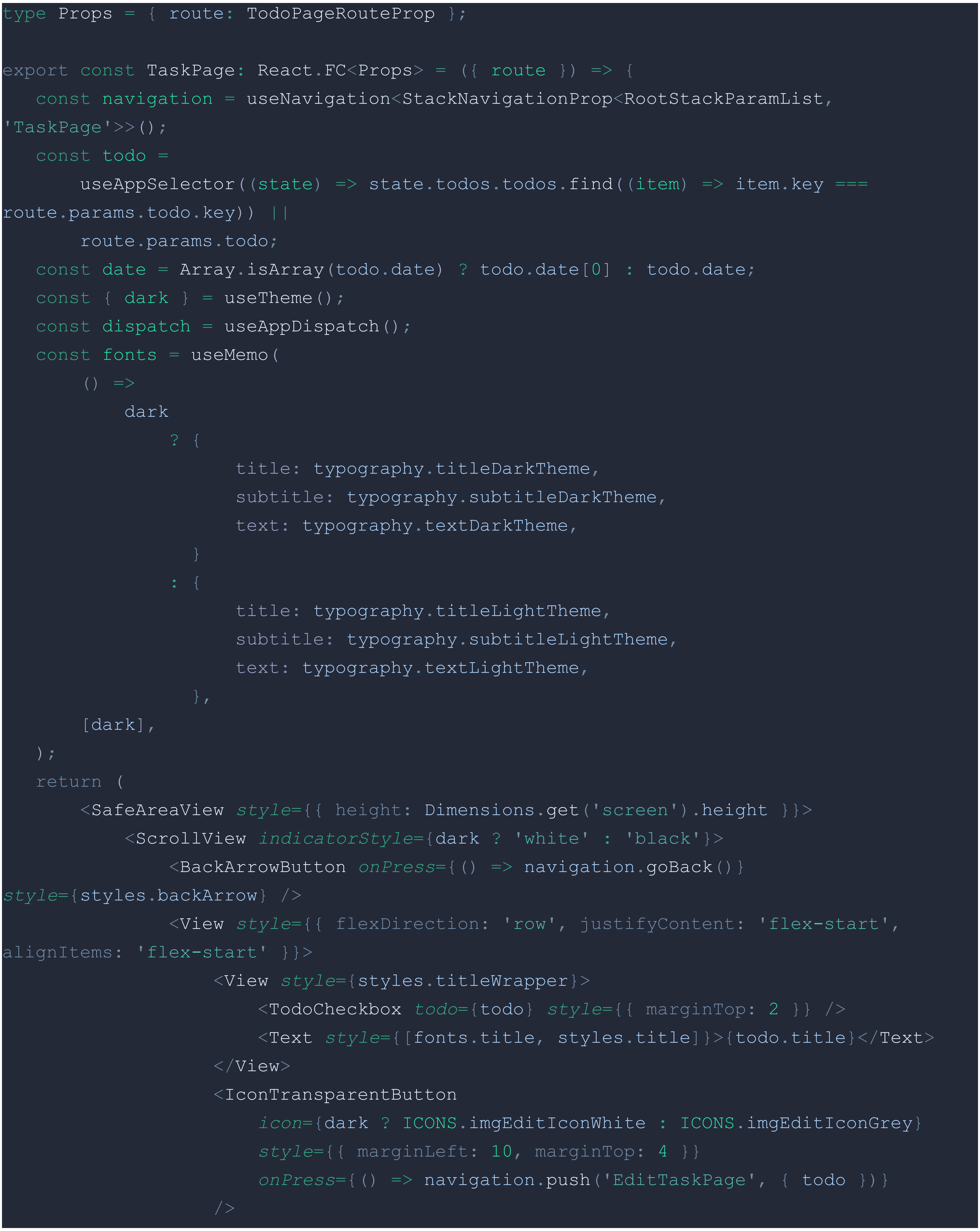
```
import React, { useRef, useState } from 'react';
import { Text, TextInput, View, SafeAreaView, Pressable } from 'react-native';
import { StackNavigationProp } from '@react-navigation/stack';
import { useNavigation, useTheme } from '@react-navigation/native';
import { RootStackParamList } from '../../RootStackParams';
```



```
import { ActionButton } from '../../atomic/ActionButton/ActionButton';
import { styles } from './styles';
import { useAppDispatch, useAppSelector } from '../../../store/hooks';
import { changeUsername } from '../../../store/appSlice';
import { Themes } from '../../../store/types';
import { ThemeButton } from '../../atomic/ThemeButton/ThemeButton';
import { useDarkTheme, useLightTheme, useSystemTheme } from
'../../../hooks/useAppTheme';
import { AnimatedSquaresOne, AnimatedSquaresTwo, AnimatedSquaresThree } from
'../../atomic/AnimatedSquares';
import { typography } from '../../../../assets/globalStyles';

type StartPageScreenProp = StackNavigationProp<RootStackParamList, 'StartPage'>;

export const StartPage: React.FC = () => {
    const inputRef = useRef<TextInput>(null);
    const { colors: themeColors, dark } = useTheme();
    const [inputValue, setInputValue] = useState('');
    const navigation = useNavigation<StartPageScreenProp>();
    const theme = useAppSelector((state) => state.app.theme);
    const dispatch = useAppDispatch();
    const setDarkTheme = useDarkTheme();
    const setLightTheme = useLightTheme();
    const setSystemTheme = useSystemTheme();
    const NavToStartPage = () => {
        dispatch(changeUsername(inputValue));
        // finally 2 should be changed to 1 may be, now it is correct number
        navigation.pop(2);
        navigation.push('MainPage');
    };
    return (
        <SafeAreaView style={styles.canvas} accessible>
            <Pressable onPress={() => (inputRef?.current ? inputRef.current.blur() :
undefined)} style={styles.canvas}>
                <View style={{ position: 'absolute', width: 375, alignSelf: 'center' }}>
                    <AnimatedSquaresOne />
                    <AnimatedSquaresTwo />
                    <AnimatedSquaresThree />
                </View>
                <View>
                    <Text style={[styles.header, dark ? typography.titleDarkTheme :
typography.titleLightTheme]}>
                        Добро пожаловать в DoTo!
                    </Text>
                    <Text style={[styles.text, dark ? typography.textDarkTheme :
typography.textLightTheme]}>
                        Введите имя пользователя
                    </Text>
                    <TextInput
                        ref={inputRef}
                        style={[styles.input, { borderBottomColor: themeColors.text,
color: themeColors.text }]}
                        onChangeText={setInputValue}
                        value={inputValue}
                    />
                    <Text style={[styles.text, { color: themeColors.text }]}>Цветовая
тема</Text>
                    <View style={{ marginLeft: 34 }}>
                        <ThemeButton title="Тёмная" onPress={setDarkTheme}
selected={theme === Themes.DARK} />
                        <ThemeButton title="Светлая" onPress={setLightTheme}
selected={theme === Themes.LIGHT} />
                        <ThemeButton title="Системная" onPress={setSystemTheme}
selected={theme === Themes.SYSTEMIC} />
                    </View>
                </View>
                <ActionButton
                    title="Начать"
                    onPress={NavToStartPage}
                    active={!!inputValue.trim()}
                    style={styles.button}
                />
            </Pressable>
        </SafeAreaView>
    );
};


./src/components/pages/StartPage/styles.ts

import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
    canvas: {
        flex: 1,
        justifyContent: 'space-between',
    },
    header: {
        alignSelf: 'center',
        marginBottom: 29,
        marginTop: 110,
    },
    text: {
        marginLeft: 34,
        marginBottom: 20,
    },
    input: {
        marginLeft: 34,
        width: 250,
        borderStyle: 'solid',
        borderBottomWidth: 1,
        fontSize: 24,
        marginBottom: 44,
    },
    button: {
        bottom: 46,
        width: 94,
        height: 40,
        marginRight: 53,
        alignSelf: 'flex-end',
    },
});


./src/components/pages/TaskPage/style.ts

import { Platform, StyleSheet } from 'react-native';
import { COLORS } from '../../../../assets/colors';

export const styles = StyleSheet.create({
    descriptionWrapper: {
        minHeight: 175,
        width: 262,
        borderBottomWidth: 1,
        borderBottomColor: COLORS.BUTTON_THEME_LIGHT,
        marginLeft: 94,
        paddingBottom: 10,
    },
    backArrow: { marginLeft: 34, marginTop: 22, marginBottom: 38 },
    titleWrapper: {
        flexDirection: 'row',
        justifyContent: 'flex-start',
        alignItems: 'flex-start',
        marginLeft: 32,
        width: 280,
    },
    title: {
        maxWidth: 230,
        marginLeft: 14,
        marginBottom: 30,
    },
    subtitle: { marginLeft: 93, marginBottom: 10, fontSize: 14 },
    date: { marginLeft: 93, marginBottom: 33, fontSize: 14 },
    categoryCard: { marginLeft: 93, marginBottom: 36, marginTop: 5 },
    deleteBtn: { left: 95, width: 100, marginTop: 50, marginBottom: Platform.OS ===
'ios' ? 45 : 90 },
});
```

./src/components/pages/TaskPage/TaskPage.tsx

```
import { RouteProp, useNavigation, useTheme } from '@react-navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import React, { useMemo } from 'react';
import { Text, SafeAreaView, ScrollView, View, Dimensions } from 'react-native';
import { ICONS } from '../../../../assets';
import { typography } from '../../../../assets/globalStyles';
import { useAppDispatch, useAppSelector } from '../../../store/hooks';
import { deleteTodo } from '../../../store/todoSlice';
import { createCustomDateStringFromDate, createCustomTimeStringFromDate } from
'../../../utils/dateConvertations';
import { BackArrowButton } from '../../atomic/BackArrowButton/BackArrowButton';
import { IconActionButton } from '../../atomic/IconActionButton/IconActionButton';
import { IconTransparentButton } from
'../../atomic/IconTransparentButton/IconTransparentButton';
import { SelectCategoryCard } from
'../../atomic/SelectCategoryCard/SelectCategoryCard';
import { TodoCheckbox } from '../../atomic/TodoCheckbox/TodoCheckbox';
import { RootStackParamList } from '../../RootStackParams';
import { styles } from './style';

type TodoPageRouteProp = RouteProp<RootStackParamList, 'TaskPage'>;
```



```
type Props = { route: TodoPageRouteProp };

export const TaskPage: React.FC<Props> = ({ route }) => {
   const navigation = useNavigation<StackNavigationProp<RootStackParamList,
'TaskPage'>>();
   const todo =
       useAppSelector((state) => state.todos.todos.find((item) => item.key ===
route.params.todo.key)) ||
       route.params.todo;
   const date = Array.isArray(todo.date) ? todo.date[0] : todo.date;
   const { dark } = useTheme();
   const dispatch = useAppDispatch();
   const fonts = useMemo(
       () =>
           dark
               ? {
                     title: typography.titleDarkTheme,
                     subtitle: typography.subtitleDarkTheme,
                     text: typography.textDarkTheme,
                 }
               : {
                     title: typography.titleLightTheme,
                     subtitle: typography.subtitleLightTheme,
                     text: typography.textLightTheme,
                 },
       [dark],
   );
   return (
       <SafeAreaView style={{ height: Dimensions.get('screen').height }}>
           <ScrollView indicatorStyle={dark ? 'white' : 'black'}>
               <BackArrowButton onPress={() => navigation.goBack()}
style={styles.backArrow} />
               <View style={{ flexDirection: 'row', justifyContent: 'flex-start',
alignItems: 'flex-start' }}>
                   <View style={styles.titleWrapper}>
                       <TodoCheckbox todo={todo} style={{ marginTop: 2 }} />
                       <Text style={[fonts.title, styles.title]}>{todo.title}</Text>
                   </View>
                   <IconTransparentButton
                       icon={dark ? ICONS.imgEditIconWhite : ICONS.imgEditIconGrey}
                       style={{ marginLeft: 10, marginTop: 4 }}
                       onPress={() => navigation.push('EditTaskPage', { todo })}
                   />
```

```
                </View>
                <Text style={[fonts.subtitle, styles.subtitle]}>Дата</Text>
                <Text style={[fonts.text, styles.date]}>
                    {date?.toDateString() !== new Date(Date.now()).toDateString() &&
                    date?.toDateString() !== new Date(Date.now() + 1000 * 60 * 60 *
24).toDateString()
                        ? (date && `${createCustomDateStringFromDate(date)}, `) || 'Без
даты'
                        : (date?.toDateString() === new Date(Date.now()).toDateString()
&& 'Сегодня, ') || 'Завтра, '}
                    {Array.isArray(todo.date)
                        ? `${createCustomTimeStringFromDate(todo.date[0])} -
${createCustomTimeStringFromDate(
                              todo.date[1],
                          )}`
                        : todo.date && createCustomTimeStringFromDate(todo.date)}
                </Text>
                <Text style={[fonts.subtitle, styles.subtitle]}>Категория</Text>
                <SelectCategoryCard style={styles.categoryCard} category={todo.category}
/>
                <Text style={[fonts.subtitle, styles.subtitle]}>Описание</Text>
                <View style={styles.descriptionWrapper}>
                    <Text style={[fonts.title, { fontSize: 18 }]}>{todo.description ||
'Нет описания'}</Text>
                </View>
                <IconActionButton
                    icon={ICONS.imgTrashIconWhite}
                    style={styles.deleteBtn}
                    iconSize={26}
                    onPress={() => {
                        navigation.goBack();
                        dispatch(deleteTodo(todo));
                    }}
                />
            </ScrollView>
        </SafeAreaView>
    );
};


./src/hooks/useAppTheme.ts

import { useCallback } from 'react';
import { changeTheme } from '../store/appSlice';
import { Themes } from '../store/types';
import { useAppDispatch, useAppSelector } from '../store/hooks';

const useAppTheme = (themeType: Themes) => {
   const theme = useAppSelector((state) => state.app.theme);
   const dispatch = useAppDispatch();
   return useCallback(() => {
       dispatch(changeTheme(themeType));
   }, [theme]);
};

export const useDarkTheme = useAppTheme.bind({}, Themes.DARK);
export const useLightTheme = useAppTheme.bind({}, Themes.LIGHT);
export const useSystemTheme = useAppTheme.bind({}, Themes.SYSTEMIC);
```

./src/hooks/useFilteredTodos.ts

```
import { useAppSelector } from '../store/hooks';

export const useTodayTodos = () =>
   useAppSelector((state) =>
       state.todos.todos.filter((todo) => {
           if (!todo.date) return false;
           return Array.isArray(todo.date)
               ? todo.date[0].toLocaleDateString() === new
Date(Date.now()).toLocaleDateString()
               : todo.date.toLocaleDateString() === new
Date(Date.now()).toLocaleDateString();
       }),
   );
```

./src/store/appSlice.ts

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { IAppState, Themes } from './types';

const initialState: IAppState = {
   username: '',
   theme: Themes.SYSTEMIC,
};
```

```
export const appSlice = createSlice({
    name: 'app',
    initialState,
    reducers: {
        changeTheme: (state, action: PayloadAction<Themes>) => {
            state.theme = action.payload;
        },
        changeUsername: (state, action: PayloadAction<string>) => {
            state.username = action.payload;
        },
    },
});

export const { changeTheme, changeUsername } = appSlice.actions;
export default appSlice.reducer;


./src/store/categorySlice.ts

import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { COLORS } from '../../assets/colors';
import { ICategory } from './types';

interface ICategoriesState {
    categories: ICategory[];
}

const initialState: ICategoriesState = {
    categories: [
        {
            key: '0',
            name: 'Другое',
            color: COLORS.VIOLET,
        },
    ],
};

export const categorySlice = createSlice({
    name: 'category',
    initialState,
    reducers: {
        addCategory: (state, action: PayloadAction<ICategory>) => {
            state.categories = state.categories.concat(action.payload);
        },
        deleteCategory: (state, action: PayloadAction<ICategory>) => {
            state.categories = state.categories.filter((item) => item.key !==
action.payload.key);
        },
        editCategory: (state, action: PayloadAction<ICategory>) => {
            state.categories = state.categories.map((item) =>
                item.key === action.payload.key ? action.payload : item,
            );
        },
    },
});

export const { addCategory, deleteCategory, editCategory } = categorySlice.actions;
export default categorySlice.reducer;


./src/store/hooks.ts

import { TypedUseSelectorHook, useDispatch, useSelector } from 'react-redux';
import type { RootState, AppDispatch } from './store';

export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;


./src/store/store.ts

import { configureStore } from '@reduxjs/toolkit';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { persistStore, persistReducer, createTransform } from 'redux-persist';
import appReducer from './appSlice';
import categoryReducer from './categorySlice';
import todosReducer, { ITodosState } from './todoSlice';
import { ITodo } from './types';

const replacer = (key: string, value: any) => (value instanceof Date ?
value.toISOString() : value);
const reviver = (key: string, value: any) =>
    typeof value === 'string' && value.match(/^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}/) ?
new Date(value) : value;
const encode = (toDeshydrate: any) => JSON.stringify(toDeshydrate, replacer);
const decode = (toRehydrate: string) => JSON.parse(toRehydrate, reviver);

const persistAppConfig = {
   key: 'app',
   storage: AsyncStorage,
};
const persistTodosConfig = {
   key: 'todos',
   storage: AsyncStorage,
   transforms: [createTransform<ITodo, string>(encode, decode)],
};

const persistCategoriesConfig = {
   key: 'categories',
   storage: AsyncStorage,
};

const persistedAppReducer = persistReducer(persistAppConfig, appReducer);
const persistedCategoryReducer = persistReducer(persistCategoriesConfig,
categoryReducer);
const persistedTodosReducer = persistReducer<ITodosState>(persistTodosConfig,
todosReducer);

export const store = configureStore({
   reducer: {
       app: persistedAppReducer,
       categories: persistedCategoryReducer,
       todos: persistedTodosReducer,
   },
   middleware: (getDefaultMiddleware) =>
       getDefaultMiddleware({
           serializableCheck: false,
       }),
});

export const persistor = persistStore(store);

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;


./src/store/todoSlice.ts
```

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { COLORS } from '../../assets/colors';
import { DateTypes, ITodo } from './types';

export interface ITodosState {
    todos: ITodo[];
}

const initialState: ITodosState = {
    todos: [
        {
            key: '0',
            title: 'Создать список задач',
            description: 'Записать свои задачи на ближайшее время в приложение',
            category: {
                key: '0',
                name: 'Другое',
                color: COLORS.VIOLET,
            },
            dateType: DateTypes.DEADLINE,
            date: new Date(Date.now()),
            completed: false,
        },
    ],
};

export const todoSlice = createSlice({
    name: 'todo',
    initialState,
    reducers: {
        addTodo: (state, action: PayloadAction<ITodo>) => {
            state.todos = state.todos.concat(action.payload);
        },
        deleteTodo: (state, action: PayloadAction<ITodo>) => {
            state.todos = state.todos.filter((item) => item.key !== action.payload.key);
        },
        toggleTodo: (state, action: PayloadAction<ITodo>) => {
            const todoIndex = state.todos.findIndex((item) => item.key ===
action.payload.key);
            state.todos[todoIndex].completed = !state.todos[todoIndex].completed;
        },
        editTodo: (state, action: PayloadAction<ITodo>) => {
            const todoIndex = state.todos.findIndex((item) => item.key ===
action.payload.key);
            state.todos[todoIndex] = action.payload;
        },
    },
});

export const { addTodo, deleteTodo, editTodo, toggleTodo } = todoSlice.actions;
export default todoSlice.reducer;


./src/store/types.ts

export enum DateTypes {
    WITHOUT = 'WITHOUT',
    DEADLINE = 'DEADLINE',
    INTERVAL = 'INTERVAL',
}
export enum Themes {
    LIGHT = 'LIGHT',
    DARK = 'DARK',
    SYSTEMIC = 'SYSTEMIC',
}
export interface ICategory {
    key: string;
    name: string;
    color: string;
}
export interface ITodo {
    key: string;
    title: string;
    dateType: DateTypes;
    date?: Date | [start: Date, end: Date];
    description: string;
    completed: boolean;
    category: ICategory;
}
export interface IAppState {
    username: string;
    theme: Themes;
}



./src/utils/chunc.ts
```

```
export function chunk<T>(arr: Array<T>, size: number): Array<Array<T>> {
    const result = [];

    for (let i = 0; i < Math.ceil(arr.length / size); i += 1) {
        result.push(arr.slice(i * size, i * size + size));
    }

    return result;
}

./src/utils/convertHexToRGBA.ts

export const convertHexToRGBA = (hexCode: string, opacity: number) => {
    let hex = hexCode.replace('#', '');
    if (hex.length === 3) {
        hex = `${hex[0]}${hex[0]}${hex[1]}${hex[1]}${hex[2]}${hex[2]}`;
    }
    const r = parseInt(hex.substring(0, 2), 16);
    const g = parseInt(hex.substring(2, 4), 16);
    const b = parseInt(hex.substring(4, 6), 16);
    return `rgba(${r},${g},${b},${opacity / 100})`;
};


./src/utils/dateConvertations.ts

export const createCustomDateStringFromDate = (date: Date) => {
    const day = date.getDate() < 10 ? '0'.concat(date.getDate().toString()) :
date.getDate().toString();
    const month = date.getMonth() < 9 ? '0'.concat((date.getMonth() + 1).toString()) :
date.getMonth().toString();
    const year = date.getFullYear().toString();
    return `${day}.${month}.${year}`;
};

export const createDateFromCustomDateString = (dateString: string) => {
    const parts = dateString.split('.');
    const res = new Date();
    res.setDate(parseInt(parts[0], 10));
    res.setMonth(parseInt(parts[1], 10) - 1);
    res.setFullYear(parseInt(parts[2], 10));
    return res;
};

export const createCustomTimeStringFromDate = (date: Date) => {
   const hours = date.getHours() < 10 ? '0'.concat(date.getHours().toString()) :
date.getHours().toString();
   const minutes = date.getMinutes() < 10 ? '0'.concat(date.getMinutes().toString()) :
date.getMinutes().toString();
   return `${hours}:${minutes}`;
};

./src/utils/dateHalfHourRound.ts

export const dateHalfHourFloor = (date: Date) =>
   date.getTime() % 1800000 === 0 ? date : new Date(date.getTime() - (date.getTime() %
1800000));

export const dateHalfHourCeil = (date: Date) =>
   date.getTime() % 1800000 === 0 ? date : new Date(date.getTime() + 1800000 -
(date.getTime() % 1800000));

./src/utils/getRUMonthFromDate.ts

export const getRUMonthFromDate = (date: Date) => {
   const months = [
       'января',
       'февраля',
       'марта',
       'апреля',
       'мая',
       'июня',
       'июля',
       'августа',
       'сентября',
       'октября',
       'ноября',
       'декабря',
   ];
   return months[date.getMonth()];
};



./src/utils/isCloseToBottom.ts
```

```
interface IsCloseToBottomProps {
   layoutMeasurement: { height: number; width: number };
   contentOffset: { x: number; y: number };
   contentSize: { height: number; width: number };
}
export const isCloseToBottom = ({ layoutMeasurement, contentOffset, contentSize }:
IsCloseToBottomProps) =>
   layoutMeasurement.height + contentOffset.y >= contentSize.height - 30;


./src/utils/todosSortByDate.ts

import { ITodo } from '../store/types';

export const todosSortByDateAscending = (todos: ITodo[]) =>
   [...todos].sort((a, b) => {
       if (!a.date) return 1;
       if (!b.date) return -1;
       const startA = Array.isArray(a.date) ? a.date[0] : a.date;
       const startB = Array.isArray(b.date) ? b.date[0] : b.date;
       return startA.getTime() - startB?.getTime();
   });
```