모바일앱개발

2023년 2학기 최종과제보고서

과제 메이트 구성표; 대표 학생난에 *를 기입

pdf파일로 변환하여 제출

보고서의 내용은 본 양식의 2번째 페이지부터 작성

대표 학생	학 번	성 명	과제 역할	기여도
*	201804248	정진우	발표 및 디자인	100 / 100
	201804118	최상욱	개발	100 / 100
	201804065	윤성훈	개발	100 / 100

과제 제출 마감일: 2023년 12월 14일 목요일 오후 10시까지

GitHub에 업로드한 URL: https://github.com/Choi-Sangwook/ReactNative_TodoList

youtube에 업로드한 URL: https://youtube.com/watch?v=Ry5PJEbc3t0&si=ISUpBWccvGN1mSJB

표절율: 12%

가. 과제는 대표 학생이 팀별 1개만 제출.

나. 과제물 필수 내용:

1. 코딩한 소스코드는 250줄 이상.
   단, 주석 및 불 필요한 개행은 제외
2. 보고서에 참고한 사이트, 문헌, 참고한 코드의 출처를 표기.
3. 보고서에 독자적으로 코딩한 부분과 인용한 코드를 명시.
4. **보고서의 첫 페이지에 팀별 팀원의 역할 분담및 기여도를 명시.**

#1. 필수사항: 팀별 과제의 프로그램를 GitHub에 업로드하기 바랍니다.
그리고, GitHub에 업로드한 URL를 보고서의 첫페이지에 기입 바랍니다.
#2. 필수사항: 팀별 과제의 데모 비디오를 youtube에 업로드하기 바랍니다.
그리고, youtube에 업로드한 URL를 보고서의 첫페이지에 기입 바랍니다.
#3. 제출하는 보고서의 표절율을 copykiller로 검사후 pdf파일로 함께 첨부 하여 제출
#4. 표절율이 40%이상인 보고서의 최종 과제 점수는 0점 처리
#5. 최종 과제 보고서에 기여도가 기입 되지 않은 학생의 경우, 최종 과제 점수는 0점 처리
#6. 최종 과제 보고서의 마감일은 12월 14일 오후 10시까지로 한다.

다. 과제 제출 마감일까지 보고서를 제출하지 않은 팀의 최종 과제의 성적은 0점 처리함

보고서 내용

#작품 제목 : MemoList(메모리스트)

#작품 개요

수업 시간에 제작한 기존 TODO List에 메모장 및 캘린더 기능을 추가하여
사용자가 일과를 효율적으로 관리할 수 있도록 업그레이드한 TODO List 앱.

#주요 기능:

1. 오늘의 일과 확인 및 수행 여부 관리
2. 캘린더를 통한 일과 조회 및 관리
3. 메모 기능. - 추가, 수정, 삭제, 검색
4. 설정 탭 테마 변경 기능.

#디자인

Figma 사용하여 디자인 후 실제 코드는 FIgma 내 CSS를 확인하여 직접 코딩.

<Figma 결과>

#기능 설명 및 코드 리뷰

- 새롭게 학습한 라이브러리 같은 부분에 대해서는 참고 문헌에 적힌 공식 사이트에서 구조 및 사용법을 학습하였으며 우리의 프로젝트에 맞게 직접 코딩을 하여 사용함.

네비게이션

네비게이션은 탭 네비게이션과 스택 네비게이션 2가지를 사용하였으며 화면의 구성은 아래의 구조와 같다.

```
<Tab>
    Home
    <Stack>
        Memo
        MemoForm
    </Stack>
    <Stack>
        Calendar
        TaskForm
    </Stack>
    Setting
</Tab>
```

홈, 메모, 캘린더, 세팅 화면이 탭 네비게이션으로 구성되어있으며, 메모와 캘린더 화면에서는 각각 메모, 일과 추가 페이지로 이동해야하기 때문에 스택 네비게이션을 사용함.

<실제 코드> - App.js

```
const MemoStackScreen = () => (
  <MemoStack.Navigator
    screenOptions={{
      headerShown: false,
    }}>
    <MemoStack.Screen name="Memo" component={MemoScreen} />
    <MemoStack.Screen name="AddMemoForm" component={MemoFormScreen} />
  </MemoStack.Navigator>
);
const CalendarStackScreen = () => (
  <CalendarStack.Navigator
  screenOptions={{
    headerShown: false,
  }}>
    <CalendarStack.Screen name="Calendar" component={CalendarScreen} />
    <CalendarStack.Screen  name="AddTaskForm"  component={CalendarFormScreen}
/>
```

```
  </CalendarStack.Navigator>
);

const App = () => {
  const { darkMode} = useTasksContext();
  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
    <NavigationContainer >
          <Tab.Navigator initialRouteName="Home">
            <Tab.Screen
              name="Home"
              component={HomeScreen}
              options={{
                title: 'Home',
                tabBarIcon: ({color, size}) => (
                  <Icon name="home" color={color} size={size} />
                ),
                headerShown: false,
              }}
            />
            <Tab.Screen
              name="Note"
              component={MemoStackScreen}
              options={{
                title: 'Note',
                tabBarIcon: ({color, size}) => (
                  <Icon name="description" color={color} size={size} />
                ),
                headerShown: false,
              }}
            />
            <Tab.Screen
              name="Task"
              component={CalendarStackScreen}
              options={{
                title: 'Calendar',
                tabBarIcon: ({color, size}) => (
                  <Icon name="event" color={color} size={size} />
                ),
                headerShown: false,
              }}
```

```
            />
            <Tab.Screen
              name="Setting"
              component={SettingScreen}
              options={{
                title: 'Settings',
                tabBarIcon: ({color, size}) => (
                  <Icon name="settings" color={color} size={size} />
                ),
                headerShown: false,
              }}
            />
          </Tab.Navigator>
    </NavigationContainer>
    </ThemeProvider>
  );
};
```

<코드 설명>

- const MemoStackScreen = () : 메모 화면을 스택 네비게이션 화면으로 구성.
  메모 조회 화면, 메모 등록 화면 총 2가지로 스택 구성. 캘린더도 동일함.
- Screen 태그 내의 name 속성은 이후 네비게이션 사용 시 화면의 이름.
- component 속성은 연결할 Screen 컴포넌트를 의미.
- 아래 참고문헌을 통해 구조 설계. 이후 속성이나 옵션은 직접 코딩으로 구현.

<참고 문헌>

네비게이션 공식 문서: https://reactnavigation.org/docs/nesting-navigators

UseContext를 사용하여 일과 목록 전역변수로 설정

일과 목록을 MainHome, Calendar, CalendarForm 등 여러 컴포넌트에서 사용하고, 그때마다 Async Storage에 접근하여 데이터를 가져와야 하는 상황 발생.
처음에는 이를 해결하기 위해 메인 화면에서 한 번만 가져오고 이후 컴포넌트에 props를 통해 넘겨주었으나 코드가 난잡해지고 실수가 잦아짐.
그래서 UseContext를 사용하여 일과 목록을 전역 변수로 만들고 관리하기로 결정.

<실제코드> - TaskContext.js

```
const TasksContext = createContext();

export const useTasksContext = () => useContext(TasksContext);

export const TasksProvider = ({ children }) => {
    const [tasks, setTasks] = useState({});
    const [darkMode, setDarkMode] = useState(false);

    const updateTasks = (newTasks) => {
        setTasks(newTasks);
        if (newTasks !== undefined) {
            AsyncStorage.setItem('tasks', JSON.stringify(newTasks))
              .catch(error => console.error('Error:', error));
        }
    };

    const updateDarkMode = (newDarkMode) => {
      setDarkMode(newDarkMode);
      AsyncStorage.setItem('darkMode', JSON.stringify(newDarkMode))
        .catch(error => console.error('Error:', error));
  };

    const _loadTasks = async () => {
        const loadedTasks = await AsyncStorage.getItem('tasks');
        setTasks(JSON.parse(loadedTasks || '{}'));
        const loadedDarkMode = await AsyncStorage.getItem('darkMode');
        setDarkMode(loadedDarkMode === 'true');
    };
    useEffect(() => {
        _loadTasks();
    }, []);
```

```
    return (
      <TasksContext.Provider value={{ tasks, updateTasks, darkMode, updateDarkMo
de }}>
        {children}
      </TasksContext.Provider>
    );
};
```

<코드 설명>

- const TasksContext = createContext() : TasksContext를 생성하는 코드.
- export const useTasksContext = () => useContext(TasksContext) : TasksContext를 사용하기 위한 커스텀 훅으로 다른 컴포넌트에서 해당 훅을 이용해 일과 목록을 가져옴.
- export const TasksProvider = ({ children }) => : 컨텍스트 내부에 정의된 값을 해당 Provider로 제공 할 수 있음.
- const [tasks, setTasks] = useState({}) : useState를 통해 일과 목록을 저장하기 위한 변수 초기화.
- const updateTasks = (newTasks) => : 일과 목록에 변동이 생겼을 경우 해당 함수를 통해 Async Storage 안에 데이터에 업데이트 시켜준다.
- const _loadTasks = async () => : 일과 목록을 AsyncStorage에서 가져와 위에서 선언한 Tasks 변수에 넣어준다.
- useEffect(() => : useEffect를 이용해 컴포넌트가 처음 렌더링 될 때 Async Storage에서 일과 데이터를 불러옴.
- const [darkMode, setDarkMode] = useState(false) : darkMode의 여부를 나타내는 상태변수. 초기값 false로 초기화.
- 아래 문서를 통해 사용법 및 구조 학습 후 직접 코딩으로 구현.

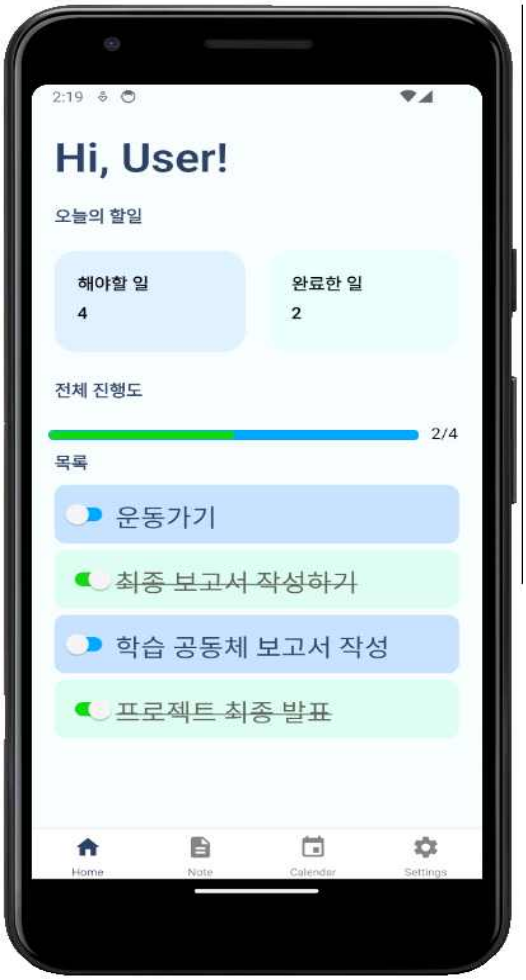
<참고 문헌>

UseContext 공식 문서 : https://ko.legacy.reactjs.org/docs/context.html

MainHome - 오늘의 일과 조회, 관리 및 진행도 확인

메인 홈 화면에서는 오늘의 일과를 확인할 수 있고 일과 수행 여부를 토글 버튼을 이용하여 사용자가 관리 가능하다. 토글 버튼을 누르면 일과의 Completed가 false에서 true로 바뀌고 정보를 나타내는 Box.js 컴포넌트와 ProgressBar.js 컴포넌트가 반응하여 변화된 카운트 값을 사용자에게 보여준다. Progress Bar는 라이브러리를 이용하여 제작했다.

<화면>



<실제 코드> - MainHome.js

```
export default function App({navigation}) {
  const width = Dimensions.get('window').width;
  const [toDayTasks, setTodayTasks] = useState({});
  const { tasks, updateTasks } = useTasksContext();
  const { darkMode } = useTasksContext();

  const _toggleTask = id => {
    const currentTasks = Object.assign({}, tasks);;
```

```
    currentTasks[id]['completed'] = !currentTasks[id]['completed'];
    updateTasks(currentTasks);
  };

  useEffect(() => {
    const today = new Date();
    const formattedToday = `${today.getFullYear()}-${String(today.getMonth() + 1).pa
dStart(2, '0')}-${String(today.getDate()).padStart(2, '0')}`;
    const todayTaskList = Object.values(tasks).filter((task) => task.date === formatted
Today);
    setTodayTasks(todayTaskList.reduce((acc, task) => {
      acc[task.id] = task;
      return acc;
    }, {}));
  }, [tasks]);

  const tasksValue = Object.values(toDayTasks);
  const length = tasksValue.length || 0;
  const completed = tasksValue.filter((task) => task.completed === true).length || 0;

  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
      <Container>
        <StatusBar
          barStyle={darkMode ? "light-content":"dark-content"}
          backgroundColor={darkMode ? darkTheme.background:lightTheme.backgro
und}
        />
        <Title>Hi, User!</Title>
        <SubTitle>오늘의 할일</SubTitle>
        <BoxConatiner width={width}>
            <Box title = "해야할 일" count = {length} width={width}/>
            <Box title = "완료한 일" count = {completed} width={width}/>
        </BoxConatiner>

        <SubTitle>전체 진행도</SubTitle>
        <PrograssBar completed={completed} length={length}/>
        <SubTitle>목록</SubTitle>
        {Object.values(toDayTasks).length === 0 ? (
          <Container>
            <SubTitle>오늘의 일과를 추가하세요.</SubTitle>
```

```
            </Container>
          ) : (
          <List width={width}>
            {Object.values(toDayTasks)
              .reverse()
              .map(item => (
                <Task
                  key={item.id}
                  item={item}
                  toggleTask={_toggleTask}
                />
              ))}
          </List>
          )}
        </Container>
      </ThemeProvider>
    );
}
```

<코드 설명>
- UseContext를 사용하여 일과 목록을 가져와 Tasks에 저장.
- tasks에 변동이 생길때마다 useEffect가 실행되고 오늘 날짜로 일과들의 Date를 필터링하여 오늘의 날짜에 해당하는 일과만 toDayTasks 변수에 저장.
- toggle 버튼 클릭시 updateToggle 함수가 실행되고 해당 함수는 일과 데이터의 Completed 값을 반전시킴.
- 오늘의 일과를 Tasks 컴포넌트를 사용해 리스트로 렌더링.

<참고 문헌>
없음.

<사용컴포넌트>
1. Box.js
2. ProgressBar.js

Memo
사용자가 간단한 메모를 기록할 수 있는 공간.

<화면>

<실제 코드>

```
export default function App({ navigation }) {
  const width = Dimensions.get('window').width;

  const [isReady, setIsReady] = useState(false);

  const [memos, setMemos] = useState({});
  const [searchKeyword, setSearchKeyword] = useState('');

  const { darkMode, updateDarkMode} = useTasksContext();
```

```
const _saveMemos = async (memoData) => {
  try {
    await AsyncStorage.setItem('memos', JSON.stringify(memoData));
    setMemos(memoData);
  } catch (error) {
    console.error(error);
  }
};

const _loadMemos = async () => {
  try {
    const loadedMemos = await AsyncStorage.getItem('memos');
    setMemos(JSON.parse(loadedMemos || '{}'));
  } catch (error) {
    console.error(error);
  }
};

useEffect(() => {
  const unsubscribe = navigation.addListener('focus', () => {
    _loadMemos();
  });

  return unsubscribe;
}, [navigation]);

const _deleteMemo = (id) => {
  const currentMemos = { ...memos };
  delete currentMemos[id];
  _saveMemos(currentMemos);
};

const _updateMemo = (id) => {
  navigation.navigate('AddMemoForm', { id: id });
};

const filteredMemos = Object.values(memos).filter(
  (memo) =>
    memo.contents.includes(searchKeyword) || memo.title.includes(searchKeyword)
);
```

```
  return isReady ? (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
      <Container>
      <StatusBar
          barStyle={darkMode ? "light-content":"dark-content"}
          backgroundColor={darkMode ? darkTheme.background:lightTheme.backgro
und}
        />
        <BoxConatiner width={width}>
          <Title>Note</Title>
          <IconButton
            type={images.update}
            onPressOut={() => navigation.navigate('AddMemoForm')}/>
        </BoxConatiner>
        <TextInput
          placeholder="메모를 검색하세요"
          value={searchKeyword}
          onChangeText={(text) => setSearchKeyword(text)}
          placeholderTextColor={darkMode ? darkTheme.main : lightTheme.main}
          style={{
            color: darkMode ? darkTheme.main : lightTheme.main,
            width: width - 40,
            borderColor: darkMode ? darkTheme.main : lightTheme.main,
            borderWidth: 1,
            borderRadius: 5,
            padding: 8,
            marginTop: 10,
            marginBottom: 10,
          }}
        />
        {Object.values(memos).length === 0 ? (
          <Container>
            <SubTitle>메모를 등록해보세요!</SubTitle>
          </Container>
        ) : (
          <List width={width}>
            {Object.values(filteredMemos)
              .reverse()
              .map(item => (
                <MemoTask
                  key={item.id}
```

```
                    item={item}
                    deleteMemo={_deleteMemo}
                    updateMemo={_updateMemo}
                  />
                ))}
            </List>
          )}
        </Container>
      </ThemeProvider>
    ) : (
      <AppLoading
        startAsync={_loadMemos}
        onFinish={() => setIsReady(true)}
        onError={console.error}
      />
    );
}
```

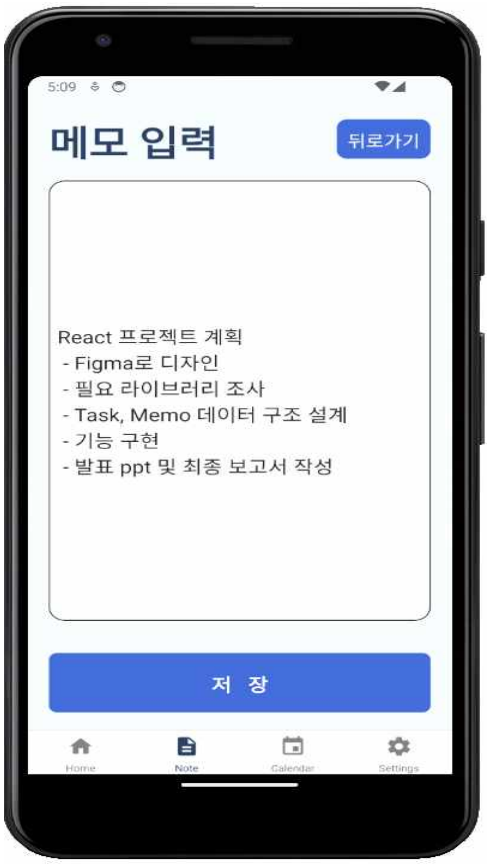
<코드 설명>
- 화면에 접근 시 useEffect를 통해 AsyncStorage에서 메모를 가져오도록 설정.
- 가져온 memo의 List는 memoTask 컴포넌트에 넣어 렌더링
- 수정이나 삭제 동작은 해당 Memo의 Id값을 이용하여 함수의 인자로 넘겨주고 동작하도록 설정.
- 만약 수정버튼 클릭시 MemoForm 화면으로 Id값을 함께 넘겨주고 form화면에서 해당 id 값을 이용하여 동작 수행.

<참고 문헌>
없음

<사용 컴포넌트>
1. MemoTask.js
2. IconButton.js

MemoForm
사용자가 메모를 등록하거나 수정할 수 있는 화면

<화면>



<실제 코드>

```
const MemoForm = ({ navigation, id }) => {
  const [memoText, setMemoText] = useState('');
  useEffect(() => {
    if (id) {
      const fetchMemoContent = async () => {
        try {
          const existingMemos = await AsyncStorage.getItem('memos');
          const memos = existingMemos ? JSON.parse(existingMemos) : {};
          if (memos[id]) {
            setMemoText(memos[id].contents);
          }
        } catch (error) {
          console.error('메모 내용 가져오기 오류:', error);
        }
```

```
      };
      fetchMemoContent();
    }
  }, [id]);

  const _updateMemo = async () => {
    const title = memoText.substring(0,7);
    const existingMemos = await AsyncStorage.getItem('memos');
    const memos = existingMemos ? JSON.parse(existingMemos) : {};
    await _saveMemo({ ...memos, [id]: { id: id, title:title ,contents: memoText } });
  };

  const _addMemo = async () => {
    const ID = Date.now().toString();
    const title = memoText.substring(0, 7);

    const newMemoObject = {
      [ID]: {id: ID, title: title, contents: memoText},
    };
    try {
      const existingMemos = await AsyncStorage.getItem('memos');
      const value = existingMemos ? JSON.parse(existingMemos) : {};
      setMemoText('');
      await _saveMemo({ ...value, ...newMemoObject });
    } catch (error) {
      console.error('메모 저장 오류:', error);
    }
  };

  const _saveMemo = async (memoData) => {
    try {
      console.log('메모를 저장하는 중...');
      await AsyncStorage.setItem('memos', JSON.stringify(memoData));
      navigation.navigate('Memo');
      console.log('메모 저장 완료:', memoData);
    } catch (error) {
      console.error('메모 저장 오류:', error);
    }
  };
  const handleMemoSubmit = () => {
    if (memoText && memoText.trim() !== '') {
```

```
        if(id === undefined){
        _addMemo();
        console.log('최초 저장된 메모:', memoText);
        }else {
          _updateMemo();
          console.log('수정된 메모:', memoText);
        }
      } else {
      }
    };

    return (
      <Container>
      <View style={{ width: Dimensions.get('window').width * 0.9,flexDirection: 'row', ju
stifyContent: 'space-between', alignItems: 'center' }}>
        <Title>메모 입력</Title>
        <BackButton onPress={() => navigation.goBack()}>
            <BackButtonText>뒤로가기</BackButtonText>
          </BackButton>
      </View>
        <Input
          placeholder="메모를 작성하세요"
          multiline
          value={memoText}
          onChangeText={(text) => setMemoText(text)}
        />
        <Button onPress={handleMemoSubmit}>
          <ButtonText>저    장</ButtonText>
        </Button>
      </Container>
    );
};

const MemoFormPage = ({ navigation, route}) => {
    const { id } = route.params || {id: undefined};
    return (
      <ThemeProvider theme={theme}>
        <StatusBar barStyle="dark-content" />
        <MemoForm navigation={navigation} id={id}/>
      </ThemeProvider>
    );
```

```
};

export default MemoFormPage;
```

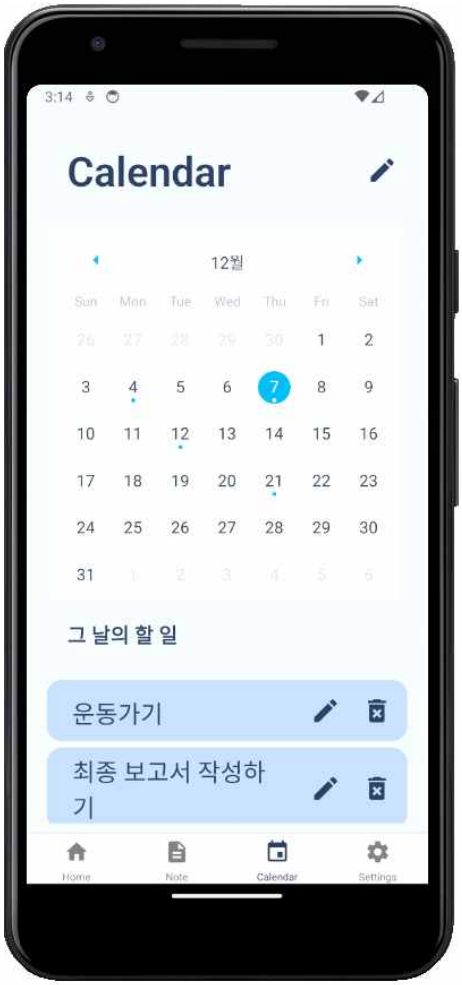
<코드 설명>
- 해당 화면으로 접근 시 메모 수정 동작인지 등록 동작인지 확인
- 만약 id값이 존재하면 수정, 존재하지 않으면 등록.
- 수정 화면이라면, 해당 하는 Id값 메모의 contents를 불러와 Text Value로 설정.
  이후에 UpdateMemo 함수를 실행하여 기존 Memo 리스트에 덮어쓰기.
- 등록 화면이라면, TextValue는 비어있는 문자열로 초기화.
  이후에 AddMemo 함수를 실행하여 기존 Memo 리스트에 새롭게 추가.

<참고 문헌>
없음

<사용 컴포넌트>
없음

#Calendar
Calendar라이브러리를 통해 화면에 나온 달력을 통해 날자를 선택하면 해당 날짜의 할 일 목록을 보여주고 새로만들기 버튼을 누르면 입력폼으로 넘어가며 목록에 있는 할 일을 수정 버튼과 삭제버튼으로 관리하는 화면

<화면>



<실제 코드>

```
import React, { useState, useEffect } from 'react';
import { StatusBar, Dimensions } from 'react-native';
import styled, { ThemeProvider } from 'styled-components/native';
import IconButton from '../components/IconButton';
import {images} from '../images';
import { Calendar } from "react-native-calendars";
import CalendarTask from '../components/CalendarTask';
import { useTasksContext } from '../TaskContext';
import {lightTheme, darkTheme} from '../theme'
```

```
const Container = styled.SafeAreaView`
  flex: 1;
  background-color: ${({ theme }) => theme.background};
  align-items: center;
  justify-content: flex-start;
`;
const Title = styled.Text`
  font-size: 40px;
  font-weight: 600;
  width: ${({ width }) => width - 150}px;
  color: ${({ theme }) => theme.main};
  align-self: flex-start;
  align-self: center;
  margin: 20px;
`;
const List = styled.ScrollView`
  margin:10px 0;
  flex: 1;
  width: ${({ width }) => width - 40}px;
`;

const SubTitle = styled.Text`
    margin: 20px;
    padding: 0 20px;
    color: ${({ theme }) => theme.main};
    font-size: 20px;
    font-weight: 700;
    align-self: flex-start;
`;

const BoxConatiner = styled.SafeAreaView`
  height:120px;
  width: ${({ width }) => width - 40}px;
  background-color: ${({ theme }) => theme.background};
  flex-direction: row;
  justify-content: space-between;
  align-items: center;
`;

const CalendarContainer = styled.View`
```

```
  width: ${({ width }) => (width - 60)}px;
`;



export default function App({ navigation }) {
  const width = Dimensions.get('window').width;
  const today = new Date();
  const year = today.getFullYear();
  const month = String(today.getMonth() + 1).padStart(2, '0');
  const day = String(today.getDate()).padStart(2, '0');
  const formattedToday = `${year}-${month}-${day}`;
  const [loadTasks, setLoadTasks] =useState({});
  const { tasks, updateTasks,darkMode } = useTasksContext();
  const [selectedDate, setSelectedDate] = useState(formattedToday);
  const [markedDates, setMarkedDates] = useState({});



  const _loadTasks = async () => {
    setLoadTasks(tasks);
    setMarkedDates({});
    const markedDatesObject = {
      ...tasks,
      [selectedDate]: {
        selected: true,
        marked: false,
        dots: [{ color: 'green' }],
      },
    };
    Object.keys(tasks).forEach((taskId) => {
    const task = tasks[taskId];
    markedDatesObject[task.date] = {
      selected: markedDatesObject[task.date]?.selected || false,
      dots: [{ color: 'green' }],
      marked: true,
    };
  });
  await setMarkedDates(markedDatesObject);
```

```
await _selectedDateTasks();
};

const _selectedDateTasks = () => {
  const selectedDateTasks = Object.keys(tasks)
  .filter(key => tasks[key].date === selectedDate)
  .reduce((obj, key) => {
    obj[key] = tasks[key];
    return obj;
  }, {});
setLoadTasks(selectedDateTasks);
};

const _deleteTask = async(id)=> {
  const currentTasks = Object.assign({}, tasks);
  delete currentTasks[id];
  await updateTasks(currentTasks);
};

useEffect(() => {
  _loadTasks();
}, [tasks]);
const _updateTask = item => {
  const currentTasks = Object.assign({}, tasks);
  currentTasks[item.id] = item;
  updateTasks(currentTasks);
};

const handleDateSelect = async(date) => {
  const dateString = date.dateString;
  setSelectedDate(dateString);
  console.log(dateString);
  const updatedMarkedDates = { ...markedDates };
  if (updatedMarkedDates[selectedDate]) {
    updatedMarkedDates[selectedDate] = {
      ...updatedMarkedDates[selectedDate],
      selected: false,
    };
  }

  updatedMarkedDates[dateString] = {
```

```
    selected: true,
    marked: updatedMarkedDates[dateString]?.marked || false,
    dots: updatedMarkedDates[dateString]?.dots || [],
    };
    console.log(updatedMarkedDates[dateString]);
    setMarkedDates(updatedMarkedDates);
    _selectedDateTasks();
  };


  useEffect(() => {
    const unsubscribe = navigation.addListener('focus', async () => {
      setSelectedDate(selectedDate!==null?selectedDate:formattedToday);
      await _loadTasks();
    });

    return unsubscribe;
  }, [navigation, formattedToday,tasks,loadTasks,markedDates]);

  useEffect(() => {
    _selectedDateTasks();
  }, [selectedDate]);

  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
      <Container>
        <StatusBar
          barStyle={darkMode ? "light-content":"dark-content"}
          backgroundColor={darkMode                                                    ?
darkTheme.background:lightTheme.background} // Android only
        />
        <BoxConatiner width={width}>
            <Title>Calendar</Title>
            <IconButton
              type={images.update}
              onPressOut={() => navigation.navigate('AddTaskForm', { selectedDate
})}/>
        </BoxConatiner>
        <CalendarContainer width={width}>
            <Calendar
            theme={{
```

```
            todayTextColor: darkMode ?'white':'black',
          }}
          showSixWeeks={false}
          borderBottomWidth = {width}
          borderBottomColor="#000000"
          onDayPress={(day) => handleDateSelect(day)}
          markedDates={markedDates}
          monthFormat={'M월'}/>
        </CalendarContainer>
        <SubTitle>그 날의 할 일</SubTitle>
        <List width={width}>
          {Object.values(loadTasks)
            .reverse()
            .map(item => (
              <CalendarTask
                key={item.id}
                item={item}
                deleteTask={_deleteTask}
                updateTask={_updateTask}
                onPressOut={() => navigation.navigate('AddTaskForm', { item })}
              />
            ))}
        </List>
      </Container>
    </ThemeProvider>
  );
}
```

<코드 설명>
-화면 진입시 오늘날짜를 가져와 오늘이 선택된 캘린더를 보여주고 일정이 있으면 해당 할 일을 목록에 보여줍니다.
-다른 날짜 선택시 캘린더의 마크가 이동되고 해당 날짜의 일정을 List에 넣어 보여줍니다.
- 일정인 있는 날은 캘린더에서 dot으로 표시가 나옵니다.

<참고 문헌>
https://www.npmjs.com/package/react-native-calendars

<사용한 컴포넌트>
1. boxContainer.js
2. IconButton.js

#CalendarForm
일정을 입력할수 있는 화면으로 선택된 날짜와 날짜를 선택할수 있는 dateTimePicker 그리고 할 일 제목과 해당 할 일의 메모를 입력할수 있는 input으로 구성되어있으며 저장을 위한 등록 버튼이 배치되어있습니다.

<화면>

<실제 코드>

```
import React, { useState } from 'react';
import { StatusBar, Dimensions, View } from 'react-native';
import styled, { ThemeProvider } from 'styled-components/native';
import DateBox from '../components/DateBox'
import TaskInput from '../components/TaskInput'
import CustonButton from '../components/CustomButton';
import { useTasksContext } from '../TaskContext';
import {lightTheme, darkTheme} from '../theme'
```

```
const Container = styled.SafeAreaView`
  flex: 1;
  background-color: ${({ theme }) => theme.background};
  align-items: center;
  justify-content: flex-start;
`;
const Title = styled.Text`
  font-size: 24px;
  font-weight: 700;
  color: ${({ theme }) => theme.main};
  align-self: flex-start;
  margin: 20px 0;
`;

const BackButton = styled.TouchableOpacity`
  height: 45px;
  background-color: ${({ darkMode }) =>
  darkMode ? darkTheme.itemCompletedBackground: '#416AD7'};
  padding: 10px;
  border-radius: 10px;
  align-self:center;
  margin: 0 10px;
`;

const BackButtonText = styled.Text`
  font-size: 18px;
  color: white;
`;

const ViewStyle = styled(View)`
  flex-direction: row;
  width: ${({ width }) => width - 40}px;
  justify-content: space-between;
`;

export default function App({navigation, route}) {
  const width = Dimensions.get('window').width;
  const [newTask, setNewTask] =
  useState(route.params.item != null?route.params.item.text:'');
  const [newMemo, setNewMemo] =
  useState(route.params.item != null?route.params.item.memo:'');
```

```
  const { tasks, updateTasks,darkMode } = useTasksContext();
  const [selectedDate, setSelectedDate] =
useState(route.params.item != null?
route.params.item.date : route.params.selectedDate);

  const _saveTasks = () => {
      navigation.navigate('Calendar');
  };

  const _addTask = () => {
    if (!newTask.trim() && !newMemo.trim()) {
      return;
    }

    const ID = route.params.item != null ?
    route.params.item.id : Date.now().toString();

    const newTaskObject = {
      id: ID,
      text: newTask,
      completed: false,
      date: selectedDate,
      memo: newMemo,
    };

    setNewTask('');
    setNewMemo('');

    if (route.params.item != null) {
      tasks[ID]=newTaskObject;
      updateTasks(tasks);
    }
      updateTasks({ ...tasks, [ID]: newTaskObject });

    _saveTasks();
  };

  const _handleTextChange = text=> {
    setNewTask(text);
  };
  const _handleTextChangeMemo = memo=> {
```

```
    setNewMemo(memo);
  };

  const handleDateChange = (date) => {
    setSelectedDate(date)
  };

  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
      <Container>
        <StatusBar
          barStyle={darkMode ? "light-content":"dark-content"}
          backgroundColor={darkMode ?
          darkTheme.background:lightTheme.background}
        />
        <ViewStyle width={width}>
          <Title>일정 등록</Title>
          <BackButton darkMode={darkMode} onPress={() => navigation.goBack() }>
            <BackButtonText>뒤로가기</BackButtonText>
          </BackButton>
        </ViewStyle>
        <DateBox date={selectedDate} width={width}
         onDateChange={handleDateChange}/>
        <TaskInput
          value={newTask}
          memo={newMemo}
          onChangeText={_handleTextChange}
          onChangeTextMemo={_handleTextChangeMemo}
          onSubmitEditing={_addTask}
        />
        <CustonButton width={width} title="등록" onPress={_addTask}/>
      </Container>
    </ThemeProvider>
   );
}
```

<코드 설명>
-Calendar화면으로부터 수정 버튼을 통해 넘어온 경우 task와 memo, seletectdate를 해당 item의 것으로 초기화 하고 새로만들기 버튼을 통해 넘어온 경우 task와 memo는 공백으로 seletedate는 오늘 날짜로 초기화 합니다.
-memo와 task에 빈칸으로 있는 경우 등록버튼을 눌러도 저장이 안되고 화면에 그대로 남게

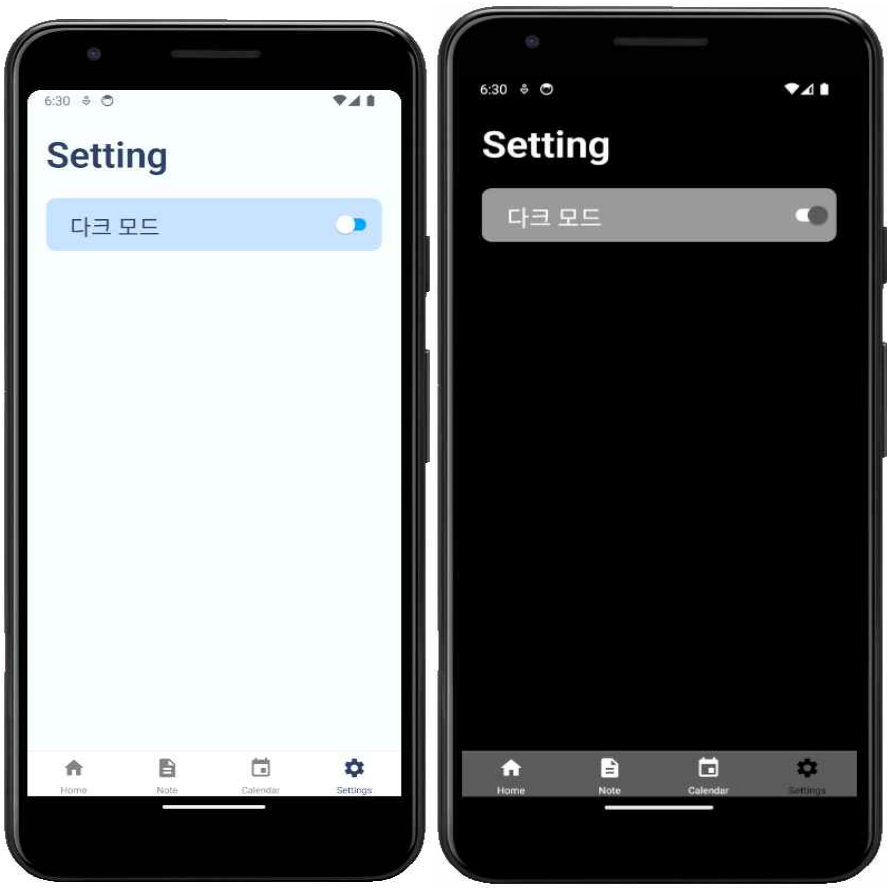
구현, 빈칸이 아닌 경우 저장과 함께 calendar화면으로 이동

<참고 문헌>
처음배우는 리액트 네이티브 (김범준 지음/한빛미디어),
https://www.npmjs.com/package/react-datetime-picker

<사용 컴포넌트>
1. DateBox.js
2. TaskInput.js
3. CustomButton.js

#Setting
세팅 스크린에 다크모드 설정이 가능한 토글버튼을 놓고 토글 시 테마 변경

<화면>



<실제 코드>

```
import React, { useState, useEffect } from 'react';
import { StatusBar } from 'react-native';
import styled, { ThemeProvider } from 'styled-components/native';
import SettingComponent from '../components/SettingComponent'
import { useTasksContext } from '../TaskContext';
import {lightTheme, darkTheme} from '../theme'

const Container = styled.SafeAreaView`
  flex: 1;
  background-color: ${({ theme }) => theme.background};
  align-items: center;
  justify-content: flex-start;
`;
```

```
const Title = styled.Text`
  font-size: 40px;
  font-weight: 600;
  color: ${({ theme }) => theme.main};
  align-self: flex-start;
  margin: 20px;
`;

export default function App({navigation}) {
  const { darkMode} = useTasksContext();

  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
      <Container>
        <status bar
          barStyle={darkMode ? "light-content":"dark-content"}
          backgroundColor={darkMode ?
          darkTheme.background:lightTheme.background}
        />
        <Title>Setting</Title>
        <SettingComponent title="다크 모드"/>
      </Container>
    </ThemeProvider>
  );
}
```

<코드 설명>
-useTasksContext로 전역변수인 darkMode 가져오기
-darkMode의 bool값으로 테마 적용
-SettingComponent로 darkMode update

<참고 문헌> 처음배우는 리액트 네이티브 (김범준 지음/한빛미디어)

<사용 컴포넌트>
1. SettingComponent.js

#IconButton
아이콘 모양의 버튼인 커스텀 컴포넌트로 props로 넘겨준 이미지와 기능으로 버튼역할을 합니다.

<실제 코드>

```
import React from 'react';
import { TouchableOpacity } from 'react-native';
import styled from 'styled-components/native';
import PropTypes from 'prop-types';
import { images } from '../images';

const Icon = styled.Image`
  tint-color: ${({ theme, completed }) =>
    completed ? theme.done : theme.text};
  width: 30px;
  height: 30px;
  margin: 10px;
`;

const IconButton = ({ type, onPressOut, id, completed }) => {
  const _onPressOut = () => {
    onPressOut(id);
  };

  return (
    <TouchableOpacity onPressOut={_onPressOut}>
      <Icon source={type} completed={completed} />
    </TouchableOpacity>
  );
};

IconButton.defaultProps = {
  onPressOut: () => {},
};

IconButton.propTypes = {
  type: PropTypes.oneOf(Object.values(images)).isRequired,
  onPressOut: PropTypes.func,
  id: PropTypes.string,
  completed: PropTypes.bool,
};
```

```
export default IconButton;
```

<코드 설명>
-props로부터 넘어온 type을 이미지 폴더에서 찾아 아이콘 이미지에 넣고 넘어온 기능 함수를 onPressOut에 할당해주어 해당 버튼이 클리이 됬을 때 함수를 실행

<참고 문헌>
처음배우는 리액트 네이티브 (김범준 지음/한빛미디어)에 iconButton만들기 인용

#CustomButton
사용자가 원하는 버튼을 재사용하여 만들 수 있는 커스텀 컴포넌트

<실제 코드>

```
import React from 'react';
import { TouchableOpacity, Text } from 'react-native';
import styled , { ThemeProvider } from 'styled-components/native';
import { useTasksContext } from '../TaskContext';
import {lightTheme, darkTheme} from '../theme'
const Contents = styled.Text`
    color: #fff;
    text-align: center;
    font-size: ${props=>props.title==="등록"?'24px':'15px'};
    font-weight: 700;
`;

const Container = styled.View`
    background: ${(props)=>props.title==="등록"?(props.darkMode ? darkTheme.itemCompletedBackground: '#416AD7'):'#3E92FF'};
    height: ${props=>props.title==="등록"?'60px':'45px'};
    width: ${({ width }) => (width - 40)}px;
    border: 1px solid ${({ darkMode }) => darkMode ? darkTheme.itemCompletedBackground: '#9EC8FF'};
    border-radius: 20px;
    padding: 10px 20px;
    margin:20px;
`;

const CustonButton = props => {
const { darkMode } = useTasksContext();
  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
    <TouchableOpacity onPressOut={props.onPress}>
        <Container width={props.width} title={props.title} darkMode={darkMode}>
            <Contents title={props.title} darkMode={darkMode}>{props.title}</Contents>
        </Container>
    </TouchableOpacity>
    </ThemeProvider>
  );
};
```

```
CustonButton.defaultProps = {
  onPressOut: () => {},
};

export default CustonButton;
```

<코드 설명>
props로 넘어온 넓이값과 버튼의 쓸 title과 다크모드로 인한 버튼 색깔을 가져와 넘겨주어 해당 커스텀 버튼을 구현

<참고 문헌>
처음배우는 리액트 네이티브 (김범준 지음/한빛미디어)

#ToggleButton
Swich를 이용한 토글을 구현

<실제 코드>
// ToggleSwitch.js

```
import React, { useState, useEffect } from 'react';
import { View, Switch, StyleSheet } from 'react-native';
import { ThemeProvider } from 'styled-components/native';
import { useTasksContext } from '../TaskContext';
import {lightTheme, darkTheme} from '../theme'

const ToggleButton = ({ onPress, completed }) => {
  const [isEnabled, setIsEnabled] = useState(completed);
  const { darkMode} = useTasksContext();

  useEffect(() => {
    setIsEnabled(completed);
  }, [completed]);

  const toggleSwitch = () => {
    setIsEnabled((prev) => !prev);
    onPress && onPress(!isEnabled);
  };

  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
    <View style={styles.container}>
      <Switch
        trackColor={{ false: darkMode ? darkTheme.toggleunfilledColor:lightTheme.toggleunfilledColor, true: darkMode ? darkTheme.toggleDone:lightTheme.toggleDone }}
        thumbColor={isEnabled ? (darkMode ? darkTheme.tabBarColor:lightTheme.tabBarColor) : (darkMode ?darkTheme.tabBarColor:lightTheme.tabBarColor)}
        ios_backgroundColor="#3e3e3e"
        onValueChange={toggleSwitch}
        value={completed}
      />
    </View>
    </ThemeProvider>
```

```
  );
};

const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
    justifyContent: 'center',
    margin: 0,
    padding: 0,
  },
});

export default ToggleButton;
```

<코드 설명>
-전역변수인 darkMode를 가져와 해당 값에 따라 Switch에 trackColor와 ThumbColor값을 선택하고 토글버튼의 bool값을 변경하여 전달할수 있게 구현

<참고 문헌>
React-native Switch 공식 문서

#Task
리스트 목록에 나타날 일정들을 표현할 커스텀 컴포넌트

<실제 코드>

```
import React, { useState } from 'react';
import styled from 'styled-components/native';
import PropTypes from 'prop-types';
import Input from './Input';
import ToggleButton from './ToggleButton';

const Container = styled.View`
  flex-direction: row;
  align-items: center;
  background-color: ${({ theme,completed }) => (completed ? theme.itemCompletedBackground : theme.itemBackground)};
  border-radius: 10px;
  padding: 5px;
  margin: 3px 0px;
`;

const Contents = styled.Text`
  flex: 1;
  font-size: 24px;
  color: ${({ theme, completed }) => (completed ? theme.done : theme.text)};
  text-decoration-line: ${({ completed }) =>
    completed ? 'line-through' : 'none'};
    margin: 0 20px;
`;

const Task = ({ item, toggleTask}) => {
  const [isEditing, setIsEditing] = useState(false);
  const [text, setText] = useState(item.text);


  const _handleTogglePress = () => {
    toggleTask(item.id);
  };

  return isEditing ? (
    <Input
      value={text}
```

```
        onChangeText={text => setText(text)}
        onSubmitEditing={_onSubmitEditing}
        onBlur={_onBlur}
      />
    ) : (
      <Container completed={item.completed}>
        <ToggleButton onPress={_handleTogglePress} completed={item.completed} />
        <Contents completed={item.completed}>{item.text}</Contents>

      </Container>
    );
};

Task.propTypes = {
  item: PropTypes.object.isRequired,
  toggleTask: PropTypes.func.isRequired,
};

export default Task;
```

<코드 설명>
-toggleButton 컴포넌트를 앞에 두어 토글 가능하게 만들고 contents에 props로부터 받아온 item의 text를 넣어 구현

<참고문헌>
처음배우는 리액트 네이티브 (김범준 지음/한빛미디어)에 task를 인용하여 변경

#Box.js
메인화면에서 해야할 일과 완료한 일을 나타내는 Box 컴포넌트.

<실제 코드>
```
const Box = props =>{
    const { darkMode} = useTasksContext()|| {};
    return (
        <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
        <Container title={props.title} width={props.width} darkMode={darkMode}>
            <Contents>{props.title}</Contents>
            <Contents>{props.count}</Contents>
        </Container>
        </ThemeProvider>
    )
};
```

<코드 설명>
-부모인 메인 홈으로부터 title과 count를 props로 넘겨받아 값을 설정한다.

<참고 문헌>
없음

#ProgressBar.js
해야할 일과 완료한 일을 게이지로 사용자가 보기쉽게 나타냄.

<실제 코드>

```
return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
    <BarView>
      <Bar>
        <Progress.Bar
          progress={progressValue}
          width={null}
          height={8}
          color={darkMode ? darkTheme.toggleDone:lightTheme.toggleDone}
          borderColor={darkMode ? darkTheme.toggleborderColor:lightTheme.toggleb
orderColor}
          unfilledColor={darkMode ? darkTheme.toggleunfilledColor:lightTheme.toggle
unfilledColor}
        />
      </Bar>
      <BarText>
        {completedValue}/{lengthValue}
      </BarText>
    </BarView>
    </ThemeProvider>
  );
};
```

<코드 설명>
-메인 홈으로부터 총 해야할 일 개수와 완료한 일의 개수를 props로 넘겨 받음.
- 이후 ProgressBar의 Value로 해당 값을 설정. completedValue / lengthValue.

<참고 문헌>
Progress 관련 npm 공식 문서: https://www.npmjs.com/package/react-native-progres
s

#MemoTask
메모화면에 리스트로 보일 메모 목록의 task를 구현한 것으로 메모내용과 수정,삭제 버튼이 할당되어있는 커스텀 컴포넌트

<실제 코드>

```
import React, { useState } from 'react';
import styled from 'styled-components/native';
import PropTypes from 'prop-types';
import IconButton from './IconButton';
import { images } from '../images';

const Container = styled.View`
  flex-direction: row;
  align-items: center;
  background-color: ${({ theme}) => theme.itemBackground};
  border-radius: 15px;
  padding: 5px;
  margin: 3px 0px;
`;

const Contents = styled.Text`
  padding:0 20px;
  flex: 1;
  font-size: 24px;
  color: ${({ theme }) => theme.text};
`;

const MemoTask = ({ item, deleteMemo, updateMemo }) => {
  return (
    <Container>
      <Contents>{item.title}</Contents>
        <IconButton
          type={images.update}
          id={item.id}
          onPressOut={updateMemo}
        />
      <IconButton
        type={images.delete}
        id={item.id}
        onPressOut={deleteMemo}
      />
```

```
      </Container>
    );
};

MemoTask.propTypes = {
    item: PropTypes.object.isRequired,
    deleteMemo: PropTypes.func.isRequired,
    updateMemo: PropTypes.func.isRequired,
};

export default MemoTask;
```

<코드 설명>
props로부터 받은 item에 title을 Contents에 넣어 text로 나타나게 하고 그옆에 iconButton을 통해 수정 버튼과 삭제 버튼을 구현하여 배치

<참고 문헌>
처음배우는 리액트 네이티브 (김범준 지음/한빛미디어)에 task를 인용하여 변경

#CalendarTask
Calendar화면에 리스트로 보여질 할 일 목록을 나타내는 커스텀 컴포넌트

<실제 코드>

```
import React, { useState } from 'react';
import styled from 'styled-components/native';
import PropTypes from 'prop-types';
import IconButton from './IconButton';
import { images } from '../images';

const Container = styled.View`
  flex-direction: row;
  align-items: center;
  background-color: ${({ theme}) => theme.itemBackground};
  border-radius: 15px;
  padding: 5px;
  margin: 3px 0px;
`;

const Contents = styled.Text`
  padding:0 20px;
  flex: 1;
  font-size: 24px;
  color: ${({ theme }) => theme.text};
`;

const CalendarTask = ({ item, deleteTask, onPressOut }) => {
  return (
    <Container>
      <Contents>{item.text}</Contents>
        <IconButton
          type={images.update}
          id={item.id}
          onPressOut={onPressOut}
        />
        <IconButton
          type={images.delete}
          id={item.id}
          onPressOut={deleteTask}
        />
    </Container>
```

```
    );
};

CalendarTask.propTypes = {
    item: PropTypes.object.isRequired,
    deleteTask: PropTypes.func.isRequired,
    updateTask: PropTypes.func.isRequired,
};

export default CalendarTask;
```

<코드 설명>
props로부터 받은 item에 title을 Contents에 넣어 text로 나타나게 하고 그옆에 iconButton을 통해 수정 버튼과 삭제 버튼을 구현하여 배치

<참고 문헌>
처음배우는 리액트 네이티브 (김범준 지음/한빛미디어)에 task를 인용하여 변경

#DateBox
DatePicker와 설정한 날짜 값을 나타내는 컴포넌트

<실제 코드>
```
const Box = ({date, width, onDateChange}) =>{
    const { darkMode} = useTasksContext();
    console.log('선택된 날짜' , date);

    return (
        <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
            <Container width={width}>
                <Contents darkMode={darkMode}>{date}</Contents>
                <DatePicker title="날짜" date = {date} onDateChange={onDateChange}
/>
            </Container>
        </ThemeProvider>
    )
};
```

<코드 설명>
- DatePicker 컴포넌트를 사용해 날짜를 선택할 수 있고 선택된 날짜는 contents로 나타냄.

<참고 문헌>
없음

<사용 컴포넌트>
1. DatePicker

#DatePicker

일정 등록시 날짜 선택을 위해 필요한 컴포넌트

<실제 코드>

```
const CustonButton = ({title, onDateChange}) => {
const [isDatePickerVisible, setDatePickerVisibility] = useState(false);
const { darkMode} = useTasksContext();
const showDatePicker = () => {
    setDatePickerVisibility(true);
};
const hideDatePicker = () => {
    setDatePickerVisibility(false);
};
const handleConfirm = (date) => {
    console.warn("A date has been picked: ", date);
    const formattedDate = date.toISOString().split('T')[0];
    onDateChange(formattedDate);
    hideDatePicker();
};
  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
    <TouchableOpacity onPressOut={showDatePicker}>
        <Container darkMode={darkMode}>
            <Contents darkMode={darkMode}>{title}</Contents>
            <DateTimePickerModal
                isVisible={isDatePickerVisible}
                mode="date"
                onConfirm={handleConfirm}
                onCancel={hideDatePicker}
            />
        </Container>
    </TouchableOpacity>
    </ThemeProvider>
  );
};
```

<코드 설명>

- 만약 DatePicker를 클릭하면 앞서 설정한 isDatePicker가 true로 바뀌고 선택창이 사용자에게 보여진다.
- 이후 사용자가 원하는 날짜를 클릭하면 해당하는 날짜 값을 가져온다.

<참고 문헌>

reactDatePicker 공식문서: https://reactdatepicker.com/

#TaskInput
일정 등록 시 제목과 내용을 따로 입력하기 위한 Input 컴포넌트.

<실제 코드>

```
const Input = ({
  value,
  memo,
  onChangeText,
  onChangeTextMemo,
}) => {
  const width = Dimensions.get('window').width;
  const { darkMode, updateDarkMode} = useTasksContext();
  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
      <Container>
        <StyledInput
          width={width}
          placeholder="할일"
          maxLength={50}
          autoCapitalize="none"
          autoCorrect={false}
          returnKeyType="done"
          keyboardAppearance="dark" // iOS only
          value={value}
          onChangeText={onChangeText}
          darkMode={darkMode}
        />
        <MemoInput
          width={width}
          placeholder="메모"
          maxLength={50}
          autoCapitalize="none"
          autoCorrect={false}
          returnKeyType="done"
          keyboardAppearance="dark" // iOS only
          value={memo}
          onChangeText={onChangeTextMemo}
          darkMode={darkMode}
        />
      </Container>
    </ThemeProvider>
```

```
  );
};

export default Input;
```

<코드 설명>
- Input 태그를 2개 합쳐서 사용자가 제목과 본문을 따로 입력 가능하도록 만든 컴포넌트.

<참고 문헌>
없음

#SettingComponent
셋팅 화면에서 목록들을 나타낼 때 사용하는 컴포넌트

<실제 코드>
```
const Task = props => {
  const { darkMode, updateDarkMode} = useTasksContext();
  const [isEnabled, setIsEnabled] = useState(darkMode);

  const toggleSwitch = () => {
    updateDarkMode(!darkMode);
  };
  return (
    <ThemeProvider theme={darkMode ? darkTheme:lightTheme}>
    <Container >
      <Contents>{props.title}</Contents>
      <Switch
        trackColor={{ false: darkMode ? darkTheme.toggleunfilledColor:lightTheme.toggleunfilledColor, true: darkMode ? darkTheme.toggleDone:lightTheme.toggleDone }}
        thumbColor={isEnabled ? (darkMode ? darkTheme.tabBarColor:lightTheme.tabBarColor) : (darkMode ?darkTheme.tabBarColor:lightTheme.tabBarColor)}
        ios_backgroundColor="#3e3e3e"
        onValueChange={toggleSwitch}
        value={darkMode}
      />
    </Container>
    </ThemeProvider>
  );
};
```

<코드 설명>
- 설정 아이템의 이름을 담을 contents와 토글버튼인 Switch로 이루어져 있다.
- 토글 버튼을 누르면 해당 아이템의 옵션이 활성화되거나 비활성화 된다.

<참고 문헌>
React Switch 공식문서 : https://www.npmjs.com/package/react-switch