

앱 개발 입문

성균관대학교 컬처애편테크놀로지융합전공 하계 부트캠프

5일차 - 2022. 08. 02 (화)

강의 슬라이드 링크

https://github.com/kunny/skku-bootcamp-2022-summer/blob/main/_slides/day5.pdf

오늘 강의에서 다룰 내용

- 플러터에서 화면을 이동하는 방법을 알아봅니다.
- 노트 목록 화면과 편집 화면을 루트에 추가합니다.
- 노트 보기 화면을 구현합니다.
- 노트 목록, 보기, 편집 화면을 연결합니다.
- 데이터베이스를 알아봅니다.

플러터에서 화면 이동 처리하기

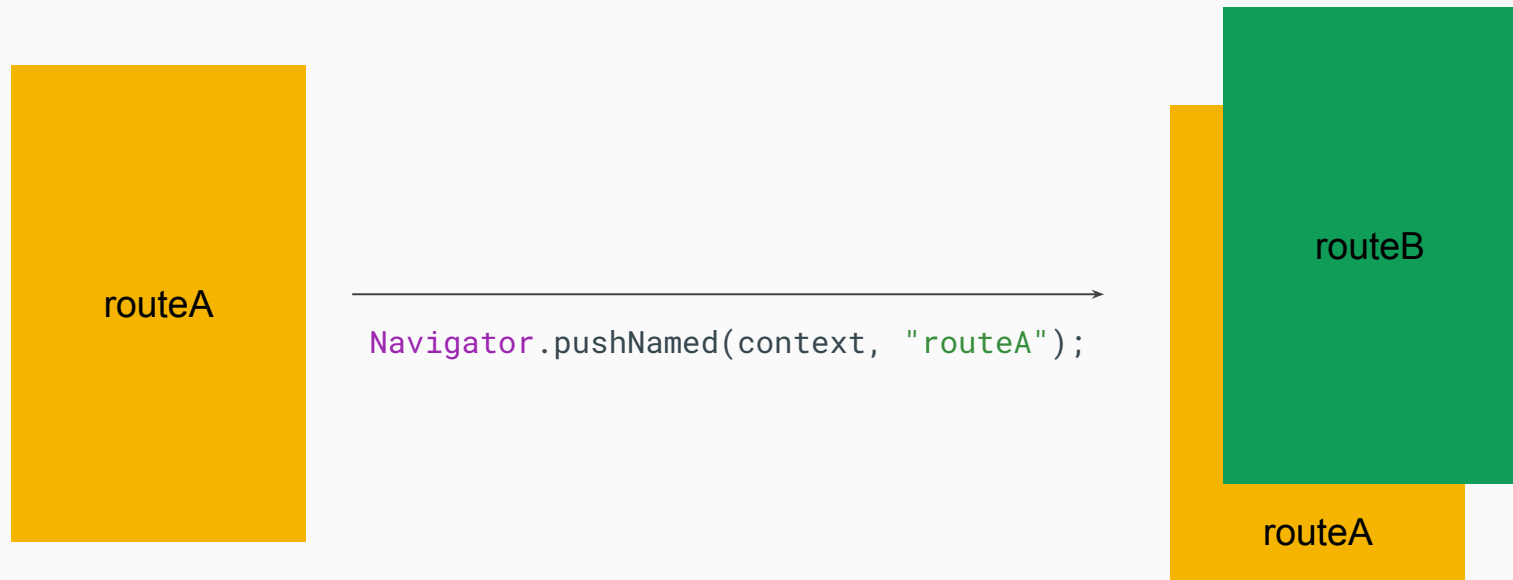
플러터에서 화면 이동 처리하기

- 각 화면은 스택(Stack) 방식으로 관리되며, **Navigator** 클래스 내 다양한 함수를 사용하여 화면을 이동합니다.
- 앱 내의 각 화면은 **루트(Route)**라 부르며, 이름을 지정한 화면은 **Named Route**라 부릅니다.
- **MaterialApp**의 **routes** 속성을 사용하여 앱에서 이동할 수 있는 화면을 정의합니다.
- Named Route를 호출할 때에는 인자(**arguments**)를 추가로 전달할 수 있습니다.
- 호출했던 화면에서 기존 화면으로 돌아왔을 때 이벤트를 받을 수 있습니다. (**Future**)

플러터에서 화면 이동 처리하기

- 새 화면 열기 (현재 화면 유지)
 - [Navigator.push\(\)](#) / [Navigator.pushNamed\(\)](#)
- 현재 화면을 새 화면으로 대체
 - [Navigator.pushReplacement\(\)](#) / [Navigator.pushReplacementNamed\(\)](#)
- 현재 화면을 닫고 기존 화면으로 복귀
 - [Navigator.pop\(\)](#)
- 현재 화면을 닫고 특정 조건을 만족하는 화면으로 복귀
 - [Navigator.popUntil\(\)](#)

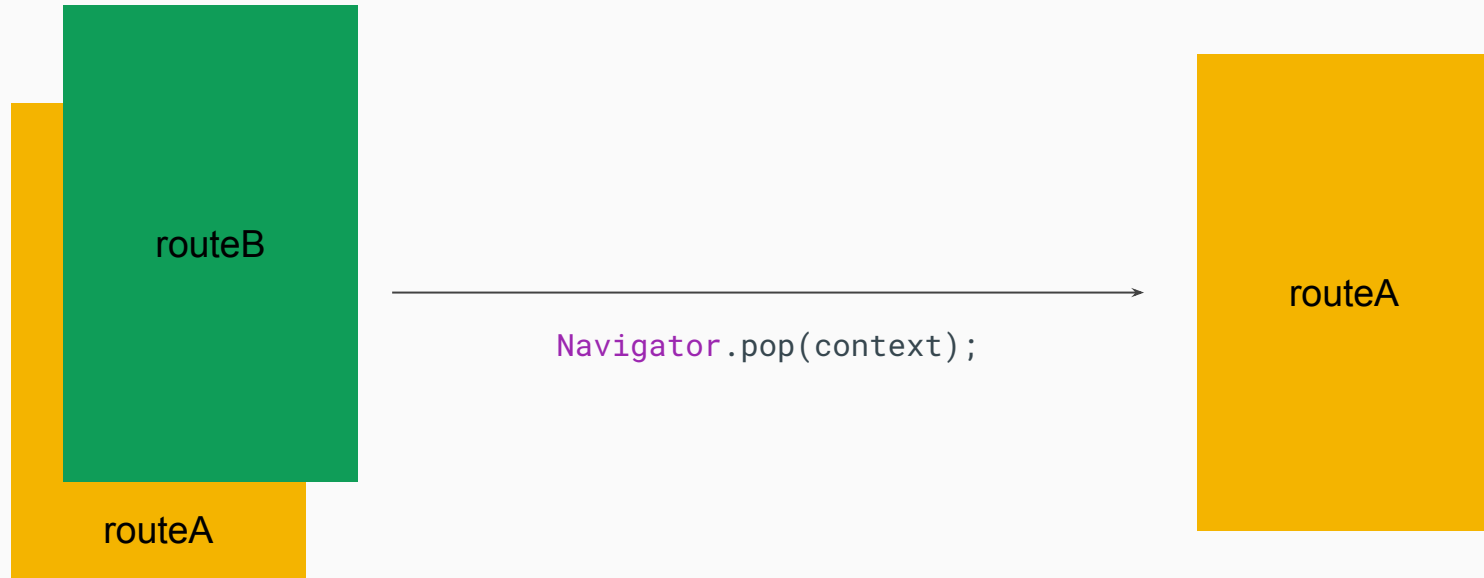
플러터에서 화면 이동 처리하기



플러터에서 화면 이동 처리하기



플러터에서 화면 이동 처리하기



플러터에서 화면 이동 처리하기



노트 목록 화면과 편집 화면을
루트에 추가하기

노트 목록 화면과 편집 화면을 루트에 추가하기

1. `NoteListPage`와 `NoteEditPage` 화면에 이름을 지어줍니다.
2. `MyApp` 클래스 내 `MaterialApp`의 `initialRoute`와 `routes` 속성을 추가하여 앱의 첫 페이지와 앱에 포함되는 화면을 구성합니다.

노트 목록 화면과 편집 화면을 루트에 추가하기

```
class NoteListPage extends StatefulWidget {  
  static const routeName = '/';  
  
  const NoteListPage({Key? key}) : super(key: key);  
  
  @override  
  State createState() => _NoteListPageState();  
}
```

노트 목록 화면과 편집 화면을 루트에 추가하기

```
class NoteEditPage extends StatefulWidget {  
  static const routeName = '/edit';  
  
  const NoteEditPage({Key? key}): super(key: key);  
  
  @override  
  State createState() => _NoteEditPageState();  
}
```

노트 목록 화면과 편집 화면을 루트에 추가하기

```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      ...  
      initialRoute: NoteListPage.routeName,  
      routes: {  
        NoteListPage.routeName: (context) => const NoteListPage(),  
        NoteEditPage.routeName: (context) => const NoteEditPage(),  
      },  
    );  
  }  
}
```

노트 보기 화면 구현하기

노트 보기 화면 구현하기

1. `page` 폴더에 `note_view_page.dart` 파일을 추가합니다.
2. `note_view_page.dart` 파일에 `NoteViewPage` 클래스를 구현합니다.
3. `MyApp` 클래스 내 `MaterialApp`의 `routes` 속성에 `NoteViewPage`를 추가합니다.

11:29



DEBUG



(제목 없음)



□□□□

노트 본문 화면 작성하기

```
class NoteViewPage extends StatefulWidget {  
  static const routeName = '/view';  
  
  final int index;  
  
  const NoteViewPage(this.index, {Key? key}) : super(key: key);  
  
  @override  
  State createState() => _NoteViewPageState();  
}
```

노트 보기 화면 작성하기

```
class _NoteViewPageState extends State<NoteViewPage> {  
  @override  
  Widget build(BuildContext context) {  
    final note = noteService().getNote(widget.index);  
    return Scaffold(  
      appBar: AppBar(  
        title: Text(note.title.isEmpty ? '(제목 없음)' : note.title),  
        actions: [  
          IconButton(  
            icon: const Icon(Icons.edit),  
            tooltip: '편집',  
            onPressed: null,  
          ),  
          IconButton(  
            icon: const Icon(Icons.delete),  
            tooltip: '삭제',  
            onPressed: null,  
          ),  
        ],  
      ),  
      body: Container(  
        width: double.infinity,  
        height: double.infinity,  
        color: note.color,  
        child: SingleChildScrollView(  
          padding: const EdgeInsets.symmetric(  
            horizontal: 12.0, vertical: 16.0),  
          child: Text(note.body),  
        ),  
      ),  
    );  
  }  
}
```

note_view_page.dart

노트 보기 화면 작성하기

```
class _NoteViewPageState extends State<NoteViewPage> {  
  ...  
  void _edit(int index) {  
    Navigator.pushNamed(  
      context,  
      NoteEditPage.routeName,  
      arguments: index,  
    ).then((value) {  
      setState(() {});  
    });  
  }  
}
```

노트 보기 화면 작성하기

```
class _NoteViewState extends State<NoteViewPage> {  
  ...  
  void _confirmDelete(int index) {  
    showDialog(  
      context: context,  
      builder: (context) {  
        return AlertDialog(  
          title: const Text('노트 삭제'),  
          content: const Text('노트를 삭제할까요?'),  
          actions: [  
            TextButton(  
              child: const Text('아니오'),  
              onPressed: () {  
                Navigator.pop(context);  
              },  
            ),  
            TextButton(  
              child: const Text('예'),  
              onPressed: () {  
                noteService().deleteNote(index);  
                Navigator.popUntil(context, (route) => route.isFirst);  
              },  
            ),  
          ],  
        );  
      },  
    );  
  }  
}
```

note_view_page.dart

노트 보기 화면 작성하기

```
class _NoteViewPageState extends State<NoteViewPage> {  
  @override  
  Widget build(BuildContext context) {  
    ...  
    return Scaffold(  
      appBar: AppBar(  
        title: Text(note.title.isEmpty ? '(제목 없음)' : note.title),  
        actions: [  
          IconButton(  
            icon: const Icon(Icons.edit),  
            tooltip: '편집',  
            onPressed: () {  
              _edit(widget.index);  
            },  
          ),  
          IconButton(  
            icon: const Icon(Icons.delete),  
            tooltip: '삭제',  
            onPressed: () {  
              _confirmDelete(widget.index);  
            },  
          ),  
        ],  
      ),  
    );  
  }  
}
```

note_view_page.dart

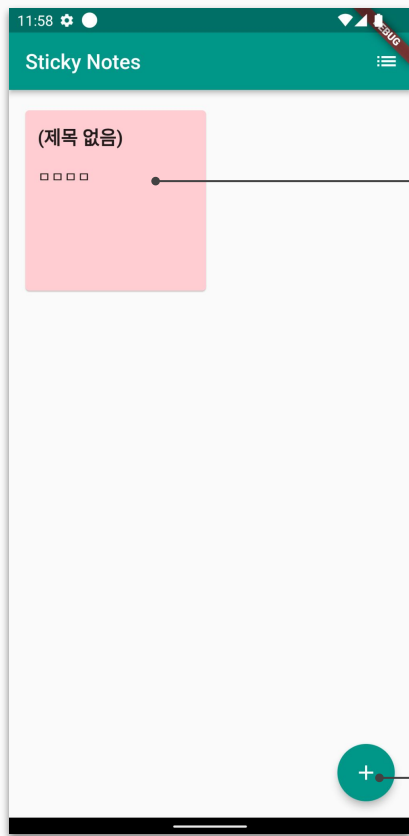
노트 보기 화면 작성하기

```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      ...  
      routes: {  
        ...  
        NoteViewPage.routeName: (context) {  
          final index = ModalRoute.of(context)!.settings.arguments as int;  
          return NoteViewPage(index);  
        },  
      },  
    );  
  }  
}
```

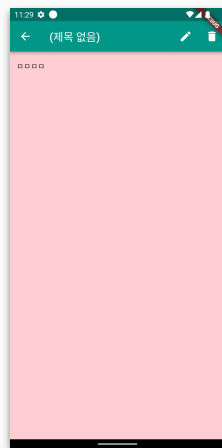

노트 목록, 보기, 편집 화면
연결하기

노트 목록, 보기, 편집 화면 연결하기

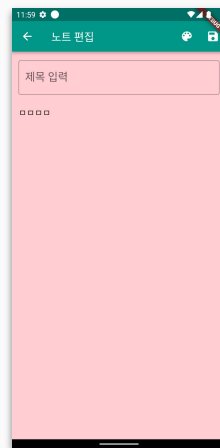
1. 노트 작성 화면이 새 노트를 작성하는 모드와 편집 모드를 지원하게끔 수정합니다.
2. 노트 목록 화면에 새 노트를 추가할 수 있는 버튼을 추가합니다.
3. 노트 목록에 표시된 노트를 눌러 노트 보기 화면으로 이동하게끔 코드를 수정합니다.



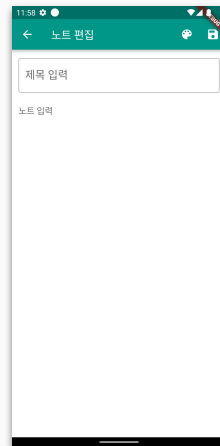
NoteListPage



NoteViewPage



NoteEditPage



노트 목록, 보기, 편집 화면 연결하기

```
class NoteEditPage extends StatefulWidget {  
  static const routeName = '/edit';  
  
  final int? index;  
  
  const NoteEditPage(this.index, {Key? key}): super(key: key);  
  
  @override  
  State createState() => _NoteEditPageState();  
}
```

노트 목록, 보기, 편집 화면 연결하기

```
class _NoteEditPageState extends State<NoteEditPage> {  
  final titleController = TextEditingController();  
  
  final bodyController = TextEditingController();  
  
  Color color = Note.colorDefault;  
  
  @override  
  void initState() {  
    super.initState();  
    final noteIndex = widget.index;  
    if (noteIndex != null) {  
      final note = noteService().getNote(noteIndex);  
      titleController.text = note.title;  
      bodyController.text = note.body;  
      color = note.color;  
    }  
  }  
}
```

note_edit_page.dart

노트 목록, 보기, 편집 화면 연결하기

```
class _NoteEditPageState extends State<NoteEditPage> {

  void _saveNote() {
    if (bodyController.text.isNotEmpty) {
      final note = Note(
        bodyController.text,
        title: titleController.text,
        color: color,
      );

      final noteIndex = widget.index;
      if (noteIndex != null) {
        noteService().updateNote(noteIndex, note);
      } else {
        noteService().addNote(note);
      }

      Navigator.pop(context);
    } else { ... }
  }
}
```

note_edit_page.dart

노트 목록, 보기, 편집 화면 연결하기

```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      ...  
      routes: {  
        ...  
        NoteEditPage.routeName: (context) {  
          final args = ModalRoute.of(context)!.settings.arguments;  
          final index = args != null ? args as int : null;  
          return NoteEditPage(index);  
        },  
      },  
    );  
  }  
}
```

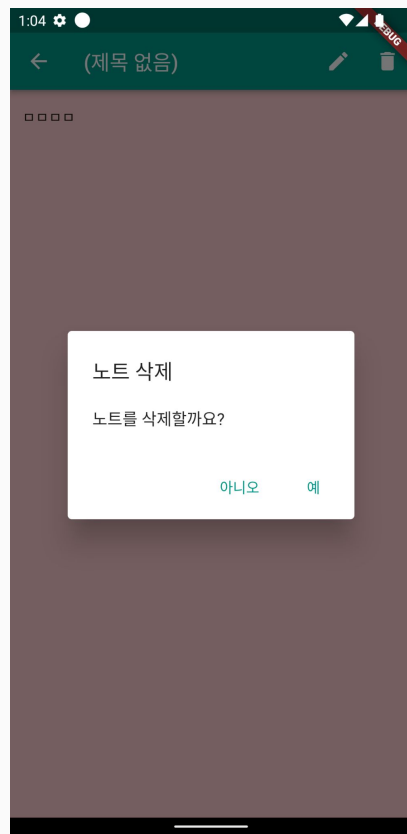
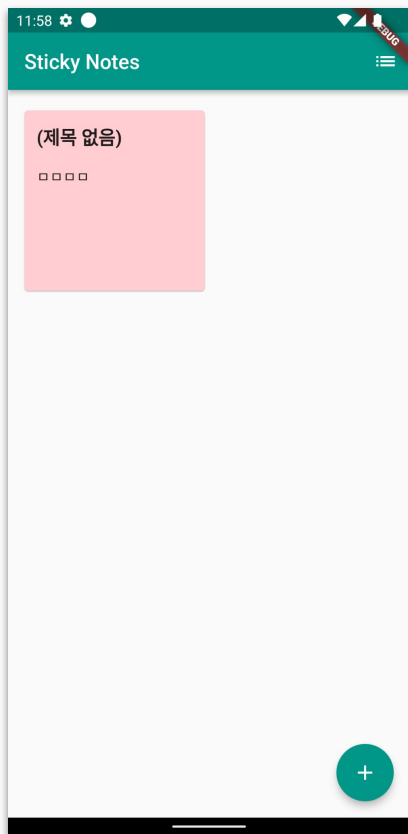
노트 목록, 보기, 편집 화면 연결하기

```
class _NoteListPageState extends State<NoteListPage> {

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      ...
      floatingActionButton: FloatingActionButton(
        tooltip: '새 노트',
        onPressed: () {
          Navigator.pushNamed(context, NoteEditPage.routeName).then((value) {
            setState(() {});
          });
        },
        child: const Icon(Icons.add),
      ),
    );
  }
}
```


노트 목록, 보기, 편집 화면 연결하기

```
class _NoteListPageState extends State<NoteListPage> {  
  ...  
  
  Widget _buildCard(int index, Note note) {  
    return InkWell(  
      onTap: () {  
        Navigator.pushNamed(  
          context,  
          NoteViewPage.routeName,  
          arguments: index,  
        ).then((value) {  
          setState(() {});  
        });  
      },  
      child: Card( ... ),  
    );  
  }  
}
```



완성된 코드

https://github.com/kunny/skku-bootcamp-2022-summer/tree/main/sticky_notes/step4

데이터베이스 알아보기

데이터베이스란?

- 논리적으로 연관된 하나 이상의 자료의 모음입니다.
- 자료를 구조화된 형식으로 저장하므로, 검색과 갱신을 효율적으로 수행할 수 있습니다.
- 안드로이드와 iOS는 간단한 데이터베이스를 처리할 수 있는 SQLite가 내장되어 있습니다.

왜 데이터베이스를 사용하나요? (파일 vs. 데이터베이스)

todo.txt

(제목 없음)

부트캠프 앱 개발
강의 신청하기

2022.07.02

shopping.txt

장보기 목록

- 🍌 양파 1망
- 🥬 양배추 1통
- 🍊 귤 1박스
- 🍷 손질 닭고기 1팩
- 🍖 삼겹살 1팩
- 🍜 우동면 1팩
- (4입)
- 🥛 우유 2팩
- 🍞 식빵 1개

022.08.01

dev_todo.txt

만들어야 할 것들

- 파이어베이스 프로젝트
- 애드몹 계정
- 구글 플레이 개발자 계정
- 애플 개발자 계정

2022.07.24

왜 데이터베이스를 사용하나요? (파일 vs. 데이터베이스)

todo.txt

(제목 없음)

부트캠프 앱 개발
강의 신청하기

2022.07.02

새 데이터를 추가하려면?

shopping.txt

장보기 목록

- 🍌 양파 1망
- 🥬 양배추 1통
- 🍊 귤 1박스
- 🐔 손질 닭고기 1팩
- 🥓 삼겹살 1팩
- 🍜 우동면 1팩
- (4입)
- 🥛 우유 2팩
- 🍞 식빵 1개

2022.08.01

dev_todo.txt

만들어야 할 것들

- 파이어베이스 프로젝트
- 애드몹 계정
- 구글 플레이 개발자 계정
- 애플 개발자 계정

2022.07.24

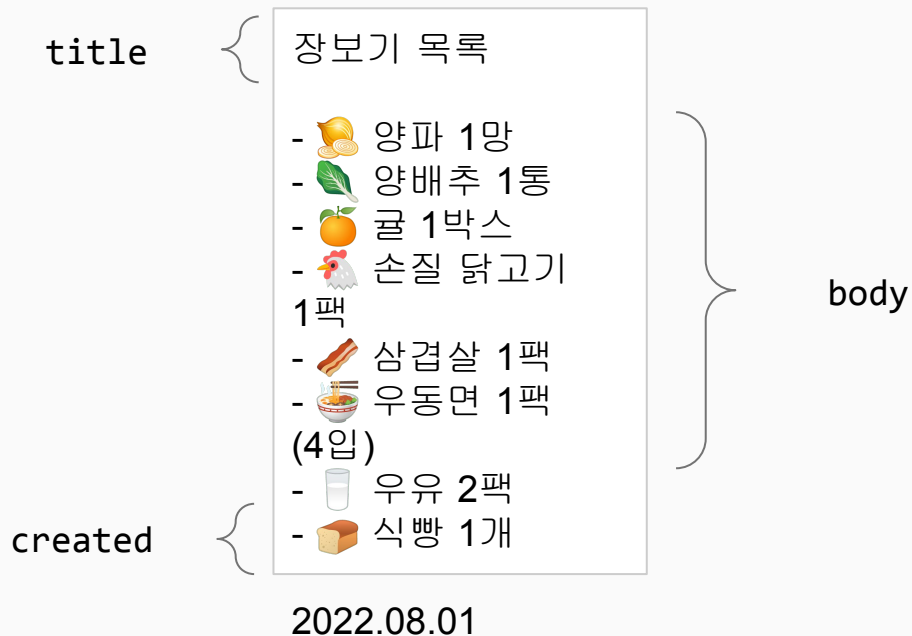
특정 날짜 이후에 생성된
메모만 찾으려면?

특정 문자열을 포함하는
메모만 검색하려면?

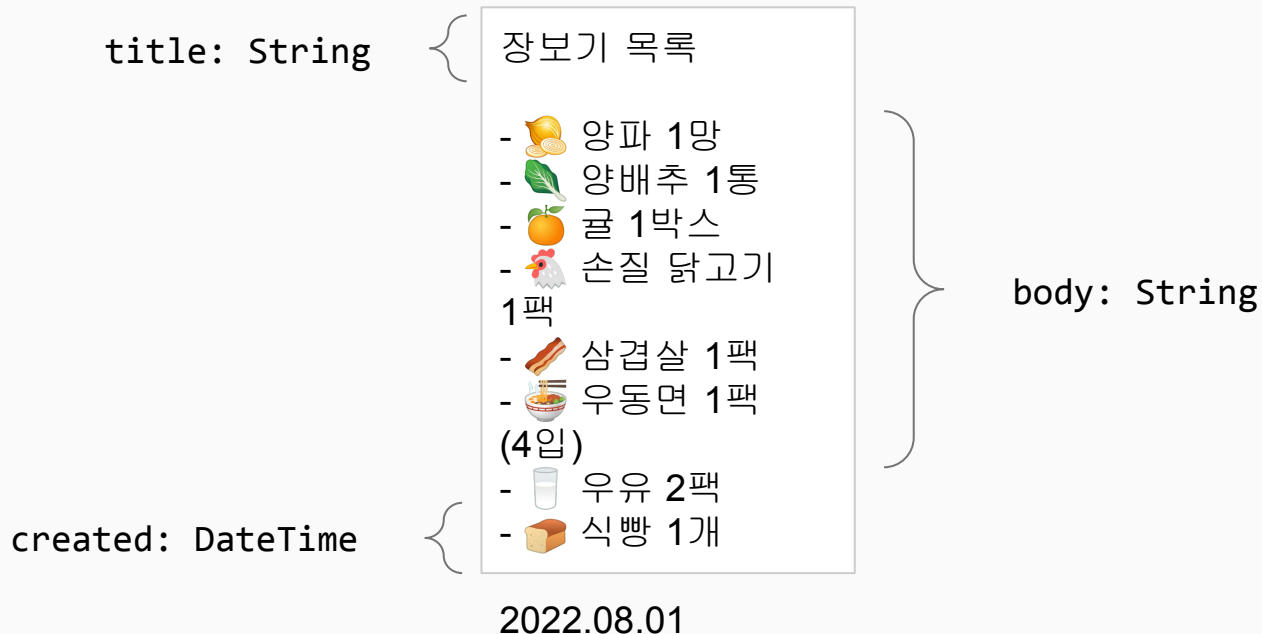
관계형 데이터베이스

- 자료가 구성되는 형태에 따라 다양한 종류의 데이터베이스가 있으며, **관계형(Relational) 데이터베이스**를 가장 많이 사용합니다.
- 관계형 데이터베이스에서 자료는 아래 구성요소로 구성됩니다.
 - 열 (Column): 자료를 구성하는 세부 구성요소로, 필드라 부르기도 합니다. (예: 연락처 자료인 경우 이름, 전화번호 등이 해당)
 - 행 (Row): 자료를 구성하는 열(Column)이 모여 이룬 의미있는 자료로, 레코드라 부르기도 합니다.
 - 고유 키 (Primary Key): 각 레코드를 고유하게 식별할 수 있는 필드를 의미합니다.
 - 테이블: 같은 특징을 갖는 레코드의 집합을 의미합니다. (예: 연락처 목록)

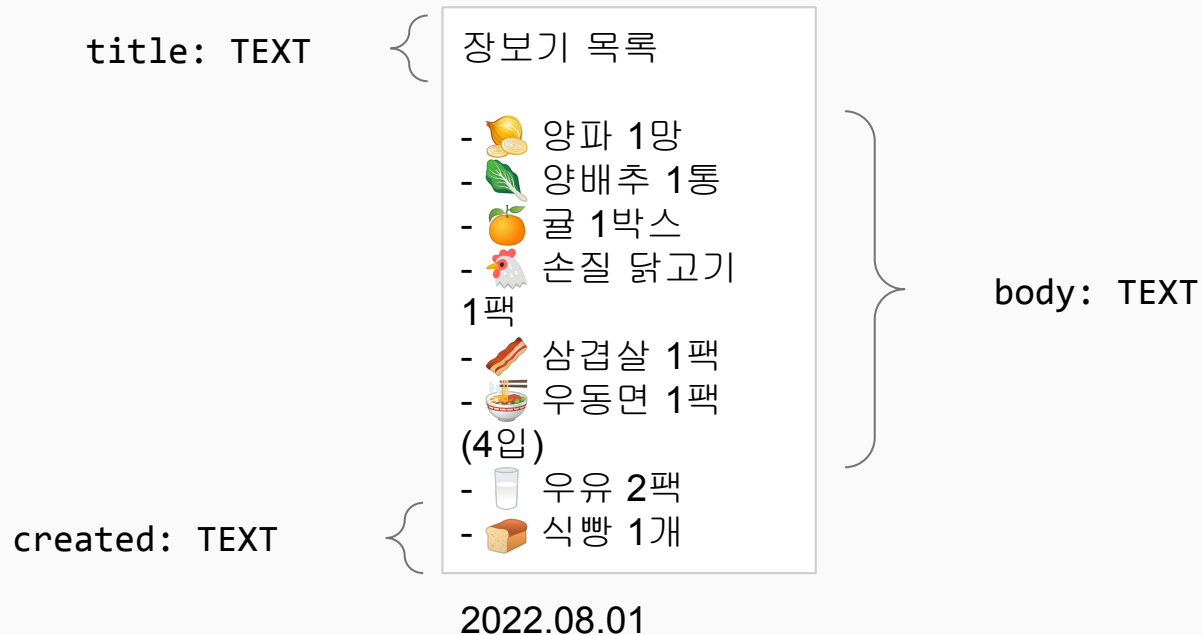
왜 데이터베이스를 사용하나요? (파일 vs. 데이터베이스)



왜 데이터베이스를 사용하나요? (파일 vs. 데이터베이스)



왜 데이터베이스를 사용하나요? (파일 vs. 데이터베이스)



왜 데이터베이스를 사용하나요? (파일 vs. 데이터베이스)

id: INTEGER (AUTO-GENERATED)

title: TEXT

장보기 목록

- 🍌 양파 1망
- 🥬 양배추 1통
- 🍊 귤 1박스
- 🐔 손질 닭고기 1팩
- 🥓 삼겹살 1팩
- 🍜 우동면 1팩 (4입)
- 🥛 우유 2팩
- 🍞 식빵 1개

body: TEXT

created: TEXT





2022.08.01

왜 데이터베이스를 사용하나요? (파일 vs. 데이터베이스)

id: INTEGER (AUTO-GENERATED)

```
title: TEXT
```

장보기 목록

-  양파 1망
-  양배추 1통
-  귤 1박스
-  손질 닭고기 1팩

body: TEXT

created: TEXT

- 우유 2팩
- 식빵 1개

2022.08.01

Record

왜 데이터베이스를 사용하나요? (파일 vs. 데이터베이스)

Primary Key

id	title	body	created
1		부트캠프 앱 개발 강의..	2022.07.02
2	장보기 목록	- 🍷 양파 1망..	2022.08.01
...

Record

Record

Table

데이터베이스 관리하기

- SQL(Structured Query Language)를 사용하여 데이터베이스를 생성하거나 테이블 내 데이터를 조작합니다.
- 데이터 정의 언어 (Data Definition Language)
 - CREATE, ALTER, DROP 등
- 데이터 조작 언어 (Data Manipulation Language)
 - SELECT, INSERT, DELETE, Update 등
- 데이터 제어 언어 (Data Control Language)
 - GRANT, REVOKE, COMMIT, ROLLBACK 등

데이터베이스 관리하기

데이터 정의 언어 예시 (테이블 생성)

```
CREATE TABLE notes(  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  title TEXT,  
  body TEXT NOT NULL,  
  color INTEGER NOT NULL  
)
```


데이터베이스 관리하기

데이터 조작 언어 예시

모든 노트 반환

```
SELECT * FROM notes;
```

노트 제목에 '장보기' 문자열을 포함하는 노트 반환

```
SELECT * FROM notes WHERE title LIKE '%장보기%';
```