# 첫걸음

가장 단순한 FastAPI 파일은 다음과 같이 보일 겁니다:

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

위를 `main.py` 에 복사합니다.

라이브 서버를 실행합니다:

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
<span style="color: green;">INFO</span>:     Started reloader process [28720]
<span style="color: green;">INFO</span>:     Started server process [28722]
<span style="color: green;">INFO</span>:     Waiting for application startup.
<span style="color: green;">INFO</span>:     Application startup complete.
```

!!! note "참고" `uvicorn main:app` 명령은 다음을 의미합니다:

```
* `main`: 파일 `main.py` (파이썬 "모듈").
* `app`: `main.py` 내부의 `app = FastAPI()` 줄에서 생성한 오브젝트.
* `--reload`: 코드 변경 후 서버 재시작. 개발에만 사용.
```

출력에 아래와 같은 줄이 있습니다:

```
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

해당 줄은 로컬에서 앱이 서비스되는 URL을 보여줍니다.

### 확인하기

브라우저로 http://127.0.0.1:8000를 여세요.

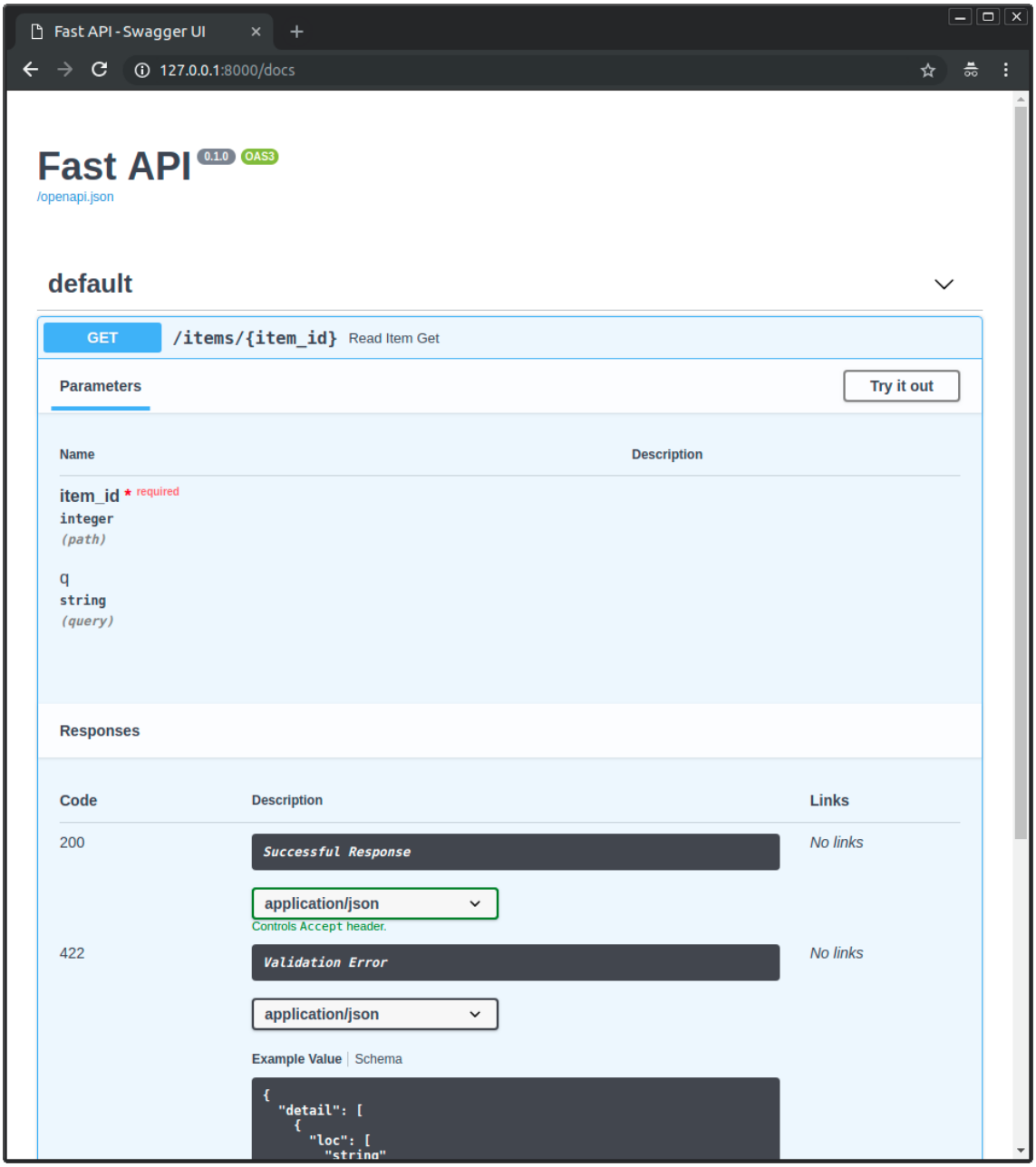
아래와 같은 JSON 응답을 볼 수 있습니다:

```
{"message": "Hello World"}
```

### 대화형 API 문서

이제 http://127.0.0.1:8000/docs로 가봅니다.

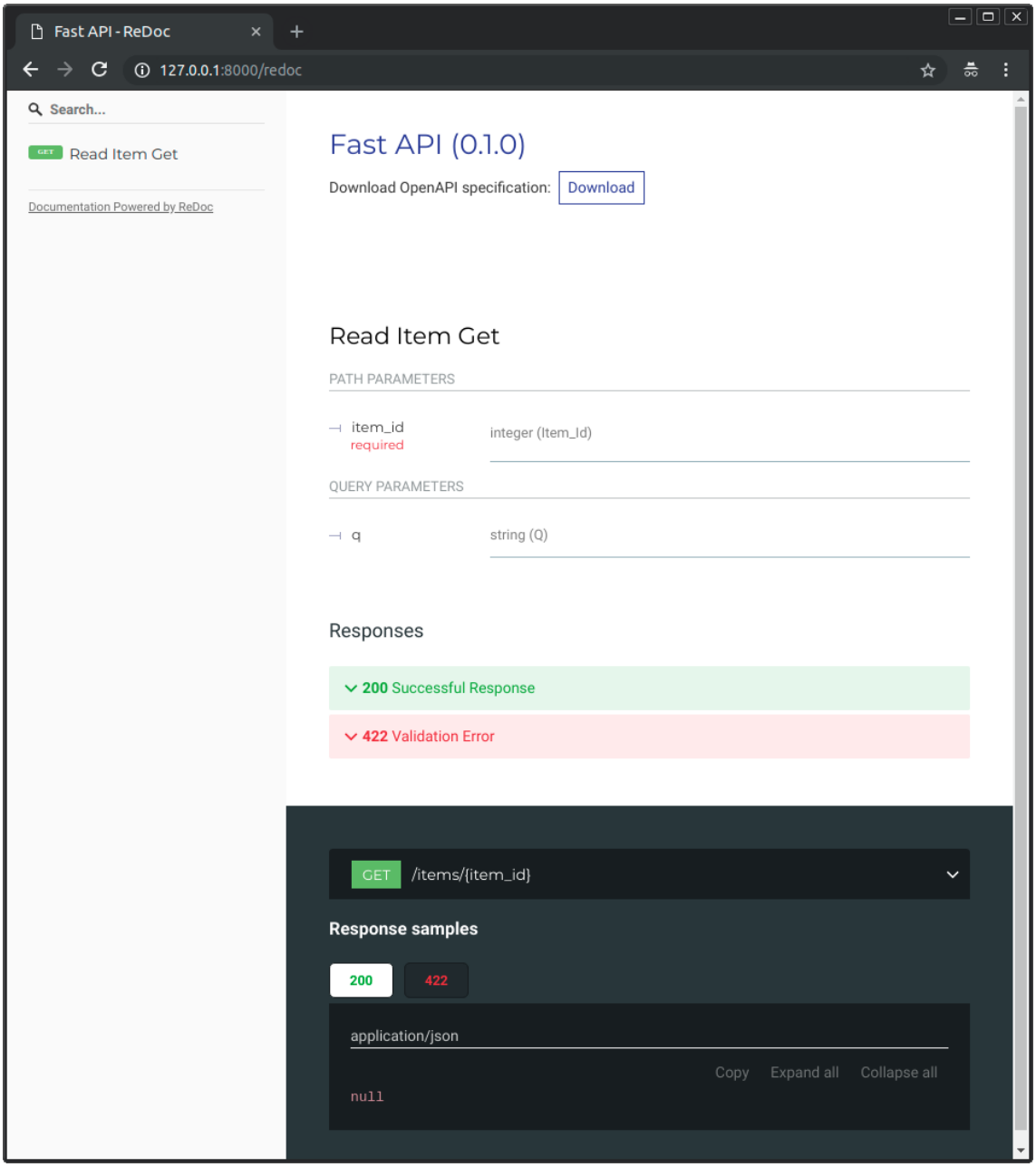
자동 대화형 API 문서를 볼 수 있습니다 (Swagger UI 제공):




### 대안 API 문서

그리고 이제, http://127.0.0.1:8000/redoc로 가봅니다.

대안 자동 문서를 볼 수 있습니다 (ReDoc 제공):




## OpenAPI

**FastAPI**는 API를 정의하기 위한 **OpenAPI** 표준을 사용하여 여러분의 모든 API를 이용해 "스키마"를 생성합니다.

**"스키마"**

"스키마"는 무언가의 정의 또는 설명입니다. 이를 구현하는 코드가 아니라 추상적인 설명일 뿐입니다.

**API "스키마"**

이 경우, OpenAPI는 API의 스키마를 어떻게 정의하는지 지시하는 규격입니다.

이 스키마 정의는 API 경로, 가능한 매개변수 등을 포함합니다.

#### 데이터 "스키마"

"스키마"라는 용어는 JSON처럼 어떤 데이터의 형태를 나타낼 수도 있습니다.

이러한 경우 JSON 속성, 가지고 있는 데이터 타입 등을 뜻합니다.

#### OpenAPI와 JSON 스키마

OpenAPI는 API에 대한 API 스키마를 정의합니다. 또한 이 스키마에는 JSON 데이터 스키마의 표준인 **JSON 스키마**를 사용하여 API에서 보내고 받은 데이터의 정의(또는 "스키마")를 포함합니다.

#### `openapi.json` 확인

가공되지 않은 OpenAPI 스키마가 어떻게 생겼는지 궁금하다면, FastAPI는 자동으로 API의 설명과 함께 JSON (스키마)를 생성합니다.

여기에서 직접 볼 수 있습니다: http://127.0.0.1:8000/openapi.json.

다음과 같이 시작하는 JSON을 확인할 수 있습니다:

```
{
    "openapi": "3.0.2",
    "info": {
        "title": "FastAPI",
        "version": "0.1.0"
    },
    "paths": {
        "/items/": {
            "get": {
                "responses": {
                    "200": {
                        "description": "Successful Response",
                        "content": {
                            "application/json": {


...
```

#### OpenAPI의 용도

OpenAPI 스키마는 포함된 두 개의 대화형 문서 시스템을 제공합니다.

그리고 OpenAPI의 모든 것을 기반으로 하는 수십 가지 대안이 있습니다. **FastAPI**로 빌드한 애플리케이션에 이러한 대안을 쉽게 추가 할 수 있습니다.

API와 통신하는 클라이언트를 위해 코드를 자동으로 생성하는 데도 사용할 수 있습니다. 예로 프론트엔드, 모바일, IoT 애플리케이션이 있습니다.

## 단계별 요약

### 1 단계: `FastAPI` 임포트

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

`FastAPI` 는 API에 대한 모든 기능을 제공하는 파이썬 클래스입니다.

!!! note "기술 세부사항" `FastAPI` 는 `Starlette` 를 직접 상속하는 클래스입니다.

`FastAPI`로 <a href="https://www.starlette.io/" class="external-link" target="_blank">Starlette</a>의 모든 기능을 사용할 수 있습니다.

### 2 단계: `FastAPI` "인스턴스" 생성

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

여기 있는 `app` 변수는 `FastAPI` 클래스의 "인스턴스"가 됩니다.

이것은 모든 API를 생성하기 위한 상호작용의 주요 지점이 될 것입니다.

이 `app` 은 다음 명령에서 `uvicorn` 이 참조하고 것과 동일합니다:

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

아래처럼 앱을 만든다면:

```
{!../../../docs_src/first_steps/tutorial002.py!}
```

이를 `main.py` 파일에 넣고, `uvicorn` 을 아래처럼 호출해야 합니다:

```
$ uvicorn main:my_awesome_api --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

### 3 단계: *경로 동작* 생성

**경로**

여기서 "경로"는 첫 번째 `/` 에서 시작하는 URL의 마지막 부분을 나타냅니다.

그러므로 아래와 같은 URL에서:

```
https://example.com/items/foo
```

...경로는 다음과 같습니다:

```
/items/foo
```

!!! info "정보" "경로"는 일반적으로 "앤드포인트" 또는 "라우트"라고도 불립니다.

API를 빌드하는 동안 "경로"는 "관심사"와 "리소스"를 분리하는 주요 방법입니다.

**동작**

여기서 "동작(Operation)"은 HTTP "메소드" 중 하나를 나타냅니다.

다음 중 하나이며:

- `POST`
- `GET`
- `PUT`
- `DELETE`

...이국적인 것들도 있습니다:

- `OPTIONS`
- `HEAD`
- `PATCH`
- `TRACE`

HTTP 프로토콜에서는 이러한 "메소드"를 하나(또는 이상) 사용하여 각 경로와 통신할 수 있습니다.

---

API를 빌드하는 동안 일반적으로 특정 행동을 수행하기 위해 특정 HTTP 메소드를 사용합니다.

일반적으로 다음을 사용합니다:

- `POST` : 데이터를 생성하기 위해.
- `GET` : 데이터를 읽기 위해.
- `PUT` : 데이터를 업데이트하기 위해.
- `DELETE` : 데이터를 삭제하기 위해.

그래서 OpenAPI에서는 각 HTTP 메소드들을 "동작"이라 부릅니다.

이제부터 우리는 메소드를 "**동작**"이라고도 부를겁니다.

***경로 동작 데코레이터* 정의**

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

`@app.get("/")` 은 **FastAPI**에게 바로 아래에 있는 함수가 다음으로 이동하는 요청을 처리한다는 것을 알려줍니다.

- 경로 `/`
- `get` 동작 사용

!!! info " `@decorator` 정보" 이 `@something` 문법은 파이썬에서 "데코레이터"라 부릅니다.

함수 맨 위에 놓습니다. 마치 예쁜 장식용(Decorative) 모자처럼(개인적으로 이 용어가 여기서 유래한거 같습니다).

"데코레이터" 아래 있는 함수를 받고 그걸 이용해 무언가 합니다.

우리의 경우, 이 데코레이터는 **FastAPI**에게 아래 함수가 **경로** `/`에 해당하는 `get` **동작** 하라고 알려줍니다.

이것이 "**경로 동작 데코레이터**"입니다.

다른 동작도 쓸 수 있습니다:

- `@app.post()`
- `@app.put()`
- `@app.delete()`

이국적인 것들도 있습니다:

- `@app.options()`
- `@app.head()`
- `@app.patch()`
- `@app.trace()`

!!! tip "팁" 각 동작(HTTP 메소드)을 원하는 대로 사용해도 됩니다.

```
**FastAPI**는 특정 의미를 강제하지 않습니다.

여기서 정보는 지침서일뿐 요구사항이 아닙니다.

예를 들어 GraphQL을 사용할때 일반적으로 `POST` 동작만 사용하여 모든 행동을 수행합니다.
```

### 4 단계: 경로 동작 함수 정의

다음은 우리의 "**경로 동작 함수**"입니다:

- **경로**: 는 `/` 입니다.
- **동작**: 은 `get` 입니다.
- **함수**: 는 "데코레이터" 아래에 있는 함수입니다 ( `@app.get("/")` 아래).

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

이것은 파이썬 함수입니다.

`GET` 동작을 사용하여 URL " `/` "에 대한 요청을 받을 때마다 **FastAPI**에 의해 호출됩니다.

위의 경우 `async` 함수입니다.

---

`async def` 대신 일반 함수로 정의할 수 있습니다:

```
{!../../../docs_src/first_steps/tutorial003.py!}
```

!!! note 참고 차이점을 모르겠다면 Async: *"In a hurry?"*{.internal-link target=_blank}을 확인하세요.

### 5 단계: 콘텐츠 반환

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

`dict` , `list` , 단일값을 가진 `str` , `int` 등을 반환할 수 있습니다.

Pydantic 모델을 반환할 수도 있습니다(나중에 더 자세히 살펴봅니다).

JSON으로 자동 변환되는 객체들과 모델들이 많이 있습니다(ORM 등을 포함해서요). 가장 마음에 드는 것을 사용하세요, 이미 지원되고 있을 겁니다.

## 요약

- `FastAPI` 임포트.
- `app` 인스턴스 생성.
- ( `@app.get("/")` 처럼) **경로 동작 데코레이터** 작성.
- (위에 있는 `def root(): ...` 처럼) **경로 동작 함수** 작성.
- ( `uvicorn main:app --reload` 처럼) 개발 서버 실행.