Prompt-Powered Kickstart

Building a Beginner's Toolkit for FastAPI

Moringa AI Capstone Project • 2026

Beginner-friendly · Hands-on · AI-assisted

FastAPI is a modern, high-performance web framework for building APIs with Python. This toolkit walks you through everything — from installation to your first running API — using GenAI prompts as your learning co-pilot.

Table of Contents

1. Title & Objective

Prompt-Powered Kickstart: Building a Beginner's Toolkit for FastAPI

This guide was created as part of the Moringa AI Capstone Project. The goal is to use Generative AI prompts as a learning companion to explore, understand, and build with FastAPI — a cutting-edge Python web framework.

Technology Chosen

- Framework: **FastAPI** — a modern, fast (high-performance) web framework for building APIs with Python.
- Language: **Python 3.8+**

Why FastAPI?

- It is one of the fastest Python frameworks available — comparable to Node.js and Go.
- Automatic interactive docs (Swagger UI & ReDoc) are generated out of the box.
- Built-in data validation using Python type hints and Pydantic.
- Strong async support via Python's `async/await`.
- Huge demand in industry for building REST APIs and microservices.

End Goal

By the end of this guide you will have a locally running FastAPI application that exposes a `GET /` root route and a `GET /items/{item_id}` parameterized route, verifiable through the browser and the built-in Swagger docs at `http://127.0.0.1:8000/docs`.

2. Quick Summary of FastAPI

What is FastAPI?

FastAPI is an open-source Python web framework created by Sebastián Ramírez in 2018. It is built on top of **Starlette** (for the web parts) and **Pydantic** (for data validation). It follows the ASGI standard, enabling high-concurrency workloads through Python's async capabilities.

Where is it Used?

- Building REST APIs and microservices.
- Machine learning model serving (e.g., wrapping a trained ML model in an API endpoint).
- Backend services for web and mobile applications.
- Data pipeline interfaces and internal tooling.

Real-World Example

Uber Eats and many data-science teams use FastAPI to expose machine-learning predictions as REST endpoints. Its automatic Swagger documentation makes it easy for frontend teams and QA engineers to test APIs without writing extra code.

■ ***Tip:*** *FastAPI generates interactive API documentation automatically. Just run your app and visit* `/docs` *— no extra setup required.*

3. System Requirements

Requirement	Details
OS	Windows 10/11, macOS 10.15+, or Linux (Ubuntu 20.04+)
Python	3.8 or higher (3.11 recommended). Check: python --version
pip	Comes bundled with Python 3.4+
Code Editor	VS Code (recommended) with the Python extension
Terminal	Any terminal — PowerShell, bash, zsh, Command Prompt
Internet	Required to install packages via pip

4. Installation & Setup

Step 1 — Verify Python is Installed

```
# Check Python version
python --version

# Expected output:
# Python 3.11.x
```

Step 2 — Create a Project Folder

```
mkdir fastapi-toolkit
cd fastapi-toolkit
```

Step 3 — Create and Activate a Virtual Environment

A virtual environment keeps your project dependencies isolated from other Python projects.

```
# Create virtual environment
python -m venv venv

# Activate — macOS/Linux:
source venv/bin/activate

# Activate — Windows:
venv\Scripts\activate

# Your prompt should now show (venv)
```

Step 4 — Install FastAPI and Uvicorn

uvicorn is the ASGI server that runs your FastAPI application.

```
pip install fastapi uvicorn

# Verify installation
pip show fastapi
pip show uvicorn
```

■ ***Tip:*** *Use* `pip install 'fastapi[all]'` *to also install optional dependencies including* `python-multipart` *for file uploads and* `httpx` *for testing.*

Step 5 — Create Your Main File

```
# Create the main application file
touch main.py      # macOS/Linux
# OR
```

```
   echo. > main.py    # Windows
```

5. Minimal Working Example

What This Example Does

The example below creates a FastAPI application with two routes: a root endpoint that returns a welcome message, and a dynamic `/items/{item_id}` endpoint that accepts a path parameter and an optional query parameter. This is the FastAPI equivalent of a 'Hello World' that also demonstrates core framework features.

main.py — Full Code

```
from fastapi import FastAPI

# Create an instance of the FastAPI application
app = FastAPI()

# Root endpoint — returns a welcome message
@app.get('/')
def read_root():
    return {'message': 'Hello, FastAPI World!'}

# Dynamic endpoint — accepts item_id as a path parameter
# and an optional 'q' query parameter
@app.get('/items/{item_id}')
def read_item(item_id: int, q: str = None):
    return {'item_id': item_id, 'query': q}
```

Running the Application

```
# Run the server with live reload enabled
uvicorn main:app --reload

# Expected terminal output:
# INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
# INFO:     Started reloader process [xxxxx]
# INFO:     Started server process [xxxxx]
# INFO:     Waiting for application startup.
# INFO:     Application startup complete.
```

Testing the API

Endpoint	URL	Expected Response
Root	http://127.0.0.1:8000/	{"message": "Hello, FastAPI World!"}
Item by ID	http://127.0.0.1:8000/items/5	{"item_id": 5, "query": null}

Item + query	http://127.0.0.1:8000/items/5?q=test	{"item_id": 5, "query": "test"}
Swagger Docs	http://127.0.0.1:8000/docs	Interactive API documentation UI
ReDoc	http://127.0.0.1:8000/redoc	Alternative documentation UI

6. AI Prompt Journal

The following prompts were used during this project on the GenAI learning platform. Each entry logs what was asked, what the AI returned, and a reflection on its usefulness.

Prompt 1	*What is FastAPI and how does it differ from Flask and Django REST Framework?*
AI Response Summary:	The AI gave a structured comparison table covering speed, async support, auto-documentation, and use cases. It clearly explained that FastAPI is async-first and uses Python type hints for validation — something Flask does not do natively.
Evaluation:	*Very helpful. The comparison table made it easy to justify choosing FastAPI for this project.*

Prompt 2	*Give me a step-by-step guide to install FastAPI and run my first application on Windows.*
AI Response Summary:	The AI walked through creating a virtual environment, installing FastAPI and uvicorn, writing a minimal main.py, and running uvicorn main:app --reload. It also warned about activating venv correctly on Windows (Scripts vs bin).
Evaluation:	*Excellent. Saved significant time — the Windows activation path difference was a common pitfall caught early.*

Prompt 3	*Explain how path parameters and query parameters work in FastAPI with a code example.*
AI Response Summary:	The AI produced a working code snippet showing both parameter types in one endpoint and explained that FastAPI automatically parses and validates them using Python type hints.
Evaluation:	*Very helpful. The annotated code made the concept immediately practical.*

Prompt 4	*I got a 422 Unprocessable Entity error in FastAPI — what does this mean and how do I fix it?*
AI Response Summary:	The AI explained that 422 errors occur when request data fails Pydantic validation, and showed how to read the error detail field in the JSON response to identify the offending field.
Evaluation:	*Helpful. Knowing where to look in the response body saved debugging time.*

Prompt 5	*What is the difference between uvicorn main:app and uvicorn main:app --reload?*
AI Response Summary:	The AI explained that --reload enables hot-reloading on file save (development only) and should never be used in production. It recommended gunicorn with uvicorn workers for production.
Evaluation:	*Useful context for understanding the development vs production distinction.*

7. Common Issues & Fixes

The following errors are commonly encountered when getting started with FastAPI.

Error	Cause	Fix
`ModuleNotFoundError: No module named 'fastapi'`	FastAPI not installed or virtual environment not activated.	Run: pip install fastapi uvicorn. Ensure venv is activated (check for (venv) in prompt).
`ModuleNotFoundError: No module named 'uvicorn'`	Uvicorn not installed.	Run: pip install uvicorn
`Address already in use (port 8000)`	Another process is using port 8000.	Run on a different port: uvicorn main:app --reload --port 8001
`422 Unprocessable Entity`	Request data failed Pydantic validation (e.g., sent a string where int expected).	Check the 'detail' field in the JSON error response. Ensure your request matches the expected types.
`404 Not Found`	Requested a route that does not exist.	Double-check the URL path. Visit /docs to see all available routes.
`ImportError: cannot import name 'FastAPI' from 'fastapi'`	Likely a corrupted or incomplete installation.	Run: pip uninstall fastapi && pip install fastapi
`uvicorn: command not found`	uvicorn is not on the PATH or venv is not activated.	Activate your venv first. Alternatively use: python -m uvicorn main:app --reload

■ ***Tip:*** *Always activate your virtual environment before running any pip or uvicorn commands. If in doubt, deactivate and reactivate:* `deactivate` *then* `source venv/bin/activate`.

8. References

Official Documentation

- **FastAPI Official Docs**
  https://fastapi.tiangolo.com
- **Uvicorn Documentation**
  https://www.uvicorn.org
- **Pydantic Documentation**
  https://docs.pydantic.dev
- **Starlette Documentation**
  https://www.starlette.io

Video Tutorials

- **FastAPI Full Course — freeCodeCamp (YouTube)**
  https://www.youtube.com/watch?v=0sOvCWFmrtA
- **FastAPI Crash Course — Traversy Media (YouTube)**
  https://www.youtube.com/watch?v=tLKKmouUams

Blog Posts & Articles

- **TestDriven.io — FastAPI Tutorial**
  https://testdriven.io/blog/fastapi-crud/
- **Real Python — FastAPI Introduction**
  https://realpython.com/fastapi-python-web-apis/

Community & Support

- **FastAPI GitHub Issues**
  https://github.com/tiangolo/fastapi/issues
- **FastAPI on Stack Overflow**
  https://stackoverflow.com/questions/tagged/fastapi
- **FastAPI Discord Community**
  https://discord.gg/VQjSZaeJmf
