



FastAPI Training Workshop

Friday, June 11 2021

*Delivered by: Bobby Drake
Adam Hill*

Plan for the day

- 10:15 - Greetings and good morning
- 10:30 - Intro to APIs and the FastAPI framework
- 11:00 - 13:00 - Practical demonstration and coding workshop 1
- 13:00 - 14:00 - LUNCH
- 14:00 - 16:00 - Practical demonstration and coding workshop 2
- 16:00 - 16:30 - Infra CCC
- 16:30+ - Informal coding workshop extension/social drinks

***All times are CEST**

Topics

- What is an API?
- What is FastAPI?
- Making your first API
- Data Models & Pydantic
- Why is stateless important!

What is an API?


- API stands for Application Programming Interface; this defines the interactions between different types of software/hardware.
- We (should) build RESTful APIs; these are APIs that conform to the constraints of REST architectural style.
 - REST = Representational State Transfer
- We use APIs as a mediator between users or clients and our resources or web services e.g.:
 - Client gets a data science model or input on demand
 - DataFlow calls a model prediction on a schedule and saves the output to a database



What makes an API RESTful?


REST is an architectural set of rules and principles to follow:

- You should get a piece of data (**a resource**) when you link to a specific URL (**a request**)
- A request is made up of:
 - The endpoint, e.g. `https://www.hal24k.com/employees/:userid`
 - The method, e.g. GET, POST, PUT, PATCH, DELETE
 - The headers
 - The data/body.
- Stateless: calls can be made independently of each other
- When data can be cached it should be

What is Fast API?


FastAPI


tiangolo/fastapi
0.65.2 31.9k 2.2k

FastAPI
FastAPI
Languages >
Features
FastAPI People
Python Types Intro
Tutorial - User Guide >
Advanced User Guide >
Concurrency and async / await
Deployment >
Project Generation - Template
Alternatives, Inspiration and Comparisons
History, Design and Future
External Links and Articles
Benchmarks
Help FastAPI - Get Help
Development - Contributing
Release Notes


FastAPI

FastAPI
FastAPI framework, high performance, easy to learn, fast to code, ready for production
Test passing coverage 100% pypi package v0.65.2

Table of contents
Sponsors
Opinions
Typer, the FastAPI of CLIs
Requirements
Installation
Example
 Create it
 Run it
 Check it
 Interactive API docs
 Alternative API docs
Example upgrade
 Interactive API docs upgrade
 Alternative API docs upgrade
Recap
Performance
Optional Dependencies
License

Documentation: <https://fastapi.tiangolo.com>
Source Code: <https://github.com/tiangolo/fastapi>

What is Fast API?

Modern, fast, Python web framework with a lot of traction in the community.

- Asynchronous (async/await)
- ASGI servers
- OpenAPI documentation
- Data classes
- Pydantic classes
- pytest



FastAPI

FastAPI Coding Exercise

Bad REST example

Demo:

What happens when when you scale an API with
memory!

Pydantic

Pydantic is a data validation and settings management library using python type annotations.

Type hints were added in Python 3.5. They are not enforced at runtime, and only are used for third-party tools such as IDEs and linters.

Pydantic enforces type hints at runtime, and provides user friendly errors when data is invalid.

```
from datetime import datetime
from typing import List, Optional
from pydantic import BaseModel

class User(BaseModel):
    id: int
    name = 'John Doe'
    signup_ts: Optional[datetime] = None
    friends: List[int] = []

external_data = {
    'id': '123',
    'signup_ts': '2019-06-01 12:22',
    'friends': [1, 2, '3'],
}
user = User(**external_data)
print(user.id)
#> 123
print(repr(user.signup_ts))
#> datetime.datetime(2019, 6, 1, 12, 22)
print(user.friends)
#> [1, 2, 3]
print(user.dict())
"""
{
    'id': 123,
    'signup_ts': datetime.datetime(2019, 6, 1, 12, 22),
    'friends': [1, 2, 3],
    'name': 'John Doe',
}
"""
```

Pydantic Models

The primary means of defining objects in *pydantic* is via models (models are simply classes which inherit from `BaseModel`).

You can think of models as similar to types in strictly typed languages, or as the requirements of a single endpoint in an API.

Untrusted data can be passed to a model, and after parsing and validation *pydantic* guarantees that the fields of the resultant model instance will conform to the field types defined on the model.

```
In [1]: from pydantic import BaseModel
...:
...: class User(BaseModel):
...:     id: int
...:     name = 'Jane Doe'
...:

In [2]: user = User(id='123')

In [3]: assert user.id == 123

In [4]: assert user.name == 'Jane Doe'
```

Recursive Model

More complex hierarchical data structures can be defined using models themselves as types in annotations.

```
In [1]: from pydantic import BaseModel
...:
...: class User(BaseModel):
...:     id: int
...:     name = 'Jane Doe'
...:
In [2]: user = User(id='123')
In [3]: assert user.id == 123
In [4]: assert user.name == 'Jane Doe'
...:
...:
...: class Foo(BaseModel):
...:     count: int
...:     size: float = None
...:
...:
...: class Bar(BaseModel):
...:     apple = 'x'
...:     banana = 'y'
...:
...:
...: class Spam(BaseModel):
...:     foo: Foo
...:     bars: List[Bar]
...:
...:
...: m = Spam(foo={'count': 4}, bars=[{'apple': 'x1'}, {'apple': 'x2'}])
...: print(m)
foo=Foo(count=4, size=None) bars=[Bar(apple='x1', banana='y'), Bar(apple='x2', banana='y')]
In [6]: print(m.dict())
{'foo': {'count': 4, 'size': None}, 'bars': [{'apple': 'x1', 'banana': 'y'}, {'apple': 'x2', 'banana': 'y'}]}
```

Field Types

Where possible *pydantic* uses [standard library types](#) to define fields, thus smoothing the learning curve. For many useful applications, however, no standard library type exists, so *pydantic* implements [many commonly used types](#).

If no existing type suits your purpose you can also implement your [own pydantic-compatible types](#) with custom properties and validation.

```
class AddressModel(BaseModel):  
    ... street: str  
    ... house_number: str  
    ... city: str  
    ... postal_code: str  
    ... country: str  
  
You, a day ago • upc  
  
You, seconds ago | 1 author (You)  
class PersonModel(BaseModel):  
    ... id: uuid.UUID = Field(default_factory=uuid.uuid4)  
    ... name: str  
    ... age: int  
    ... address: AddressModel = Field(default=AddressModel())
```

Validators

Custom validation and complex relationships between objects can be achieved using the validator decorator.

```
In [7]: from pydantic import BaseModel, ValidationError, validator
...:
...: class UserModel(BaseModel):
...:     name: str
...:     username: str
...:     password1: str
...:     password2: str
...:
...:     @validator('name')
...:     def name_must_contain_space(cls, v):
...:         if ' ' not in v:
...:             raise ValueError('must contain a space')
...:         return v.title()
...:
...:     @validator('password2')
...:     def passwords_match(cls, v, values, **kwargs):
...:         if 'password1' in values and v != values['password1']:
...:             raise ValueError('passwords do not match')
...:         return v
...:
...:     @validator('username')
...:     def username_alphanumeric(cls, v):
...:         assert v.isalnum(), 'must be alphanumeric'
...:         return v
...:
...:     user = UserModel(
...:         name='samuel colvin',
...:         username='scolvin',
...:         password1='zxcvbn',
...:         password2='zxcvbn',
...:     )
...:     print(user)
name='Samuel Colvin' username='scolvin' password1='zxcvbn' password2='zxcvbn'

In [8]: try:
...:     UserModel(
...:         name='samuel',
...:         username='scolvin',
...:         password1='zxcvbn',
...:         password2='zxcvbn2',
...:     )
...: except ValidationError as e:
...:     print(e)
...:
2 validation errors for UserModel
name
  must contain a space (type=value_error)
password2
  passwords do not match (type=value_error)
```

Pipenv

Pipenv is similar to existing tools, if you have some familiarity with Ruby's Bundler or Node's Npm. Pipenv is both a package and virtual environment management tool that uses the *Pipfile* and *Pipfile.lock* files to achieve these goals.

The *Pipfile* file is intended to specify packages requirements for your Python application or library, both to development and execution.

The *Pipfile.lock* is intended to specify, based on the packages present in *Pipfile*, which specific version of those should be used, avoiding the risks of automatically upgrading packages that depend upon each other and breaking your project dependency tree.

In production, install packages from the Piplock with the `pipenv install --ignore-pipfile`

Or `pipenv install --system`

Other similar package managers for Python include Conda and Poetry.

Pydantic

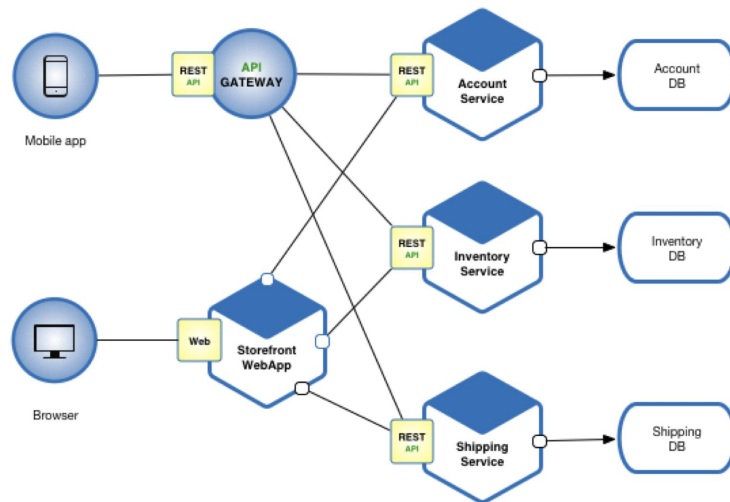
Pydantic Coding Exercise

Advanced API Design

Micro Services, Domain Driven Design, Hexagonal
Architecture and Test Driven Development

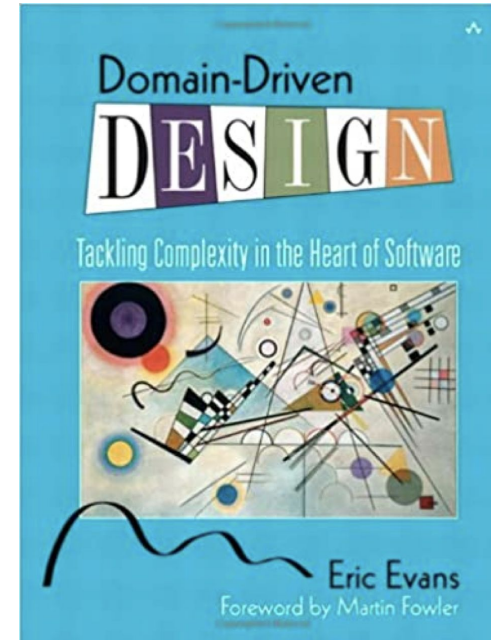
Micro Services

- Micro-services - also known as the micro-service architecture - is an architectural style that structures an application as a collection of services that are:
 - Highly maintainable and testable
 - Loosely coupled
 - Independently deployable
 - Organised around business capabilities
 - Owned by a small team

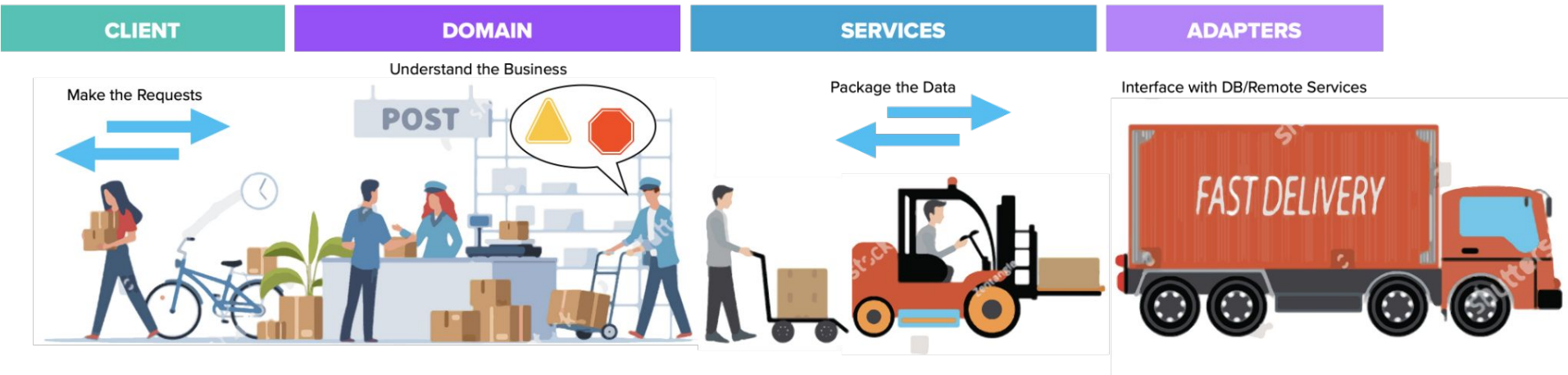


Domain Driven Design

Domain-driven design (DDD) is the concept that the structure and language of software code (class names, class methods, class variables) should match the business domain. For example, if a software processes loan applications, it might have classes such as `LoanApplication` and `Customer`, and methods such as `AcceptOffer` and `Withdraw`.



Postal Domain



What are the Benefits of DDD?

- **Patterns:** DDD gives software developers the principles and patterns to solve tough problems in software and, at times, in business.
- **Business Logic:** DDD creates business logic by explaining requirements from a domain perspective. Conceptualising the system software in terms of the business domain, reducing the risk of misunderstanding between the domain experts and the development team.
- **Communication:** DDD simplifies one factor that is complex in every professional relationship: communication. DDD allows developers, domain experts, DBAs, business owners, and (most importantly) clients to communicate effectively with each other in order to solve problems.
- **Successful History:** DDD a history of success with complex projects, aligning with the experience of software developers and software applications developed.
- **Testable Code:** DDD is an effective way to write clear, testable code, and provides principles and patterns to solve difficult problems. Clients are typically not interested in how the code works or even how the software is built, but they are interested in a product that works.

Further Reading

- Micro-Service Architecture Concepts:
<https://hal24k.atlassian.net/wiki/spaces/CD/pages/2355331132/Micro-Service+Architecture+Concepts>
- Domain Driven Design: <https://www.bol.com/nl/f/domain-driven-design/9200000002151217/>
- Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Micro-Services: <https://www.amazon.com/Architecture-Patterns-Python-Domain-Driven-Microservices/dp/1492052205>
- DDD Resources:
<https://www.domainlanguage.com/ddd/>
- Microsoft - Design a DDD-oriented micro-service:
<https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice>
- Hexagonal Architecture in Python - <https://douwevandermeij.medium.com/hexagonal-architecture-in-python-7468c2606b63>



Thank you

