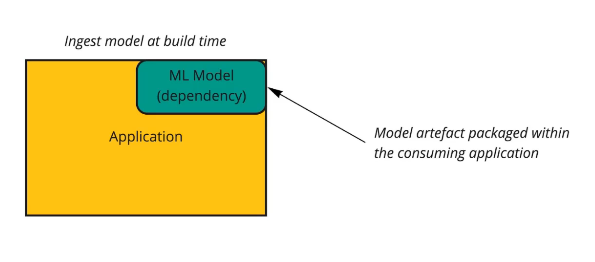
# Understanding the architecture of the API

1. ML system architectures

* Model embedded in application
* Served via a dedicated service
* Model published as data (streaming)
* Batch prediction (offline process)

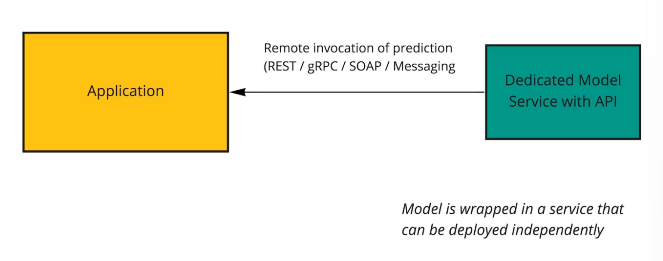
1. Architecture 1: embedded

* Pre-trained: yes
* Predict on the fly: yes
* Variations: embedded on mobile device (e.g., Core ML), running on the browser (tensorflow.js)



1. Architecture 2: dedicated model API

* Pre-trained: yes
* Predict on the fly: yes
* Variations: many. See also architecture 3

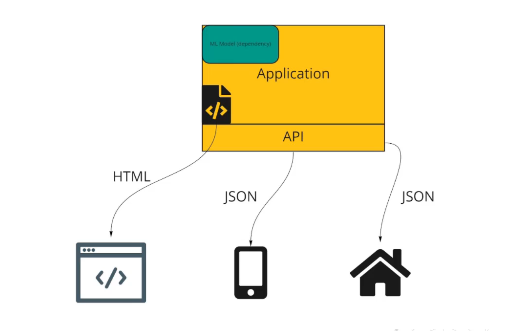


1. Specifics of the sample project

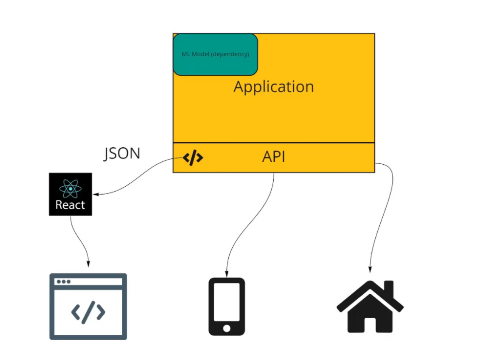
* Representative state transfer (REST) API loosely
* Can be used as an example for either embedded. / dedicated architecture

1. How can our API can be consumed?

* Web browser
* Mobile devices
* IoT
* Other applications

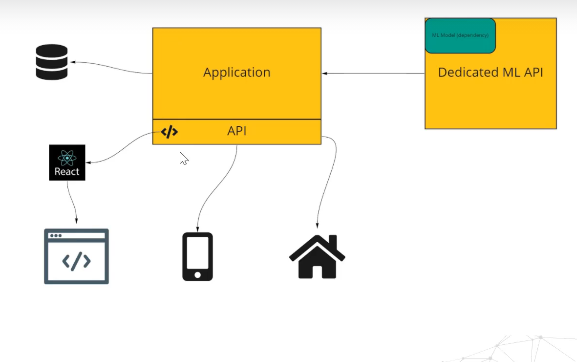


1. Modern frontend approaches



1. Dedicated ML API + microservices

* Refer to section 3



# Introduction to FastAPI

1. Web frameworks

* Provide tools to handle & automate standard tasks when building web apps
* Session mgmt.
* Templating
* Database access
* Much more

1. Python web frameworks

* Flask
* Django
* FastAPI (newer, rapidly gaining momentum)

1. Features of FastAPI

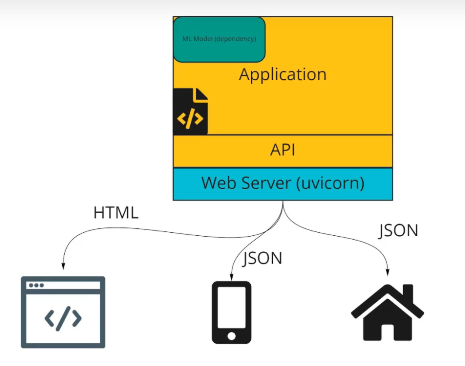
* Leverages python async capabilities -> fast
* Validation with type hints and pydantic
* Automatic documentation
* Dependency injection
* Etc.

# Using schemas in our API

# The Uvicorn web server

1. Production components: web server

* For a prod deployment, we require a web server
* Although we tend to use the terms interchangeably, technically, the server deals with handling income requests and outgoing messages
* A server implements a server gateway interface

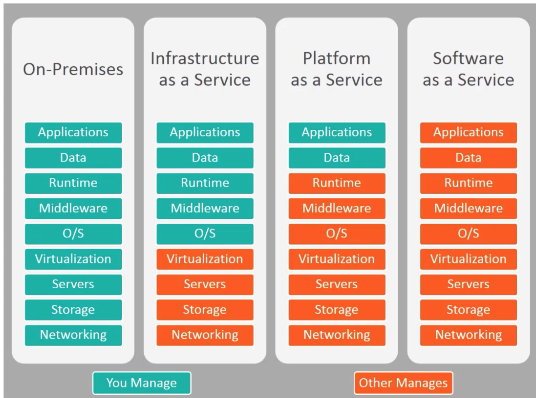


1. WSGI & ASGI

* Historically, python web frameworks have used the Web Server Gateway Interface (WSGI) introduced in PEP 333 (and revised in PEP 333)
* With the introduction of python’s asyncio library it was possible to go beyond the constraints of WSGI. Hence the Asynchronous Server Gateway Interface (ASGI) was born
* Uvicorn is an implementation of this interface

# PaaS

1. PaaS – understanding the range of possibilities



1. PaaS – Pros and Cons

* Pros:
* Simple to setup, maintain and deploy
* Easy to scale to moderate size
* Allows devs to focus on apps
* Easy creation of dev/test env
* Cons:
* Hard/impossible to scale to a very large size
* Tends to be more expensive than IaaS
* Vulnerable to PaaS downtime
* Limitations on config

# Railway App as our PaaS

1. Example PaaS providers

* AWS Elastic Beanstalk
* Microsoft Azure App Service
* Heroku (no longer free)
* Deta
* Vercel
* Digital Ocean App Platform
* PythonAnywhere

1. Why Railway app?

* Easy to use
* 500 hours for free (no credit card, no risk)
* Supports large dependency sizes on free tier
* Nice CLI so we can deploy in CI pipelines
* Works with Docker (not docker-compose)
* Good github integration
* Better than the other options