APIs and More

Agenda

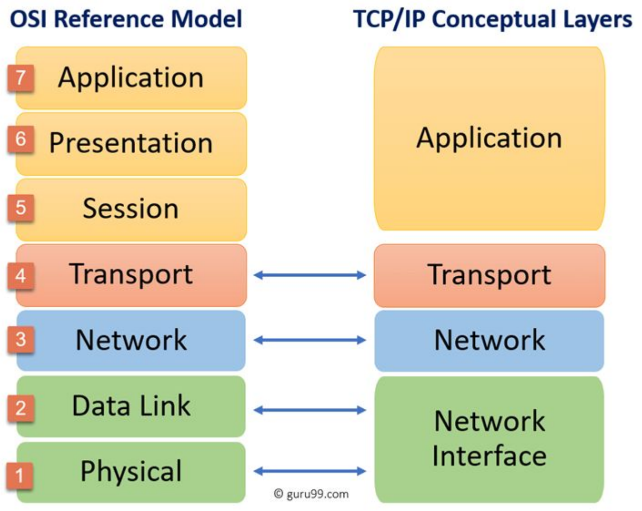
01 Networks Quickstart
02 APIs
03 Example
04 Flask
05 Resources

Learning objectives

- Basic Computer Network Knowledge
- Design a Restful API
- Flask Basics

Networks

Computer Networks



Network Layer

- IP
- Identification and Addressing
- IPv4 - 192.0.2.1
- IPv6 - 2001:0db8:0000:0000:0000:8a2e:0370:7334

Transport Layer

- UDP (User Datagram Protocol)
    - Connectionless
    - Fast
- TCP (Transmission Control Protocol)
    - Established connection - Three Way handshake
    - Extensive error checking and acknowledgment of data
    - Guarantee delivery of data to the destination router

Transport Layer

- UDP
  - Video conferencing, streaming, DNS, VoIP, Gaming, etc..
- TCP
  - HTTPS, HTTP, SMTP, POP, FTP, etc

Application Layer

- HTTP
- Mail
- File Transfer Protocol
- DNS

HTTP

- Request–response protocol in the client–server model
- Text protocol
- HTTP Methods - GET/POST/OPTIONS/HEAD …
- HTTP Status - 2XX, 3XX, 4XX, 5XX,

Using HTTP

- Postman - Convenient way for making HTTP
- Developer tools
- File Transfer Protocol
- DNS

APIs

API

- API - Application Programming Interface
- A way two or more pieces of software communicate
- Contrast with User Interface

Good API Design

- Easy to Read and work with
- Hard to misuse
- Complete and Concise

REST

- Representational state transfer
- Standard Architectural Style
- Separation of Client and Server
- Stateless
- Resources

Our Goal - Restful CRUD APIs

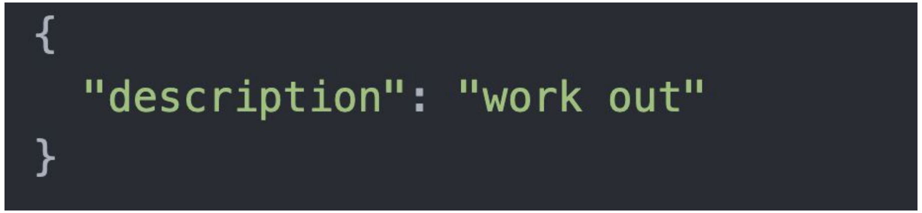
- CRUD - create, read, update, delete

Example: Task Organizer Application

- Build an API for personal organizer application
- Create, Delete, Update and Read functionality
- Task
  - Description
  - Status
  - ID

Create: Post a Tasks

- URL:**<URL>/tasks**
- HTTP METHOD: **POST**
- Request Body:

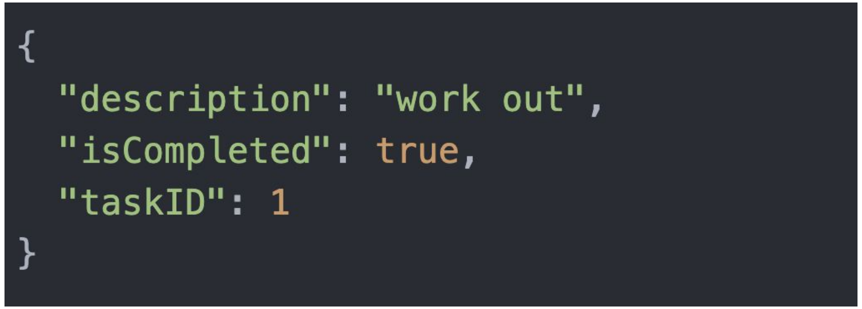


```
{
  "description": "work out"
}
```

- Response - empty response with status code 200

READ: Get a single Tasks

- URL: **<URL>/tasks/1**
- HTTP METHOD: **GET**
- Request Body: **Empy**
- Response: **the target task**



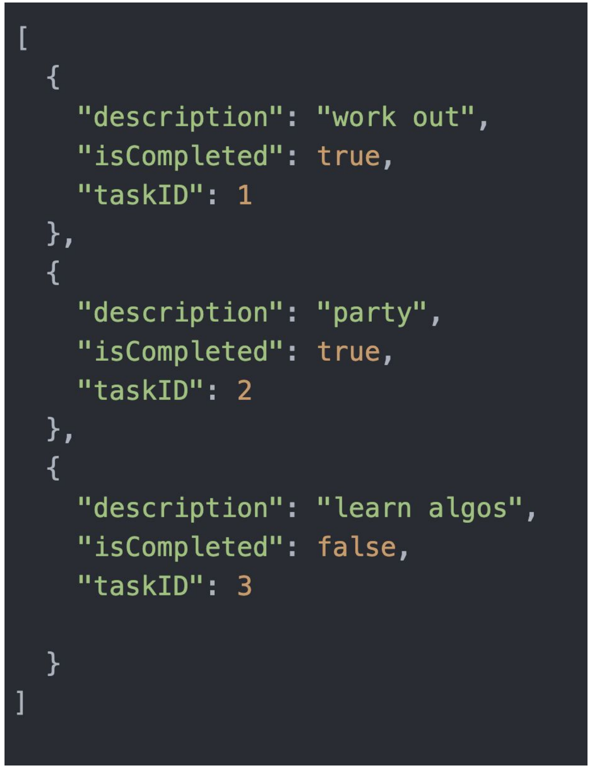
```
{
  "description": "work out",
  "isCompleted": true,
  "taskID": 1
}
```

READ: Get a single Task(NOT FOUND)

- URL: **<URL>/tasks/9999**
- HTTP METHOD: **GET**
- Request Body: **Empty**
- Response: **Status Code 404 Not**

READ: Get All Tasks

- URL: **<URL>/tasks**
- HTTP METHOD: **GET**
- Request Body: **Empty**
- Response: all tasks, status code 200

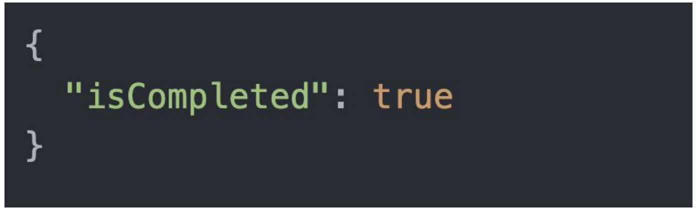


```
[
  {
    "description": "work out",
    "isCompleted": true,
    "taskID": 1
  },
  {
    "description": "party",
    "isCompleted": true,
    "taskID": 2
  },
  {
    "description": "learn algos",
    "isCompleted": false,
    "taskID": 3

  }
]
```

Update: Completion Status of a Task

- URL: myorganizer.com/tasks/2
- HTTP METHOD: PUT
- Request Body:



```
{
  "isCompleted": true
}
```

- Response - empty response with 200 OK status code

Update: Bad Request

- URL: **<URL>/tasks/2**
- HTTP METHOD: **PUT**
- Request Body:

```
{
  "isCompleted": "foncho"
}
```

- Response: **empty with code 400 Bad Request**

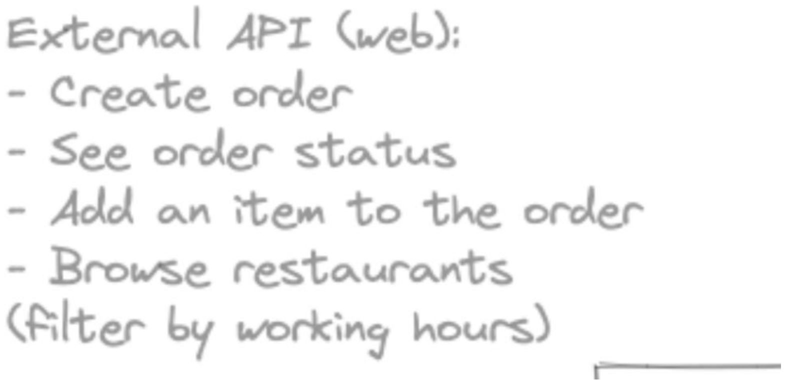
Delete: Remove a task

- URL: **<URL>/tasks/1**
- HTTP METHOD: **DELETE**
- Request Body: **Empty**
- Response: **empty with status code 200**

Designing APIs

Determine Use Cases

- Who is going to use the software
- What goals do they have
- What are the functional requirements



External API (web):
- Create order
- See order status
- Add an item to the order
- Browse restaurants
(filter by working hours)

Design Models

- Example 1: Task with a status completed and a description

```json
{
  "description": "work out",
  "isCompleted": true,
  "taskID": 1
}
```

- Example 2: Order with multiple Cart Items

Draft the API and get feedback

- Draft endpoints
- Communicate with stakeholders and get agreement

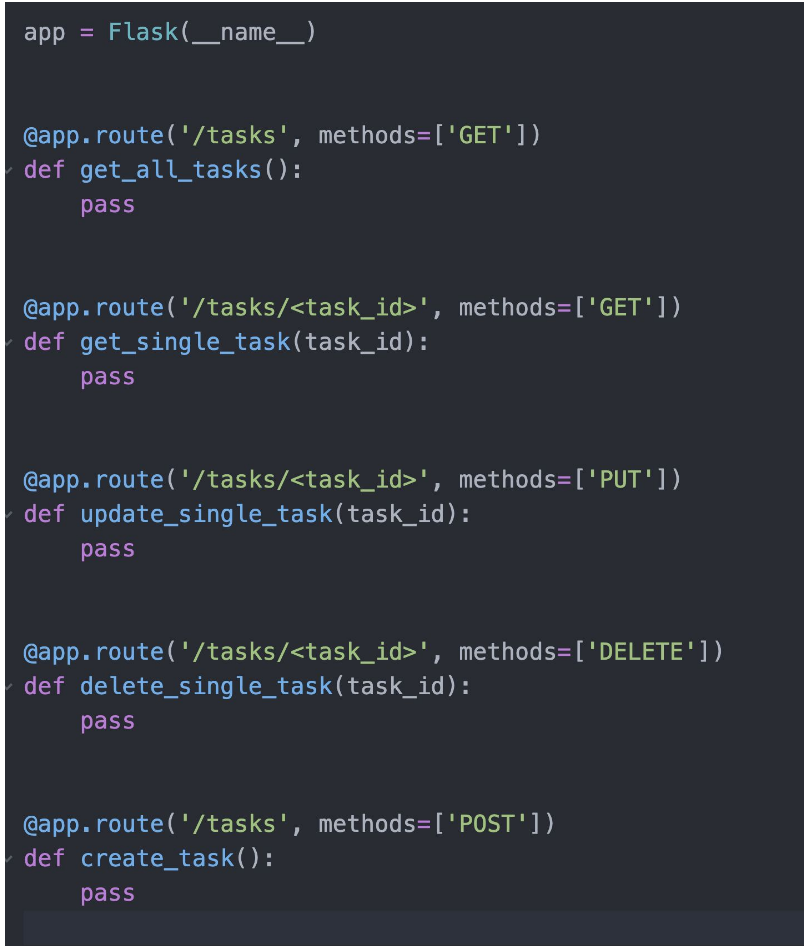
Detach Client and server

- Client and Server can be developed separately
- Stubs
- Mountebank

Flask

Flask

- Lightweight micro framework for web development
- Extension for everything
- Very convenient



```python
app = Flask(__name__)


@app.route('/tasks', methods=['GET'])
def get_all_tasks():
    pass


@app.route('/tasks/<task_id>', methods=['GET'])
def get_single_task(task_id):
    pass


@app.route('/tasks/<task_id>', methods=['PUT'])
def update_single_task(task_id):
    pass


@app.route('/tasks/<task_id>', methods=['DELETE'])
def delete_single_task(task_id):
    pass


@app.route('/tasks', methods=['POST'])
def create_task():
    pass
```

Resources

- Flask Web Development: Developing Web Applications with Python
- Flask Quickstart
- Network Direction