

Web APIs

Software Architecture

Brae Webb

April 20, 2026

Goals

- Review existing networking knowledge.

Goals

- Review existing networking knowledge.
- Understand *URLs*.

Goals

- Review existing networking knowledge.
- Understand *URLs*.
- Understand *HTTP* protocol and methods.

Goals

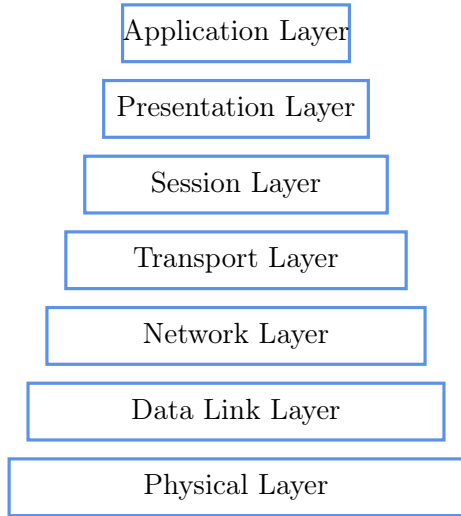
- Review existing networking knowledge.
- Understand *URLs*.
- Understand *HTTP* protocol and methods.
- Understand *RESTful* APIs.

Goals

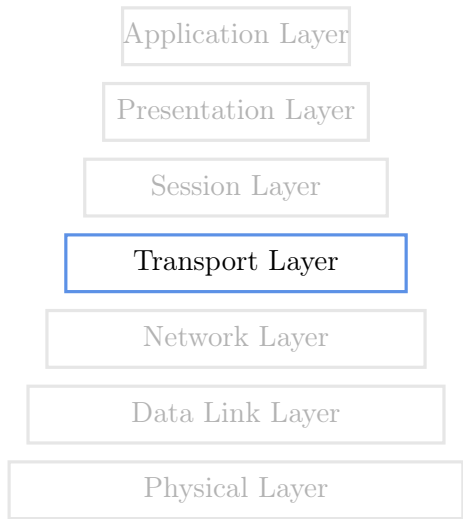
- Review existing networking knowledge.
- Understand *URLs*.
- Understand *HTTP* protocol and methods.
- Understand *RESTful* APIs.
- *Build* a basic RESTful API.

§ Networking

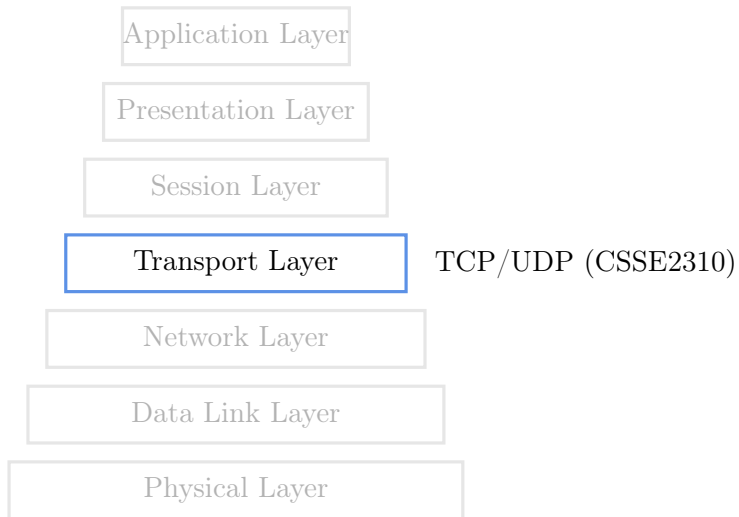
OSI Model



OSI Model



OSI Model



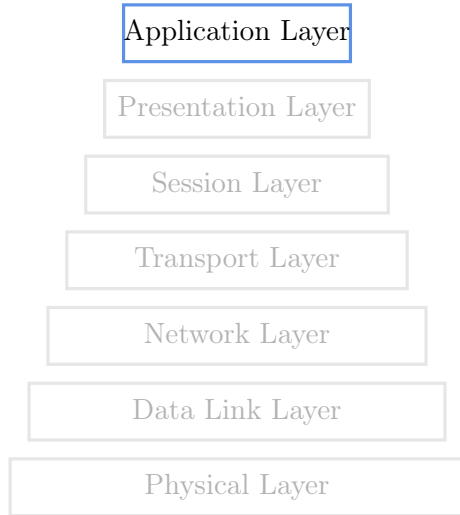
TCP/UDP

Low-level with *minimal abstraction*.

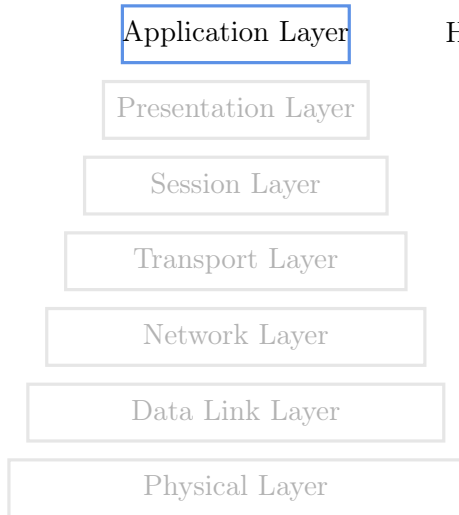
TCP/UDP

Impractical for building web APIs.

OSI Model



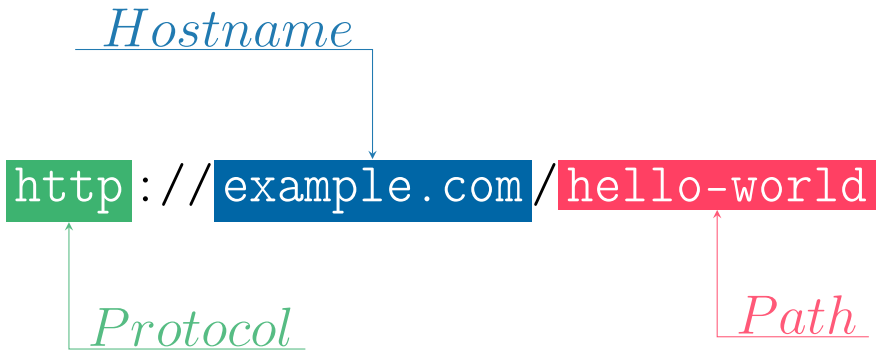
OSI Model

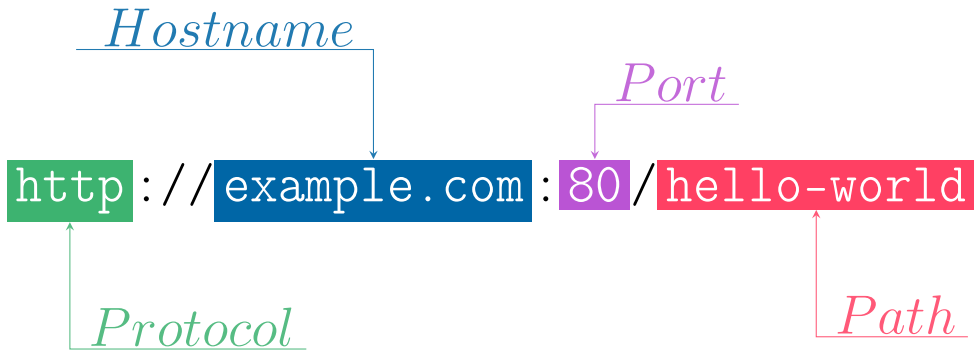


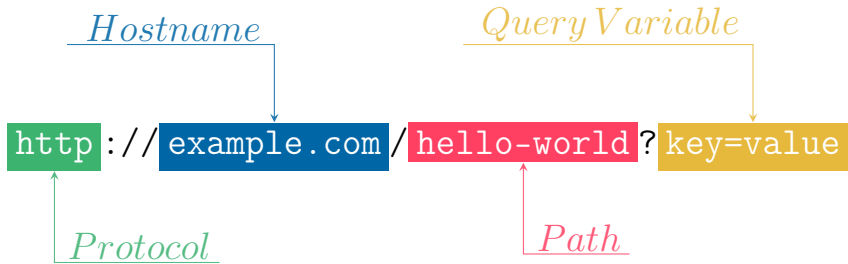
HTTP/HTTPS (CSSE6400)

\S *URLs*

The anatomy of
URLs







§ HTTP

HTTP

A *request-response* abstraction for networking.

HTTP Request

URL An endpoint to send request to.

Method Described later.

Headers Specify type of data, e.g. JSON, HTML, etc.

Body Optional extra data to include.

HTTP Response

- Status Code A number between 100 and 599 giving details about the response.
- Headers Specify type of response data, e.g. JSON, HTML, etc.
- Body Content of the response.

Status Codes

200s Indicate the request was *successful*, 200 is most common.

300s *Redirects* the client to another location.

400s Indicates that the *request was wrong*

e.g. 404 meaning that the request was for something that doesn't exist.

500s Indicates that the *server had a problem* fulfilling the request.

Types of HTTP communication

HTTP Methods

HTTP Methods

GET *Query* for information.

HTTP Methods

GET *Query* for information.

POST *Create* resource.

HTTP Methods

GET *Query* for information.

POST *Create* resource.

PUT *Update* resource.

HTTP Methods

GET *Query* for information.

POST *Create* resource.

PUT *Update* resource.

DELETE *Delete* resource.

§ API Examples

```
» cat app.py
```

```
1  from flask import Flask
```

```
3  app = Flask(__name__)
```

```
5  @app.route("/")
```

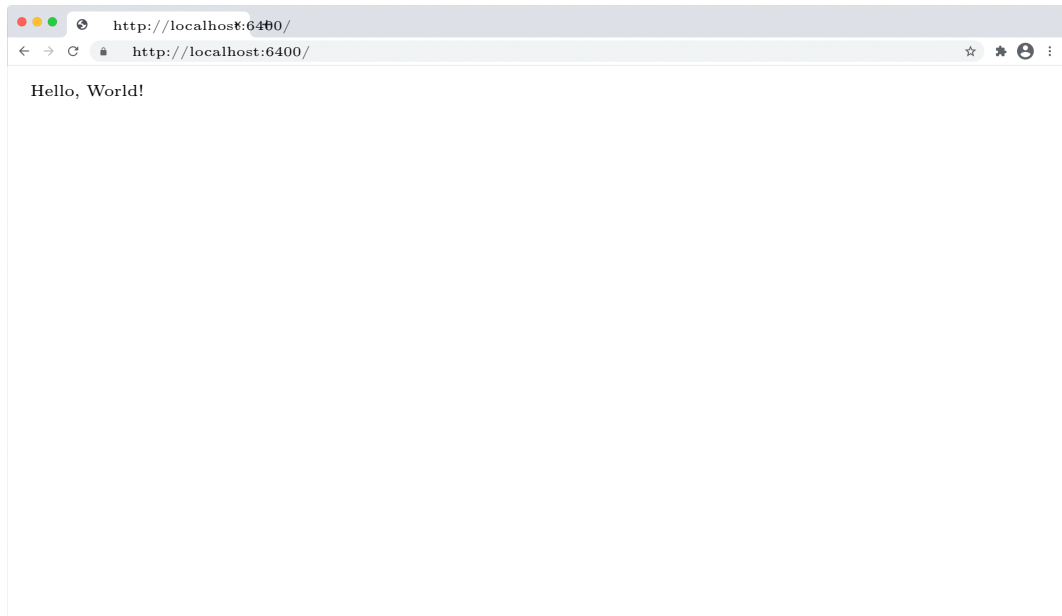
```
6  def hello_world():
```

```
7      return "Hello, World!"
```

```
9  if __name__ == "__main__":
```

```
10     app.run(port=6400)
```

Result




```
» cat app.js
```

```
1  const express = require('express')
2  const app = express()
3  const port = 6400

5  app.get('/', (req, res) => {
6    res.send('Hello, World!')
7  })

9  app.listen(port, () => {
10    console.log(`Example app listening on port ${port}`)
11  })
```

```
» cat app.py
```

```
1  from flask import Flask
```

```
3  app = Flask(__name__)
```

```
5  @app.route("/health")
```

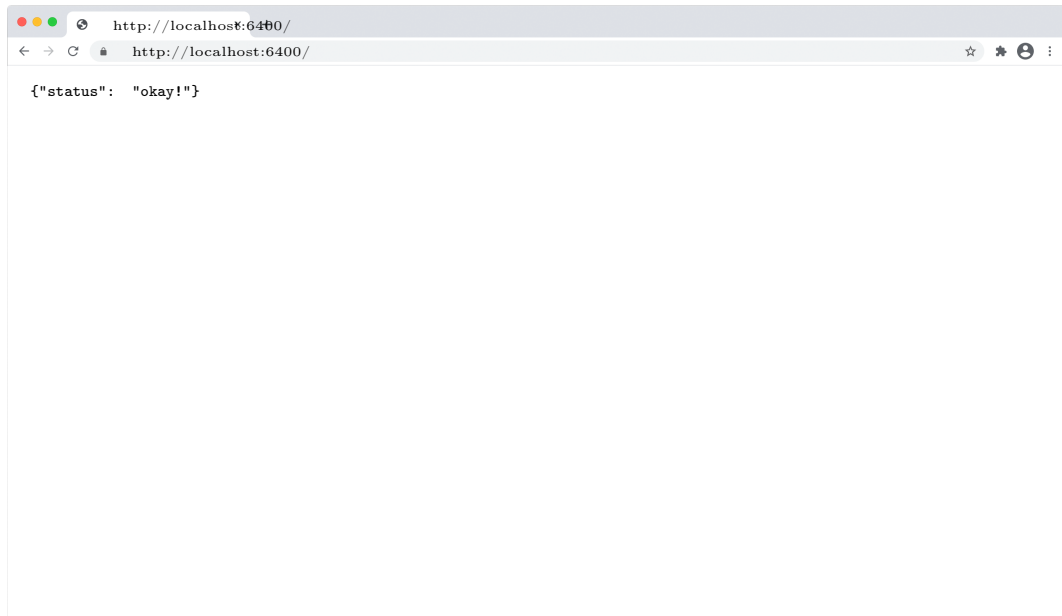
```
6  def hello_world():
```

```
7      return {"status": "okay!"}
```

```
9  if __name__ == "__main__":
```

```
10     app.run(port=6400)
```

Result



```
» cat app.js
```

```
1  const express = require('express')
2  const app = express()
3  const port = 6400

5  app.get('/', (req, res) => {
6      res.send({"status": "okay!"})
7  })

9  app.listen(port, () => {
10     console.log(`Example app listening on port ${port}`)
11 })
```

```
» cat app.py
```

```
1  from flask import Flask
2  from flask import request

4  app = Flask(__name__)

6  @app.route("/echo", methods=["POST"])
7  def hello_world():
8      return request.json.say

10 if __name__ == "__main__":
11     app.run(port=6400)
```

```
1 >>> curl -X POST \  
2 -H "Accept: application/json" \  
3 -H "Content-Type: application/json" \  
4 "http://localhost:6400" \  
5 -d '{  
6     "say" : "Hello, World",  
7 }'  
8 Hello, World
```

```
» cat app.js
```

```
1  const express = require('express')
2  const app = express()
3  const port = 6400

5  app.post('/', express.json(), (req, res) => {
6      res.send(req.body.say)
7  })

9  app.listen(port, () => {
10     console.log(`Example app listening on port ${port}`)
11 })
```