COMP1531

Full-Stack - HTTP Servers

Lecture 4.2

Author(s): Hayden Smith

(Download as PDF)

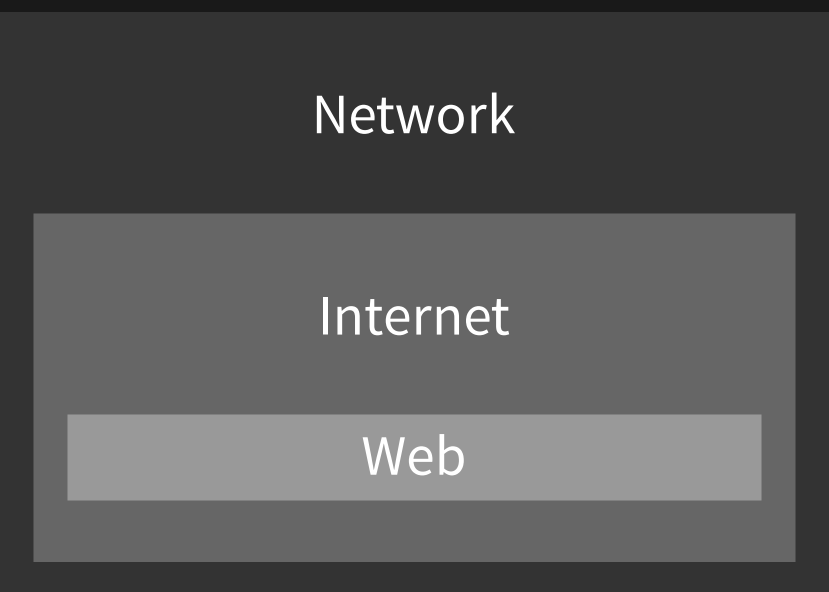
In This Lecture

- **Why? 🤔**
  - Web servers are fundamental part of web-based full-stack software

- **What? 📰**
  - Networks
  - Express Server
  - APIs
  - Crud

Networks

- **Network**: A group of interconnected computers that can communicate
- **Internet:** A global infrastructure for networking computers around the entire world together
- **World Wide Web**: A system of documents and resources linked together, accessible via URLs



If you want to learn more about networking, go and study COMP3331.

🌐 Network Protocols

- Communication over networks must have a certain "structure" so everyone can understand.
- Humans do this all the time - waving, handshakes, clapping. Standard operation procedure that structures how we share info.
- Different "structures" (protocols) are used for different types of communication.

🌐 Network Protocols

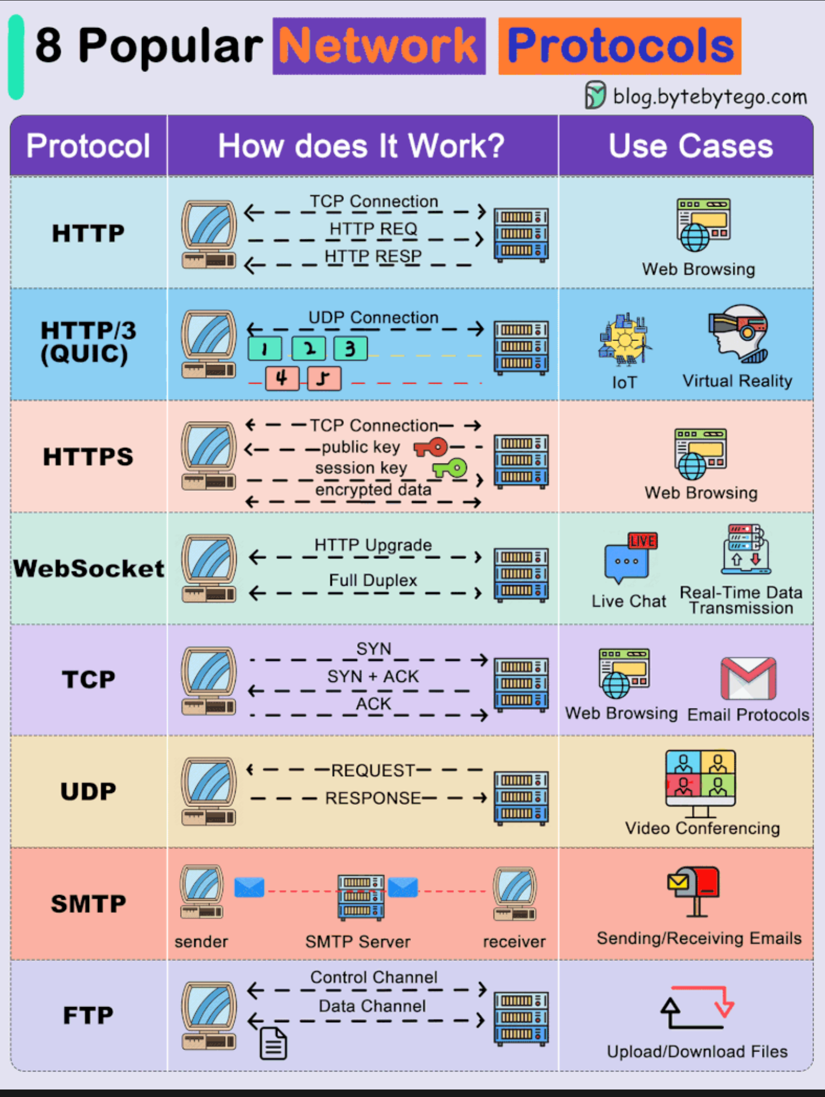
source

HTTP is an example of one of the protocols. It is the protocol of the web. The primary protocol you use to access URLs in your web browser.

Examples

Network Protocols

Examples



HTTP: Hypertext Transfer Protocol

I.E. Protocol for sending and receiving HTML documents (nowadays much more)

Web Browsers (Client)

Web Servers

Web browsers are applications to request and receive HTTP.

🍕 NodeJS Express Server

A very popular `npm` library exists to allow you to run your own HTTP server with NodeJS.

It's called Express Server.

🍕 NodeJS Express Server

```
 1 import express from 'express';
 2
 3 const app = express();
 4 const port = 3000;
 5
 6 app.use(express.text());
 7
 8 app.get('/hello', (req, res) => {
 9   res.send('Hello!');
10 });
11
12 app.get('/whats/up', (req, res) => {
13   res.send(JSON.stringify({
14     value: 'not much',
15   }));
16 });
17
18 app.listen(port, () => {
19   console.log(`Listening on port ${port}`);
20 });
```

express_basic.ts

🍕 NodeJS Express Server

```
1  import express from 'express';
2
3  const app = express();
4  const port = 3000;
5
6  app.use(express.text());
7
8  app.get('/hello', (req, res) => {
9    res.send('Hello!');
10 });
11
12 app.get('/whats/up', (req, res) => {
13   res.send(JSON.stringify({
14     value: 'not much',
15   }));
16 });
17
18 app.listen(port, () => {
19   console.log(`Listening on port ${port}`);
20 });
```

express_basic.ts

This is us importing the express library

🍕 NodeJS Express Server

```
1  import express from 'express';
2
3  const app = express();
4  const port = 3000;
5
6  app.use(express.text());
7
8  app.get('/hello', (req, res) => {
9    res.send('Hello!');
10 });
11
12 app.get('/whats/up', (req, res) => {
13   res.send(JSON.stringify({
14     value: 'not much',
15   }));
16 });
17
18 app.listen(port, () => {
19   console.log(`Listening on port ${port}`);
20 });
```

express_basic.ts

This creates an instance of a server, and we define the network port to run on. A port is essentially one of the roads in and out of a computer's network. There are often 65,000-ish.

🍕 NodeJS Express Server

```
1  import express from 'express';
2
3  const app = express();
4  const port = 3000;
5
6  app.use(express.text());
7
8  app.get('/hello', (req, res) => {
9    res.send('Hello!');
10 });
11
12 app.get('/whats/up', (req, res) => {
13   res.send(JSON.stringify({
14     value: 'not much',
15   }));
16 });
17
18 app.listen(port, () => {
19   console.log(`Listening on port ${port}`);
20 });
```

express_basic.ts

This line is a quirk of `express` that is required in order for the data of many requests to be interpreted.

🍕 NodeJS Express Server

```
 1 import express from 'express';
 2
 3 const app = express();
 4 const port = 3000;
 5
 6 app.use(express.text());
 7
 8 app.get('/hello', (req, res) => {
 9   res.send('Hello!');
10 });
11
12 app.get('/whats/up', (req, res) => {
13   res.send(JSON.stringify({
14     value: 'not much',
15   }));
16 });
17
18 app.listen(port, () => {
19   console.log(`Listening on port ${port}`);
20 });
```

express_basic.ts

This says that "when URL / is accessed, call this function". The function sends some text to the person accessing that URL.

If we want our server to do more, we need to write lots more.

🍕 NodeJS Express Server

```
import express from 'express';

const app = express();
const port = 3000;

app.use(express.text());

app.get('/hello', (req, res) => {
  res.send('Hello!');
});

app.get('/whats/up', (req, res) => {
  res.send(JSON.stringify({
    value: 'not much',
  }));
});

app.listen(port, () => {
  console.log(`Listening on port ${port}`);
});
```

express_basic.ts

This line actually starts the server (on a particular port). It essentially runs an infinite loop so the program runs forever constantly waiting for new people to "access" it via a certain URL.

🧑‍💻 What Servers Are Used For

In the past, servers were just used to serve files back to client's browsers.

Nowadays, they're used to quietly exchange large amounts of information back and forth between client and server behind the scenes. Often this happens whilst the server is acting as an API.

This is probably done by sending JSON instead of text/HTML.

Servers Often Respond With JSON

```
 1 import express from 'express';
 2
 3 const app = express();
 4 const port = 3000;
 5
 6 app.use(express.json());
 7
 8 app.get('/whats/up', (req, res) => {
 9   res.json({
10     value: 'not much',
11   });
12 });
13
14 app.listen(port, () => {
15   console.log(`Listening on port ${port}`);
16 });
```

express_json.ts

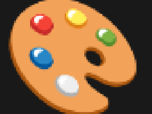
We can send javascript objects as JSON.

Servers Often Respond With JSON

```
 1 import express from 'express';
 2
 3 const app = express();
 4 const port = 3000;
 5
 6 app.use(express.json());
 7
 8 app.get('/whats/up', (req, res) => {
 9   res.json({
10     value: 'not much',
11   });
12 });
13
14 app.listen(port, () => {
15   console.log(`Listening on port ${port}`);
16 });
```

express_json.ts

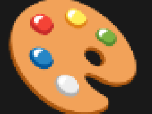
We can send javascript objects as JSON.

 How Servers Receive Information

```
1  import express from 'express';
2
3  const app = express();
4  const port = 3000;
5
6  app.use(express.json());
7
8  app.get('/my/url', (req, res) => {
9    const name = req.query.name;
10   const age = req.query.age;
11   res.json({
12     name: `Name is ${name}`,
13     age: `Age is ${age}`,
14   });
15 });
16
17 app.listen(port, () => {
18   console.log(`Listening on port ${port}`);
19 });
```

input_query.ts

Servers can receive information through the URL. This is called the `query` data.

 How Servers Receive Information

```
1  import express from 'express';
2
3  const app = express();
4  const port = 3000;
5
6  app.use(express.json());
7
8  app.get('/my/url', (req, res) => {
9    const name = req.query.name;
10   const age = req.query.age;
11   res.json({
12     name: `Name is ${name}`,
13     age: `Age is ${age}`,
14   });
15 });
16
17 app.listen(port, () => {
18   console.log(`Listening on port ${port}`);
19 });
```

input_query.ts

Servers can receive information through the URL. This is called the `query` data.

How Servers Receive Information

```
1 import express from 'express';
2
3 const app = express();
4 const port = 3000;
5
6 app.use(express.json());
7
8 app.get('/my/url/:name', (req, res) => {
9   const name = req.params.name;
10  res.json({
11    name: `Name is ${name}`,
12  });
13 });
14
15 app.listen(port, () => {
16   console.log(`Listening on port ${port}`);
17 });
```

input_params.ts

Servers can receive information through the URL. This is called the `params` data.

How Servers Receive Information

```
1  import express from 'express';
2
3  const app = express();
4  const port = 3000;
5
6  app.use(express.json());
7
8  app.get('/my/url/:name', (req, res) => {
9    const name = req.params.name;
10   res.json({
11     name: `Name is ${name}`,
12   });
13 });
14
15 app.listen(port, () => {
16   console.log(`Listening on port ${port}`);
17 });
```

input_params.ts

Servers can receive information through the URL. This is called the `params` data.

👀 Alterantives To `app.get`

We actually have an alternative to `app.get` called `app.post` which is where we send information through a body and not over the URL.

```
1  import express from 'express';
2
3  const app = express();
4  const port = 3000;
5
6  app.use(express.json());
7
8  app.post('/hello', (req, res) => {
9    const name = req.query.body;
10   res.json({
11     name: `Name is ${name}`,
12   });
13 });
14
15 app.listen(port, () => {
16   console.log(`Listening on port ${port}`);
17 });
```

post_example.ts

👀 Alterantives To `app.get`

Sure, but we can't call this in the browser anymore? How can we talk to it?

🔊 API Clients To Talk To Servers

We can use API clients to talk to servers.

Locally you can install "Advanced Rest Client" or on the CSE machines you can run `1531 arc`.

API Clients To Talk To Servers

It brings up an app like this.

ARC

Request

HTTP request

Socket

History

Send a request and recall it from here

Once you made a request it will appear in this place.

Saved

Save a request and recall it from here

Use ctrl+s to save a request. It will appear in this place.

Install new ARC with new features!

Method

GET

Request URL

An URL is required.

SEND

Parameters

Headers

Variables

Toggle source mode

Insert headers set

Header name

Header value

ADD HEADER

Headers are valid

Headers size: bytes

Selected environment: Default

🔊 API Clients To Talk To Servers

Web browsers always make a `get` request to a server, but this is what allows us to make `post` requests.

🥦 HTTP Crud Methods

`get` and `post` aren't the only types of requests we can send servers...

🥦 All The HTTP Methods

These are the remaining methods. They each have their own meaning about what the underlying action does.

Method	Operation
POST	**C**reate
GET	**R**ead
PUT	**U**pdate
DELETE	**D**elete

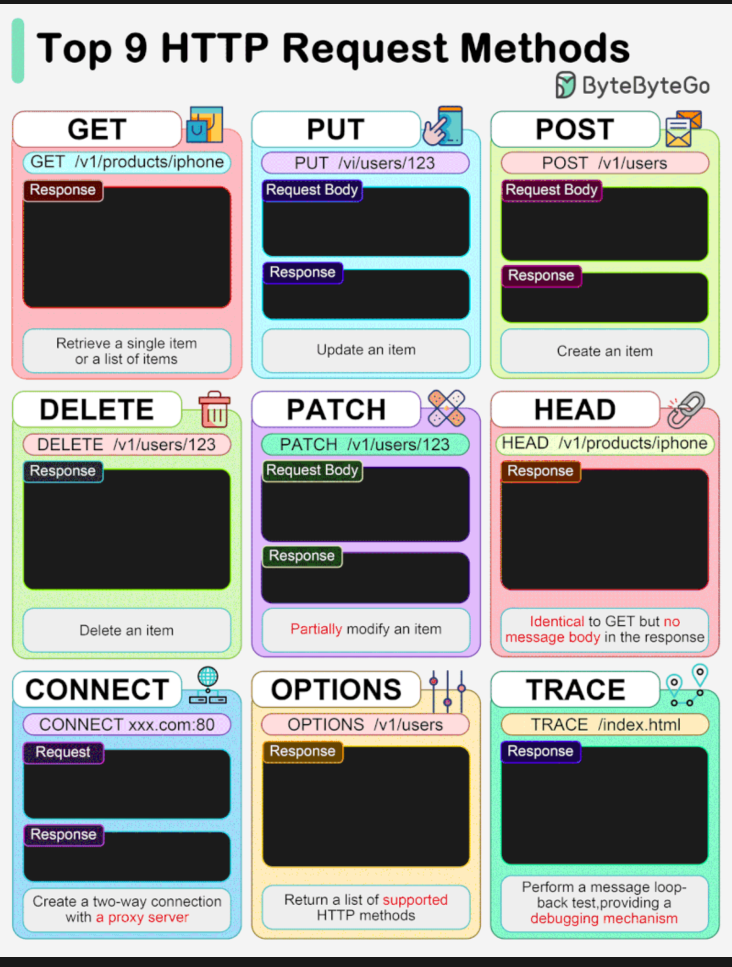
🥦 All The HTTP Methods

These are the remaining methods. They each have their own meaning about what the underlying action does.

Method	Operation	req.params	req.query	req.body	Response
POST	Create	✅	❌	✅	res.json()
GET	Read	✅	✅	❌	res.json()
PUT	Update	✅	❌	✅	res.json()
DELETE	Delete	✅	✅	❌	res.json()

All The HTTP Methods

Well not quite all...



API

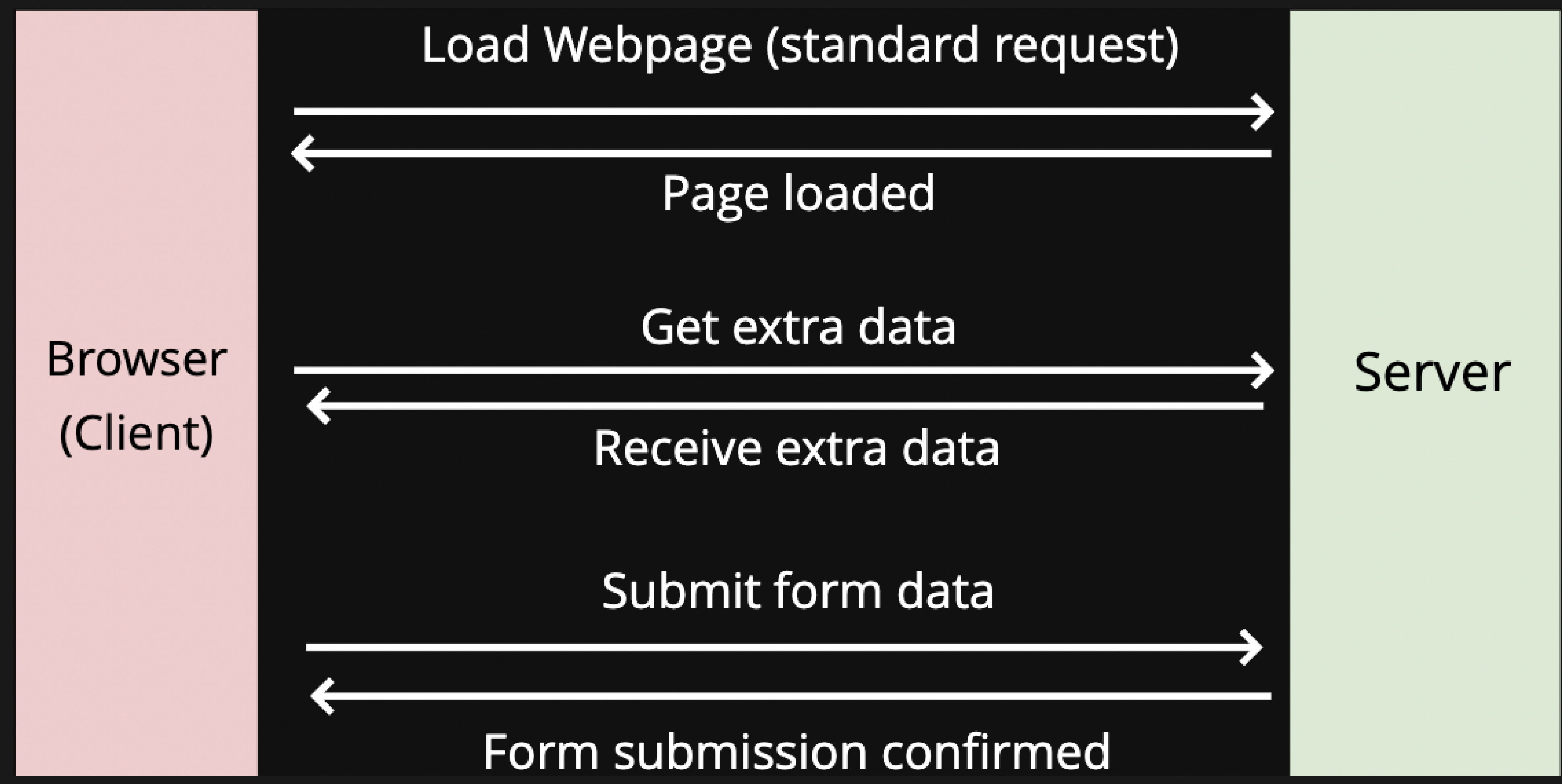
What you're actually seeing is the building blocks of an API...

API

An API (Application Programming Interface) refers to an interface exposed by a particular piece of software.

The most common usage of "API" is for Web APIs, which refer to a "contract" that a particular service provides. The interface of the service acts as a black box and indicates that for particular endpoints, and given particular input, the client can expect to receive particular output.

🪟 Web API



🪟 RESTful API

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. These 4 methods describe the "nature" of different API requests.

🔊 Requests Made Via Code

We don't need an API client to send requests to servers, we can actually send requests via code!

We can use an npm package `sync-request-curl` to allow us to programatically send RESTful API requests. `npm install sync-request-curl`.

We can send them to our previous server.

```
import request from 'sync-request-curl';

const res = request(
  'GET',
  'http://localhost:3001/apple',
  {
    qs: {
      name: 'Hayden',
    }
  }
);

console.log(JSON.parse(String(res.getBody())));
```

requests.ts

We can use an npm package `sync-request-curl` to allow us to programatically send RESTful API requests. `npm install sync-request-curl`.

We can send them to our previous server.

```
1  import request from 'sync-request-curl';
2
3  const res = request(
4    'GET',
5    'http://localhost:3001/apple',
6    {
7      qs: {
8        name: 'Hayden',
9      }
10   }
11 );
12
13 console.log(JSON.parse(String(res.getBody())));
```

requests.ts

Let's see what we can do with this...

🍅 Working With `jest`

```
import request from 'sync-request';

function get(route: string, qs: any) {
  const res = request(
    'GET',
    `http://localhost:3001${route}`,
    {
      qs: qs,
    }
  );
  return JSON.parse(String(res.getBody()));
}

describe('Test Apple', () => {
  test('If it returns a name string successfully', () => {
    const bodyObj = get('/apple', {
      name: 'Hayden',
    });
    expect(bodyObj.msg).toBe('Hi Hayden, thanks for sending apple!');
  });
});
describe('Test Orange', () => {
  test('If it returns a name string successfully', () => {
    const res = request(
      'POST',
      'http://localhost:3001/orange',
      {
        json: { name: 'Hayden' },
      }
    );
    const bodyObj = JSON.parse(String(res.getBody()));
    expect(bodyObj.msg).toBe('Hi Hayden, thanks for sending orange!');
  });
});
```

requests.test.ts

🤗 How To Wrap Into Project

In general, iteration 2 requires that you implement an HTTP server. However! Many of the routes that exist in iteration 2 are just wrappers of your iteration 1 functions.

Therefore it should be easy to "wrap" your iteration 1 functions with an HTTP server. I.E. Most of the "server" stuff you'll do is just routing, gathering bodies, and returning responses, while treating your iteration 1 functions like blackboxes.

Returning Errors

Restful APIs can also return errors:

```
import express from 'express';

const app = express();
const port = 3001;

app.use(express.text());

app.get('/apple/:name', (req, res) => {
  const name = req.params.name;
  if (name === 'Hayden') {
    res.status(400).json({
      error: 'Bad name',
    });
  } else {
    res.json({
      msg: `Hi ${name}, thanks for sending apple!`,
    });
  }
});

app.listen(port, () => {
  console.log(`Listening on port ${port}`);
});
```

error.ts

🍊 Other Express Helpers

Sometimes we put version numbers in our routes to help us keep track, e.g.:

- /v1/names/list
- /v2/names/list
- /v3/names/list

You will work with this principle in your major project.

🍊 Other Express Helpers

Final note - remember that routes will be "checked" in order of definition to see which one is most applicable for a particular call.

This means for wildcard routes we need to be careful about their position, e.g.

- `/game/:item`
- `/game`

will give us problems.

⛑️ Other - Token Encoding

A token is generally stringified for sending over HTTP - since everything over an HTTP request needs to be stringified. This is typically done with JSON.

If you pass a JSONified object (as opposed to just a string or a number) as a token, we recommend that you use encodeURIComponent and decodeURIComponent to encode it to be friendly for transfer over URLs.

Other - General Standards Standards

Not all applications follow the CRUD / Restful standard explicitly. Let's discuss two quick examples:

- Ignoring side effects to avoid everything being POST
- Sometimes 403/404's are returned in place of 400 to avoid fuzzing attacks

🎁 Optional! Making Life Easier.

Did you know we can make `node` auto restart if new files are compiled?

If we:

- Run `npm install --save-dev nodemon`
- Replace `node` with `nodemon` in `package.tson`

Then run `npm run start` in a separate terminal.

🎁 Optional! Making Life Easier.

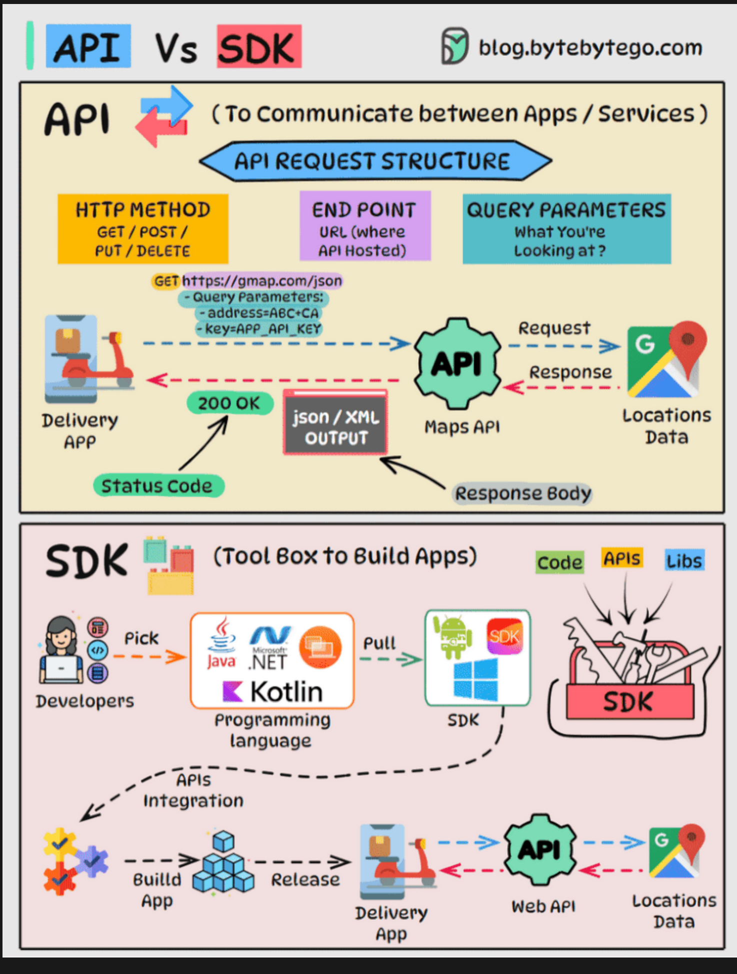
Did you know we can make `tsc` auto run if source files are changed?

If we:

- Add `--watch` flag to `tsc` command

Then run `npm run tsc` in a separate terminal.

🥷 API Vs SDK



👂 Feedback

Or go to the form here.