COMP1531

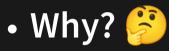
Lecture 4.2

Author(s): Hayden Smith



(Download as PDF)

In This Lecture



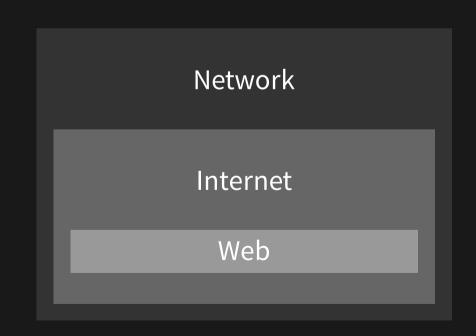
Web servers are fundamental part of web-based full-stack software

• What?

- Networks
- Express Server
- APIs
- Crud



- 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.

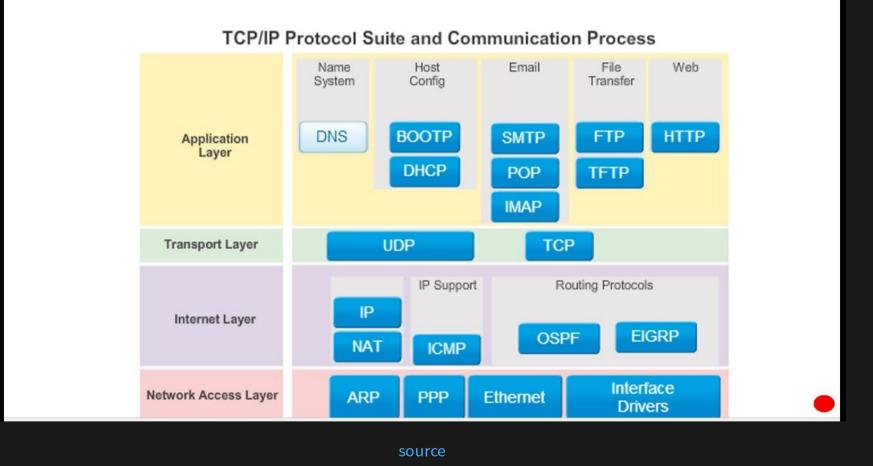


- Communication over networks must have a certain "structure" so everyone can understand.
- Humans do this all the time waving, handshakes, clapping. Standard operation procedure that structures how we share info.
- Different "structures" (protocols) are used for different types of communication.



Astrit Krasniqi CCNA/CCNP Certified Instructor - CCNA Cyber Ops, Chapter 4: Network Protocols and Services

TCP/IP Protocol Suite and Communication



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

🔁 FileZilla File Edit View Transfer Server Bookmarks Help			- 🗆 X			
₩ • ■ T T # Q № Q 1x 1/ T Q	S 🚓					
Host: Username: Passwe	ord: Port:	Quickconnect				
			~	-		
Local site: C:\Users\smith\OneDrive\Documents\Adobe\	 Remote 	site:	~	-		
Adobe Gustom Office Templates	🖻 Write: (no subje	ect) - Thunderbird				
Filename Filesize Filetype Last modified	File Edit View	osen Format Options Spelling 🗸 🔒 Securit	Tools Help			
	F <u>r</u> om:	Hayden Smith <z34180< td=""><td>)3@ad.unsw.edu.au></td><td>R PuTTY Configuration</td><td>×</td></z34180<>)3@ad.unsw.edu.au>	R PuTTY Configuration	×	
	✓ To:			Category:		
	<u>Subject:</u>			 Session Logging Terminal Keyboard Bell Features 	Options controlling session logging Session logging: None Printable output All session output SSH packets SSH packets and raw data	
3 directories				- Window	Log file name:	
Server/Local file Direc Remote file Queued files Failed transfers Successful transfers	Paragraph ➤ Hayden Smith	oh ✔ Variable Width ✔ ➡ AA ✔ A		- Appearance - Behaviour - Translation - Selection - Colours - Connection - Data - Proxy - Telnet - Blacin	putty log Browse (Log file name can contain &Y, &M, &D for date, &T for time, and &H for host name) What to do if the log file already exists: Always overwrite it Always append to the end of it Ak the user every time Image: Flush log file frequently	
	Lecturer-in-charge COMP1531 School of Computer Science and Engineering UNSW Sydney			About	Options specific to SSH packet logging Omit known password fields Omit session data Open Cancel	



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.



A very popular npm library exists to allow you to run your own HTTP server with NodeJS. It's called Express Server.



```
import express from 'express';
 1
 2
 3 const app = express();
   const port = 3000;
 4
 5
 6 app.use(express.text());
 7
   app.get('/', (req, res) => {
 8
    res.send('Hello World!');
 9
10 });
11
12 app.listen(port, () => {
     console.log(`Listening on port ${port}`);
13
14 });
```

4.2_express_basic.ts



```
1 import express from 'express';
2
3 const app = express();
4 const port = 3000;
5
6 app.use(express.text());
7
8 app.get('/', (req, res) => {
9 res.send('Hello World!');
10 });
11
12 app.listen(port, () => {
13 console.log(`Listening on port ${port}`);
14 });
```

4.2_express_basic.ts

This is us importing the express library



```
1 import express from 'express';
2
3 const app = express();
4 const port = 3000;
5
6 app.use(express.text());
7
8 app.get('/', (req, res) => {
9 res.send('Hello World!');
10 });
11
12 app.listen(port, () => {
13 console.log(`Listening on port ${port}`);
14 });
```

4.2_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.



```
1 import express from 'express';
2
3 const app = express();
4 const port = 3000;
5
6 app.use(express.text());
7
8 app.get('/', (req, res) => {
9 res.send('Hello World!');
10 });
11
12 app.listen(port, () => {
13 console.log(`Listening on port ${port}`);
14 });
```

4.2_express_basic.ts

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



```
1 import express from 'express';
2
3 const app = express();
4 const port = 3000;
5
6 app.use(express.text());
7
8 app.get('/', (req, res) => {
9 res.send('Hello World!');
10 });
11
12 app.listen(port, () => {
13 console.log(`Listening on port ${port}`);
14 });
```

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 of these.

^{4.2}_express_basic.ts



```
1 import express from 'express';
2
3 const app = express();
4 const port = 3000;
5
6 app.use(express.text());
7
8 app.get('/', (req, res) => {
9 res.send('Hello World!');
10 });
11
12 app.listen(port, () => {
13 console.log(`Listening on port ${port}`);
14 });
```

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.

^{4.2}_express_basic.ts



```
1 import express from 'express';
2
3 const app = express();
4 const port = 3000;
5
6 app.use(express.text());
7
8 app.get('/', (req, res) => {
9 res.send('Hello World!');
10 });
11
12 app.listen(port, () => {
13 console.log(`Listening on port ${port}`);
14 });
```

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.

Let's take a step back to learn about what servers are used for.

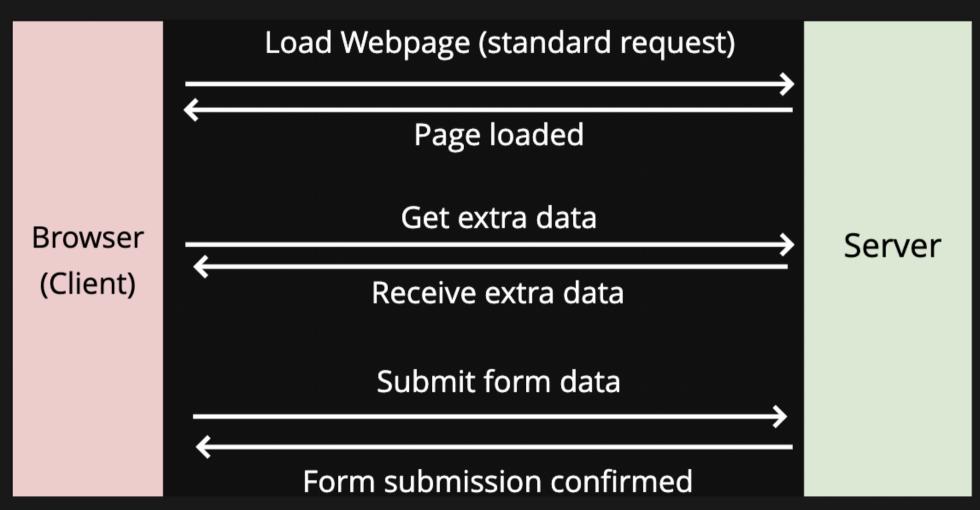
^{4.2}_express_basic.ts



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.







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.

GET, PUT, POST, DELETE are HTTP Methods. They refer to CRUD operations.

Method Operation

meenoa	operation
POST	Create
GET	Read
PUT	Update
DELETE	Delete



Different CRUD properties require different approaches for input. All output are the same.

```
1 import express from 'express';
 3 const app = express();
 4 const port = 3001;
 6 app.use(express.text());
8 app.get('/apple', (req, res) => {
     const name = req.query.name;
 9
     res.send(JSON.stringify({
10
       msg: `Hi ${name}, thanks for sending apple!`,
11
12
     }));
13 });
14
15 // same for .put and .delete
16 app.post('/orange', (req, res) => {
     const body = JSON.parse(req.body);
17
     const name = body.name;
18
     res.send(JSON.stringify({
19
       msg: `Hi ${name}, thanks for sending orange!`,
20
    }));
21
22 });
23
24 app.listen(port, () => {
     console.log(`Listening on port ${port}`);
25
26 });
```

4.2_crud.ts



Different CRUD properties require different approaches for input.

- For inputs:
 - GET | DELETE: via req.query (capture URL)
 - PUT|POST: via req.body (capture body)
- For outputs:
 - All outputs should be packaged up as JSON



If we are embrace the use of JSON everywhere, we can make use of other library feature to clean up the code.

```
1 import express from 'express';
 3 const app = express();
 4 const port = 3001;
 5
 6 app.use(express.json());
8 app.get('/apple', (req, res) => {
     const name = req.query.name;
 9
     res.json({
10
11
       msg: `Hi ${name}, thanks for sending apple!`,
12
   });
13 });
14
15 // same for .put and .delete
16 app.post('/orange', (req, res) => {
     const name = req.body.name;
17
     res.json({
18
       msg: `Hi ${name}, thanks for sending orange!`,
19
    });
20
21 });
22
23 app.listen(port, () => {
     console.log(`Listening on port ${port}`);
24
25 });
```



If we are embrace the use of JSON everywhere, we can make use of other library feature to clean up the code.

```
6 app.use(express.json());
    const name = reg.guery.name;
     res.json({
10
11
       msg: `Hi ${name}, thanks for sending apple!`,
16 app.post('/orange', (req, res) => {
     const name = req.body.name;
17
     res.json({
18
23 app.listen(port, () => {
```



Task

Create a web server that uses CRUD to allow you to create, update, read, and delete a point via HTTP requests

Use a global variable to manage the state.

Talking To A Web Server

How can we talk to a web server as a client?

- 1. API client
- 2. Web Browser
- 3. An NPM library: sync-requests



How to download/install ARC:

- Open google chrome
- Google "ARC client"
- Install the addon and open it
- Follow the demo in the lectures

You may need to use a tool like this in the final exam.

🔊 API Client (ARC, Postman)

	Request	()	:
HTTP request	Method GET - Request URL		• • •
Socket	An URL is required. Parameters		
History ^	Headers Variables		
Send a request and recall it from here	Image: Toggle source mode + Insert headers set Header name Header value ADD HEADER		×
Once you made a request it will appear in this place.	A Headers are valid	Headers size: I	bytes
Saved ^			

Save a request and recall it from here



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

()



127.0.0.1:5000/hello	× +	- 0	×
← → C ③ 127.0.0.1	:5000/hello	•	:
Hello World!	🕞 🖬 Elements Console Sources Network Performance Memory Application	»	×
	🖲 🛇 🛛 🔽 🔍 🗆 Preserve log 🗆 Disable cache 🛛 Online 🔹 🖈 🛓		\$
	Filter 🗌 Hide data URLs 📶 XHR JS CSS Img Media Font Doc WS Man	nifest Other	
	10 ms 20 ms 30 ms 40 ms 50 ms 60 ms 70 ms 80 ms 90 ms	100 ms	110
	Name × Headers Preview Response Timing		
	hello General		
	Request URL: http://127.0.0.1:5000/hello		
	Request Method: GET		
	Status Code: 👄 200 OK		
	Remote Address: 127.0.0.1:5000		
	Referrer Policy: no-referrer-when-downgrade		
	▼ Response Headers view source		
	Content-Length: 12		
	Content-Type: text/html; charset=utf-8		
	Date: Wed, 09 Oct 2019 13:26:05 GMT		
	Server: Werkzeug/0.16.0 Python/3.5.3		
	▼ Request Headers view source		
	Accept: text/html,application/xhtml+xml,application/xml;q=0.5	9,image/webp	,i
	<pre>mage/apng,*/*;q=0.8,application/signed-exchange;v=b3</pre>		
	Accept-Encoding: gzip, deflate, br		
	Accept-Language: en-GB,en-US;q=0.9,en;q=0.8		
	1 requests 166 B transferre Cache-Control: max-age=0		-



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

We can send them to our previous server.

```
1 import request from 'sync-request';
2
3 const res = request(
4 'GET',
5 'http://localhost:3001/apple?name=Hayden'
6 );
7 console.log(JSON.parse(String(res.getBody())));
```

4.2_requests.ts



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

We can send them to our previous server.

```
1 import request from 'sync-request';
2
3 const res = request(
4 'GET',
5 'http://localhost:3001/apple?name=Hayden'
6 );
7 console.log(JSON.parse(String(res.getBody())));
4.2 requests.ts
```

Let's look at the sync-request library to see if we can remove name=Hayden from URL.



```
1 import request from 'sync-request';
   describe('Test Apple', () => {
 3
     test('If it returns a name string successfully', () => {
 5
       const res = request(
         'GET',
 6
         'http://localhost:3001/apple',
 8
 9
           qs: {
             name: 'Hayden',
10
11
           },
12
         }
13
       );
14
       const bodyObj = JSON.parse(String(res.getBody()));
       expect(bodyObj.msg).toBe('Hi Hayden, thanks for sending apple!');
15
16
     });
17 });
18 describe('Test Orange', () => {
     test('If it returns a name string successfully', () => {
19
20
       const res = request(
21
         'POST',
22
         'http://localhost:3001/orange',
23
           body: JSON.stringify({ name: 'Hayden' }),
24
           headers: {
25
              'Content-type': 'application/json',
26
27
           },
28
         }
29
       );
       const bodyObj = JSON.parse(String(res.getBody()));
30
       expect(bodyObj.msg).toBe('Hi Hayden, thanks for sending orange!');
31
32
     });
```

33 });

4.2_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.

Optional! Making Life Easier.

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

If we:

- Runnpm 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.





Or go to the form here.