



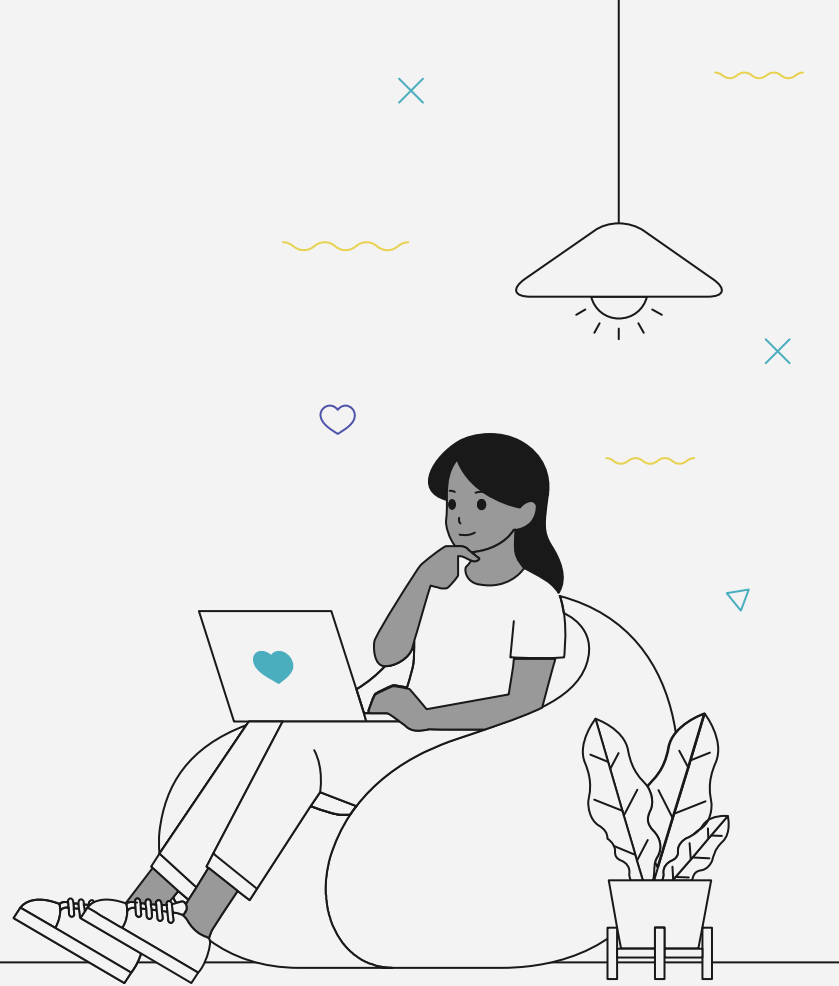
Github

<https://github.com/PersoSirEduard/HackMcGill-Backend-Workshop>



Backend Workshop

HackMcGill





Eduard Anton

What is a backend?

The backend (or “server side”) is the portion of the website you don’t see. It’s responsible for storing, [processing] and organizing data, and ensuring everything on the client-side actually works. The backend communicates with the frontend, sending and receiving information to be displayed as a web page. Source: [CareerFoundry](#)



Why use a backend?

- Persistence
- Authentication and Authorization
- External Services
- Security
- Etc.



Application Programming Interface (API)

- Allows software to communicate
- Specify standards (interface)
- Software design involved
- **APIs are not necessarily provided by a remote server! (e.g. Windows API)**



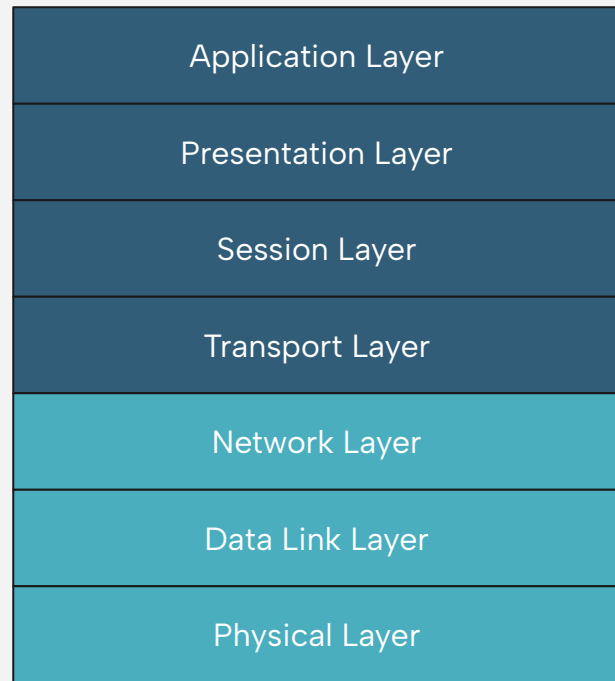
Open Systems Interconnection (OSI)

- Standardization of communication (AKA protocols)
- Communication over the network
- **Layered abstraction**

From the **Application Layer** and up we care about:

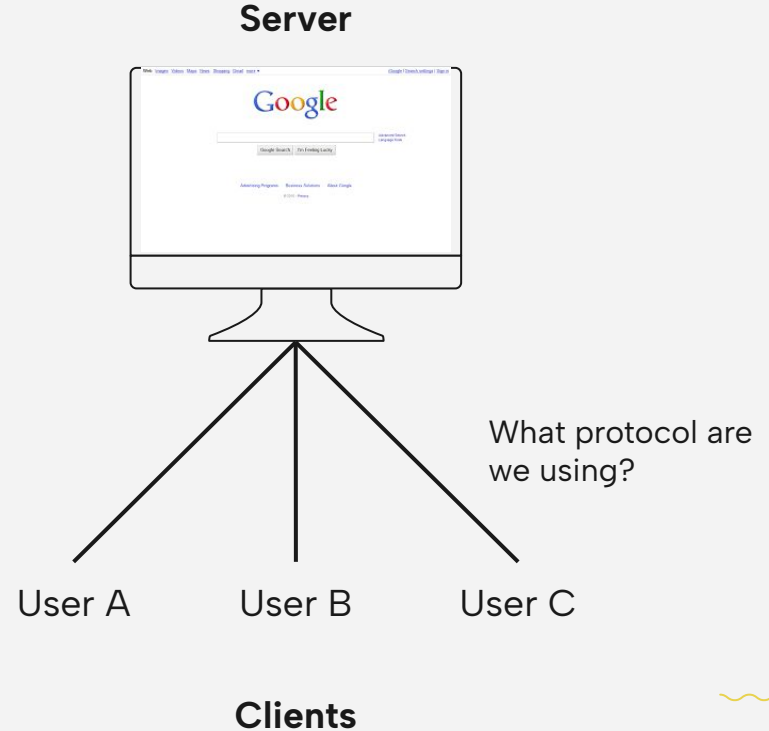
- The message we want to send
- The destination of our message

Response data flow

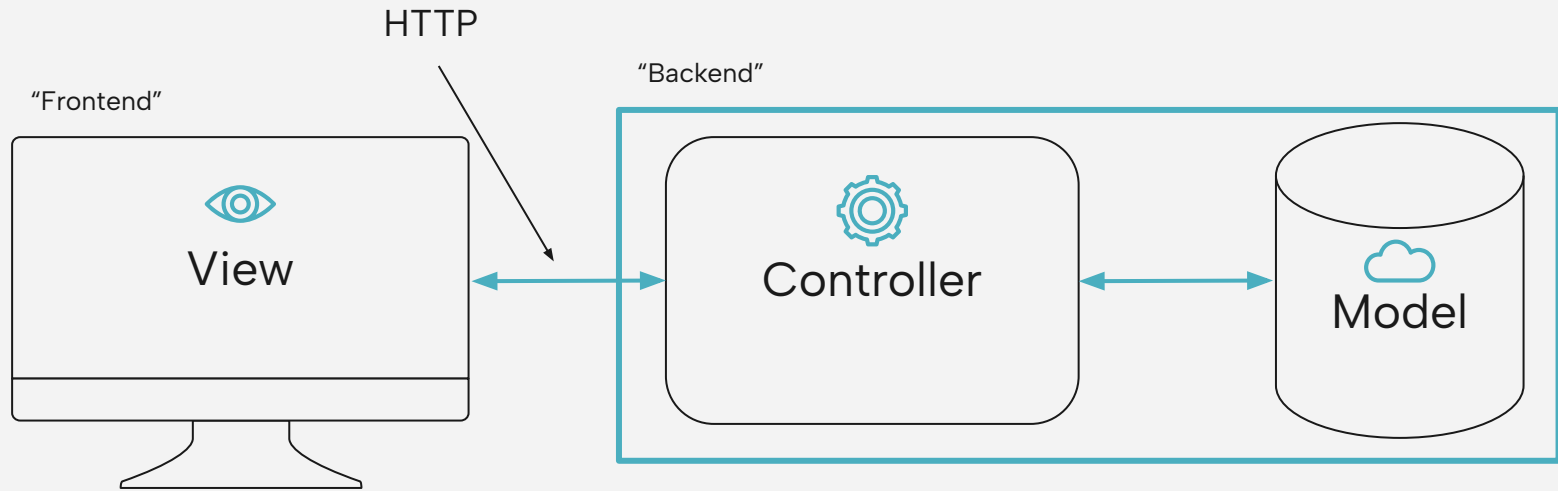


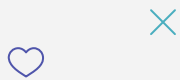
Client-Server

- Client requests a “service” provided by the server
- Agreement on the API
- Centralized architecture
- Simple
- Many-to-one connections
- Careful with traffic management. There is a risk of Denial-of-service (DoS) attacks
- Other architectures: P2P



MVC (Model-View-Controller)





Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes, such as machine-to-machine communication, programmatic access to APIs, and more.

Source: [Mozilla](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <p>This is an example of a simple HTML
page with one paragraph.</p>
  </body>
</html>
```

Note: HTTP by itself is not our API





HTTP Requests and Responses



Request

HTTP Method
Path
HTTP Version

GET / HTTP/2

Host: www.google.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:133.0) Gecko/20100101 Firefox/133.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
DNT: 1
Sec-GPC: 1
Connection: keep-alive
Cookie: Example
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Priority: u=0, i
TE: trailers

HTTP Headers

Response

HTTP Version
Return Code

HTTP/2 200 OK

Date: Wed, 22 Jan 2025 17:08:01 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Strict-Transport-Security: max-age=31536000
Content-Length: 63479
x-xss-protection: 0

HTTP Headers

Response Body

```
<!DOCTYPE html>
<html>
  <body>
    <p>Google</p>
  </body>
</html>
```





HTTP Versions

1.0

- First version, introduced in 1996
- Stateless (no sessions)

1.1

- Improved performance and security
- Persistent connection
- Caching

1.2

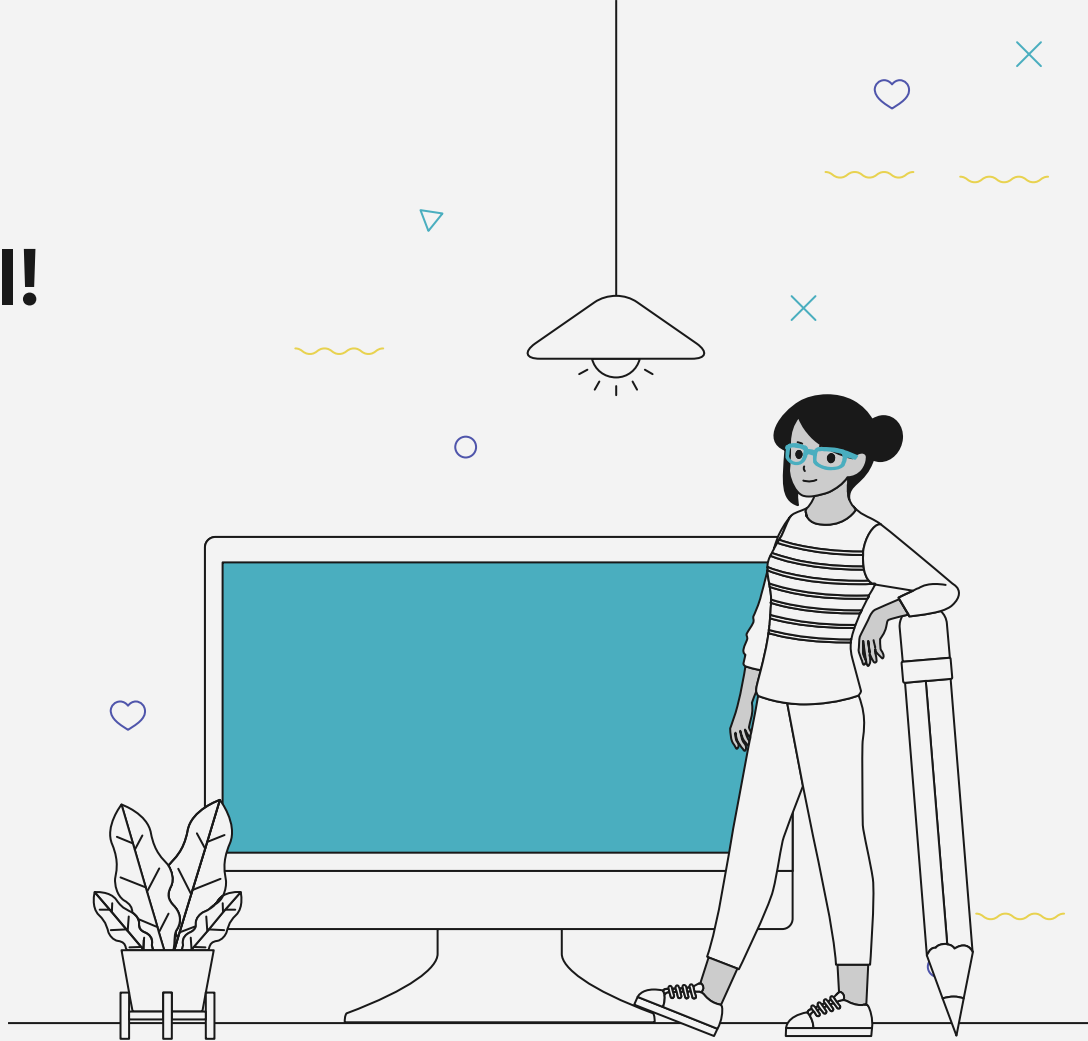
- Pipelining (send multiple requests before a response)
- Server push (proactive server)
- Header compression

2.0

- Prioritization
- Streams
- Compression



Let's build our API!



Express.js

“Most basic RESTful server”

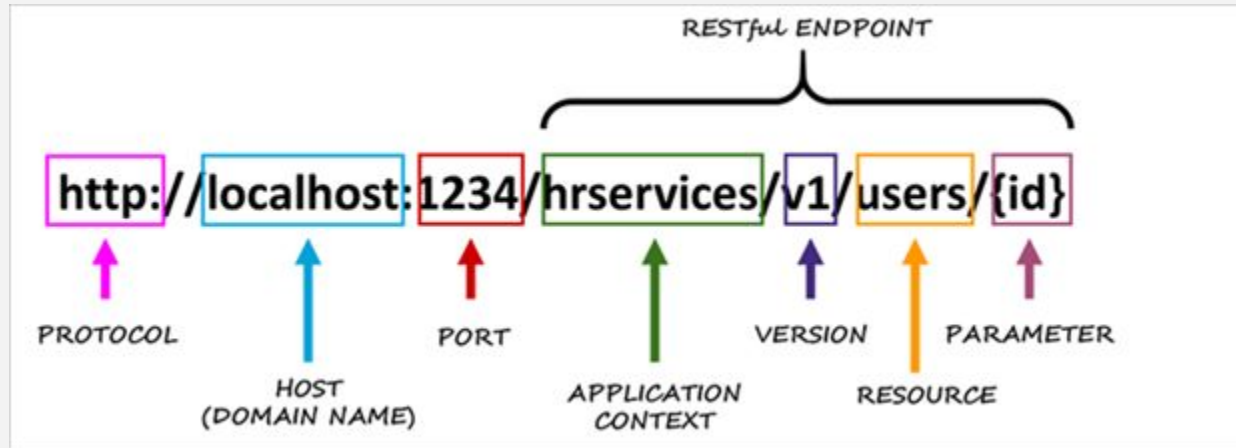
```
const express = require('express')
const app = express()
const port = 3000

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

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



REST URL



Fetch API

"Promises"

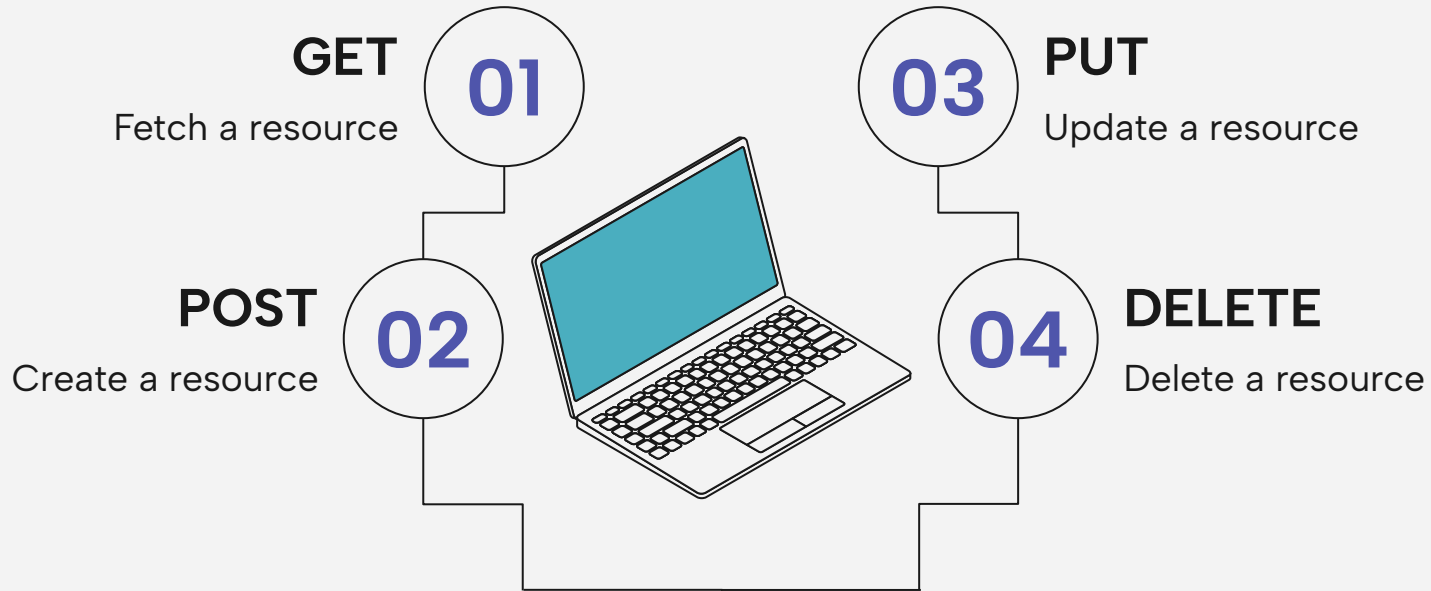
```
1 // Specify the API endpoint for user data
2 const apiUrl = 'https://api.example.com/users/123';
3
4 // Make a GET request using the Fetch API
5 fetch(apiUrl)
6   .then(response => {
7     if (!response.ok) {
8       throw new Error('Network response was not ok');
9     }
10    return response.json();
11  })
12   .then(userData => {
13     // Process the retrieved user data
14     console.log('User Data:', userData);
15  })
16   .catch(error => {
17     console.error('Error:', error);
18  });
```

"Async/Await"

```
1 // Specify the API endpoint for user data
2 const apiUrl = 'https://api.example.com/users/123';
3
4 try {
5   // Make a GET request using the Fetch API
6   const response = await fetch(apiUrl)
7   const userData = await response.json()
8
9   // Process the retrieved user data
10  console.log('User Data:', userData)
11 } catch (error) {
12   console.error('Error:', error);
13 }
```




HTTP Methods



"CRUD": Create, Read, Update, Delete



HTTP Headers

- Authentication
- Caching
- Conditionals
- Connection management
- Content negotiation
- Cross-origin resource sharing (CORS)
- Body description
- Custom
- Etc.



Thanks!

Do you have any questions?

CREDITS: This presentation template was created by [Slidesgo](#), and includes icons by [Flaticon](#), and infographics & images by [Freepik](#)

