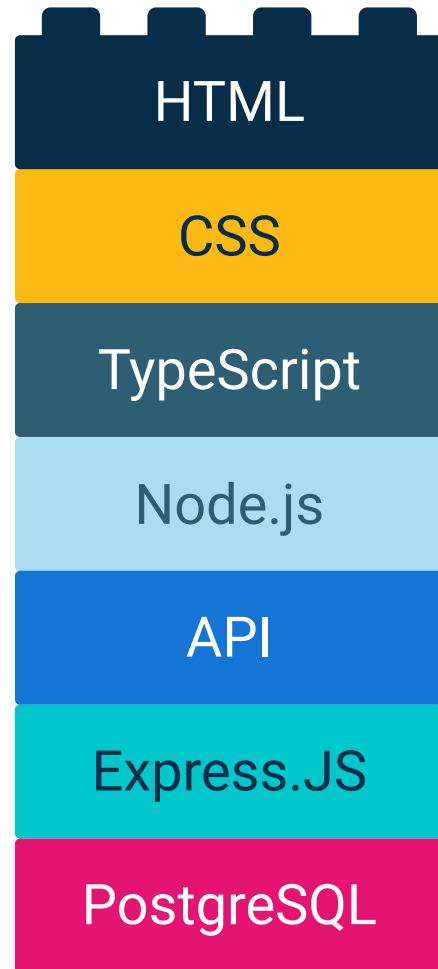Software Development

Servers and APIs

Module 09

What is **full-stack development?**

Full-Stack Web Development

Full-stack web development encompasses the suite of tools required to build both the front end and the back end of a web application.

HTML

CSS

TypeScript

Node.js

API

Express.JS

PostgreSQL

How much of **the stack** do we know?

Client-Side

- So far, we have learned about client-side development, using HTML, CSS, and TypeScript as the three primary components of client-side code.

- We have also learned about Node.js and how to run logic outside the browser, in the back end, or **server-side**.
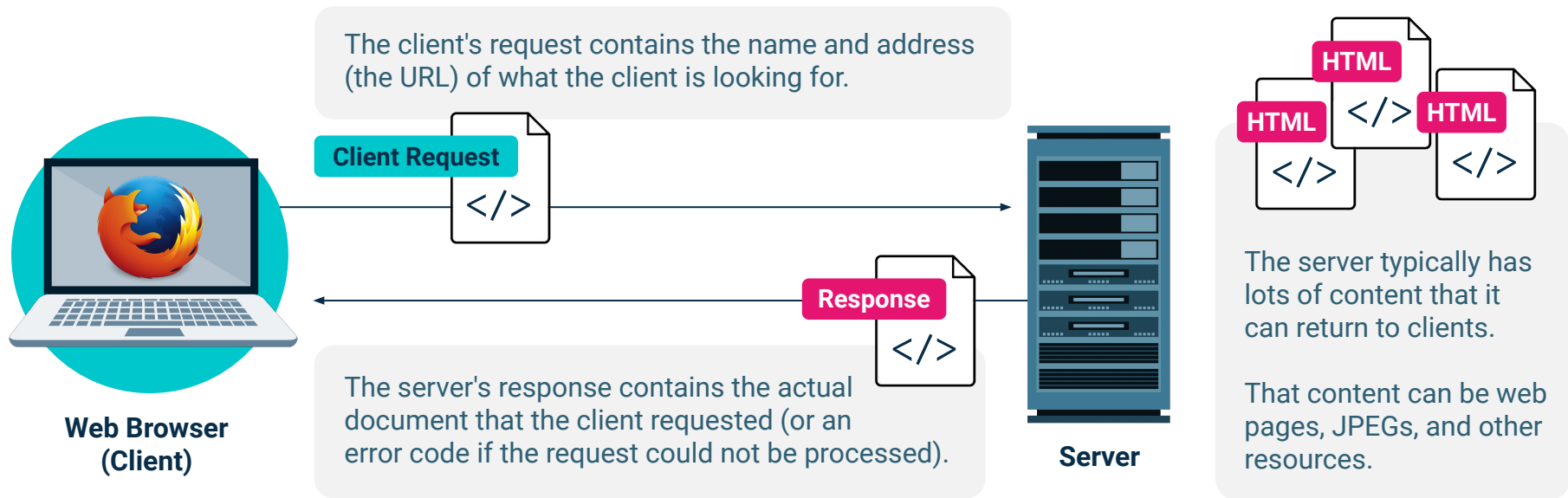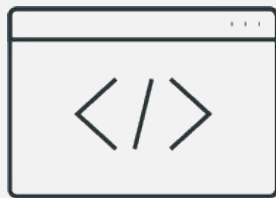
HTML

CSS

TypeScript

Node.js

API

Express.JS

PostgreSQL
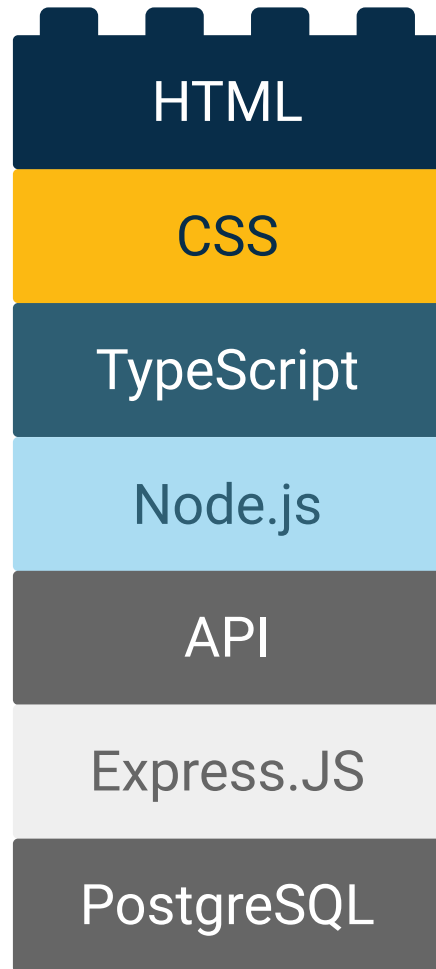
What is the
client–server model?

The Client–Server Model

In modern web applications, there is constant back-and-forth communication between the visuals displayed on the user's browser (the front end) and the data and logic stored on the server (the back end). Clients make requests, and servers respond.

The client's request contains the name and address (the URL) of what the client is looking for.

Client Request

</>

Response

</>

The server's response contains the actual document that the client requested (or an error code if the request could not be processed).

Web Browser (Client)

Server

HTML

HTML

HTML

</>

</>

</>

The server typically has lots of content that it can return to clients.

That content can be web pages, JPEGs, and other resources.
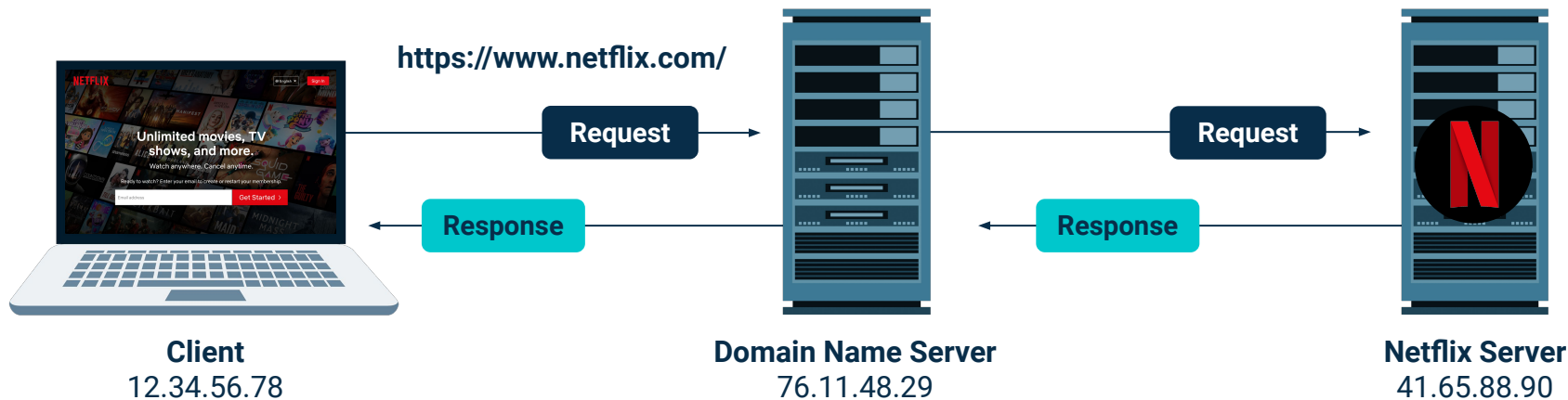
Client

Where do **web applications** live?
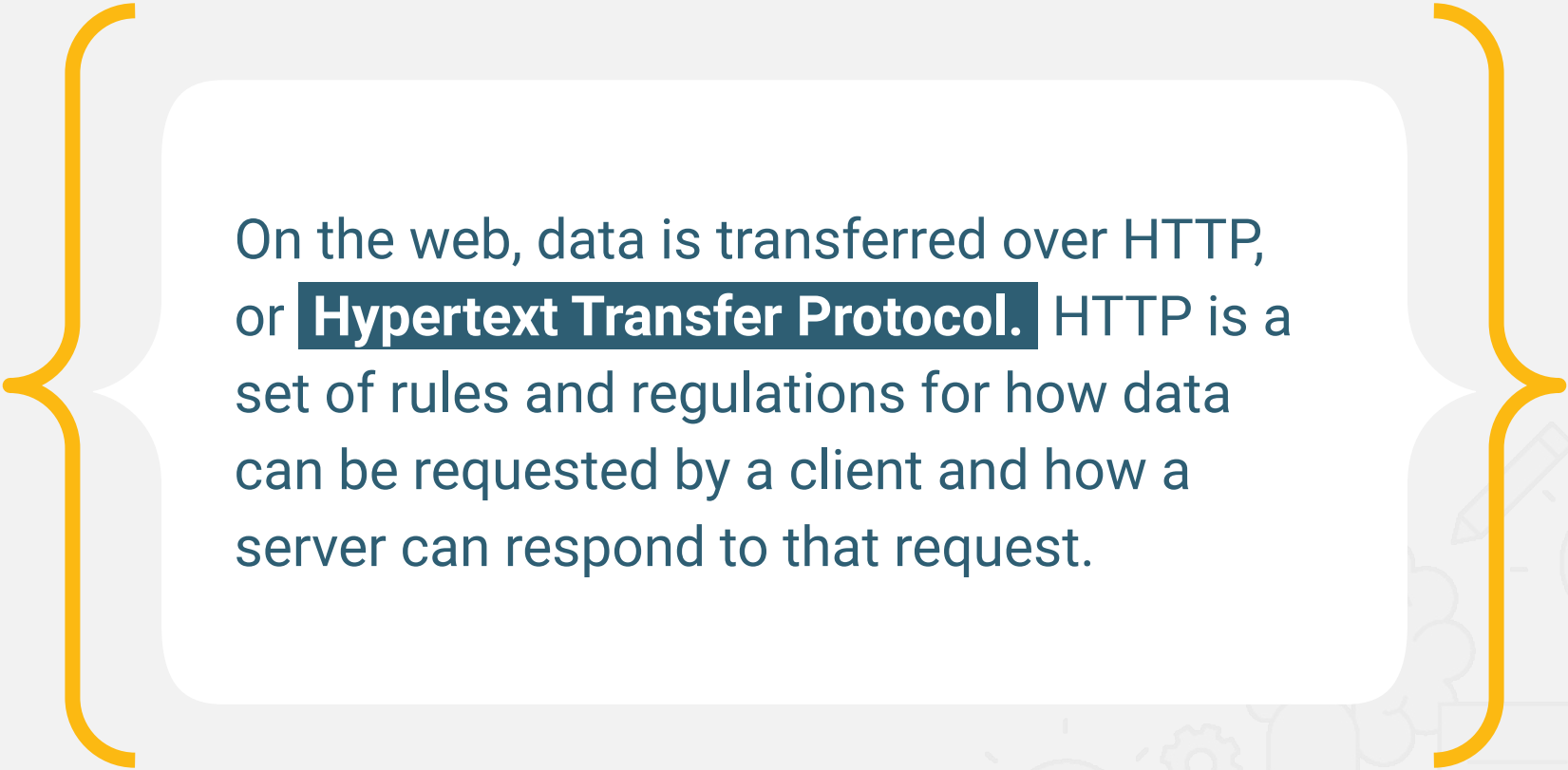
Web Applications Live on Servers

Web servers are typically nothing more than specialized computers running software with the specific task of waiting for an internet request to come in and ask for data in return.

https://www.netflix.com/

Request

Response

Request

Response

Client
12.34.56.78

Domain Name Server
76.11.48.29
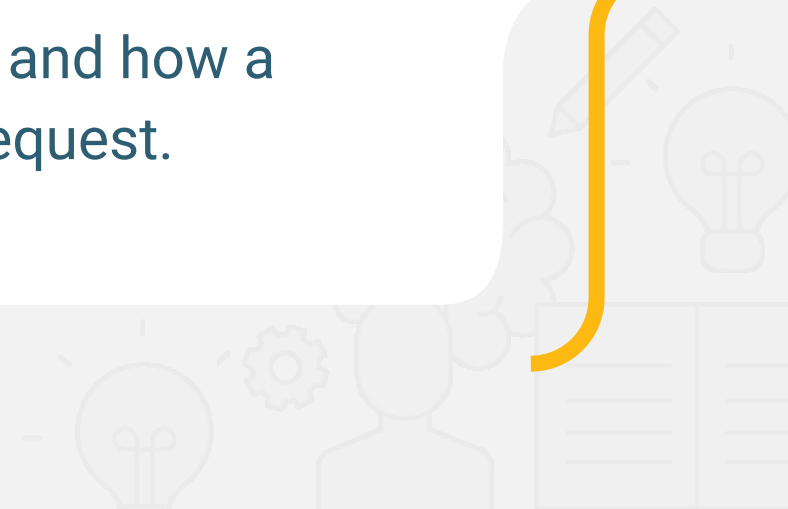
Netflix Server
41.65.88.90

How are these **requests** made?

On the web, data is transferred over HTTP, or **Hypertext Transfer Protocol.** HTTP is a set of rules and regulations for how data can be requested by a client and how a server can respond to that request.

Across all internet-connected devices, we constantly make **HTTP requests to web servers** for different types of data, like in the following examples:
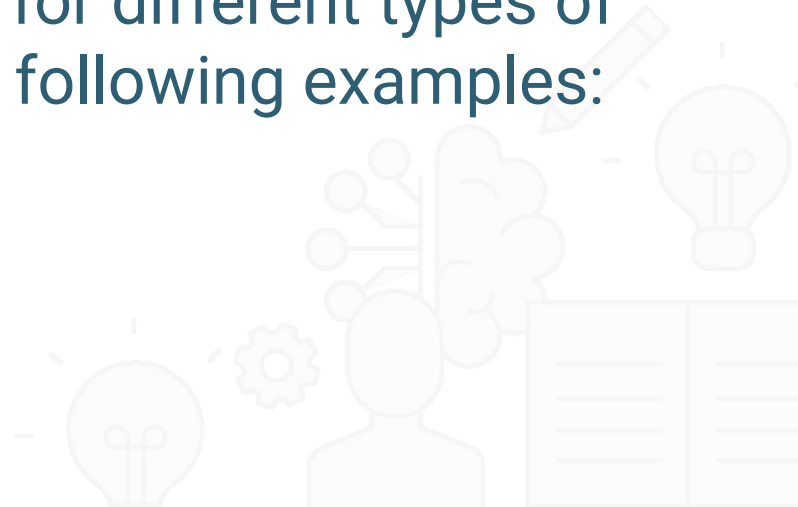
We Request Data Over HTTP

1 When we visit deployed applications at `<username>.github.io.`

2 When our phone or watch automatically updates the weather forecast.

3 When we use a media streaming service.

Can we use **data from other servers** in an application?
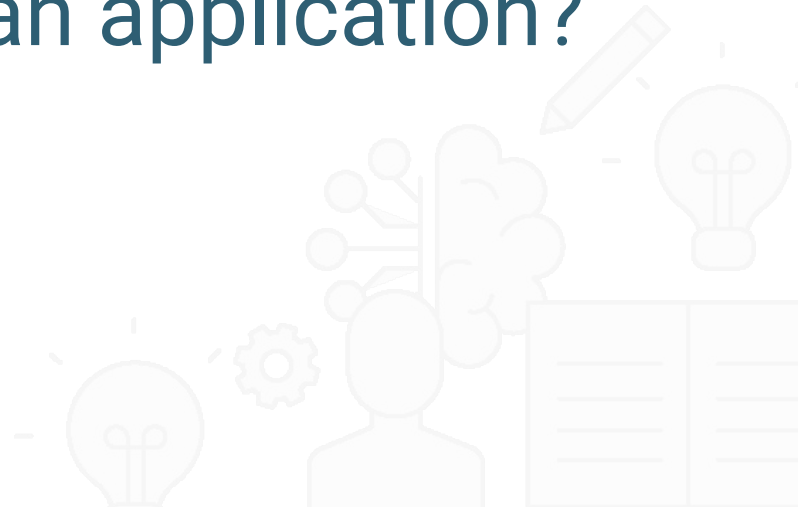
Yes, we can! We can use a specific functionality to **request data over HTTP** and use that data in an application.
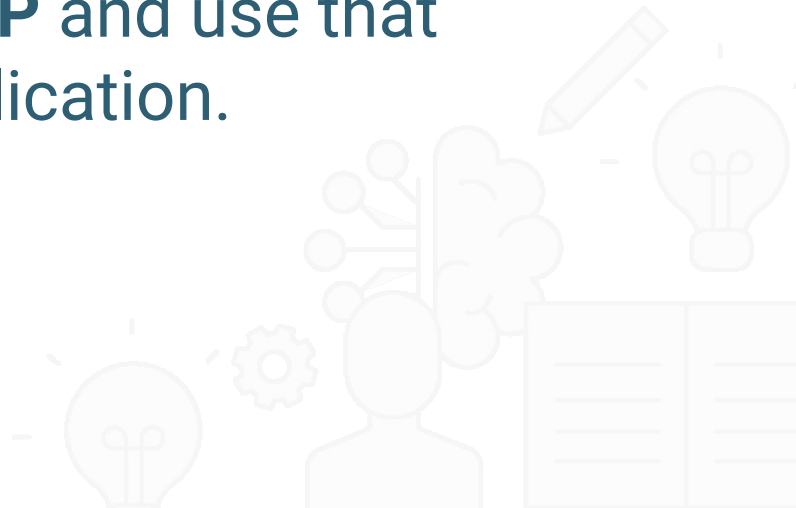
JSON (JavaScript Object Notation)

This data usually comes in a special type of text format known as JSON (JavaScript Object Notation).

https://www.google.com/maps

Get Request

JSON Response

Client
12.34.56.78

Google Server
10.23.97.54

JSON (JavaScript Object Notation)

With this data, we can do any of the following in an application:

1. Retrieve weather data to display in an application.

2. Use Google Maps to help create a trip itinerary.

3. Manage Spotify or YouTube playlists.

4. Control lights, alarms, and other devices.

5. And much, much more!

What are **server-side APIs?**

Servers that hold data that we can request are known as **server-side APIs** (Application Programming Interfaces).
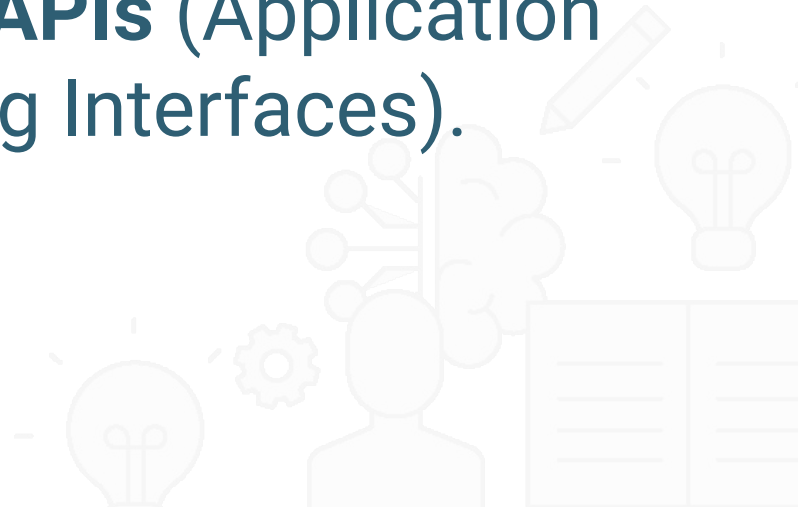
In the context of **APIs, "Application"** refers to any software with a distinct function. **"Interface"** can be thought of as a contract of service between two applications. This contract defines how the two communicate using **requests and responses.**

– "What is an API?"
(AWS Amazon)

How can we learn to **use and implement** these types of **APIs?**

How to Learn Server-Side APIs

Like other APIs we've used, implementing server-side APIs depends on the solution the API provides. Some are very simple, while others are complex and powerful, so we must determine which parts to use, if any.

How to Learn Server-Side APIs

You can try the following strategies to learn more about specific APIs:

1 Read the official documentation, and practice with the provided examples.

2 Reverse-engineer finished code to see how something was accomplished.

3 Build something from scratch and debug it using the Chrome DevTools.

4 Ask questions!

What is the **Fetch API?**

What is the Fetch API?

- We use the Fetch API to make requests to a server-side API.

- Fetch will make requests to specific API endpoints and process the response.

- Fetch can notify the API which type of request the client is sending, and which kind of response to return, using HTTP Methods and other options.

https://www.google.com/maps

Get Request

JSON Response

Client
12.34.56.78

Google Server
10.23.97.54
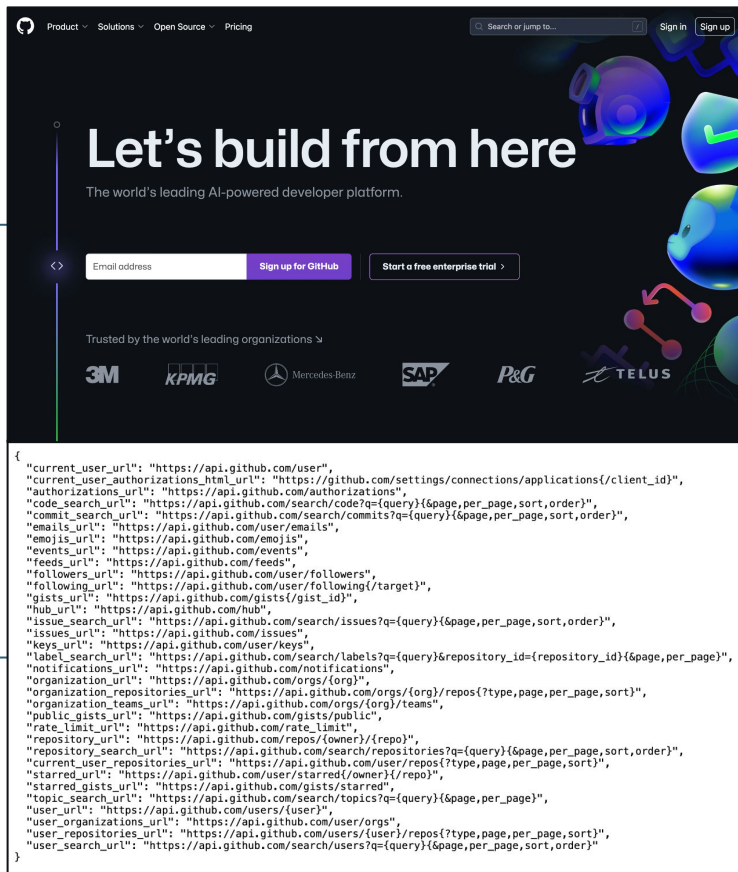
What is an API endpoint?

1 An API endpoint is the ***address*** to the specific location in an API that accepts requests and sends back responses.

2 **Uniform Resource Locators** (URLs) hold these addresses, whether they are web page locations or API endpoints.

Example Endpoint

For example,
`github.com` leads
to the homepage of
the GitHub website.

Whereas
`api.github.com`
leads to a page of
JSON data.

```
{
  "current_user_url": "https://api.github.com/user",
  "current_user_authorizations_html_url": "https://github.com/settings/connections/applications{/client_id}",
  "authorizations_url": "https://api.github.com/authorizations",
  "code_search_url": "https://api.github.com/search/code?q={query}{&page,per_page,sort,order}",
  "commit_search_url": "https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}",
  "emails_url": "https://api.github.com/user/emails",
  "emojis_url": "https://api.github.com/emojis",
  "events_url": "https://api.github.com/events",
  "feeds_url": "https://api.github.com/feeds",
  "followers_url": "https://api.github.com/user/followers",
  "following_url": "https://api.github.com/user/following{/target}",
  "gists_url": "https://api.github.com/gists{/gist_id}",
  "hub_url": "https://api.github.com/hub",
  "issue_search_url": "https://api.github.com/search/issues?q={query}{&page,per_page,sort,order}",
  "issues_url": "https://api.github.com/issues",
  "keys_url": "https://api.github.com/user/keys",
  "label_search_url": "https://api.github.com/search/labels?q={query}&repository_id={repository_id}{&page,per_page}",
  "notifications_url": "https://api.github.com/notifications",
  "organization_url": "https://api.github.com/orgs/{org}",
  "organization_repositories_url": "https://api.github.com/orgs/{org}/repos{?type,page,per_page,sort}",
  "organization_teams_url": "https://api.github.com/orgs/{org}/teams",
  "public_gists_url": "https://api.github.com/gists/public",
  "rate_limit_url": "https://api.github.com/rate_limit",
  "repository_url": "https://api.github.com/repos/{owner}/{repo}",
  "repository_search_url": "https://api.github.com/search/repositories?q={query}{&page,per_page,sort,order}",
  "current_user_repositories_url": "https://api.github.com/user/repos{?type,page,per_page,sort}",
  "starred_url": "https://api.github.com/user/starred{/owner}{/repo}",
  "starred_gists_url": "https://api.github.com/gists/starred",
  "topic_search_url": "https://api.github.com/search/topics?q={query}{&page,per_page}",
  "user_url": "https://api.github.com/users/{user}",
  "user_organizations_url": "https://api.github.com/user/orgs",
  "user_repositories_url": "https://api.github.com/users/{user}/repos{?type,page,per_page,sort}",
  "user_search_url": "https://api.github.com/search/users?q={query}{&page,per_page,sort,order}"
}
```

Example Endpoint

GET
http://api.github.com/orgs/nodejs/repos

```
[
  {
    "id": 186749,
    "node_id": "MDEwOlJlcG9zaXRvcnkxODY3NDk=",
    "name": "http-parser",
    "full_name": "nodejs/http-parser",
    "private": false,
    "owner": {
      "login": "nodejs",
      "id": 9950313,
      "node_id": "MDEyOk9yZ2FuaXphdGlvbjk5NTAzMTM=",
      "avatar_url": "https://avatars.githubusercontent.com/u/9950313?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/nodejs",
      "html_url": "https://github.com/nodejs",
      "followers_url": "https://api.github.com/users/nodejs/followers",
      "following_url": "https://api.github.com/users/nodejs/following{/other_user}",
      "gists_url": "https://api.github.com/users/nodejs/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/nodejs/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/nodejs/subscriptions",
      "organizations_url": "https://api.github.com/users/nodejs/orgs",
      "repos_url": "https://api.github.com/users/nodejs/repos",
      "events_url": "https://api.github.com/users/nodejs/events{/privacy}",
      "received_events_url": "https://api.github.com/users/nodejs/received_events",
      "type": "Organization",
      "site_admin": false
    },
    "html_url": "https://github.com/nodejs/http-parser",
    "description": "http request/response parser for c",
    "fork": false,
    "url": "https://api.github.com/repos/nodejs/http-parser",
    "forks_url": "https://api.github.com/repos/nodejs/http-parser/forks",
    "keys_url": "https://api.github.com/repos/nodejs/http-parser/keys{/key_id}",
    "collaborators_url": "https://api.github.com/repos/nodejs/http-parser/collaborators{/collaborator}",
    "teams_url": "https://api.github.com/repos/nodejs/http-parser/teams",
    "hooks_url": "https://api.github.com/repos/nodejs/http-parser/hooks",
    "issue_events_url": "https://api.github.com/repos/nodejs/http-parser/issues/events{/number}",
    "events_url": "https://api.github.com/repos/nodejs/http-parser/events",
    "assignees_url": "https://api.github.com/repos/nodejs/http-parser/assignees{/user}",
    "branches_url": "https://api.github.com/repos/nodejs/http-parser/branches{/branch}",
    "tags_url": "https://api.github.com/repos/nodejs/http-parser/tags",
    "blobs_url": "https://api.github.com/repos/nodejs/http-parser/git/blobs{/sha}",
    "git_tags_url": "https://api.github.com/repos/nodejs/http-parser/git/tags{/sha}",
    "git_refs_url": "https://api.github.com/repos/nodejs/http-parser/git/refs{/sha}",
    "trees_url": "https://api.github.com/repos/nodejs/http-parser/git/trees{/sha}",
    "statuses_url": "https://api.github.com/repos/nodejs/http-parser/statuses/{sha}",
    "languages_url": "https://api.github.com/repos/nodejs/http-parser/languages",
    "stargazers_url": "https://api.github.com/repos/nodejs/http-parser/stargazers",
    "contributors_url": "https://api.github.com/repos/nodejs/http-parser/contributors",
    "subscribers_url": "https://api.github.com/repos/nodejs/http-parser/subscribers",
    "subscription_url": "https://api.github.com/repos/nodejs/http-parser/subscription",
    "commits_url": "https://api.github.com/repos/nodejs/http-parser/commits{/sha}",
    "git_commits_url": "https://api.github.com/repos/nodejs/http-parser/git/commits{/sha}",
    "comments_url": "https://api.github.com/repos/nodejs/http-parser/comments{/number}",
```

Furthermore, **api.github.com /orgs/nodejs/r epos** leads to a massive list of JSON data.

/orgs/nod ejs/repos is the endpoint for that specific list of data.

What are the **HTTP methods?**

HTTP methods are a defined set of request methods that indicate the desired action to be performed on a resource.

– MDN Web Docs,
HTTP request methods
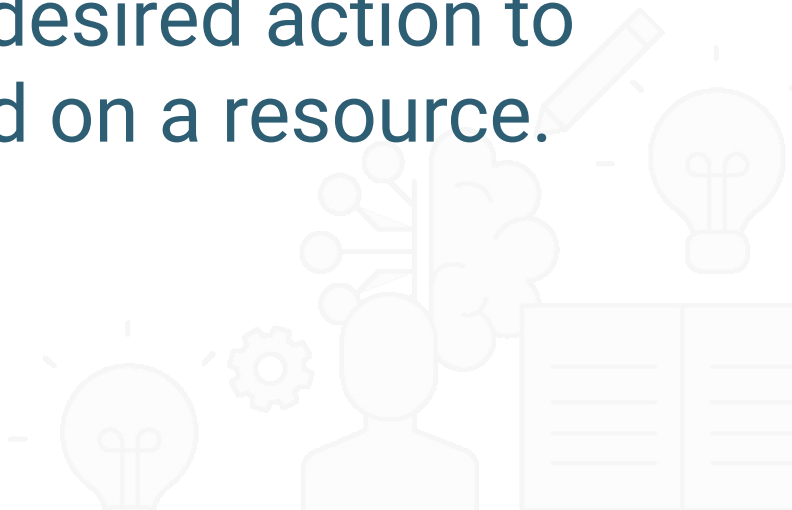
We will use four main HTTP methods. These methods perform the following actions: **Create, Read, Update, and Delete** (CRUD).

HTTP methods

However, HTTP uses the following syntax for these CRUD methods:

| POST	Submits data to the specified resource, often causing a change on the server. (CREATE)
GET	Retrieves a resource from the server. (READ)
PUT	Replaces a specified resource with a payload. (UPDATE)
DELETE	Deletes a specified resource. (DELETE)
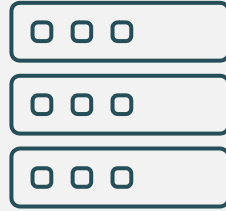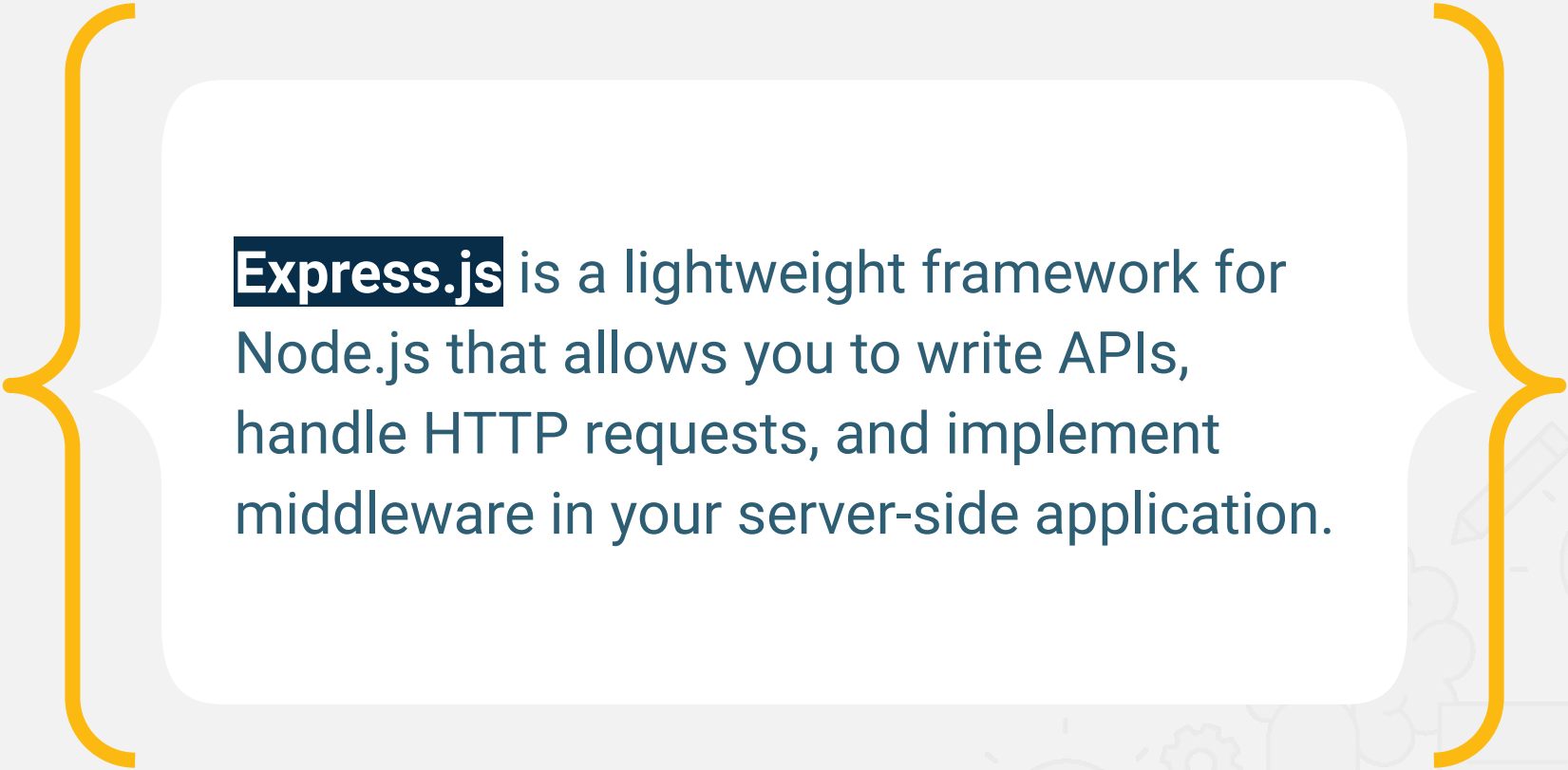
Server
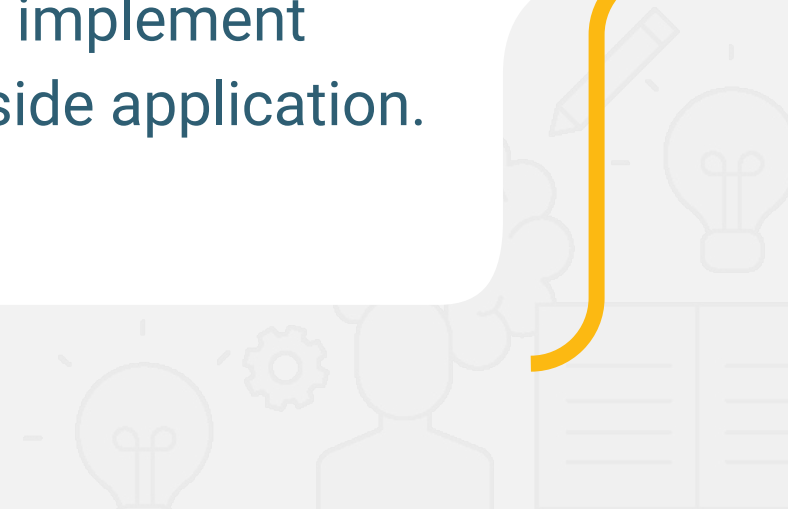
What is **Express.js?**

Express.js is a lightweight framework for Node.js that allows you to write APIs, handle HTTP requests, and implement middleware in your server-side application.

Express.js

1   **Express.js** exists on the back end of an application.

2   **Express.js** is considered the de facto standard for creating routes in **Node.js** applications.
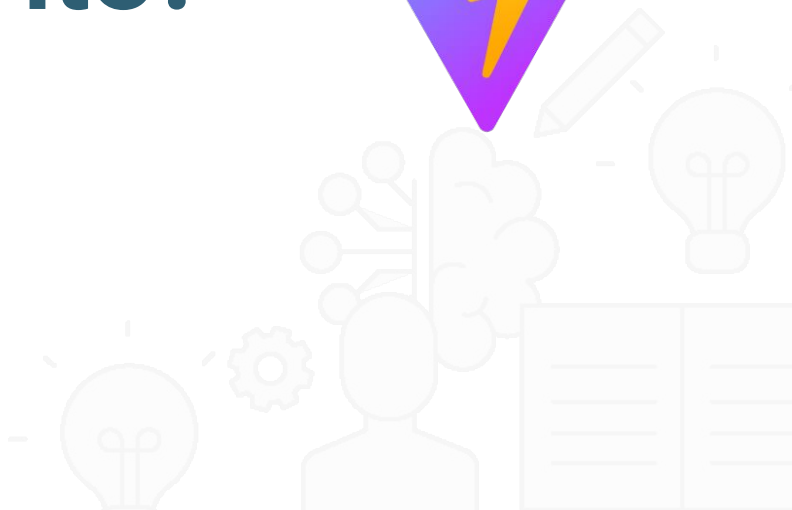
express
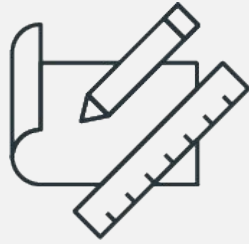
Tools

What is **Vite?**

What Is Vite?

When developing an application using TypeScript, HTML, and CSS, you will quickly discover that the TypeScript Compiler `(tsc)` will only compile the `.ts` files, which makes it tedious to get the correct files to talk to each other.

To avoid this problem, we will use a build tool called **Vite.**

Vite will bundle and run all of our client-side code together.

Async/await is a feature that makes asynchronous code look and behave more like synchronous code, improving the readability and maintainability of code that involves asynchronous operations. It's built on top of Promises, but it provides a more elegant way to work with them.

What are some differences between **Promises** and **async/await?**

Some of the Differences Between Promises and Async/Await:

Promises:

- Using `.then()` and `.catch()` to handle asynchronous operations can lead to nested structures and return new Promises in each step.

Async/await:

- Using **async** and **await** keywords leads to a more linear and readable code structure and assigns values directly from asynchronous operations.
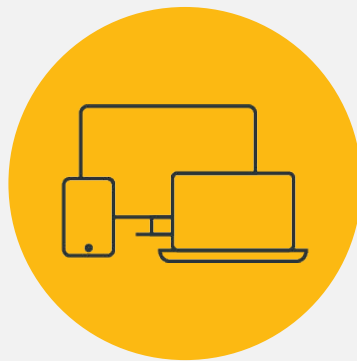
Instructor **Demonstration**
Mini-Project

Questions?

The End