

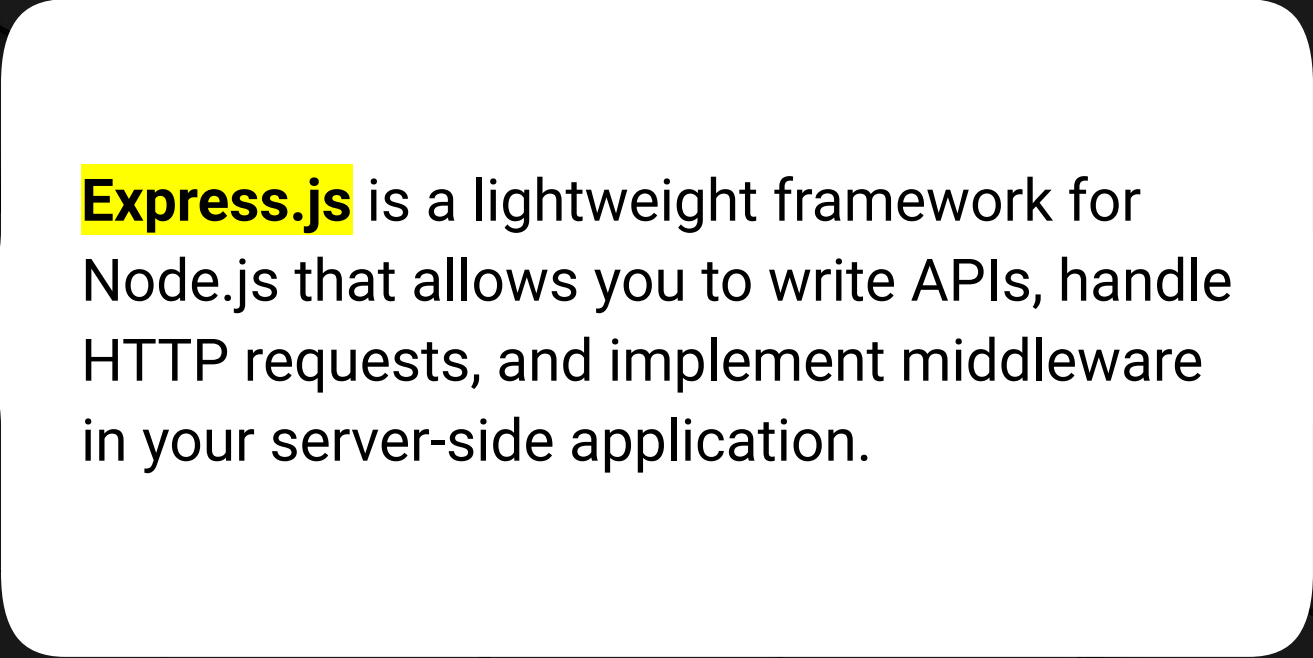
Coding Boot Camp

Module 11





**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

---



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



Express.js is considered the de facto standard for creating routes in Node.js applications.

express



**What is a route?**

# Routes

---

Routes are a lot like traffic lanes at an airport. Certain lanes are designated for dropping people off, picking up passengers, picking up luggage, and so on.



# Routes

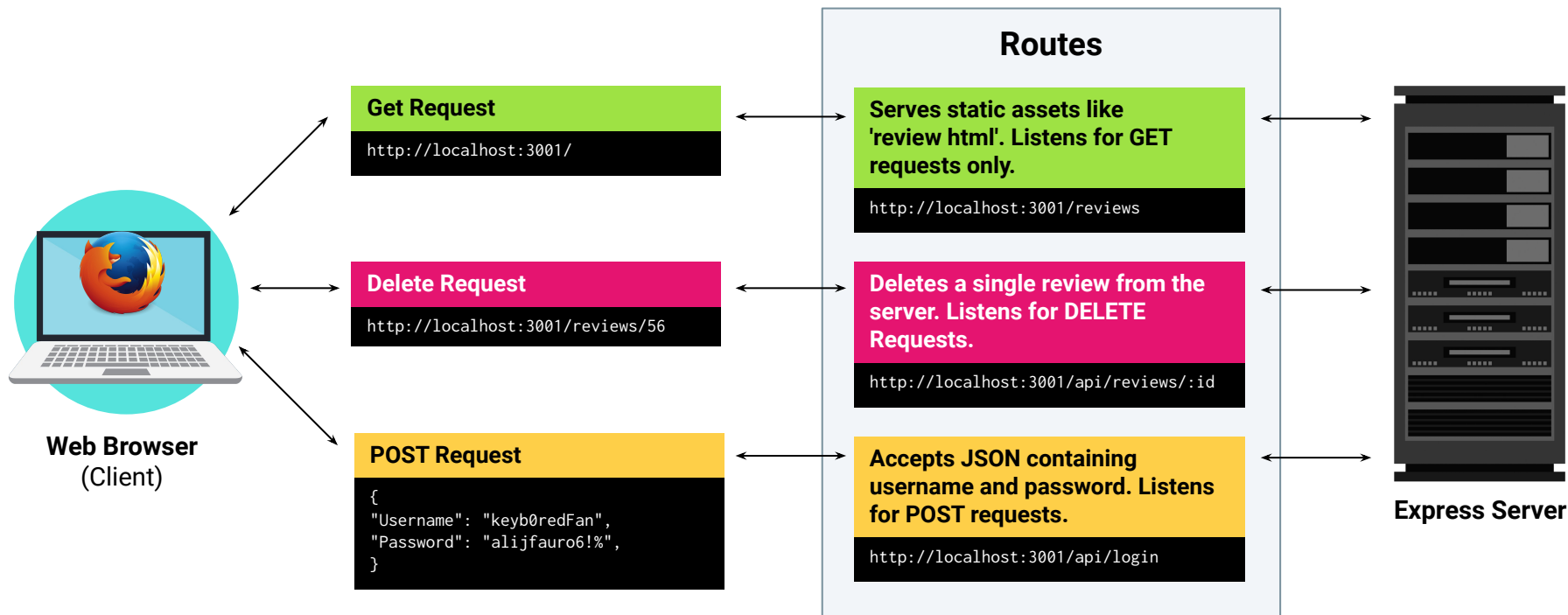
---

Similarly, routes allow us to send and receive data depending on which route and **HTTP method** we use. A route can be used for different kinds of requests, to create, read, update, and delete data.

POST	Submits data to the specified resource, often causing a change on the server.
GET	Retrieves a resource from the server.
DELETE	Deletes a specified resource.
PUT	Replaces a specified resource with a payload.

# Routes

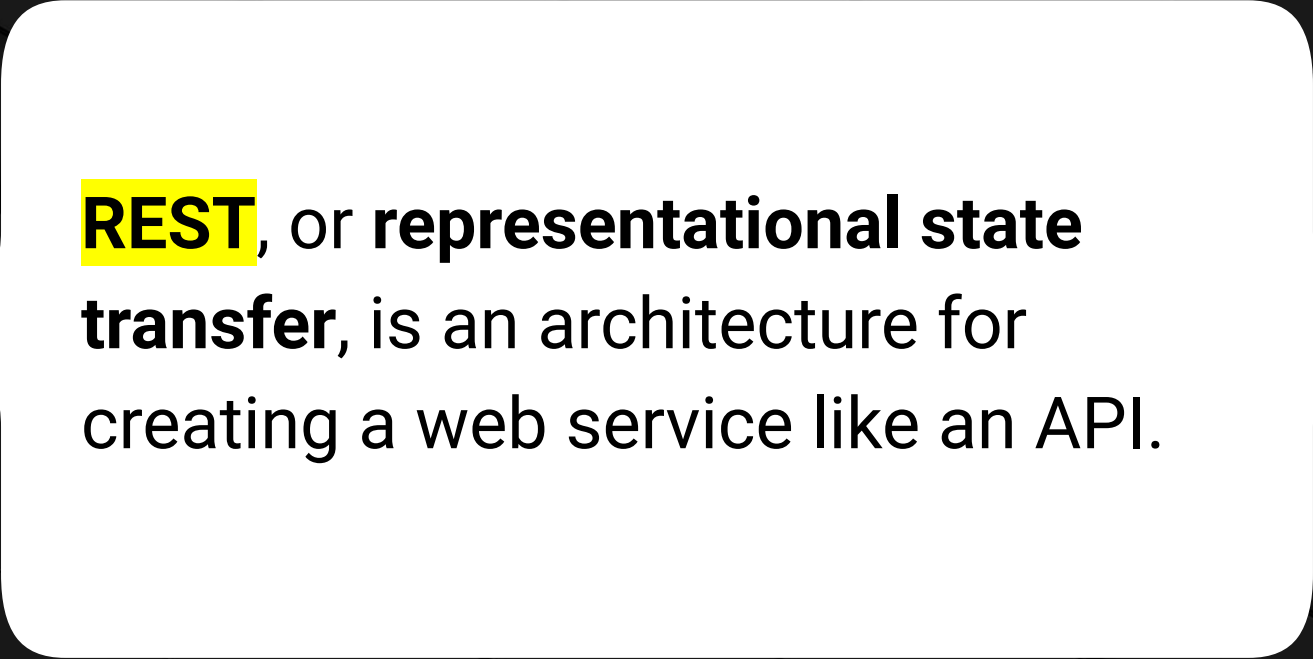
Here is an overview of how client-side requests are routed:







**What is a RESTful API?**



**REST**, or **representational state transfer**, is an architecture for creating a web service like an API.

# What is a RESTful API?

---

RESTful APIs must meet the following criteria:



Comprise clients, servers, resources and requests (via HTTP).



Use stateless communications between client and server.



Serve cached objects to reduce bandwidth.



Maintain a uniform interface between the client and the server so that they can evolve separately.



Optionally, can perform code on demand.



**What are the HTTP methods?**

# HTTP methods

---

You will use the following four main HTTP methods:

<b>POST</b>	Submits data to the specified resource, often causing a change on the server.
<b>GET</b>	Retrieves a resource from the server.
<b>DELETE</b>	Deletes a specified resource.
<b>PUT</b>	Replaces a specified resource with a payload.



**What does the code look like?**

# Code Snippets

---

Here we have an example of a few Express.js routes:



Use `get()`, `post()`, `delete()`, and similar methods to create routes.



The first argument is the path, `/api/reviews`.

```
// GET route for static homepage
app.get('/', (req, res) =>
  res.sendFile('index.html');

// GET route for all reviews
app.get('/api/reviews', (req, res) =>
  res.json(reviewData));
```

# Code Snippets (Continued)

---

Here we have an example of a POST route:



The **path** is the part of the route that comes after the base URL.



POST routes also accept the path as the first argument.



The second argument is a callback: `(req, res) => { }`.

```
// POST route to add a single review
app.post('/api/reviews', (req, res) => {
  const newReview = req.body
  writeFile(destination, newReview)

  res.json(`${req.method} received`);
});
```





**How does this relate to the front end?**

# Client-Side Requests

---

We use the Fetch API to make requests to the Express.js server.



We can create `fetch()` requests that the server-side routes understand and respond to.



POST requests will send a request body that we capture server-side.

```
// Fetch request to add a new pet
const addPet = (pet) => {
  fetch('/api/pets', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(pet),
  })
  .then((res) => res.json())
  .then((pets) => console.log(pets));
};
```



Making `fetch()` requests will be no different than making calls to a third-party API. The only difference is that this API will run locally.

# Resolving Requests

---



Requests must be concluded to prevent the client application from hanging indefinitely.



Methods attached to the response object allow us to conclude a request-response cycle.

```
app.put('/api/pets/:pet_id', (req, res) => {  
  // Logic to update a pet  
  res.json('Pet updated');  
});
```



# Instructor Demonstration

---

## Mini-Project