Express.js

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 backend of an application.

Express.js is considered the de facto standard for creating “routes”, or API “endpoints” in Node.js applications.

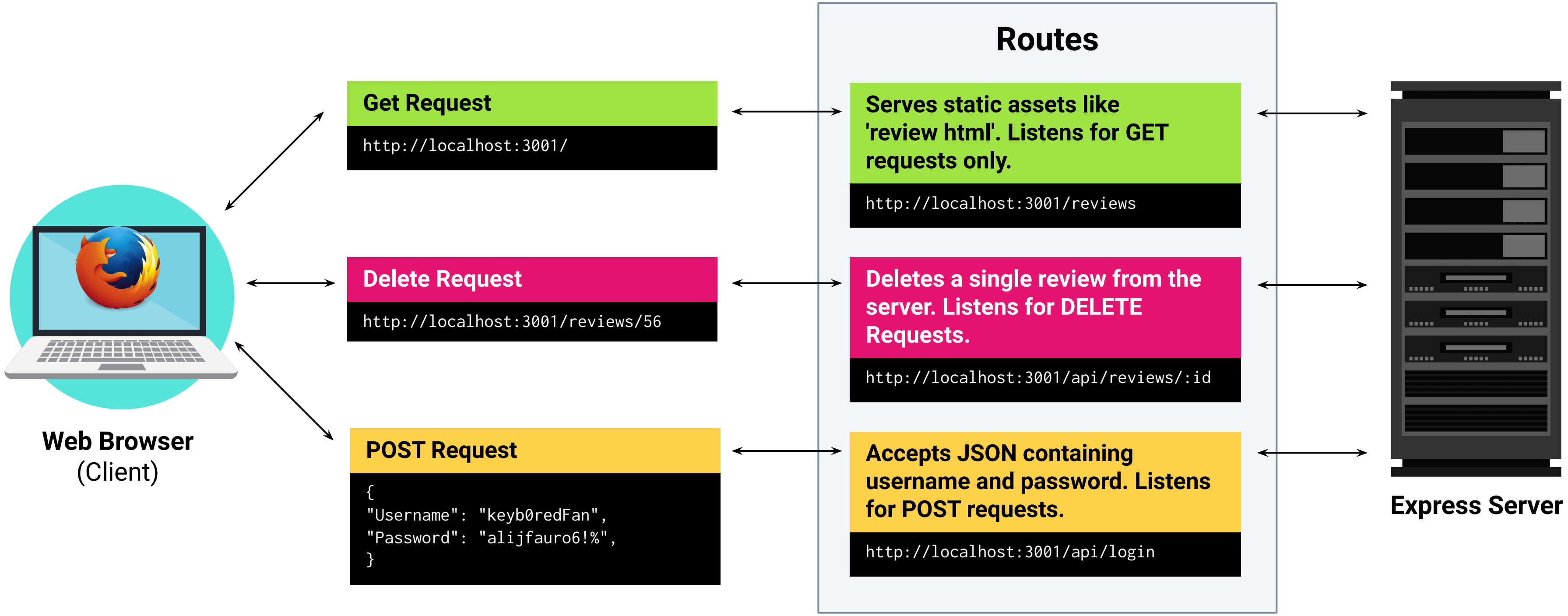
What is a “route” vs. an “endpoint”?

Route vs. Endpoint

- Route
  - Example:
    - http://www.mywebsite.com**/profile**
  - In the above example, the **/profile** part of the URL is the “route”
  - It controls the flow of a website and represents an individual page
- Endpoint
  - Example:
    - http://www.mywebiste.com/api/v1**/books/1029291**
  - In the above example the full URL is the “endpoint”, but the important part to pay attention to is:
    - **/books/1029291**
  - Typically are called only for data, not HTML content
  - A series of endpoints makes up an API

Routes

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



Web Browser
(Client)
Get Request
Delete Request
POST Request
Routes
Serves static assets like 'review html'. Listens for GET requests only.
Deletes a single review from the server. Listens for DELETE Requests.
Accepts JSON containing username and password. Listens for POST requests.
Express Server

```
http://localhost:3001/
```

```
http://localhost:3001/reviews/56
```

```
{
"Username": "keyb0redFan",
"Password": "alijfauro6!%",
}
```

```
http://localhost:3001/reviews
```

```
http://localhost:3001/api/reviews/:id
```

```
http://localhost:3001/api/login
```

What is a RESTful API?

REST, or **representational state transfer**, is an architectural design pattern 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:

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.

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
   writeToFile(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.

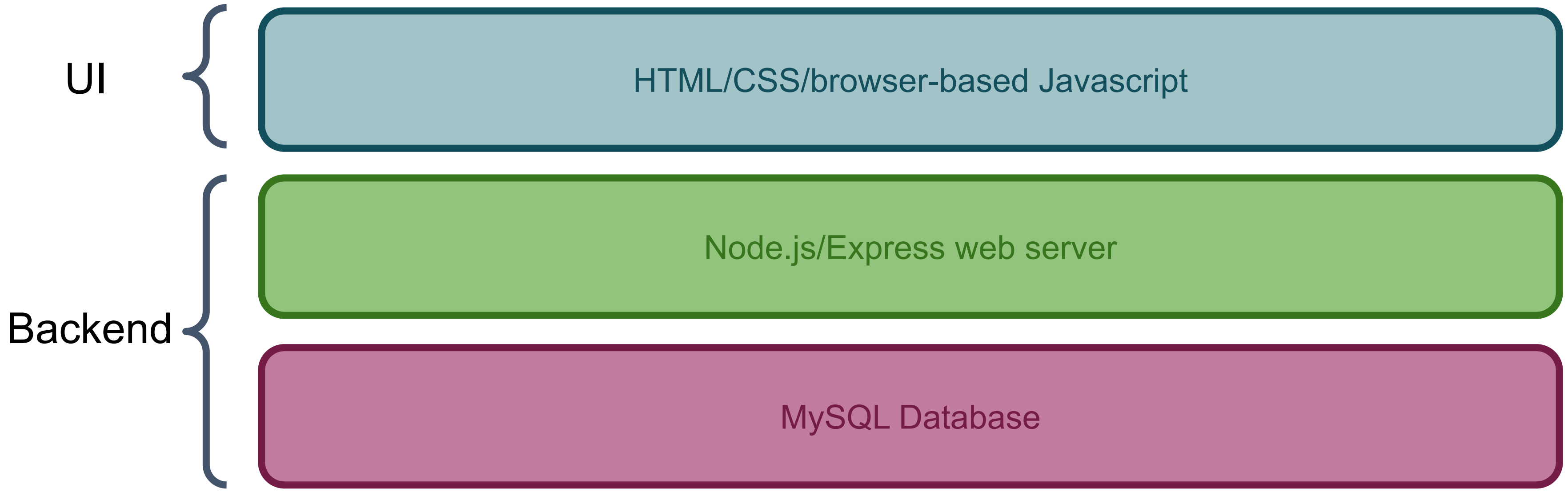
Methods attached to the response object allow us to conclude a request-response cycle.

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

REMEMBER!

In HTTP communications, there is a 1:1 relationship between requests and responses.

The Big Picture - The Stack



Instructor Demonstration

Mini-Project