Intro to Express.js

CS 390 – Web Application Development

J. Setpal

October 4, 2023

Outline

Outline

WIWYT – Express.js

- Express is pretty incredible: it allows us to develop full-fledged APIs like they're Hello World projects.

WIWYT – Express.js

- Express is pretty incredible: it allows us to develop full-fledged APIs like they're Hello World projects.
- Express is explicitly *unopiniontated*. This is important, since it allows us to self-select an implementation strategy for our application.

WIWYT – Middleware

- Express works fundamentally as an abstraction layer over the traditional API implementation. It's a big reason why it's so easy to work with.

WIWYT – Middleware

- Express works fundamentally as an abstraction layer over the traditional API implementation. It's a big reason why it's so easy to work with.
- Middleware allows us fine-grained control over the routing process within the API, enabling us to extend the Express functionality depending on the use-case.

Outline

What is an API?

API stands for **Application Programming Interface**.

What is an API?

API stands for **Application Programming Interface**.

Q: What is a User Interface (ex. GUI: Graphical User Interface) for?

What is an API?

API stands for **Application Programming Interface**.

Q: What is a User Interface (ex. GUI: Graphical User Interface) for?
A: Communicating between the user and the computer!

What is an API?

API stands for **Application Programming Interface**.

Q: What is a User Interface (ex. GUI: Graphical User Interface) for?
A: Communicating between the user and the computer!

Similarly; APIs allow for communication between two computers (usually a client and server).

What is an API?

API stands for **Application Programming Interface**.

Q: What is a User Interface (ex. GUI: Graphical User Interface) for?
A: Communicating between the user and the computer!

Similarly; APIs allow for communication between two computers (usually a client and server).

An endpoint is the network location of a resource or application.

What is an API?

API stands for **Application Programming Interface**.

Q: What is a User Interface (ex. GUI: Graphical User Interface) for?
A: Communicating between the user and the computer!

Similarly; APIs allow for communication between two computers (usually a client and server).

An endpoint is the network location of a resource or application. During development, the API endpoint we will use is `http://localhost:1337/`.

HTTP Request Methods

HTTP Methods define the type of action requested by a user at a given endpoint. The following HTTP methods are especially relevant:

Method	**Use**
GET	Retrieve information

HTTP Request Methods

HTTP Methods define the type of action requested by a user at a given endpoint. The following HTTP methods are especially relevant:

Method	**Use**
GET	Retrieve information
POST	Insert new information

HTTP Request Methods

HTTP Methods define the type of action requested by a user at a given endpoint. The following HTTP methods are especially relevant:

Method	**Use**
GET	Retrieve information
POST	Insert new information
PUT	Update information

HTTP Request Methods

HTTP Methods define the type of action requested by a user at a given endpoint. The following HTTP methods are especially relevant:

Method	**Use**
GET	Retrieve information
POST	Insert new information
PUT	Update information
DELETE	Delete a specified resource

HTTP Response Codes

Express.js allows us to build an API over HTTP.

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	Use
1xx	Information

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	**Use**
1xx	Information
2xx	Success

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	Use
1xx	Information
2xx	Success
3xx	Redirection Requests

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	Use
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	**Use**
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	**Use**
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	**Use**
200	OK

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	**Use**
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	**Use**
200	OK
400	Bad Request

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	Use
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	Use
200	OK
400	Bad Request
401	Unauthorized

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	**Use**
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	**Use**
200	OK
400	Bad Request
401	Unauthorized
403	Forbidden

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	**Use**
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	**Use**
200	OK
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	Use
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	Use
200	OK
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
418	I'm a Teapot

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	Use
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	Use
200	OK
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
418	I'm a Teapot
500	Internal Server Error

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	**Use**
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	**Use**
200	OK
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
418	I'm a Teapot
500	Internal Server Error
502	Bad Gateway

HTTP Response Codes

Express.js allows us to build an API over HTTP. Each response sent by our API has a response code.

Response codes denotes the response type by status level:

Status Level	**Use**
1xx	Information
2xx	Success
3xx	Redirection Requests
4xx	Client Error
5xx	Server Error

Some useful status codes:

Status Code	**Use**
200	OK
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
418	I'm a Teapot
500	Internal Server Error
502	Bad Gateway
503	Service Unavailable

Outline

Basic Routing

Express relies on routing to determine how the API responds to a client request.

Basic Routing

Express relies on routing to determine how the API responds to a client request.

Syntax: `app.<method>(<path>, <middleware>);`
Example: `app.get('/', (req, res) => { res.send('Hello World!'); });`
The above example responds to a `GET` request sent to the root endpoint.

Basic Routing

Express relies on routing to determine how the API responds to a client request.

Syntax: `app.<method>(<path>, <middleware>);`
Example: `app.get(‘/’, (req, res) => { res.send(‘Hello World!’); });`
The above example responds to a `GET` request sent to the root endpoint.

We can also use `all` in place of a method, to respond to every method with a single function.

MVP from Monday (Express.js Version)

If you can view this screen, I am making a mistake.

Static Files

The syntax for serving a static file is straightforward:

```
// ... some code
app.use('/location', express.static('path/to/dir'))
// ... more code
```

Static Files

The syntax for serving a static file is straightforward:

```
// ... some code
app.use('/location', express.static('path/to/dir'))
// ... more code
```

The directory is crucial to **containerize** file serving!

Static Files

The syntax for serving a static file is straightforward:

```
// ... some code
app.use('/location', express.static('path/to/dir'))
// ... more code
```

The directory is crucial to **containerize** file serving!

This can also be used to serve HTML files, by making a GET request to the endpoint followed by the path.

Static Files

The syntax for serving a static file is straightforward:

```
// ... some code
app.use('/location', express.static('path/to/dir'))
// ... more code
```

The directory is crucial to **containerize** file serving!

This can also be used to serve HTML files, by making a `GET` request to the endpoint followed by the path.

Alternatively, you can send a file without exposing a directory:
Syntax: `res.sendFile('path/to/file.html');`

Route Chaining

Implementing multiple methods for the same route requires re-specifying the endpoint. This violates D.R.Y.

Route Chaining

Implementing multiple methods for the same route requires re-specifying the endpoint. This violates D.R.Y.

One solution is route-chaining; multiple methods by specifying `route` outside the methods.
Syntax: `app.route('/path').get(f).post(f).put(f).delete(f);`

Route Chaining

Implementing multiple methods for the same route requires re-specifying the endpoint. This violates D.R.Y.

One solution is route-chaining; multiple methods by specifying `route` outside the methods.
Syntax: `app.route('/path').get(f).post(f).put(f).delete(f);`

Instead of:

```
app.get('/path', f);
app.post('/path', f);
app.put('/path', f);
app.delete('/path', f);
```

Where `f` is a function handling the operation for the route.

Module-Based Routing

Routing scales signifitantly as the API's complexity increases, even with chaining.

Module-Based Routing

Routing scales signifitantly as the API's complexity increases, even with chaining.

We can explicitly set up modules for certain routes (ex. `'/v1/'`) and integrate it to the main module, as a way of structuring the application.

Module-Based Routing

Routing scales signifitantly as the API's complexity increases, even with chaining.

We can explicitly set up modules for certain routes (ex. '`/v1/`') and integrate it to the main module, as a way of structuring the application.

We can export it the same way as a normal node module, by adding `exports.<route>` and importing the module path with `require`.

CookieCutter Express.js

`express-generator` allows us to quickly generate a skeletal workflow with recommended best practices.

CookieCutter Express.js

`express-generator` allows us to quickly generate a skeletal workflow with recommended best practices.

We can create it using: `npx express-generator --view pug`
On older node versions: `npm i -g express-generator ; express`

CookieCutter Express.js

`express-generator` allows us to quickly generate a skeletal workflow with recommended best practices.

We can create it using: `npx express-generator --view pug`
On older node versions: `npm i -g express-generator ; express`

We'll modify it to:

- Remove `bin/`.

CookieCutter Express.js

`express-generator` allows us to quickly generate a skeletal workflow with recommended best practices.

We can create it using: `npx express-generator --view pug`
On older node versions: `npm i -g express-generator ; express`

We'll modify it to:

- Remove `bin/`.
- Prune `app.js`.

CookieCutter Express.js

`express-generator` allows us to quickly generate a skeletal workflow with recommended best practices.

We can create it using: `npx express-generator --view pug`
On older node versions: `npm i -g express-generator ; express`

We'll modify it to:

- Remove `bin/`.
- Prune `app.js`.
- Add `nodemon` as a dev dependency.

CookieCutter Express.js

`express-generator` allows us to quickly generate a skeletal workflow with recommended best practices.

We can create it using: `npx express-generator --view pug`
On older node versions: `npm i -g express-generator ; express`

We'll modify it to:

- Remove `bin/`.
- Prune `app.js`.
- Add `nodemon` as a dev dependency.
- Update `package.json` to reflect the above changes.

Let’s Build a Static File Server!

If you can view this screen, I am making a mistake.

Outline

Middleware – Understanding Hooks

Idea: Everything is Middleware!

Middleware – Understanding Hooks

Idea: Everything is Middleware!

Q: What happens when we run an Express function? How does Express interpret it?

Middleware – Understanding Hooks

Idea: Everything is Middleware!

Q: What happens when we run an Express function? How does Express interpret it?
A: It runs a series of functions sequentially - like traversing a linked list.

Middleware – Understanding Hooks

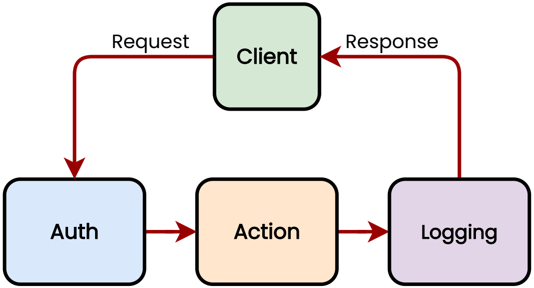
Idea: Everything is Middleware!

Q: What happens when we run an Express function? How does Express interpret it?
A: It runs a series of functions sequentially - like traversing a linked list. It propagates forward when `next();` is called.

Middleware – Understanding Hooks

Idea: Everything is Middleware!

Q: What happens when we run an Express function? How does Express interpret it?
A: It runs a series of functions sequentially - like traversing a linked list. It propagates forward when `next();` is called. Once all function calls are completed, the response is returned.



Middleware – Syntax

Let's break down some sample code:

```
// ... some code
app.get('/', f, (req, res) => {
                    res.send('Hello from the Express
                        API!!');
            })

function f(req, res, next) {
          console.log('f');
          next();
}
// ... some code
```

Middleware – Passing Values Between Functions

Passing data is incredibly important; it's what allows functions to communicate.

Middleware – Passing Values Between Functions

Passing data is incredibly important; it's what allows functions to communicate. Here's how we do it:

```
// ... some code
function f(req, res, next) {
        console.log('f');
        req.f = true;
        next();
}

app.get('/', f, (req, res) => {
                  console.log(`${req.f}`);
                  res.send('Hello from the Express
                      API!!');
           })
// ... some code
```

Middleware – Some Nuance

There's **two** pitfalls to avoid. Firstly:

- `next()` is **not** a `return`. It remains in the stack, and is called at the end of the chain.

Middleware – Some Nuance

There's **two** pitfalls to avoid. Firstly:

- `next()` is **not** a `return`. It remains in the stack, and is called at the end of the chain.
- We can't update the {`req, res`} variables after next is called. At the end of the chain, the result is sent to the client.

Middleware – Some Nuance

There's **two** pitfalls to avoid. Firstly:

- `next()` is **not** a `return`. It remains in the stack, and is called at the end of the chain.
- We can't update the {`req, res`} variables after next is called. At the end of the chain, the result is sent to the client.

Secondly:

- Middleware is called in order of declaration.

Middleware – Some Nuance

There's **two** pitfalls to avoid. Firstly:

- `next()` is **not** a `return`. It remains in the stack, and is called at the end of the chain.
- We can't update the {`req, res`} variables after next is called. At the end of the chain, the result is sent to the client.

Secondly:

- Middleware is called in order of declaration.
- Don't accidentally call authentication after the action!

Let's Implement Server Logging!

If you can view this screen, I am making a mistake (again).

Outline

Reminder – Project Proposal

Due on 10th October, 2023 @ 11:59pm.

Project Template in Brightspace under CONTENT > Project > Project Proposal Template.

Teams are ideally between 2-4 contributors. If you'd want to deviate from this, please email us or post on piazza!

Thank you!

Have an awesome rest of your day!

Slides:

`https://cs.purdue.edu/homes/jsetpal/slides/intro-express.pdf`

If anything's incorrect or unclear, please ping `jsetpal@purdue.edu`
I'll patch it ASAP.