A simple HTTP server

Node HTTP module

```javascript
const http = require('http');
const server = http.createServer((request, response) => {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end(`Don’t Panic`);  // Manually construct the response
}).listen(5042);
console.log('Listening on port %d', server.address().port);
```

In action:

```
$ curl http://localhost:5042/
Don’t Panic
```

With this low-level interface we are responsible for everything, interpreting the request and building the entire response. As you expect there is an opportunity for frameworks that implement the common features of a web server.
A simple HTTP server with Express

```javascript
const http = require('http');
const express = require('express');

const app = express();
app.get('/', (request, response) => {
  response.send("Don’t Panic");
});

app.get('/:name', (request, response) => {
  response.send("Don’t panic "+ request.params.name);

const server = http.createServer(app).listen(5042);
console.log('Listening on port %d', server.address().port);
```

There is a one-to-one mapping between API routes and Express routes

Express provides `response.send`, which automatically sends the headers, sets the correct content type, and calls `end()` (so it will only be called once during a response). It will even convert JavaScript objects to JSON for us!

We build up our server step by step by adding more routes (implementing more controllers).
Example middleware:
- body-parser: Parse JSON request body
- static: Return static assets, like HTML or CSS files
Aspect-oriented Programming (AOP)

- Design pattern for implementing “cross-cutting” concerns
  - Middleware is an example of AOP
- “Cross cutting” concerns are those that affect many parts (or concerns) of the code
  - Many requests require body parsing
- AOP is a general set of techniques for DRYing up “cross cutting” concerns

We will see other examples of "cross cutting" concerns soon, notably in implementing validations for models (in the MVC sense).

Advice is a specific piece of code that implements a cross-cutting concern
Pointcuts are the places you want to “inject” advice at runtime
Advice+Pointcut = Aspect
The Express routes often function at the controller (in the MVC sense). What about the Model?
In this context, the model is a movie. There is no explicit model class, just a JavaScript object. And for such a simple application in which we are not persisting any changes (that is ratings don't persist through restarting the server), we might not need much more. But as we want to add features, we will quickly find that we could benefit from established design patterns and library support.

---

**Movie model (M in MVC)**

Movie “resource” is a simple JavaScript object

Good enough for now, but what about?

- Validate user rating is 0-5? **Can’t trust the client!**
- Express associations between models
- Support different persistence layers (e.g. databases)

*We can use ORMs and other libraries to provide this “cross cutting” functionality*
The models are typically the RESTful resources

<table>
<thead>
<tr>
<th>Route</th>
<th>Controller Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /movies</td>
<td>Create new movie from request data</td>
</tr>
<tr>
<td>GET /movies/:id</td>
<td>Read data of movie with id == :id</td>
</tr>
<tr>
<td>PUT /movies/:id</td>
<td>Update movie with id == :id from request data</td>
</tr>
<tr>
<td>DELETE /movies/:id</td>
<td>Delete movie with id == :id</td>
</tr>
<tr>
<td>GET /movies</td>
<td>List (read) all movies</td>
</tr>
</tbody>
</table>

A single model: Movie
OO modeling: A tiny bit of UML

- **Unified Modeling Language**: A visual language for describing artifacts in a OO system
- UML is expansive, we are focused just on class modeling features

Relationships are shown by the arrows (and fill)...

Adapted from Armando Fox and David Patterson (Berkeley cs169) under CC-BY-SA-NC license.
Adapted from Armando Fox and David Patterson (Berkeley cs169) under CC-BY-SA-NC license.
First associations...
A movie has many genres
There is a many-to-many relationship between Users and Movies via the ratings.
Often called a “has many through” association.

Adapted from Armando Fox and David Patterson (Berkeley cs169) under CC-BY-SA-NC license.
The nouns in the user stories (blue) often correspond to models, while the verbs (red) correspond to associations between models and/or methods on the models.
Student Advice: CRC cards and designing up front

- “Having a solid design & schema saved us a lot of pain”
- “MVC's separation of concerns really made for a nice app structure”
- “Designing rich client-side and server-side in SOA made it easy to decouple development”
- “We wish we had designed the object model and schema more thoroughly”

Adapted from Armando Fox and David Patterson (Berkeley cs169) under CC-BY-SA-NC license.