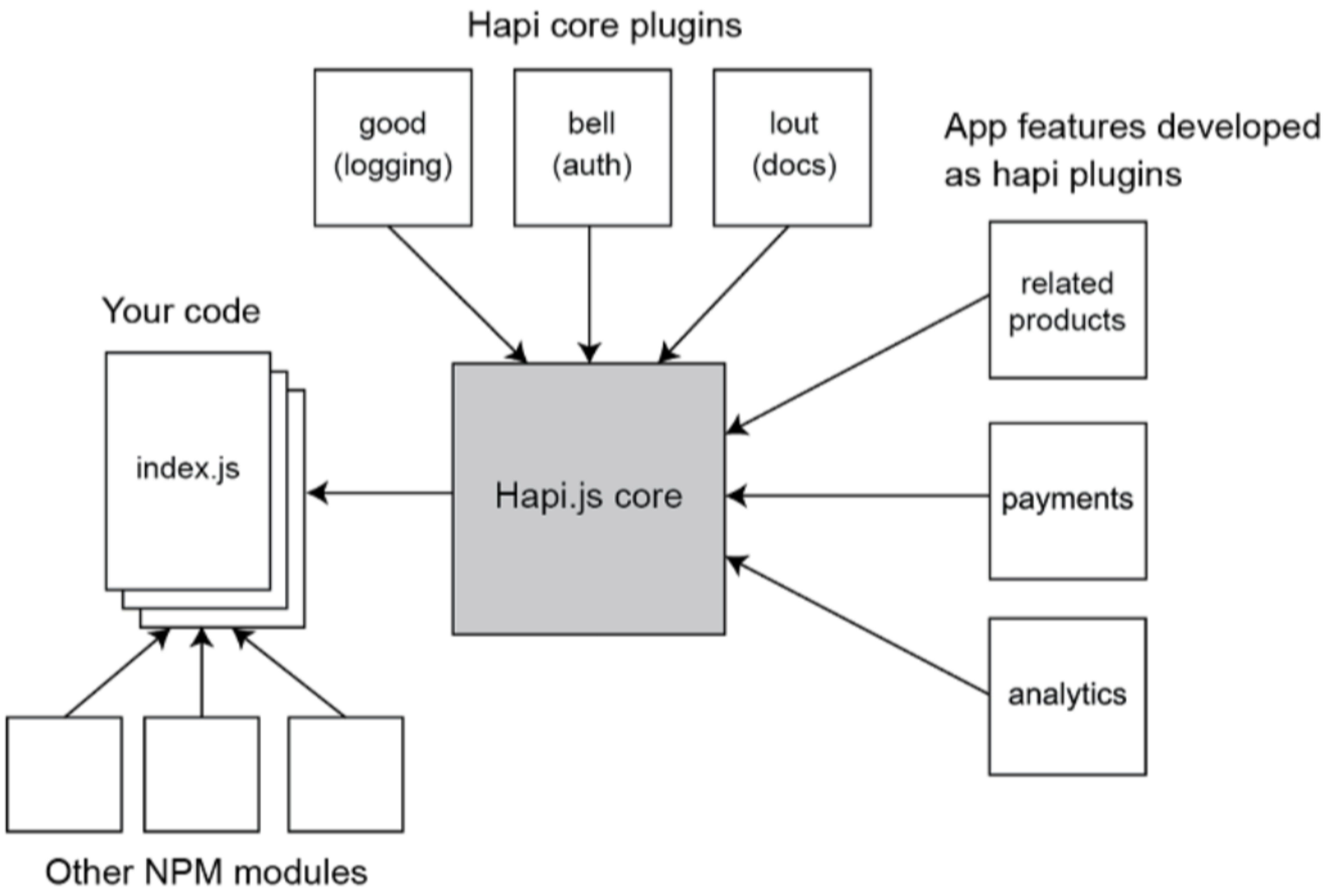
HAPI Building Blocks

Example Hapi Application Structure



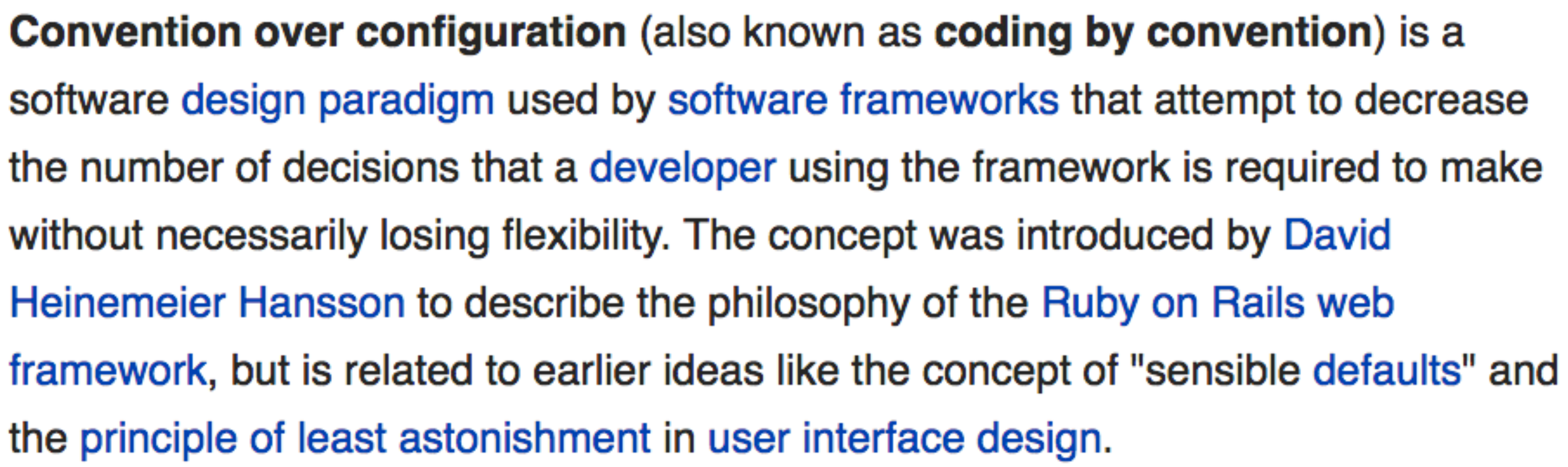
Convention over Configuration

I LOVE TO WRITE A BUNCH OF CONFIGURATION FILES BEFORE WRITING ACTUAL CODE

- Said no one ever

Reasonable defaults
Only specify the unconventional bits
Reduce number of decisions to be made
Eliminate distractions

Convention over Configuration



Convention over configuration (also known as **coding by convention**) is a software design paradigm used by software frameworks that attempt to decrease the number of decisions that a developer using the framework is required to make without necessarily losing flexibility. The concept was introduced by David Heinemeier Hansson to describe the philosophy of the Ruby on Rails web framework, but is related to earlier ideas like the concept of "sensible defaults" and the principle of least astonishment in user interface design.

https://en.wikipedia.org/wiki/Convention_over_configuration

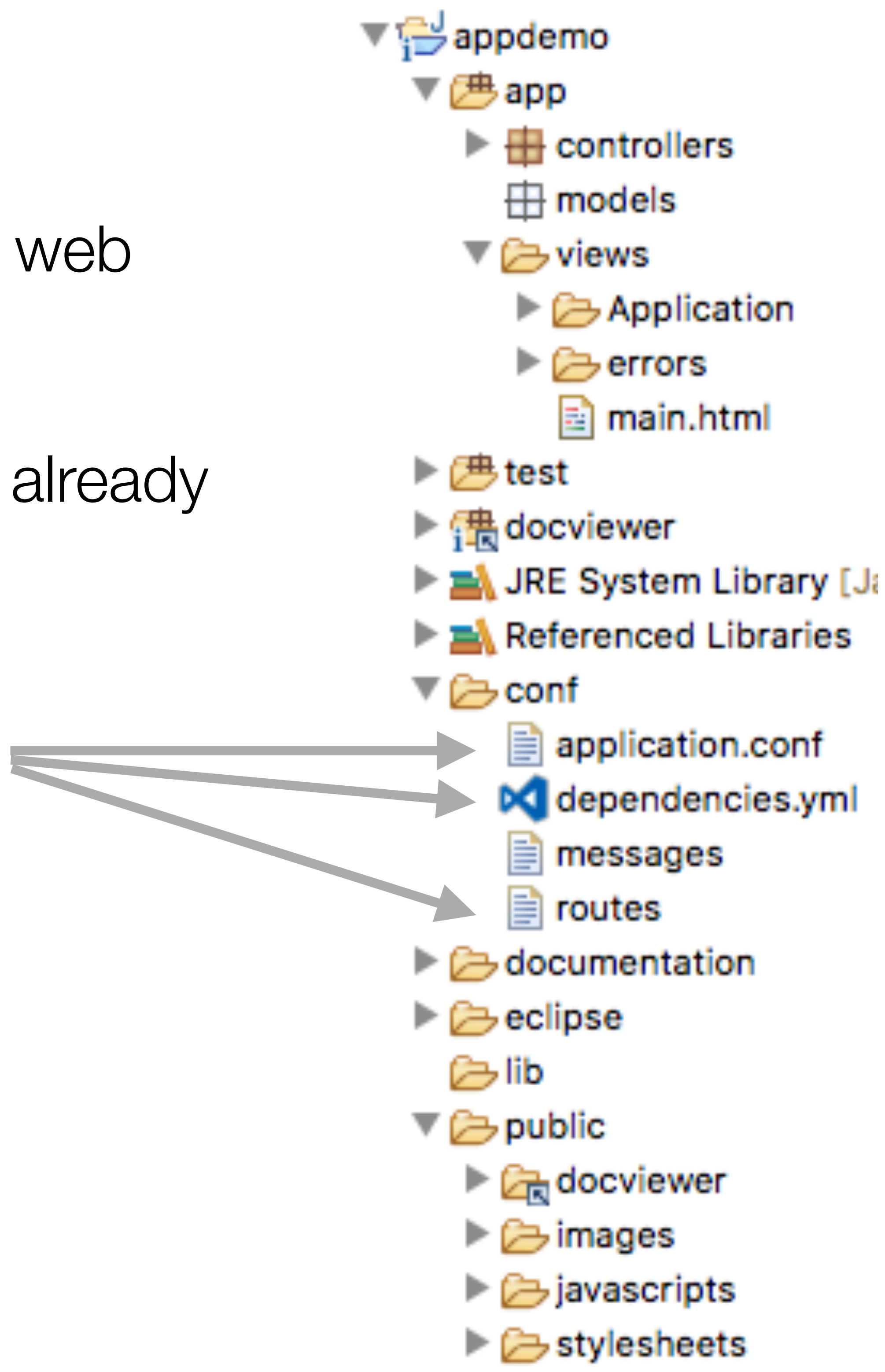
Convention over Configuration in Play 1

play new

Generates a complete working web app

Considerable range of defaults already configured to 'just work'

Default can be changed by



Convention over Code - Example

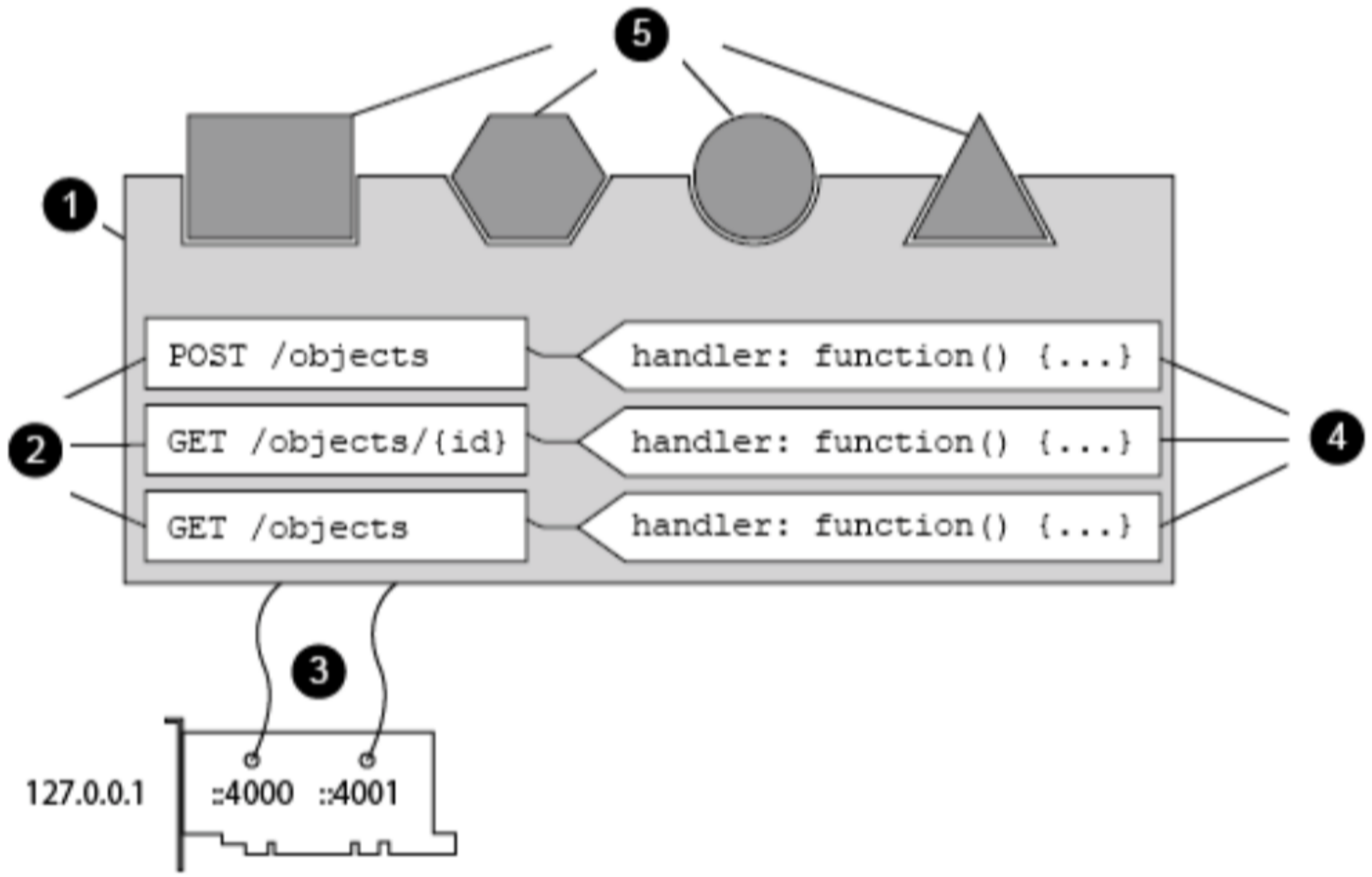
```
const bean = require('jellybean');

bean.setName('Coffee');
bean.setColor('brown');
bean.setSpeckles(false);
```

- Verbose - 3 method calls on the bean object to configure the jellybean.
- Configuration part of the program logic

```
const bean = require(‘jellybean’);

const options = {
  name: 'Tutti Frutti',
  color: 'mixed',
  speckles: true
};

bean.config(options);
```

- config method takes options argument
- More flexible because it separates the configuration from the code
- Place all the configurations of jellybeans in a separate file and include them.
- To change the configurations later just update the config.

Hapi Building Blocks



5
1
POST /objects
GET /objects/{id}
GET /objects
2
handler: function() {...}
handler: function() {...}
handler: function() {...}
4
3
127.0.0.1
::4000
::4001

1. Server
2. Routes
3. Connections
4. Handlers
5. Plugins

Server

index.js

```
'use strict';

const Hapi = require('hapi');

var server = new Hapi.Server();
server.connection({ port: process.env.PORT || 4000 });

server.start(err => {
  if (err) {
    throw err;
  }

  console.log('Server listening at:', server.info.uri);
});
```

- A server is the container for the hapi application.
- All other Hapi objects are created or used in the context of a server.
- A hapi server doesn't directly listen on a network port.
- Make connections from your server so the app can speak to the outside world.

Routes

- Routes in hapi are a way of telling the framework that you're interested in certain types of request.
- Create a route with a set of options, including the HTTP verb (such as GET, POST) and path (for example /about) that you wish to respond to, and add it to a server.

routes.js

```
const Controller = require('./controller.js');

module.exports = [

  { method: 'GET', path: '/', config: Controller.index },

];
```

Connection

index.js

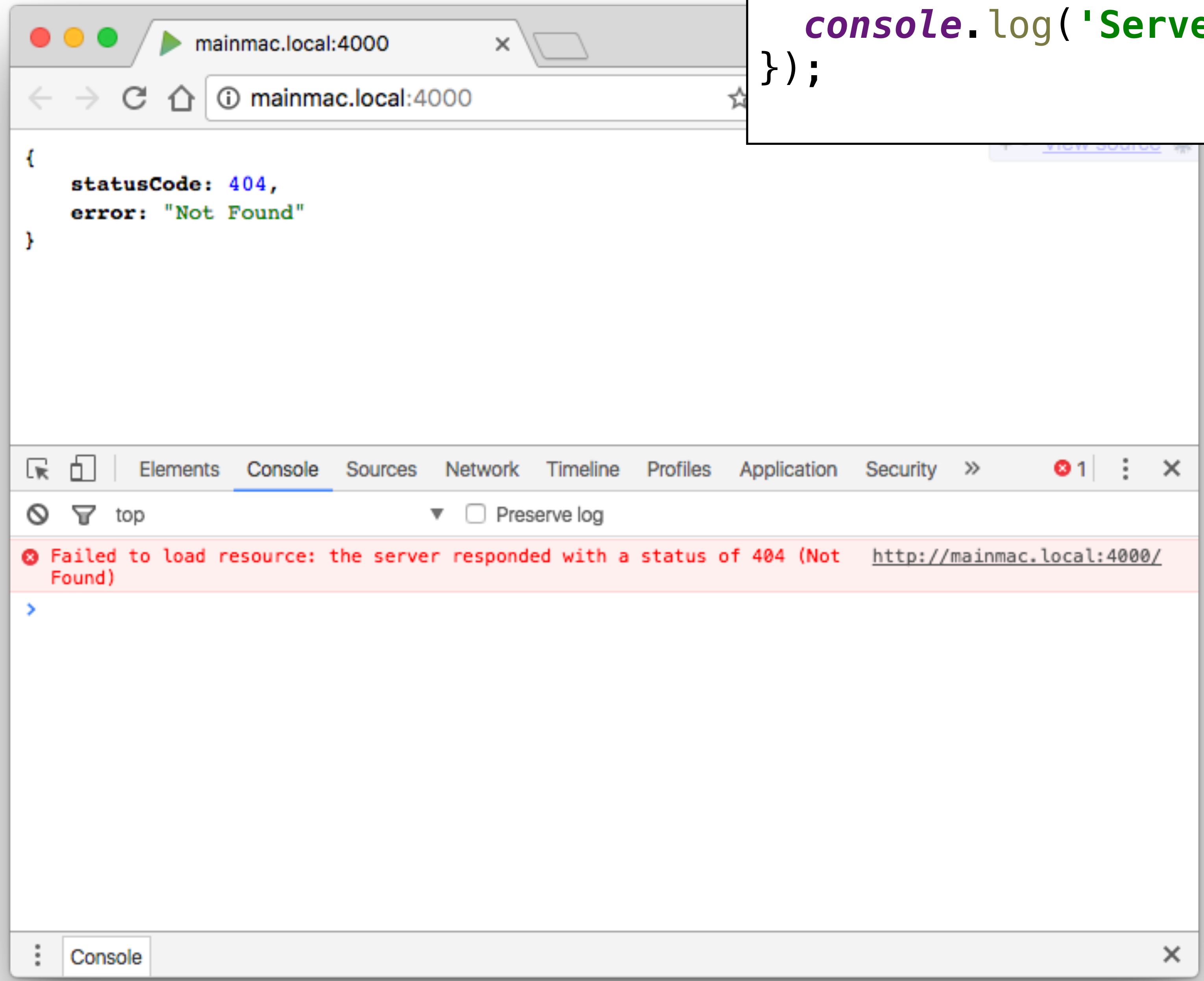
```
'use strict';

const Hapi = require('hapi');

var server = new Hapi.Server();
server.connection({ port: process.env.PORT || 4000 });

server.start(err => {
  if (err) {
    throw err;
  }

  console.log('Server listening at:', server.info.uri);
});
```

- Use connections to attach a hapi server to a network interface,
- It can start accepting incoming requests on this interface
- Connections allow a single hapi server listen on multiple ports

Connection but no Routes Configured

index.js

```
'use strict';

const Hapi = require('hapi');

var server = new Hapi.Server();
server.connection({ port: process.env.PORT || 4000 });

server.start(err => {
  if (err) {
    throw err;
  }

  console.log('Server listening at:', server.info.uri);
});
```



```
{
    statusCode: 404,
    error: "Not Found"
}
```

```
Failed to load resource: the server responded with a status of 404 (Not Found)    http://mainmac.local:4000/
```

Configuring Routes

- When a new request arrives at the server, hapi will attempt to find one of the routes that matches the request.
- If it successfully pairs up the request with one of your routes, it will look to your route handler for how to handle the request.

routes.js

```
const Controller = require('./controller.js');

module.exports = [

  { method: 'GET', path: '/', config: Controller.index },

];
```

index.js

```
'use strict';

const Hapi = require('hapi');

var server = new Hapi.Server();
server.connection({ port: process.env.PORT || 4000 });

server.route(require('./routes'));

server.start(err => {
  if (err) {
    throw err;
  }

  console.log('Server listening at:', server.info.uri);
});
```

Starting the Server

- server.start called when server is launched.
- If there us an error on startup, error details passed in ‘err’ parameter.
- If no error, the server is running, awaiting requests and dispatching to handlers based on the installed routes

index.js

```
'use strict';

const Hapi = require('hapi');

var server = new Hapi.Server();
server.connection({ port: process.env.PORT || 4000 });

server.route(require('./routes'));

server.start(err => {
  if (err) {
    throw err;
  }

  console.log('Server listening at:', server.info.uri);
});
```

Handlers

- Handlers are the way to tell hapi how it should respond to an HTTP request.
- A handler can take several forms.
- The simplest handler is defined as a JavaScript function with access to a request object and a reply interface.
- The request object provides details about the request.
- Use the reply interface to respond to the request

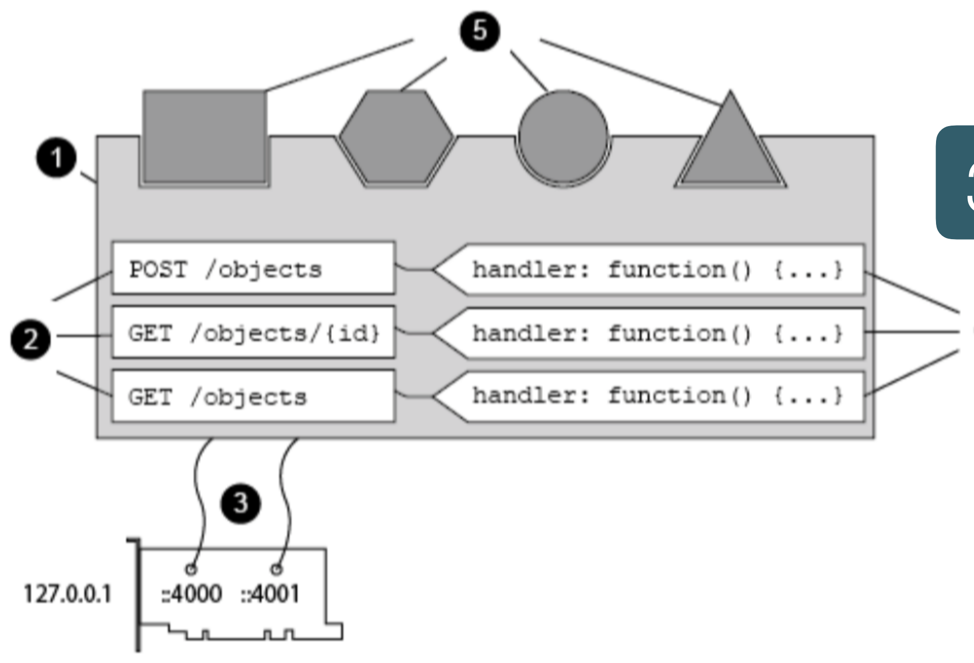
routes.js

```
const Controller = require('./controller.js');

module.exports = [

  { method: 'GET', path: '/', config: Controller.index },

];
```

controller.js

```
exports.index = {

  handler: function (request, reply) {
    reply('Hello!');
  }

};
```

1. Servers
2. Connections
3. Routes
4. Handlers



5
1
POST /objects
GET /objects/{id}
GET /objects
handler: function() {...}
handler: function() {...}
handler: function() {...}
2
4
3
127.0.0.1
::4000
::4001

controller.js

```
exports.index = {

  handler: function (request, reply) {        4
    reply('Hello!');
  }

};
```

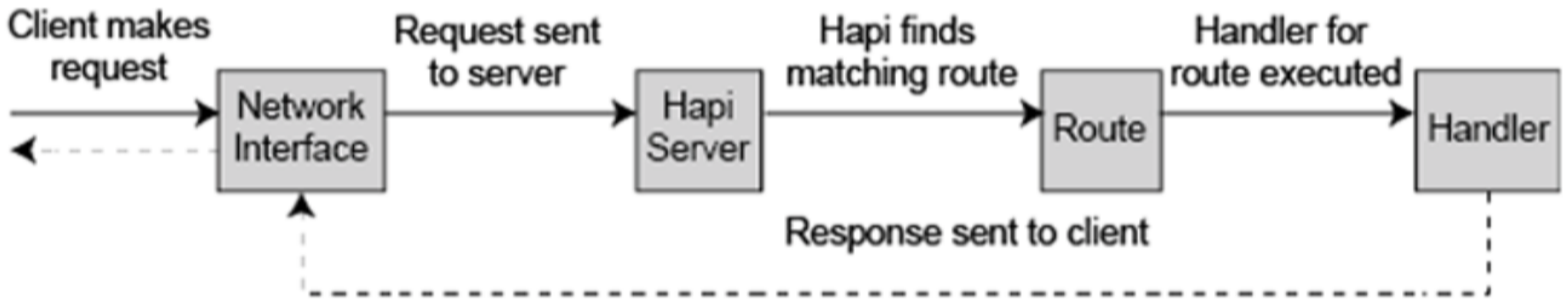
routes.js

```
const Controller = require('./controller.js');

module.exports = [

3  { method: 'GET', path: '/', config: Controller.index },
                                                4
];
```

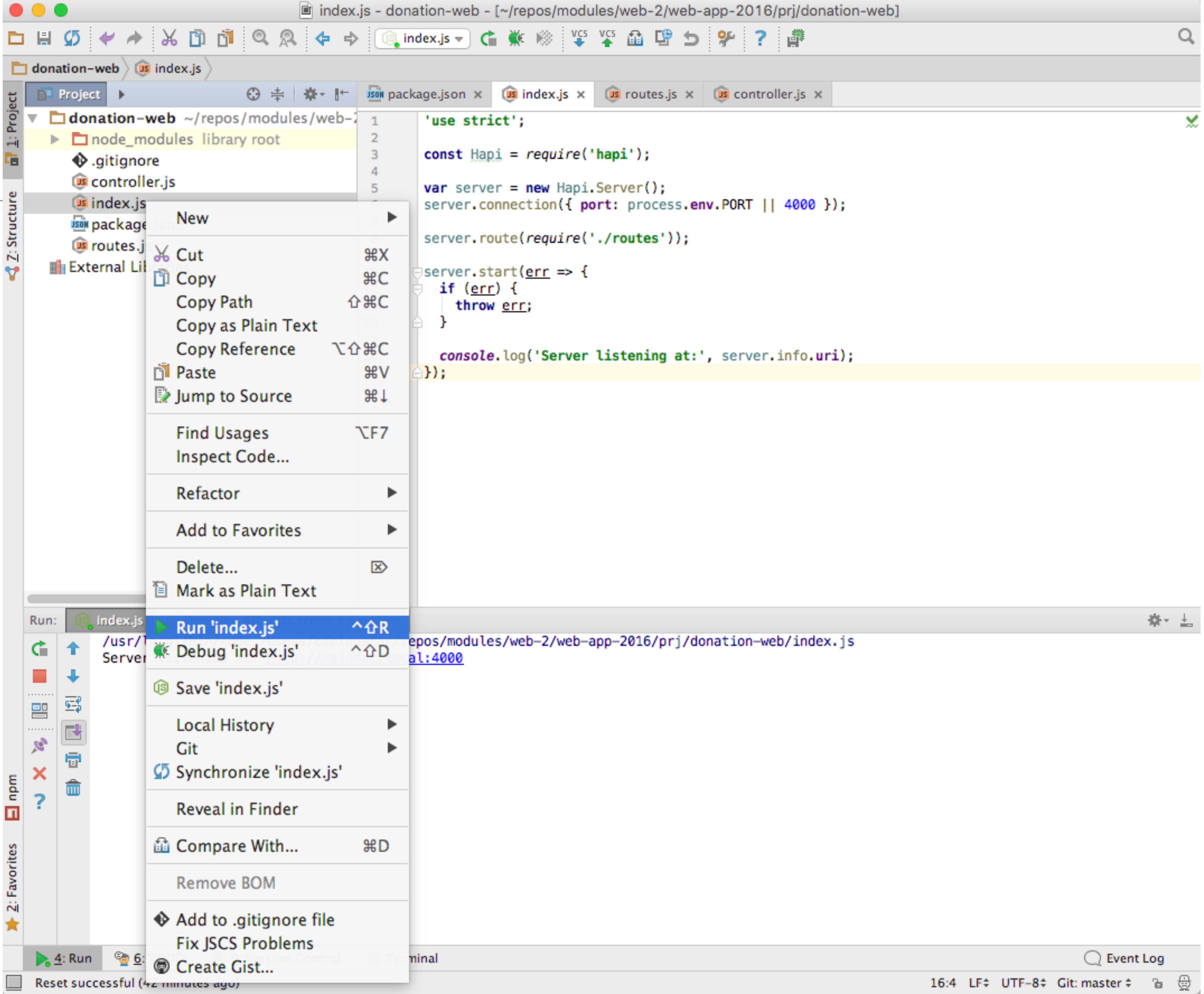
index.js

```
'use strict';

const Hapi = require('hapi');
                                   1
var server = new Hapi.Server();
server.connection({ port: process.env.PORT || 4000 });   2

3 server.route(require('./routes'));

server.start(err => {
  if (err) {
    throw err;          1
  }

  console.log('Server listening at:', server.info.uri);
});
```

Hapi Request Handing

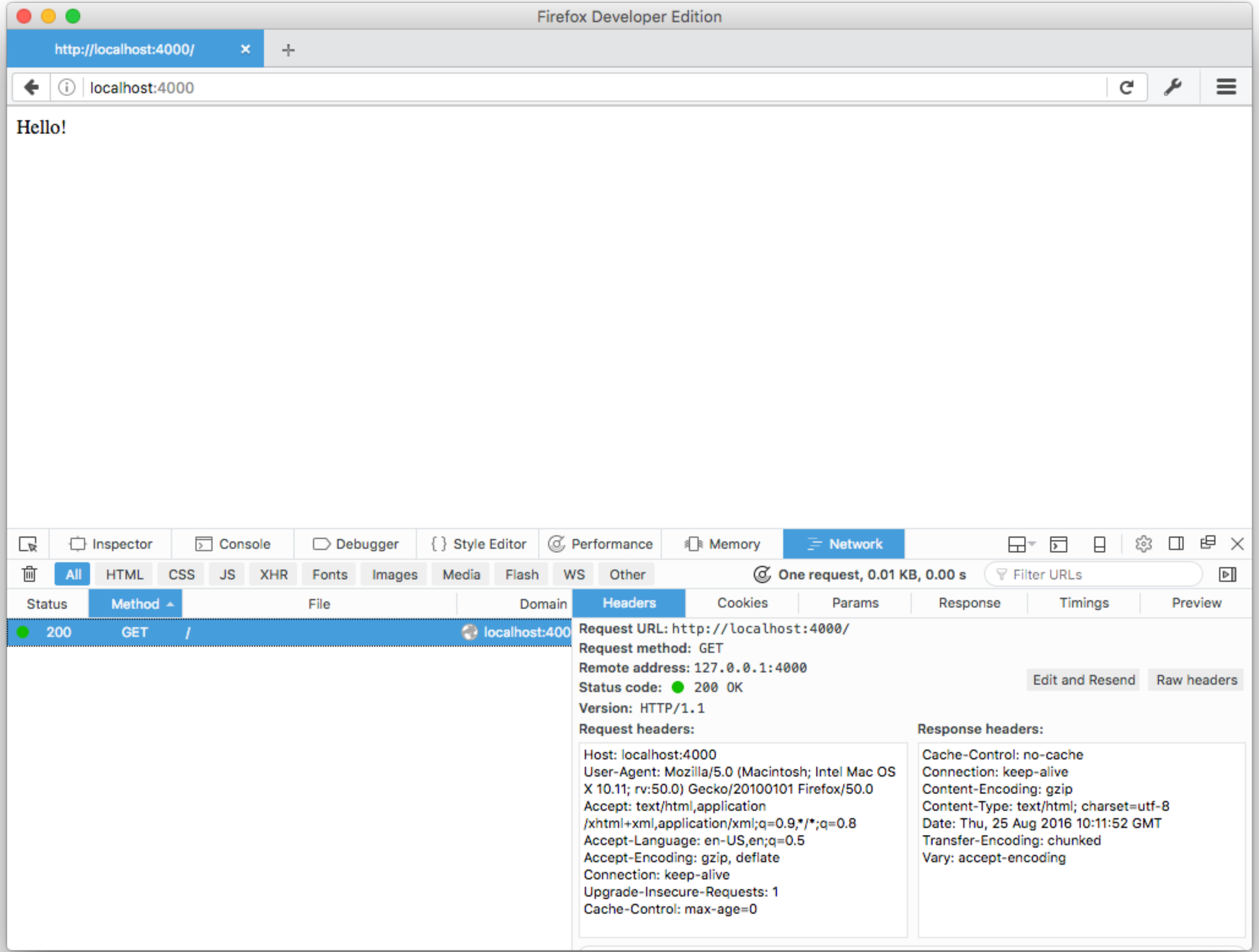


Connection -> Server -> Route -> Handler



```
'use strict';

const Hapi = require('hapi');

var server = new Hapi.Server();
server.connection({ port: process.env.PORT || 4000 });

server.route(require('./routes'));

server.start(err => {
  if (err) {
    throw err;
  }

  console.log('Server listening at:', server.info.uri);
});
```

```
/usr/l                                          epos/modules/web-2/web-app-2016/prj/donation-web/index.js
Server                                          al:4000
```



Hello!

One request, 0.01 KB, 0.00 s

200 GET / localhost:400

Request URL: `http://localhost:4000/`
Request method: GET
Remote address: 127.0.0.1:4000
Status code: ● 200 OK
Version: HTTP/1.1

Request headers:

```
Host: localhost:4000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS
X 10.11; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application
/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Response headers:

```
Cache-Control: no-cache
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Thu, 25 Aug 2016 10:11:52 GMT
Transfer-Encoding: chunked
Vary: accept-encoding
```

Plugins

- Plugins are a way of extending servers with new functionality.

- Plugins can extend a server with some global utility such as logging all requests or adding caching to responses.

- There are many existing plugins available as npm packages that deal with things like authentication and logging, written by the hapi core team and community.

- It’s also possible to create your own plugins that divide your application into smaller logical chunks that are easier to maintain or even replace or remove altogether at a later date.

Plugins Example

- Logging: **good**
- good is a hapi plugin to monitor and report on a variety of hapi server events as well as ops information from the host machine.
- It listens for events emitted by hapi server instances and pushes standardized events to a collection of streams.

```
$ npm install good
$ npm install good-console
```

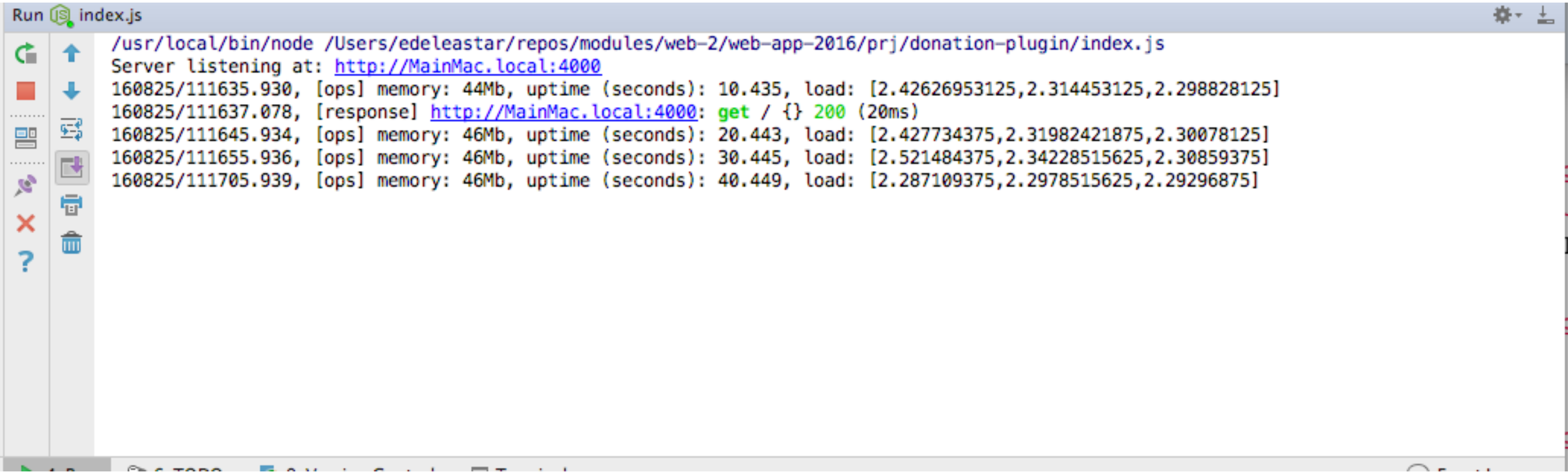
Plugin Configuration & Registration

- Plugins often take their configuration as an object, specifying various feature initial values
- Plugins are then registered - and only when this is complete is the service started

index.js

```
const Hapi = require('hapi');
const server = new Hapi.Server();
server.connection({ port: process.env.PORT || 4000 });

const options = {
  ops: {
    interval: 10000
  },
  reporters: {
    myConsoleReporter: [{
      module: 'good-console'
    }, 'stdout']
  }
};

server.register({ register: require('good'), options, }, (err) => {
  if (err) {
    throw err;
  }

  server.route(require('./routes'));

  server.start((err) => {
    if (err) {
      throw err;
    }

    console.log('Server listening at:', server.info.uri);
  });

});
```

Good Logging



```
/usr/local/bin/node /Users/edeleastar/repos/modules/web-2/web-app-2016/prj/donation-plugin/index.js
Server listening at: http://MainMac.local:4000
160825/111635.930, [ops] memory: 44Mb, uptime (seconds): 10.435, load: [2.42626953125,2.314453125,2.298828125]
160825/111637.078, [response] http://MainMac.local:4000: get / {} 200 (20ms)
160825/111645.934, [ops] memory: 46Mb, uptime (seconds): 20.443, load: [2.427734375,2.31982421875,2.30078125]
160825/111655.936, [ops] memory: 46Mb, uptime (seconds): 30.445, load: [2.521484375,2.34228515625,2.30859375]
160825/111705.939, [ops] memory: 46Mb, uptime (seconds): 40.449, load: [2.287109375,2.2978515625,2.29296875]
```