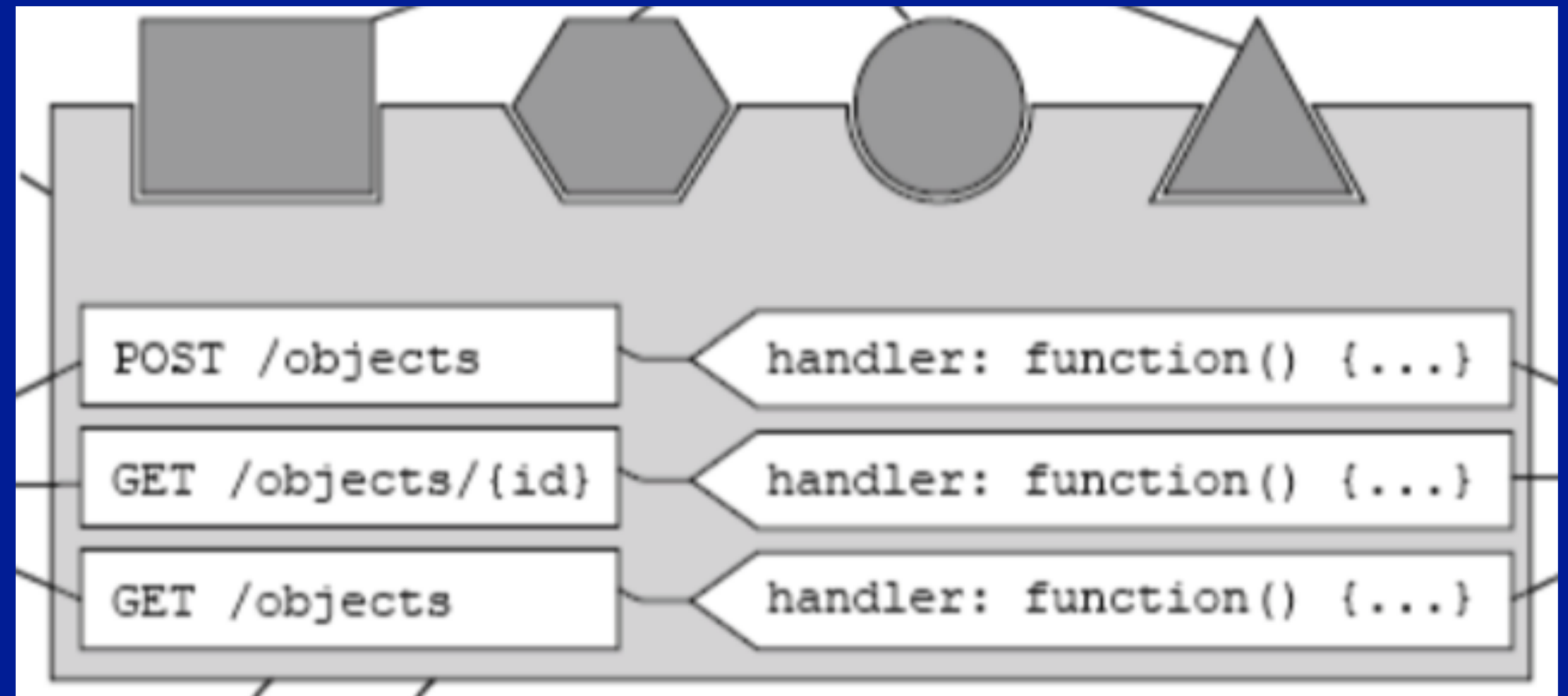
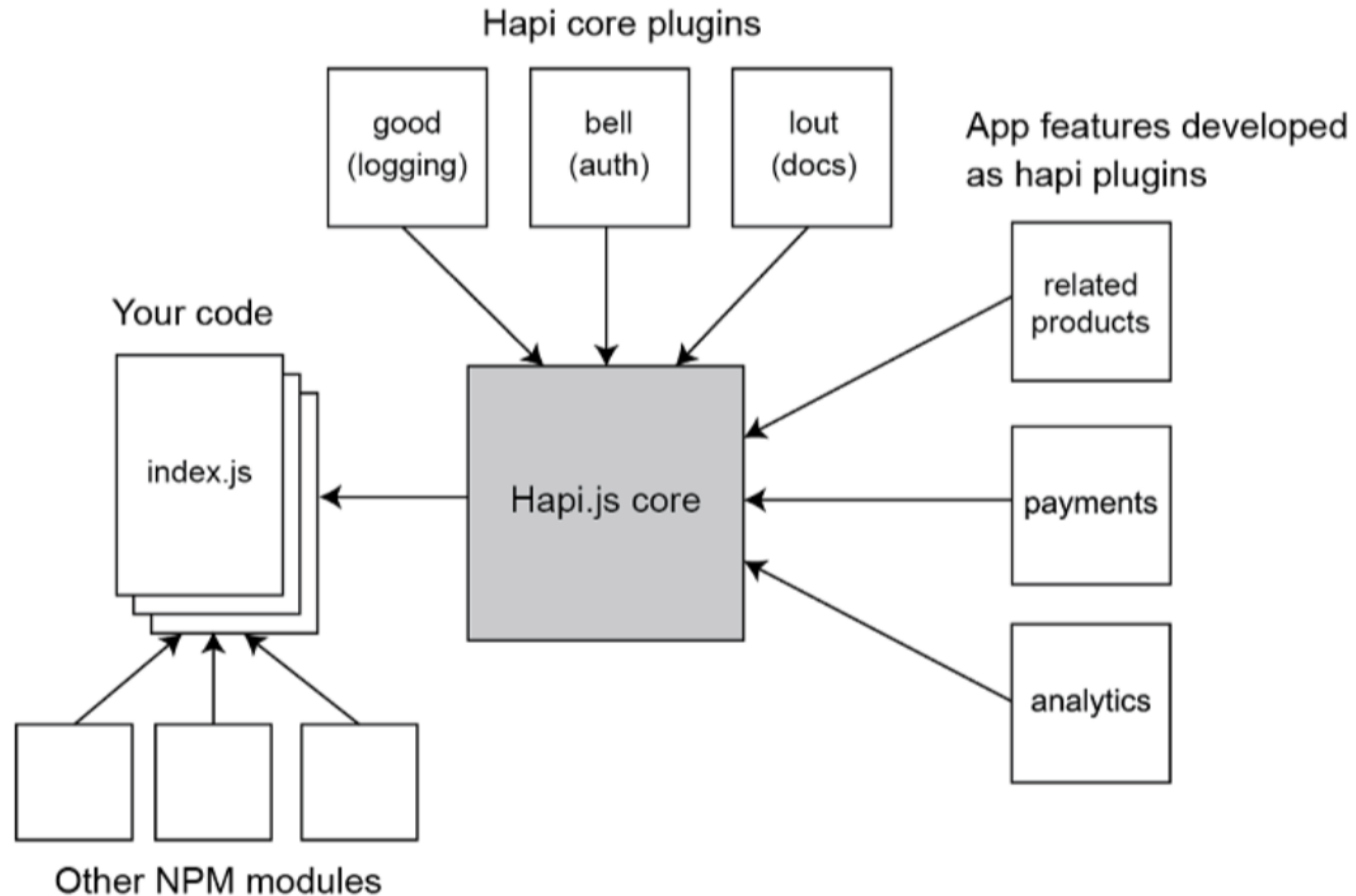


# HAPI Building Blocks



Full Stack Web Development

# Example Hapi Application Structure

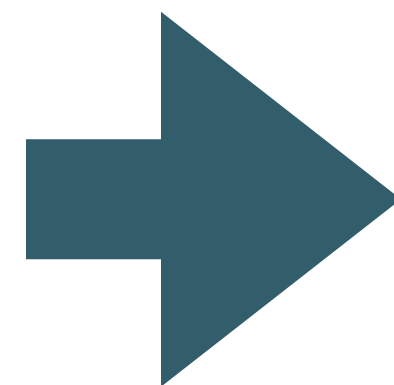


# Convention over Configuration

---

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**.

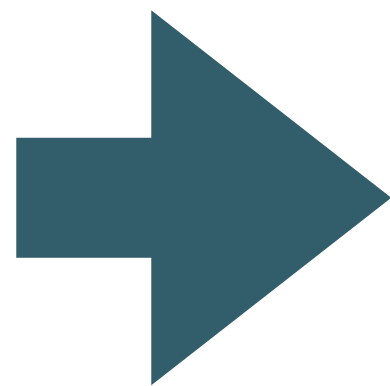
[https://en.wikipedia.org/wiki/Convention\\_over\\_configuration](https://en.wikipedia.org/wiki/Convention_over_configuration)



# Convention over Configuration in Play 1

---

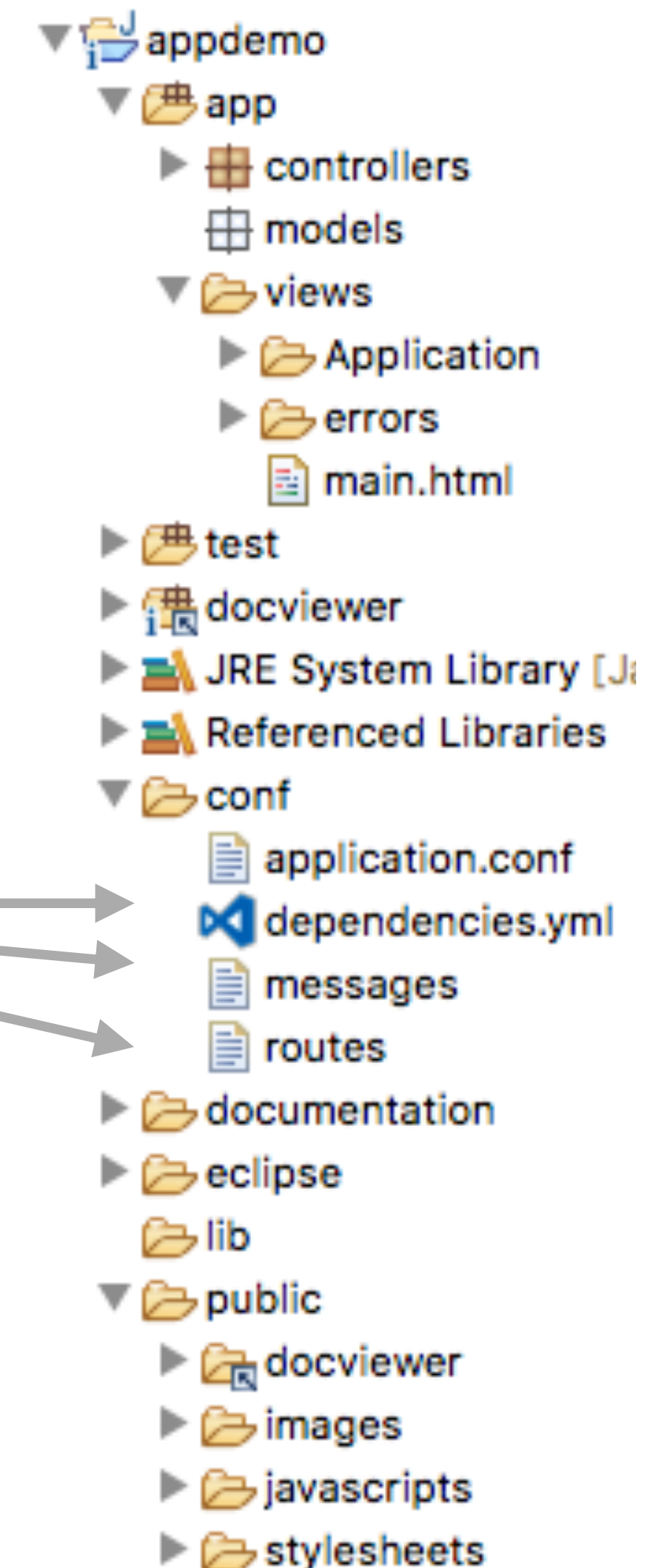
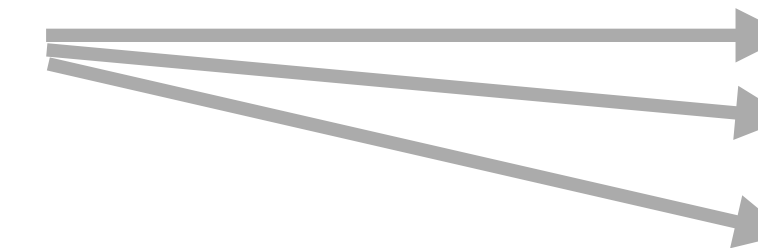
`play new`



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 = {...}  
  
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

# Convention over Code - Example

```
const bean = {...}  
  
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.

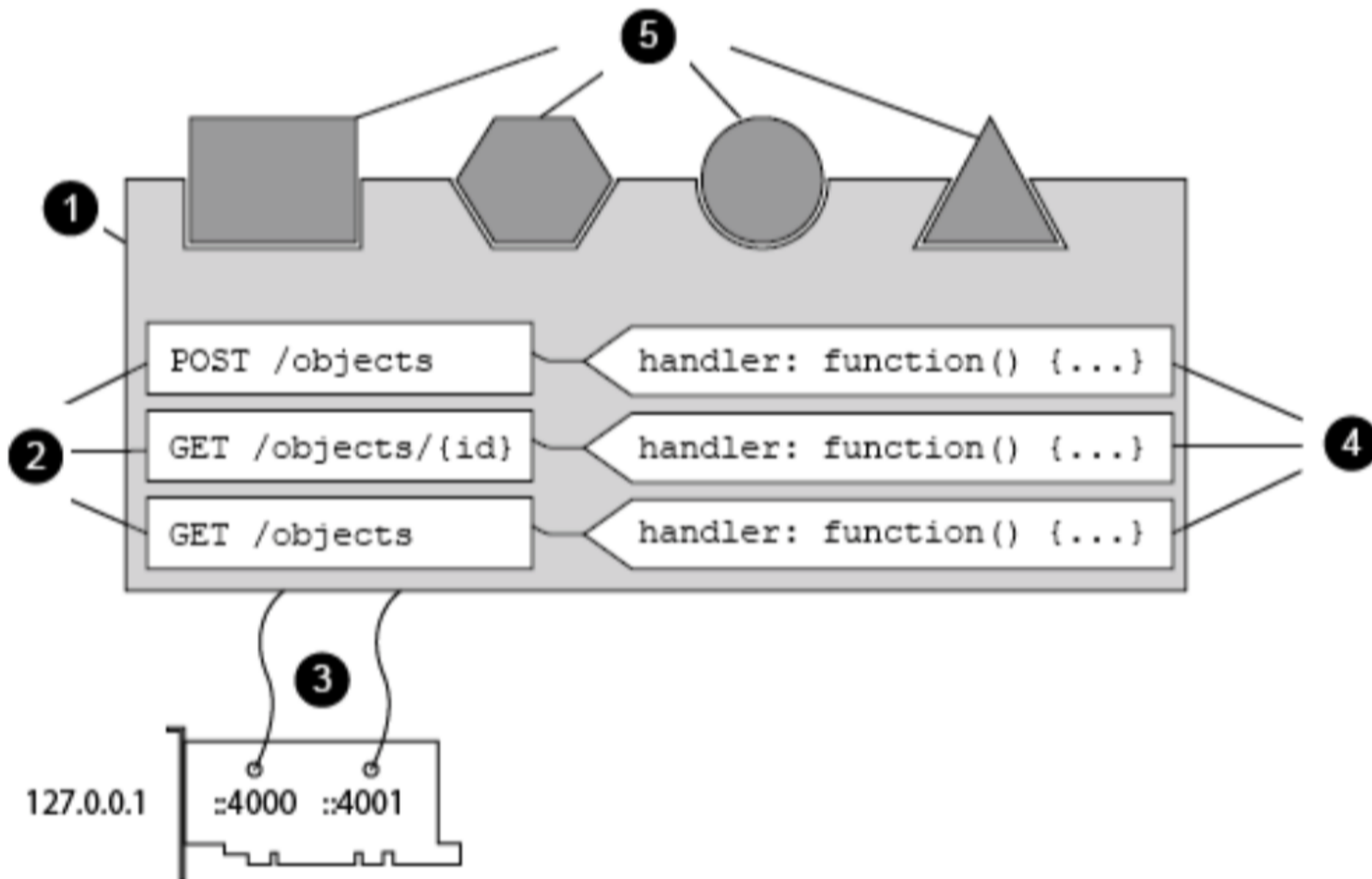
```
const bean = {...}  
bean.setName( 'Coffee' );  
bean.setColor( 'brown' );  
bean.setSpeckles( false );
```



```
const bean = {...}  
  
const options = {  
  name: 'Tutti Frutti',  
  color: 'mixed',  
  speckles: true  
};  
  
bean.config(options);
```



# Hapi Building Blocks

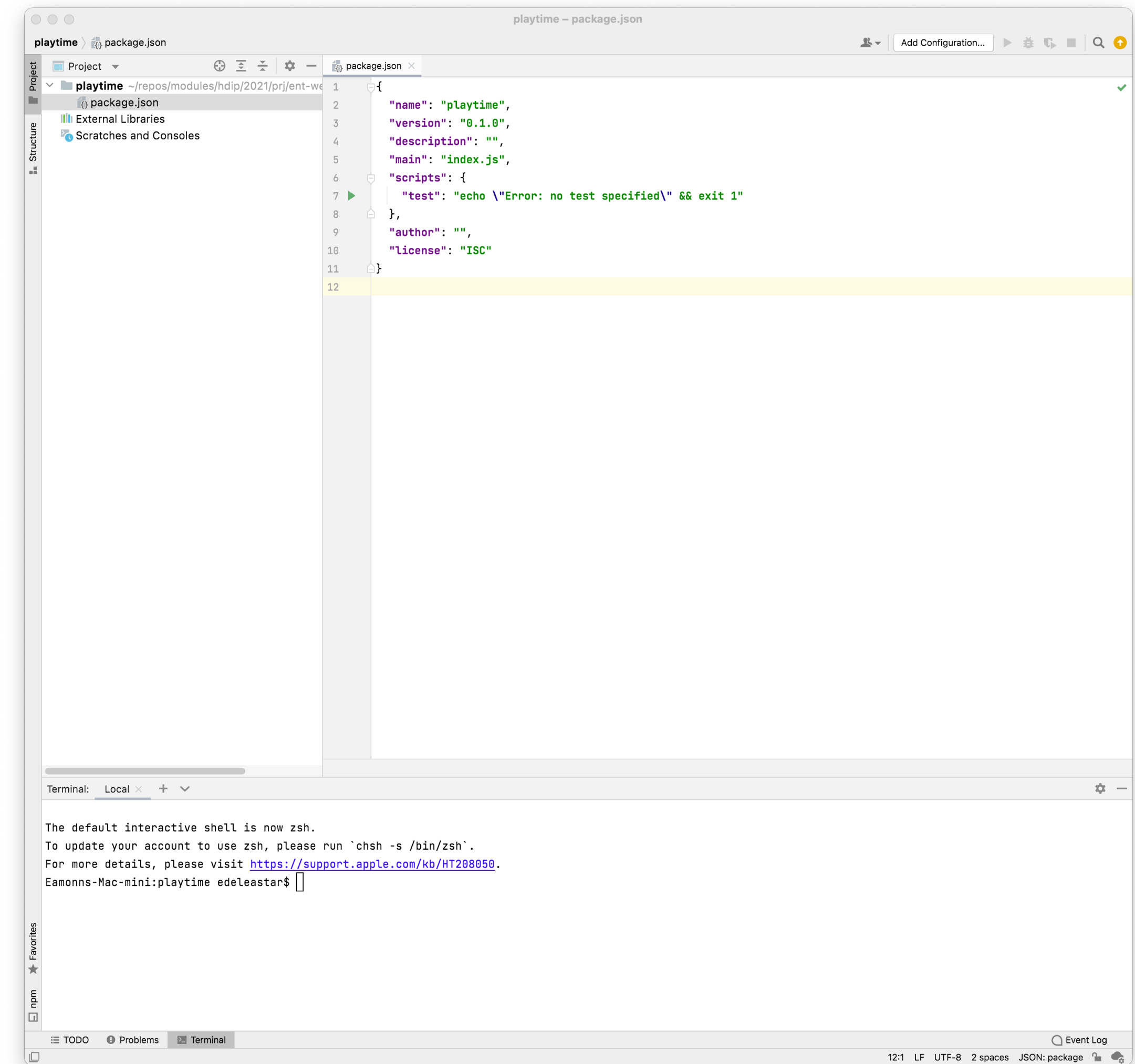


1. Server
2. Routes
3. Connections
4. Handlers
5. Plugins

# Simplest Node Application

## package.json

```
{
  "name": "playtime",
  "version": "0.1.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```



# Minimal Hapi Application

## src/server.js

```
import Hapi from "@hapi/hapi";
import path from "path";

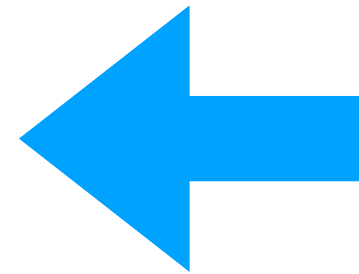
import { fileURLToPath } from "url";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

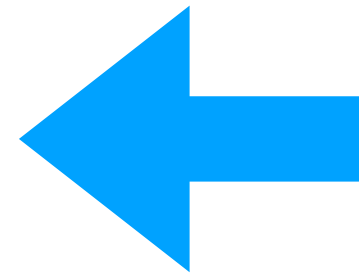
async function init() {
  const server = Hapi.server({
    port: 3000,
    host: "localhost",
  });
  await server.start();
  console.log("Server running on %s", server.info.uri);
}

process.on("unhandledRejection", (err) => {
  console.log(err);
  process.exit(1);
});

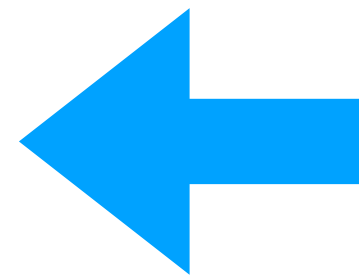
init();
```



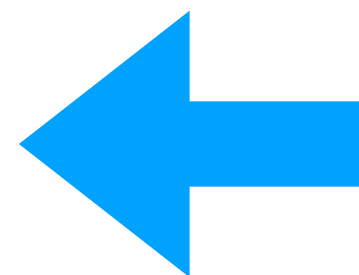
Import Hapi



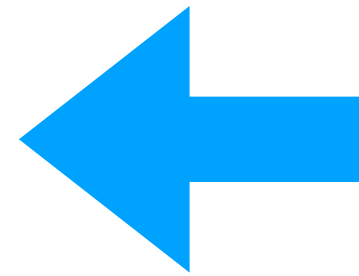
Import utilities to locate folder paths



Initialise a server



Handle failure to start



Start the app

# Application Scripts

## package.json

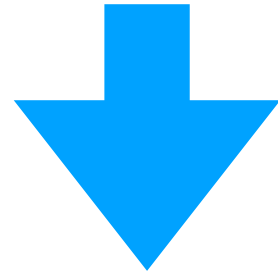
```
{
  "name": "playtime",
  "version": "0.1.0",
  "description": "Playtime: a Hapi/node application for managing Play]",
  "main": "src/server.js",
  "type": "module",
  "scripts": {
    "start": "node src/server.js",
  },
}
```

1 script:  
"start"

```
npm run start
```

# Running the application

- Start the application ...



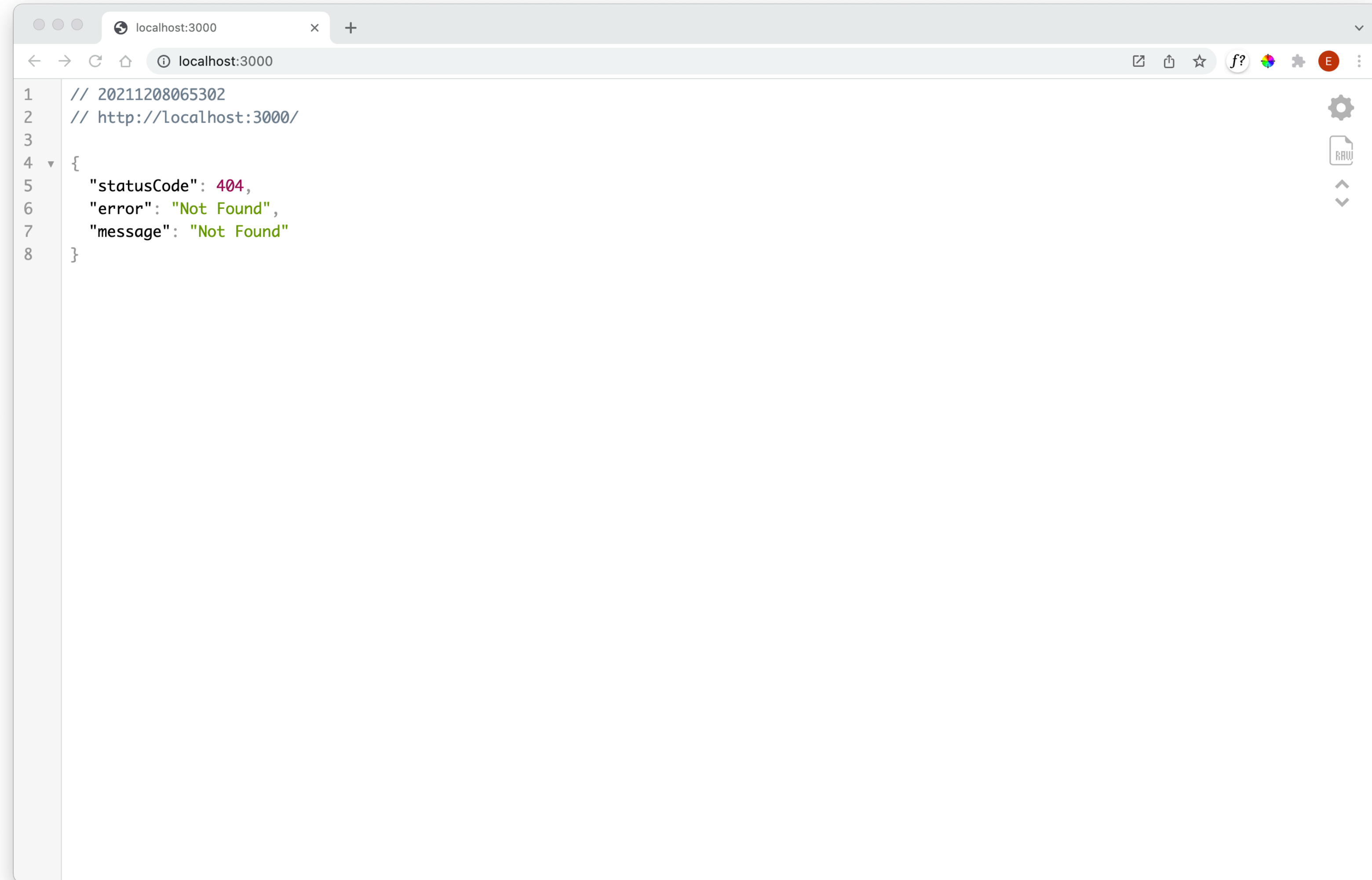
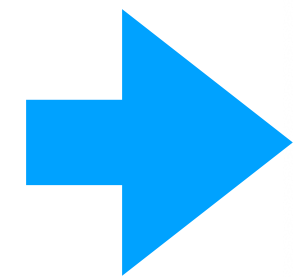
```
npm run start
```

```
> playtime@0.1.0 start
```

```
> node src/server.js
```

```
Server running on http://localhost:3000
```

- ...no routes defined yet



# 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.

## web-routes.js

```
import { dashboardController } from "../controllers/dashboard-controller.js";  
  
export const webRoutes = [{ method: "GET", path: "/", config: dashboardController.index }];
```



# Configuring Routes

## web-routes.js

```
import { dashboardController } from "../controllers/dashboard-controller.js";

export const webRoutes = [{ method: "GET", path: "/", config: dashboardController.index }];
```

- 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.



```
async function init() {
  const server = Hapi.server({
    port: 3000,
    host: "localhost",
  });
  await server.register(Vision);
  server.route(webRoutes);
  await server.start();
  console.log("Server running on %s", server.info.uri);
}
```


# Starting the Server

## web-routes.js

```
import { dashboardController } from "../controllers/dashboard-controller.js";

export const webRoutes = [{ method: "GET", path: "/", config: dashboardController.index }];
```

- server.start called to start the server.
- If there is an error on startup, the error handler will one called
- If no error, the server is running, awaiting requests and dispatching to handlers based on the installed routes



```
async function init() {
  const server = Hapi.server({
    port: 3000,
    host: "localhost",
  });
  await server.register(Vision);
  server.route(webRoutes);
  await server.start();
  console.log("Server running on %s", server.info.uri);
}
```

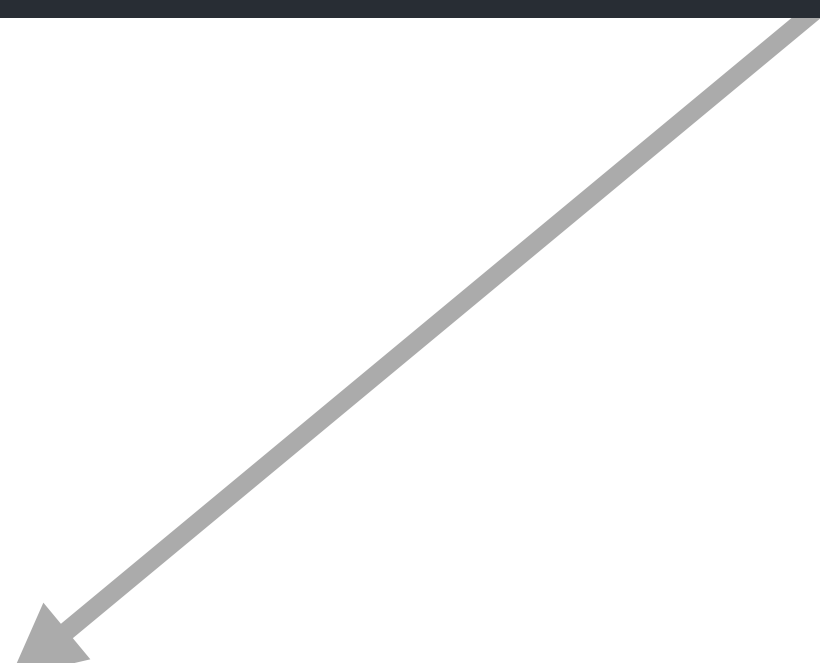
```
process.on("unhandledRejection", (err) => {
  console.log(err);
  process.exit(1);
});
```

# 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.
- Return a value to be rendered by the browser

## web-routes.js

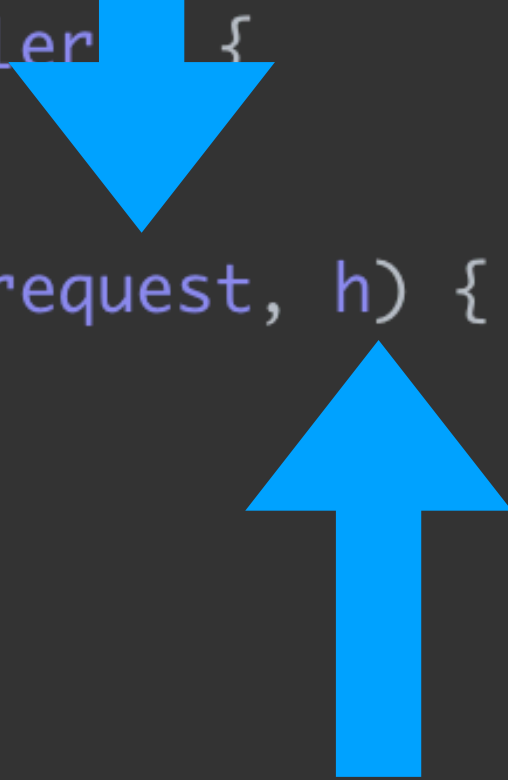
```
import { dashboardController } from "../controllers/dashboard-controller.js";  
  
export const webRoutes = [{ method: "GET", path: "/", config: dashboardController.index }];
```



## dashboard-controller.js

```
export const dashboardController = {  
  index: {  
    handler: async function (request, h) {  
      return 'Hello!';  
    },  
  },  
};
```

- The request parameter is an object with details about the end user's request, such as path parameters, an associated payload, authentication information, headers, etc.



```
export const dashboardController = {  
  index: {  
    handler: async function (request, h) {  
      return 'Hello!';  
    },  
  },  
};
```

The diagram consists of two large blue arrows. One arrow points downwards from the text 'The request parameter' to the 'request' argument in the function signature of the code snippet. The other arrow points upwards from the 'h' argument in the function signature to the text 'h is the response toolkit'.

- h is the response toolkit, an object with several methods used to respond to the request.

1. Servers

2. Connections

3. Routes

4. Handlers

3

```
export const webRoutes = [{ method: "GET", path: "/", config: dashboardController.index }];
```

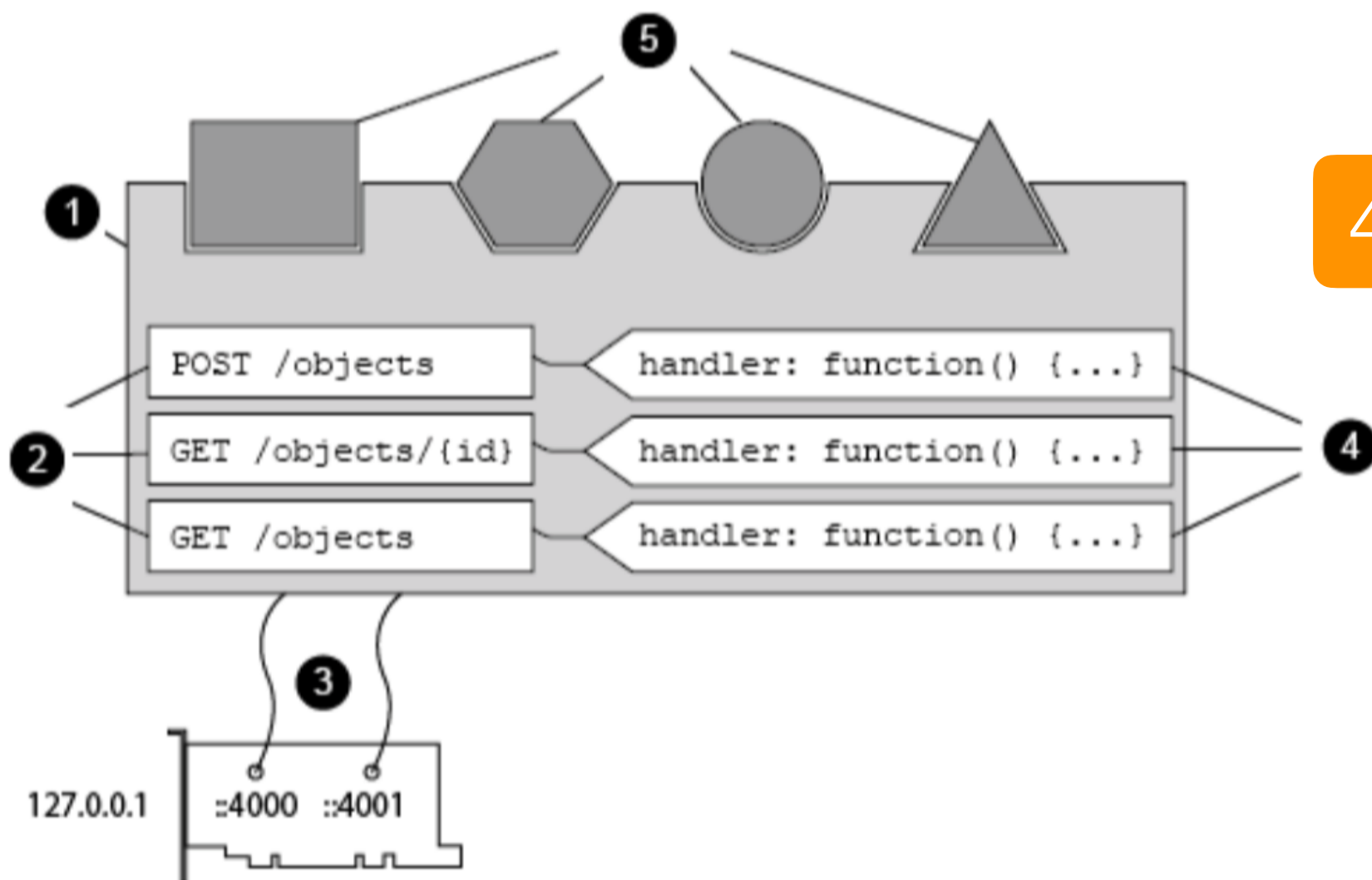
1

2

```
async function init() {  
  const server = Hapi.server({  
    port: 3000,  
    host: "localhost",  
  });  
  await server.register(Vision);  
  server.route(webRoutes);  
  await server.start();  
  console.log("Server running on %s", server.info.uri);  
}
```

4

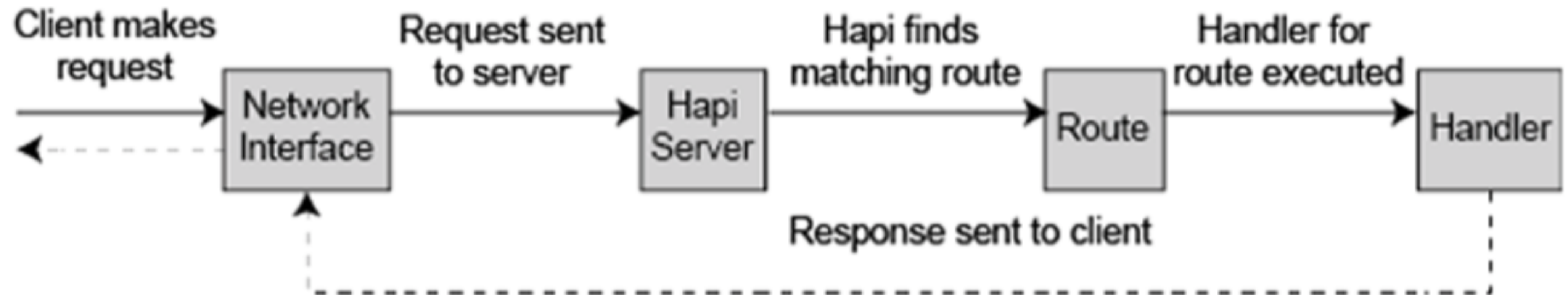
```
export const dashboardController = {  
  index: {  
    handler: async function (request, h) {  
      return 'Hello!';  
    },  
  },  
};
```





# Hapi Request Handling

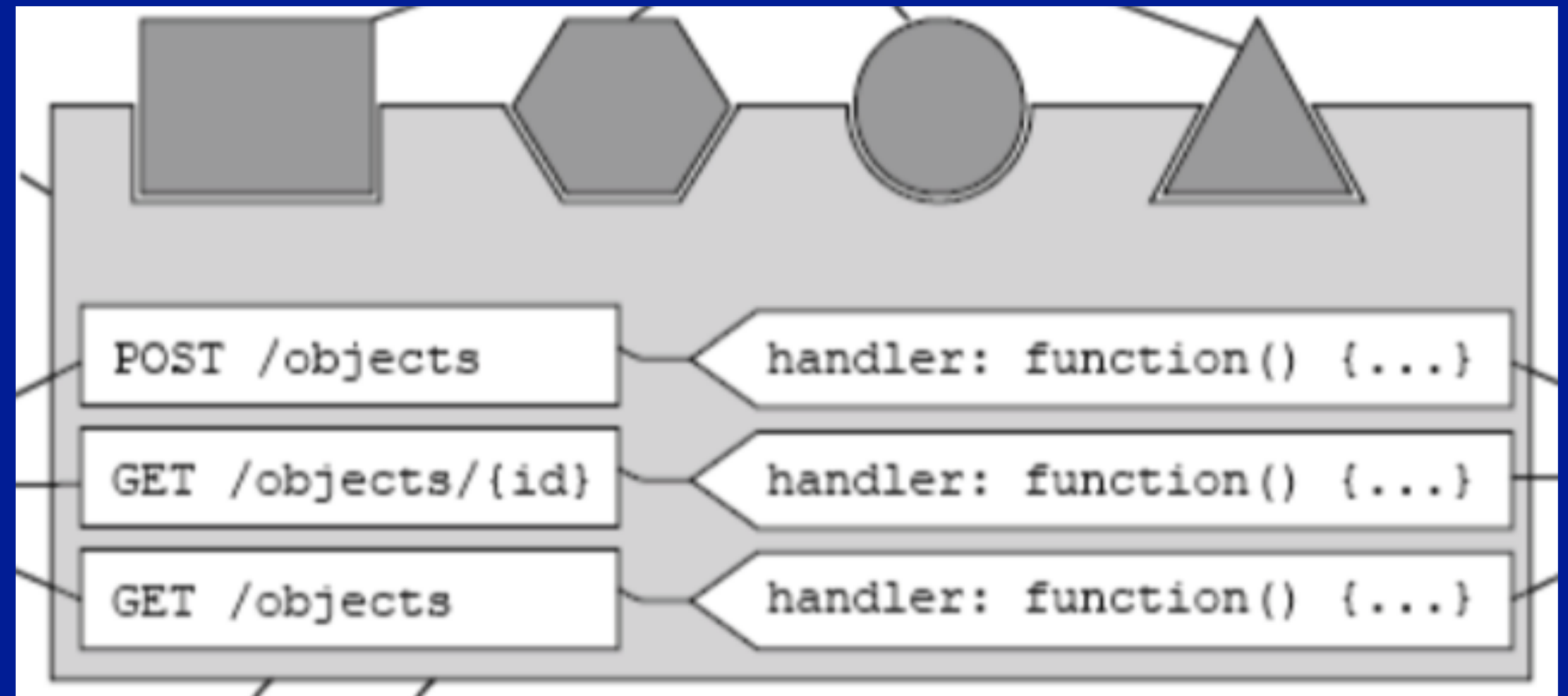
---



Connection -> Server -> Route -> Handler



# HAPI Building Blocks



Full Stack Web Development