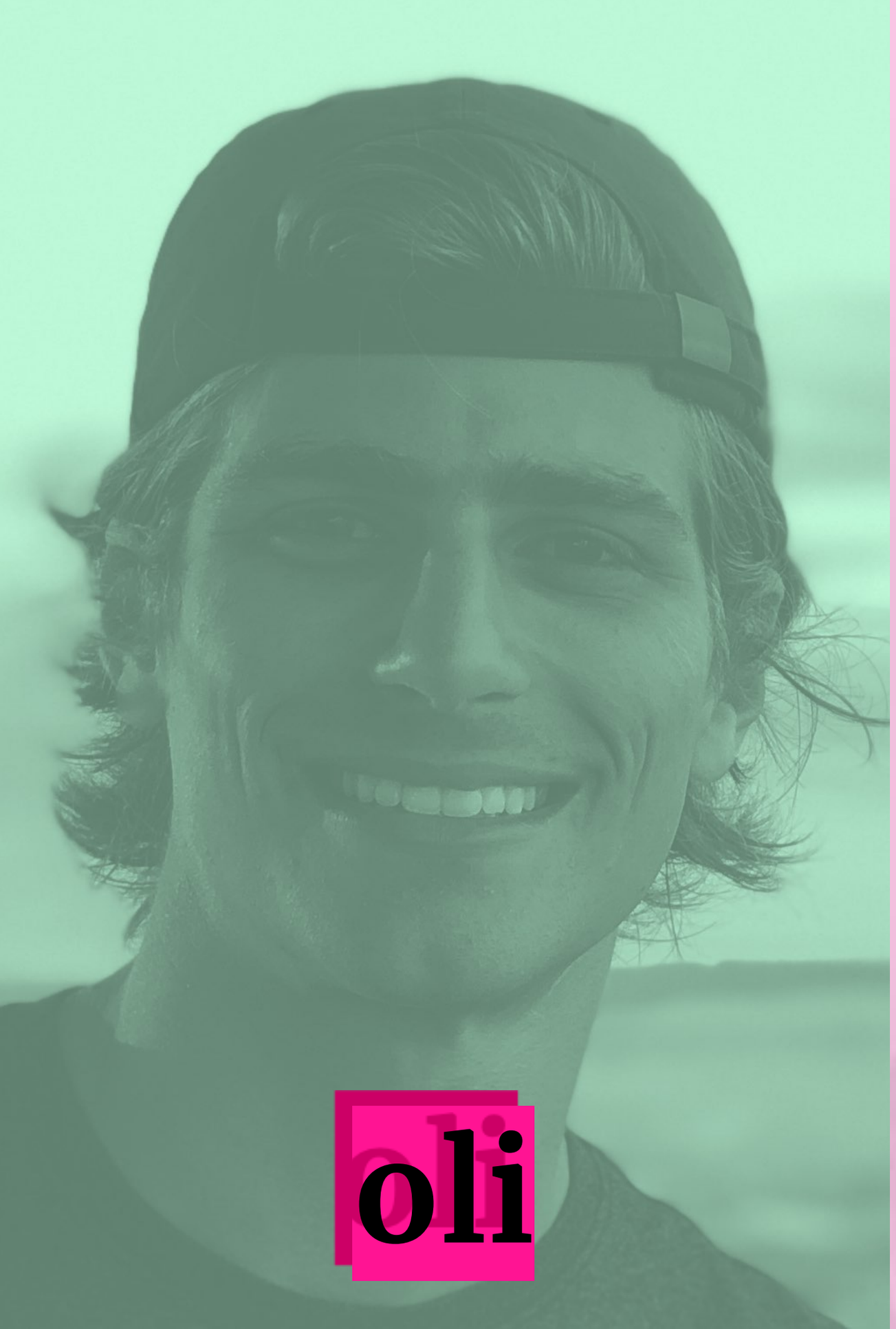
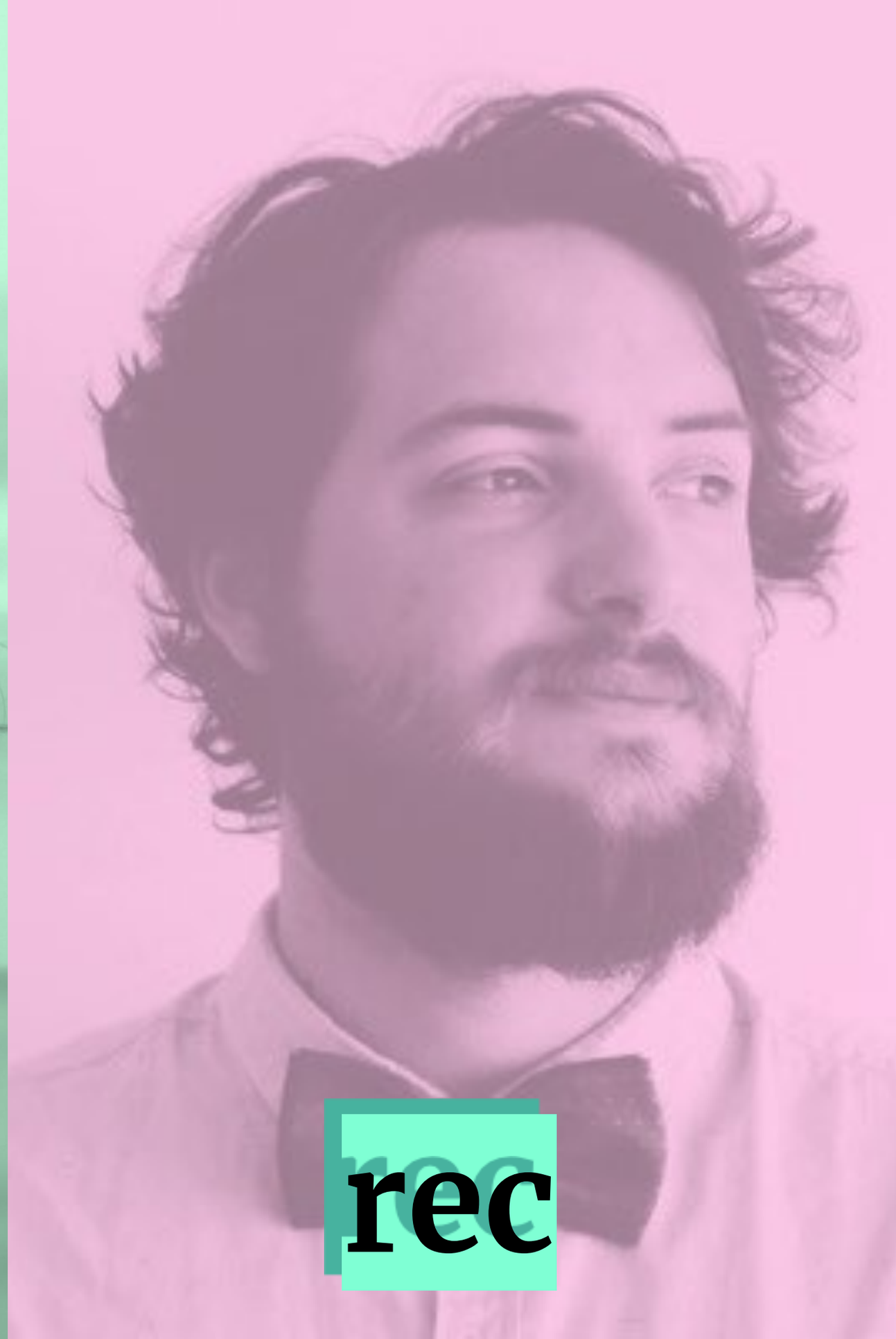
web programming

intro to node.js


oli


rec

agenda

1. intro to node.js
2. modules
3. project structure
4. intro to hapijs

1. intro to node.js

what's node.js?

- platform to execute JavaScript
  (outside a browser)

- built on top of JavaScript V8
  (JavaScript engine that powers Google Chrome)

- uses a non-blocking I/O model
  (event-driven, perfect for real-time applications)

what's node.js?

- its code is absolutely open-source
  (you can find and collaborate with it in GitHub)
- written in C++ and JavaScript
  (most of its codebase is written in JavaScript)

node vs. browser

browser

`window` global variable

node.js

`process` global variable

start a node process

```
$ node
```

start a node process

```
$ node <file-name.js>
```

node *hello-world.js*

```
setTimeout(function() {
  console.log('world');
}, 2000);

console.log('Hello');
```

php alternative

```
echo('Hello');
sleep(2);
echo('world');
```

“Node never sleeps.”

–Ryan Dahl

2. modules

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

require

- module system for referencing to other files
- these files can be located:
  - node's standard library
  - relative to our codebase
  - third-party, via package manager

require (std lib)

```
const fs = require('fs');

const onDone = function() {
  console.log('Success!');
};

fs.writeFile('./test.txt', '😊', 'utf8', onDone);
```

require (relative)

```
// add.js
module.exports = function(a, b) {
  return a + b;
};

// index.js
const add = require('./add');

add(1, 2);
```

require (3rd-party)

```
// we need to install it first!
// npm install --save hapi

const Hapi = require('hapi');

const server = Hapi();
```

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

Added in: v0.1.22

- `<Object>`

A collection of all the standard HTTP response status codes, and the short description of each. For example, `http.STATUS_CODES[404] === 'Not Found'`.

http.createServer([options][, requestListener]) [src]

▸ History

- `options` `<Object>`
  - `IncomingMessage` `<http.IncomingMessage>` Specifies the `IncomingMessage` class to be used. Useful for extending the original `IncomingMessage`. **Default:** `IncomingMessage`.
  - `ServerResponse` `<http.ServerResponse>` Specifies the `ServerResponse` class to be used. Useful for extending the original `ServerResponse`. **Default:** `ServerResponse`.
- `requestListener` `<Function>`
- Returns: `<http.Server>`

Returns a new instance of `http.Server`.

The `requestListener` is a function which is automatically added to the `'request'` event.

http.get(options[, callback]) [src]

http.get(url[, options][, callback]) [src]

▸ History

- `url` `<string>` | `<URL>`
- `options` `<Object>` Accepts the same `options` as `http.request()`, with the `method` always set to `GET`. Properties that are inherited from the prototype

Go to https://nodejs.org/api for more documentation

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

```javascript
const http = require(‘http');
let serverCounter = 0;

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Visitor number' + (++serverCounter));
});

server.listen(8000);
```

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.write(‘hello');

  setTimeout(function() {
    res.end(‘world’);
  }, 2000);
});

server.listen(8000);
```

3. project structure

dividing by *envs*

client

sub-project for the web interface

server

our abstraction layer for the data

dividing by more specific *envs*

web

sub-project for the web interface

api

our abstraction layer for the data

EXERCISE

Set up your project

repository

- go to github.com
- click on "New repository"
- choose the following options and then click "Create repository":
  - name: `my-proj`
  - public
  - README: yes
  - .gitignore: Node
  - license: MIT
- click on "Clone" and copy the URL

```
$ git clone <repo_url> .
```

```
$ npm init
```

This utility will walk you through
creating a **package.json** file.
It only covers the most common items,
and tries to guess sensible defaults.

See `**npm help json**` for definitive
documentation on these fields
and exactly what they do.

Use `**npm install <pkg>**` afterwards to
install a package and
save it as a dependency in the
package.json file.

```
Press ^C at any time to quit.
package name: (<…>)
version: (1.0.0)
description:
git repository:
keywords:
author:
license: (ISC)
About to write to <…>:
```

package.json

```
{
  "name": "my-proj",
  "scripts": {
    ...
  },
  "dependencies": {
    ...
  }
  "devDependencies": {
    ...
  }
}
```

package.json

```
{
  "name": "web-programming-boilerplate",
  "scripts": {
    "lint:js": "eslint .",
    "lint:css": "stylelint **/*.css",
    "lint": "npm run lint:js && npm run lint:css",
    "pretest": "npm run lint",
    "test": "jest --passWithNoTests"
  },
  "devDependencies": {
    "@ucudal/eslint-config": "...",
    "@ucudal/stylelint-config": "...",
    ...
  }
}
```

dev environment setup

```
$ git checkout -b setup
```

linting

- **static analysis** of code
  - **code-quality** rules: find problematic patterns in code
  - **formatting** rules: find code that doesn't adhere to style guidelines

linting – ESLint

- 
- open source JavaScript linting utility
- we will use it for code-quality rules only (not formatting)
- config: `@ucudal/eslint-config`

.eslintrc.json

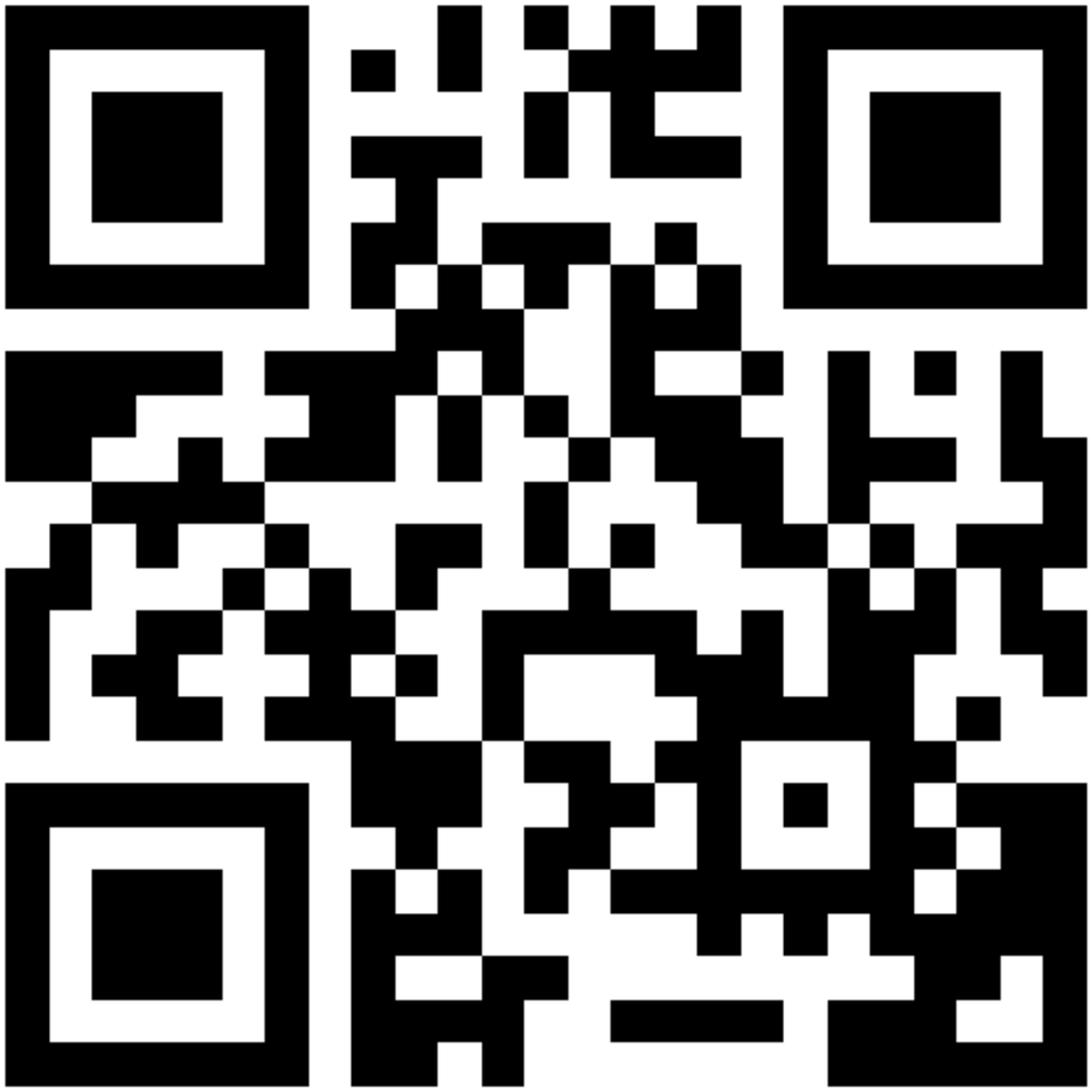
```
{
  "extends": "@ucudal/eslint-config"
}
```

reference: https://eslint.org/docs/user-guide/configuring

formatting – Prettier

- 
- code formatter
- opinionated
- we will use it in conjunction with ESLint and stylelint

set up Prettier



goo.gl/L3wCRN

.prettierrc.json

```
{
  "singleQuote": true,
  "trailingComma": "all"
}
```

reference: https://prettier.io/docs/en/configuration.html

.eslintrc.json

```
{
  "extends": [
    "@ucudal/eslint-config",
    "plugin:prettier/recommended"
  ]
}
```

testing – jest

- 
- zero configuration
- jsdom built-in

```
$ npm i -D jest
```

jest.config.js

```
module.exports = {
  testEnvironment: 'node',
  verbose: true,
};
```

reference: https://jestjs.io/docs/en/configuration

package.json

```
{
  "name": "web-programming-boilerplate",
  "scripts": {
    "lint": "eslint .",
    "pretest": "npm run lint",
    "test": "jest --passWithNoTests"
  },
  "devDependencies": {
    "@ucudal/eslint-config": "...",
    ...
  }
}
```

pull request

- go to the `my-proj` repo page on github.com
- switch to "setup" branch
- click on "New pull request"
- give it a name and click on "Create pull request"
- Wait for Travis to run and merge on success

EXERCISE

Initialize the api package

```
$ mkdir api && cd api
$ npm init
```

```
$ npm install --save @hapi/hapi
```

api/index.js

```
const Hapi = require('@hapi/hapi');

const init = async () => {

    const server = Hapi.server({
        port: 3000,
        host: 'localhost'
    });

    server.route({
        method: 'GET',
        path:'/',
        handler: (request, h) => {

            return [{ _id: 1, name: ‘Product 1’}, { _id: 2, name: ‘Product 2’ }];
        }
    });

    await server.start();
    console.log('Server running on %s', server.info.uri);
};
```

api/package.json

```
{
  …,
  "scripts": {
    "start": “node index.js”,
    …
  },
  …,
}
```

```
$ npm start
```

3. intro to hapi.js

what's hapijs?

- it's one out of many node.js **web frameworks**.
- developed to handle huge traffic.
- allows to create **APIs** or **server pages** and **static files**.

what's hapijs?

- has a **routing** mechanism that maps path patterns and http methods.
- it allows you to specify how to render **templates**, if you need to do so.

a hapi server

```
const Hapi = require(‘@hapi/hapi‘);

const server = Hapi.server({
  port: 8000,
  host: ‘localhost’
});

server.start();
```

a hapi server

```
const Hapi = require(‘@hapi/hapi‘);

const server = Hapi.server({ hapi instance
  port: 8000,
  host: ‘localhost’
});

server.start();
```

a hapi server

```
const Hapi = require(‘@hapi/hapi‘);

const server = Hapi.server({
  port: 8000, port where we are listening
  host: ‘localhost’
});

server.start();
```

a hapi server

```
const Hapi = require(‘@hapi/hapi‘);

const server = Hapi.server({
  port: 8000,
  host: 'localhost'
});

server.start();
```

host from where we serve

a hapi server

```
const Hapi = require('@hapi/hapi');

const server = Hapi.server({
  port: 8000,
  host: ‘localhost’
});

server.start();
```

method to start listening to requests

a hapi route

```
const server = Hapi.server({
        port: 3000,
        host: 'localhost'
    });

    server.route({
        method: 'GET',
        path:'/',
        handler: (request, h) => {

            return { myAttribute: ‘This is an example’ };
        }
    });

    await server.start();
```

```
$ curl -i localhost:8000
```

a hapi response

could be:

- **simple text**

```
return ‘Some example test’
```

- **json**

```
return { mySampleObj: true }
```

<EXERCISE>

Make multi-resource API in hapi.js

- hapi.js serving to a port of choice.
- Resolve /products to a list of two products.
- Resolve /products/:id to a single product.
- Each product has an _id of 1 or 2 and a random name.
- Do the same for /users.

thanks!