SENG365 Web Computing Architecture: #2 JavaScript cont. and Asynchronous behavior

Ben Adams
Course Coordinator & Lecturer
benjamin.adams@canterbury.ac.nz
310, Erskine Building

Reminder of last week

- What is a web application? And why a course on it?
- Reference model for web applications, for SENG365
- HTTP servers
- JavaScript

- Course administration
  - Teaching team
  - Labs
  - Assessments
    - Assignments

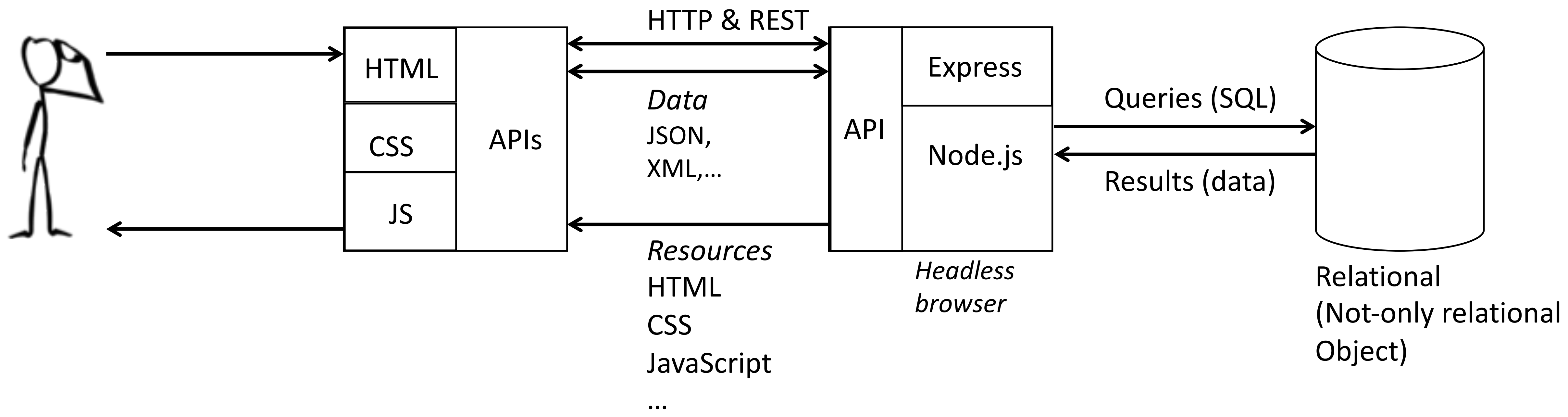
The story so far…

In the lectures

- Introduction to web computing
  - e.g. client-server
- HTTP, REST, & Intro to APIs
- Some JavaScript concepts
- Introduction to the assignments

In the labs

- Self-study labs x 3
- Introduction to Node.js
- Introduction to persistence
- Structuring your server application



User	HTTP client	HTTP Server	Database
Human	Machine	Machine	Machine

In the lecture this week

- News and admin, assignment 1
- Asynchronous behaviours
    - Event loop
    - Callback hell
    - Promises
    - `async/await` syntax
- Module dependency
- API versioning
- (Closures)

Admin

Student reps

- Contact information on the Learn page

Assignment 1

Assignment 1 repo on eng-git (GitLab)

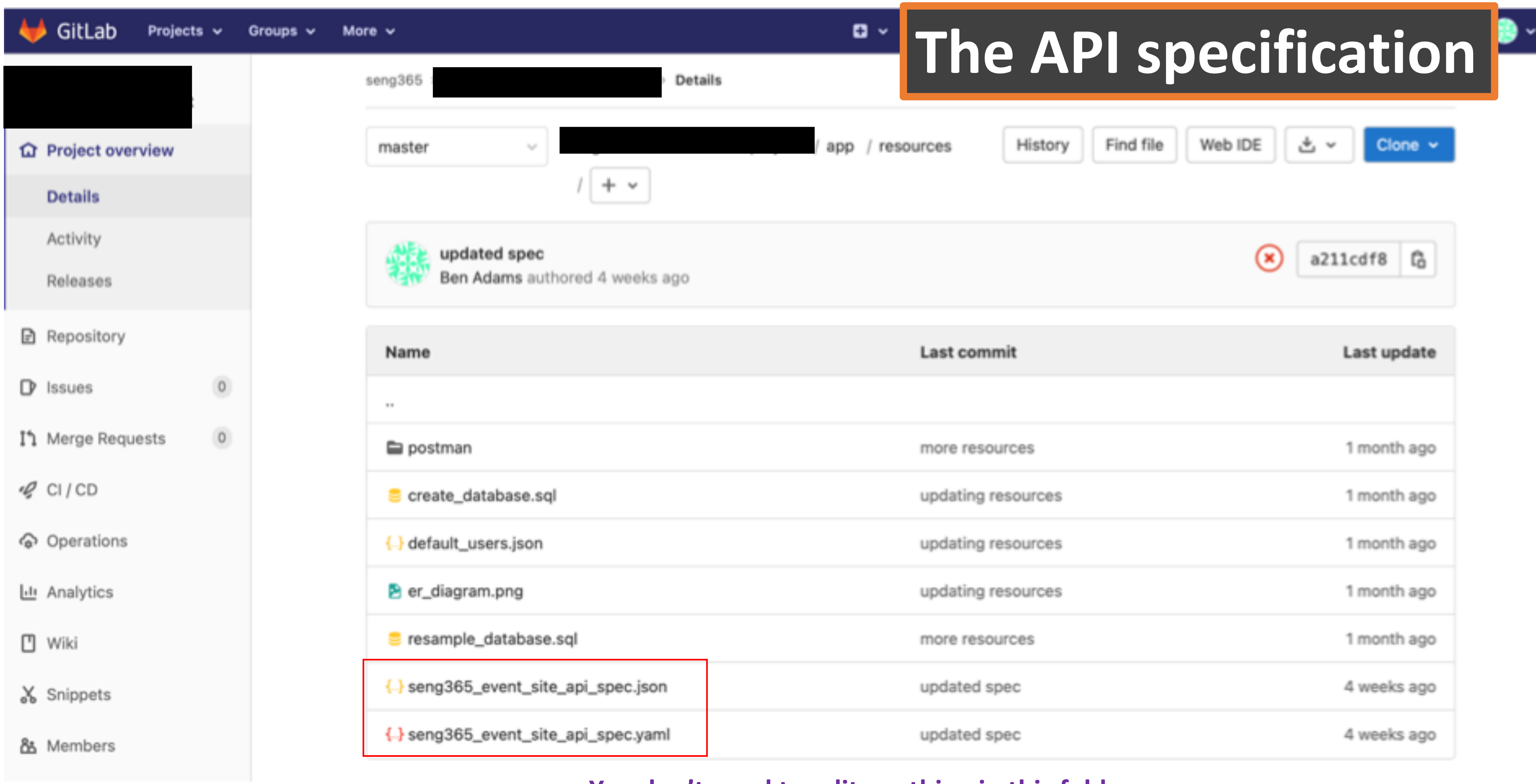
- You must log in to eng-git to activate your account. And then…
- Your eng-git repo has been created
  - skeleton project
    - Clone from eng-git into your own development environment
    - Install node modules: `npm install`
    - Create a `.env` file in the root directory of your project
      - Add `.env` to `.gitignore`
    - Add your specific environment variables to `.env`
  - API specification
    - See next slide(s)
  - README.md
    - See next slide(s) for your allocated port number in the VM server

The assignment in essence

- (Assignment Briefing on Learn)
- Implement the API specification provided in the repo
- We will assess the implementation using a suite of automated tests.
  - Assessing API coverage: how much of the API was implemented?
  - Assessing API correctness: was an endpoint correctly implemented?
- The automated tests are available for you
  - See the information in the README.md
  - You can see how well you are progressing
- **For the actual assessment we will use different data**, but intend to use the same (or similar) automated test suite.

Finding the assignment briefing and spec

The API specification



[redacted]

seng365 [redacted] Details

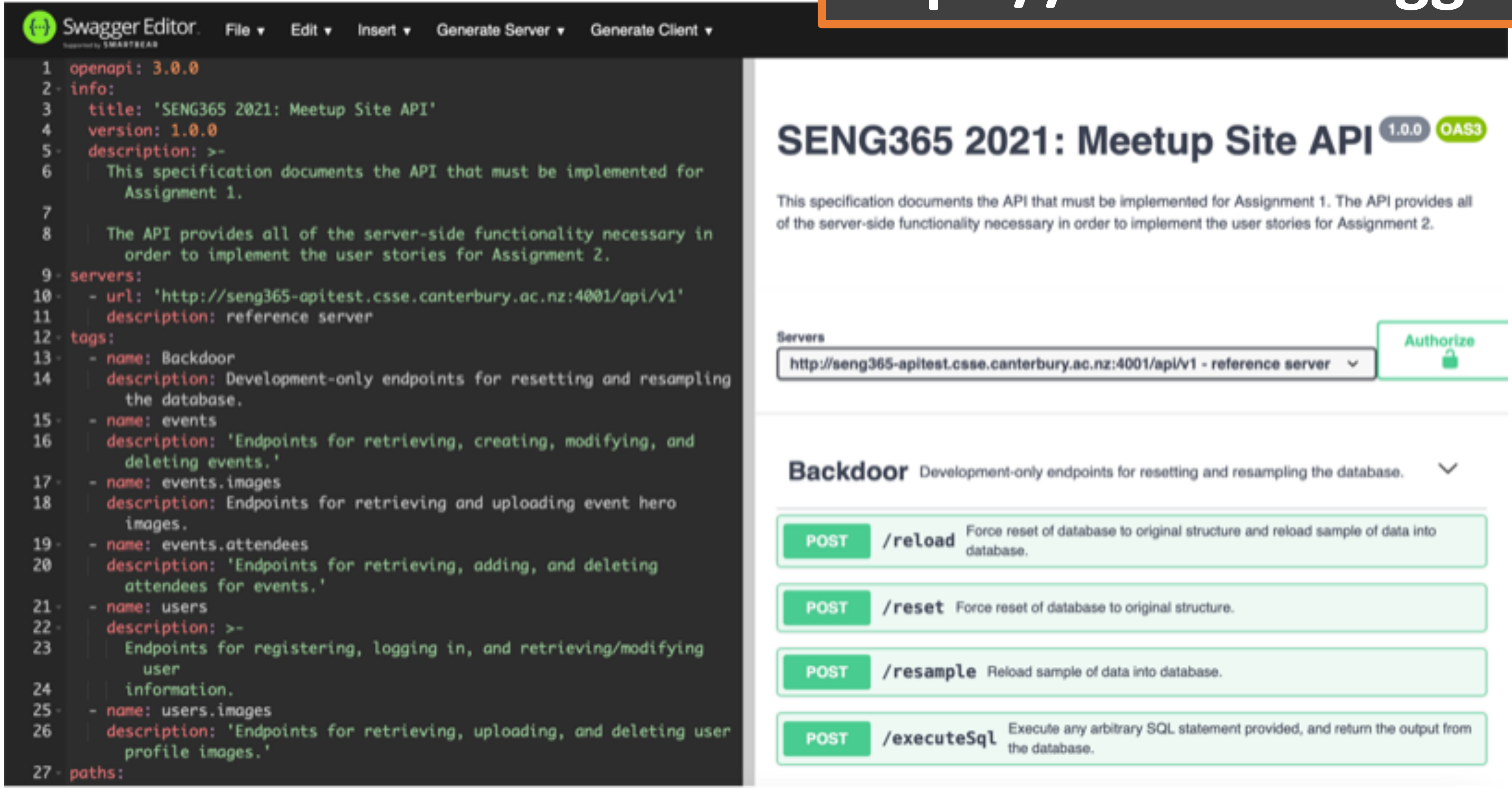
master [redacted] / app / resources

updated spec
Ben Adams authored 4 weeks ago
a211cdf8

Name	Last commit	Last update
..		
postman	more resources	1 month ago
create_database.sql	updating resources	1 month ago
default_users.json	updating resources	1 month ago
er_diagram.png	updating resources	1 month ago
resample_database.sql	more resources	1 month ago
seng365_event_site_api_spec.json	updated spec	4 weeks ago
seng365_event_site_api_spec.yaml	updated spec	4 weeks ago

You don't need to edit anything in this folder
It's all read only.

https://editor.swagger.io



```yaml
openapi: 3.0.0
info:
  title: 'SENG365 2021: Meetup Site API'
  version: 1.0.0
  description: >-
    This specification documents the API that must be implemented for
      Assignment 1.

    The API provides all of the server-side functionality necessary in
      order to implement the user stories for Assignment 2.
servers:
  - url: 'http://seng365-apitest.csse.canterbury.ac.nz:4001/api/v1'
    description: reference server
tags:
  - name: Backdoor
    description: Development-only endpoints for resetting and resampling
      the database.
  - name: events
    description: 'Endpoints for retrieving, creating, modifying, and
      deleting events.'
  - name: events.images
    description: Endpoints for retrieving and uploading event hero
      images.
  - name: events.attendees
    description: 'Endpoints for retrieving, adding, and deleting
      attendees for events.'
  - name: users
    description: >-
      Endpoints for registering, logging in, and retrieving/modifying
        user
      information.
  - name: users.images
    description: 'Endpoints for retrieving, uploading, and deleting user
      profile images.'
paths:
```

SENG365 2021: Meetup Site API 1.0.0 OAS3

This specification documents the API that must be implemented for Assignment 1. The API provides all of the server-side functionality necessary in order to implement the user stories for Assignment 2.

Servers
http://seng365-apitest.csse.canterbury.ac.nz:4001/api/v1 - reference server

Authorize

Backdoor Development-only endpoints for resetting and resampling the database.

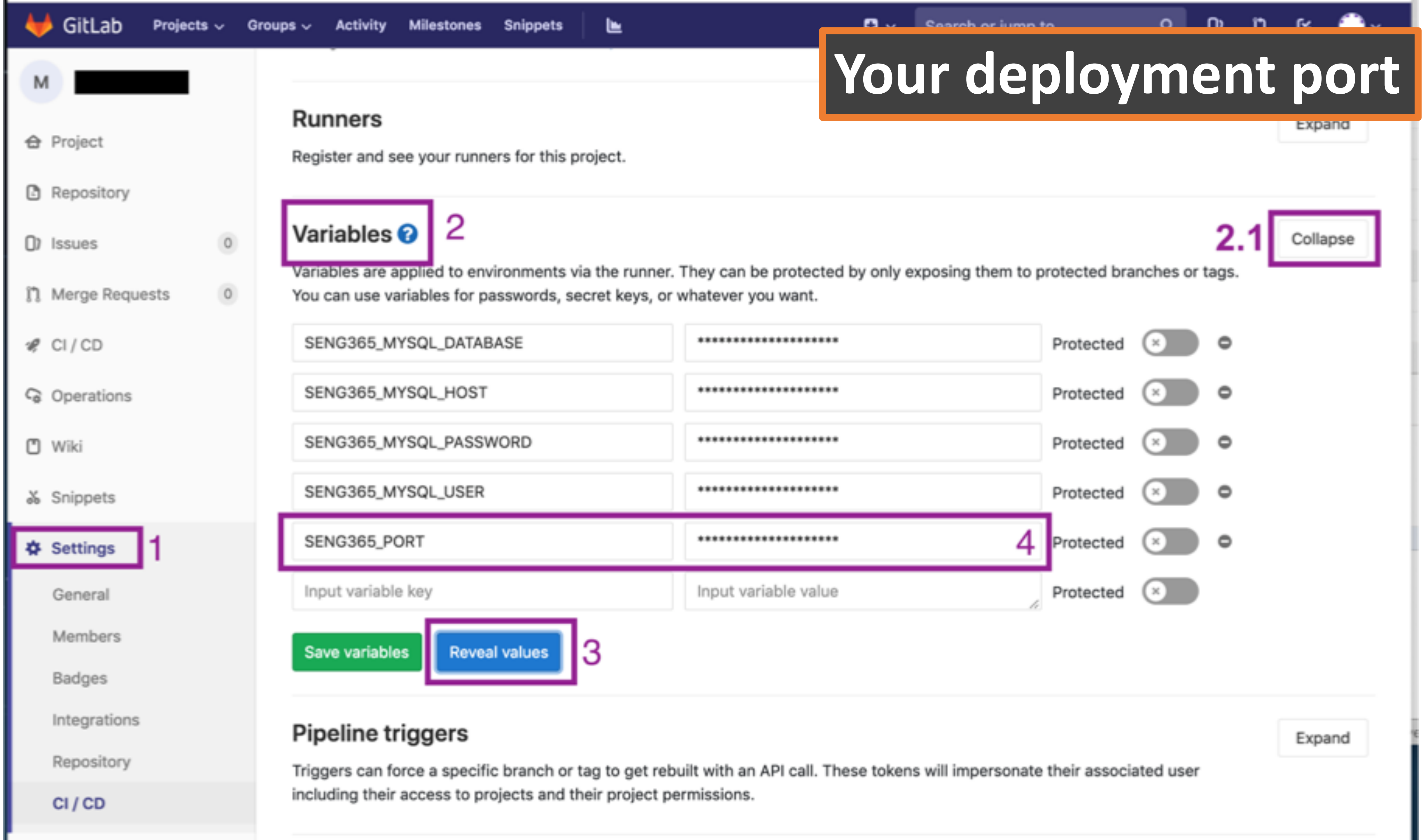
POST /reload Force reset of database to original structure and reload sample of data into database.

POST /reset Force reset of database to original structure.

POST /resample Reload sample of data into database.

POST /executeSql Execute any arbitrary SQL statement provided, and return the output from the database.

Your deployment port



[redacted]

Runners

Register and see your runners for this project.

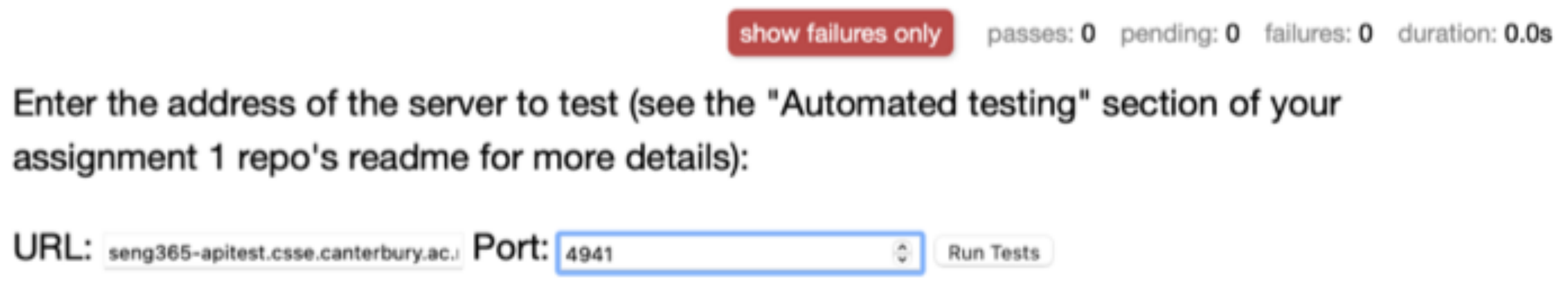
Variables

Variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. You can use variables for passwords, secret keys, or whatever you want.

SENG365_MYSQL_DATABASE	********************	Protected
SENG365_MYSQL_HOST	********************	Protected
SENG365_MYSQL_PASSWORD	********************	Protected
SENG365_MYSQL_USER	********************	Protected
SENG365_PORT	********************	Protected
Input variable key	Input variable value	Protected

Pipeline triggers

Triggers can force a specific branch or tag to get rebuilt with an API call. These tokens will impersonate their associated user including their access to projects and their project permissions.



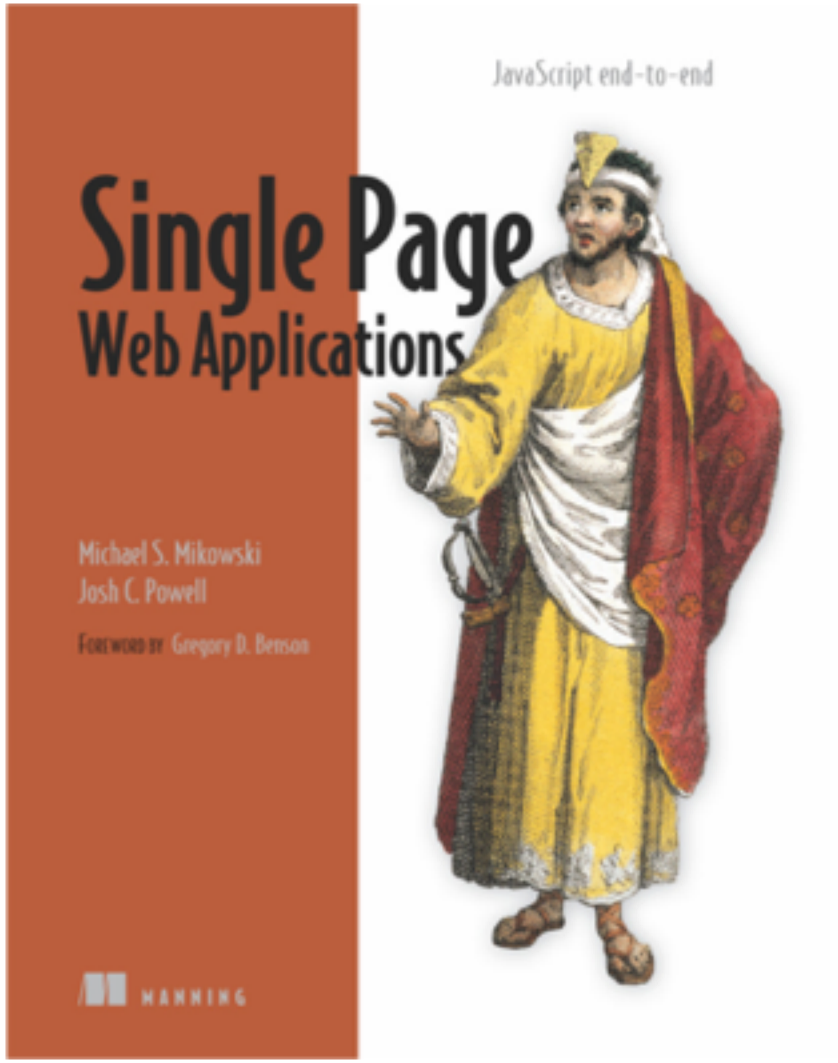
Enter the address of the server to test (see the "Automated testing" section of your assignment 1 repo's readme for more details):

URL: seng365-apitest.csse.canterbury.ac. Port: 4941

Enter the port number assigned to you in the eng-git SENG365_PORT variable

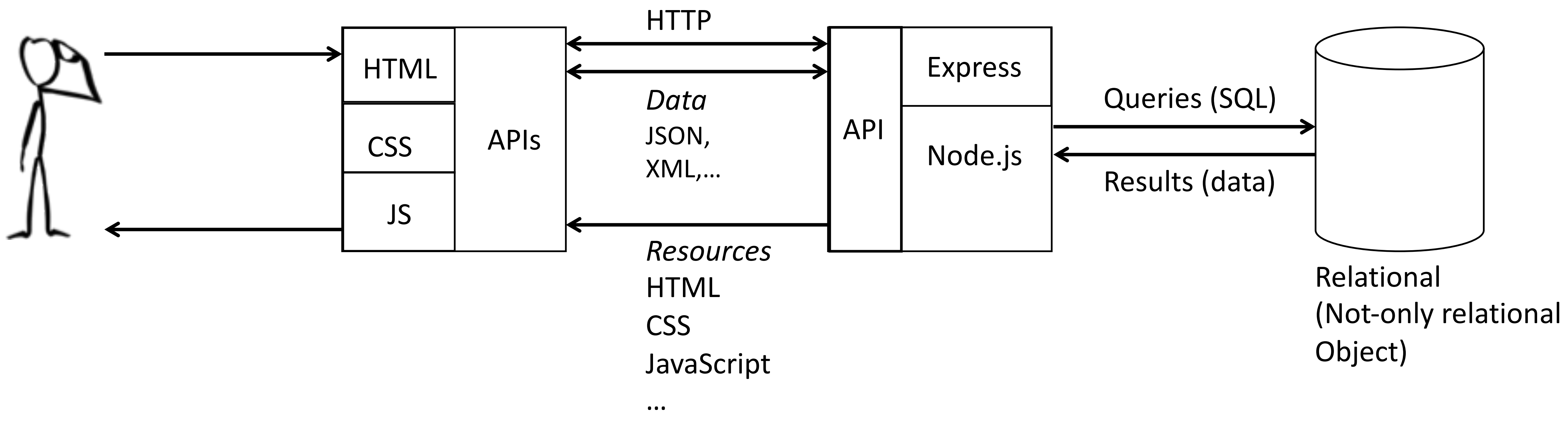
When testing locally in a lab machine (e.g. at localhost) then use 4941

Assumes JavaScript in the browser rather than in Node.js



Asynchronous behaviours

When will x happen? And in what order will X happen?



User	Client	Server	Database
Human	Machine	Machine	Machine

How does JavaScript handle asychronicity

Javascript is a:

- single threaded
- single concurrent language

meaning JavaScript can:

- handle one task at a time or a piece of code at a time.

It has a single **call stack** which along with other parts like heap, queue constitutes the Javascript Concurrency Model (implemented inside of V8).

https://medium.com/@gaurav.pandvia/understanding-javascript-function-executions-tasks-event-loop-call-stack-more-part-1-5683dea1f5ec

The event loop (JavaScript Concurrency Model)

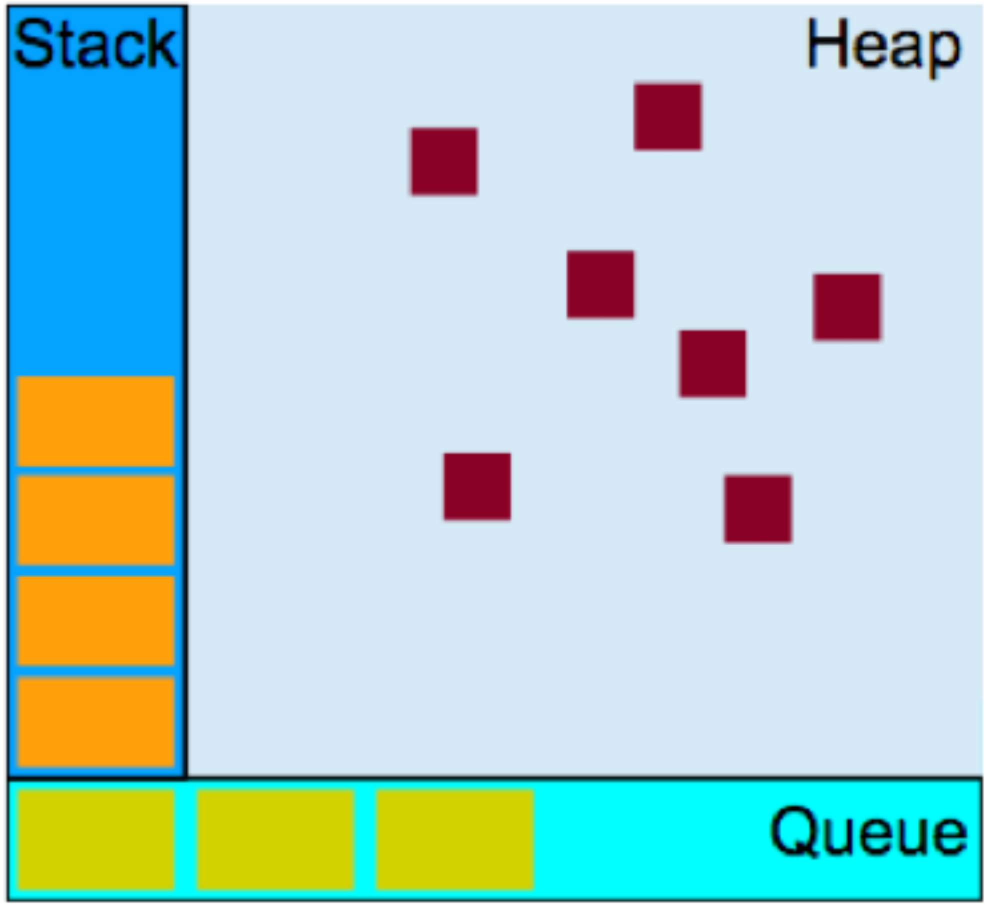
Call Stack: a data structure to maintain record of function calls.

- Call a function to execute: push something on to the stack
- Return from a function: pop off the top of the stack.

(The single thread.)

Heap: Memory allocation to variables and objects.

Queue:a list of messages to be processed and the associated callback functions to execute.

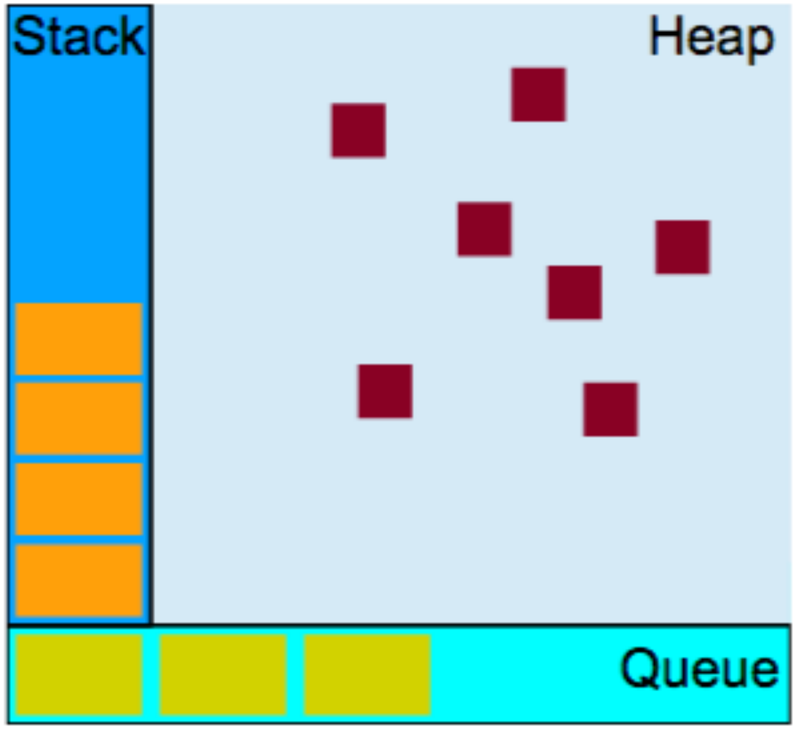


https://developer.mozilla.org/en/docs/Web/JavaScript/EventLoop

An initial example

What will complete first?... **... and why?**

```
Line  Code
1.    setTimeout(() =>
      console.log('first'), 0);
2.    console.log('second')
```

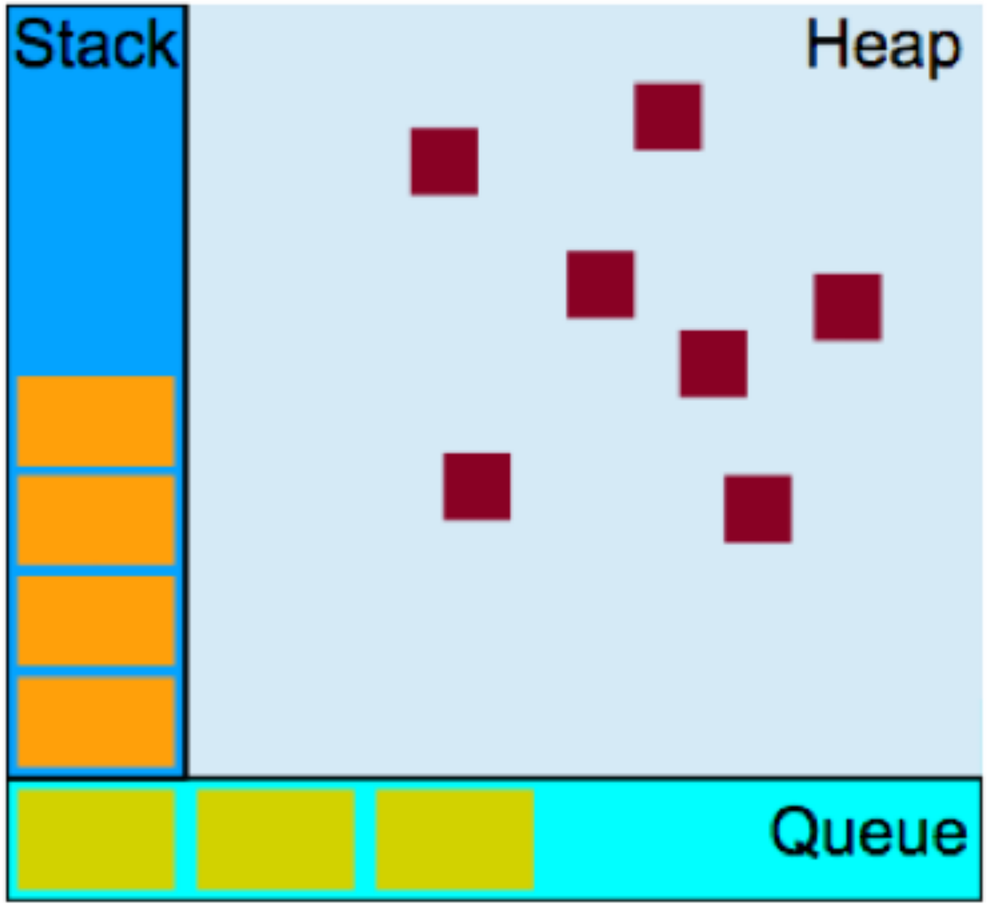


Another example

```
mysql.query('SELECT * from a_table', function
  (error, results, fields){
     if (error) throw error;
     console.log('Done’);
  }
);
```

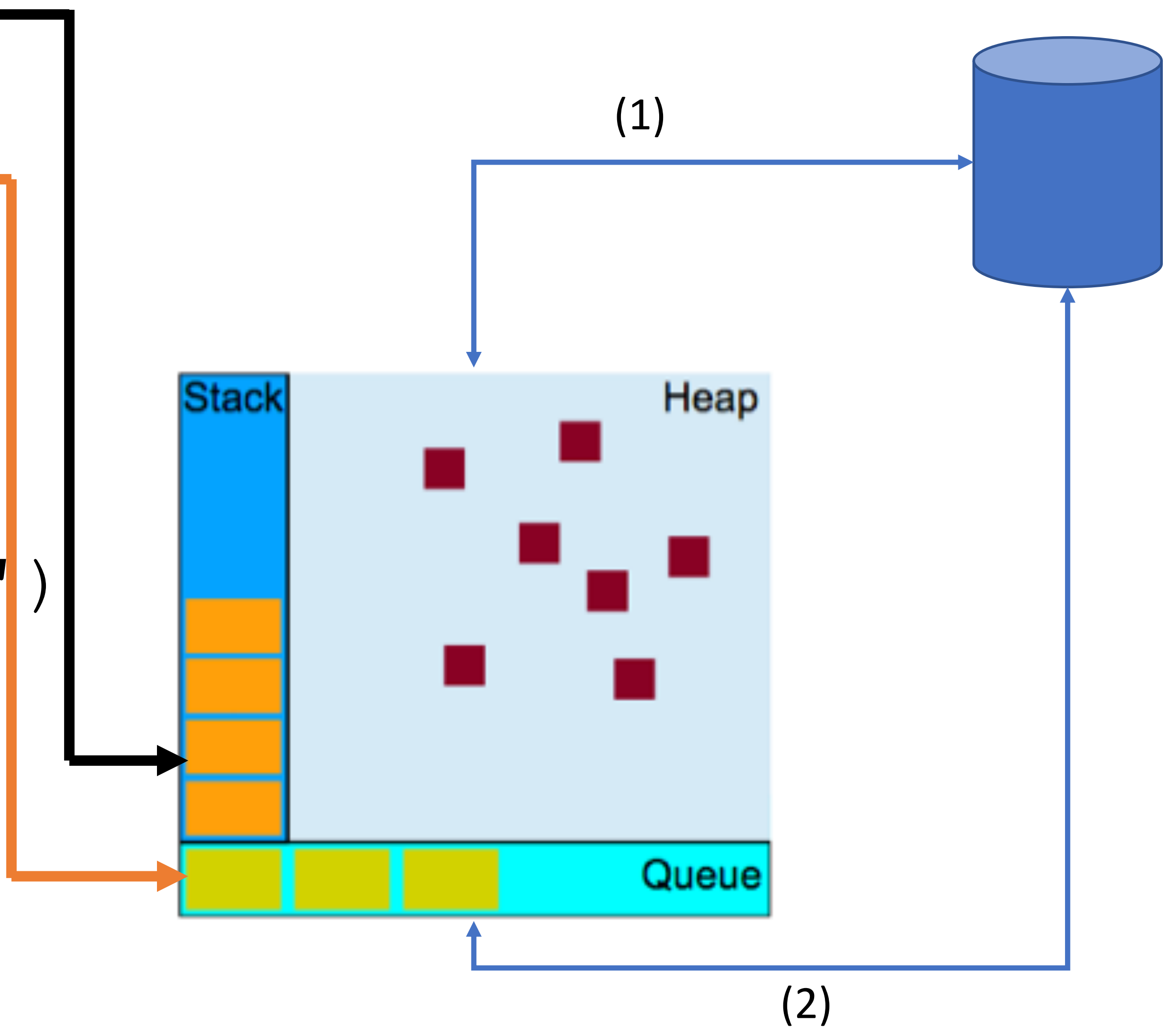
The event loop (JavaScript Concurrency Model)

```
mysql.query('SELECT * from
  a_table', function
  (error, results, fields)
  {
    if (error) throw error;
     console.log('Done');
   }
);
```



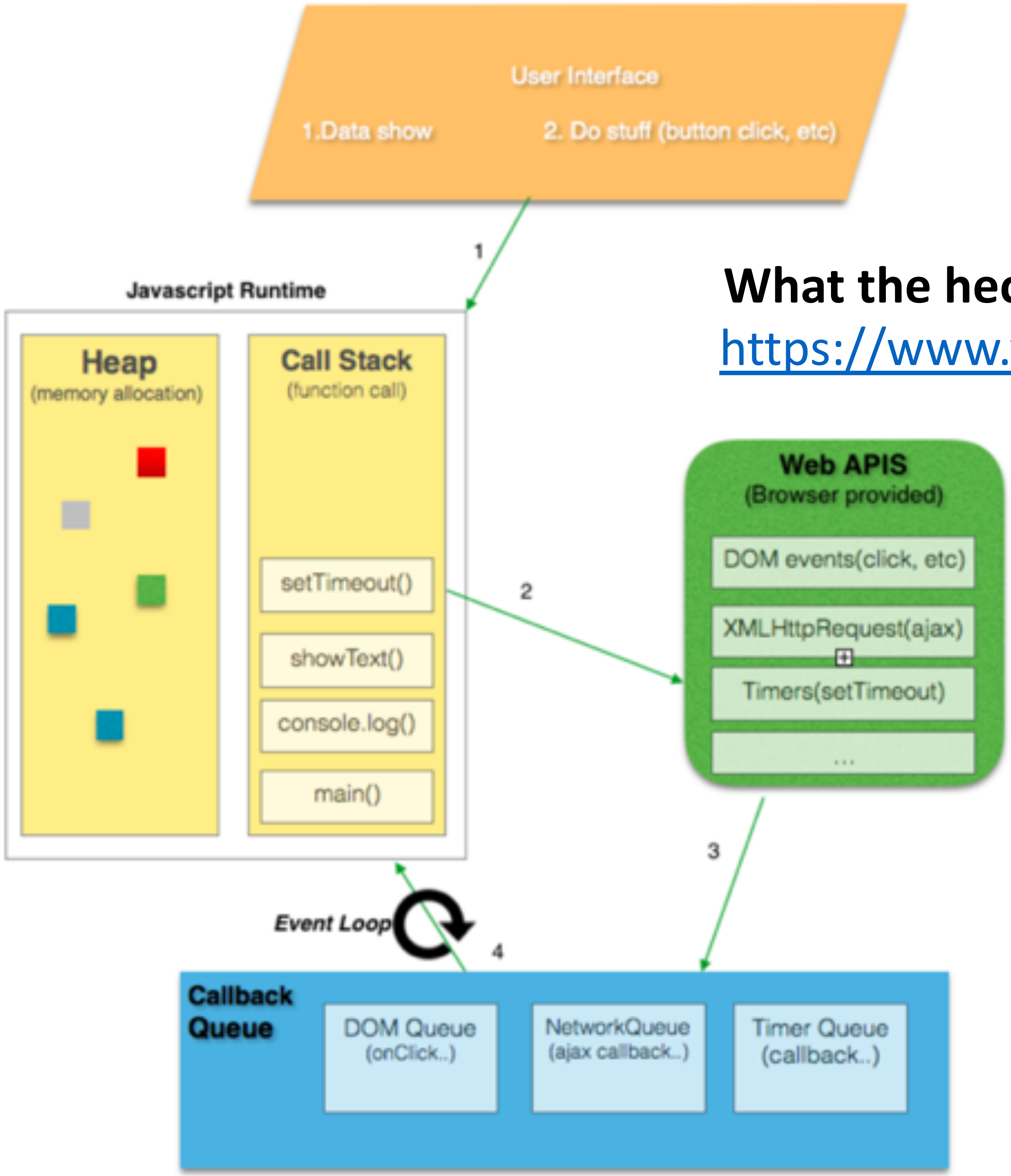
The event loop (JavaScript Concurrency Model)

```
mysql.query('SELECT *
  from a_table',
  function (error,
  results, fields) {
   if (error) throw
  error;
     console.log('Done')
  ;
});
```

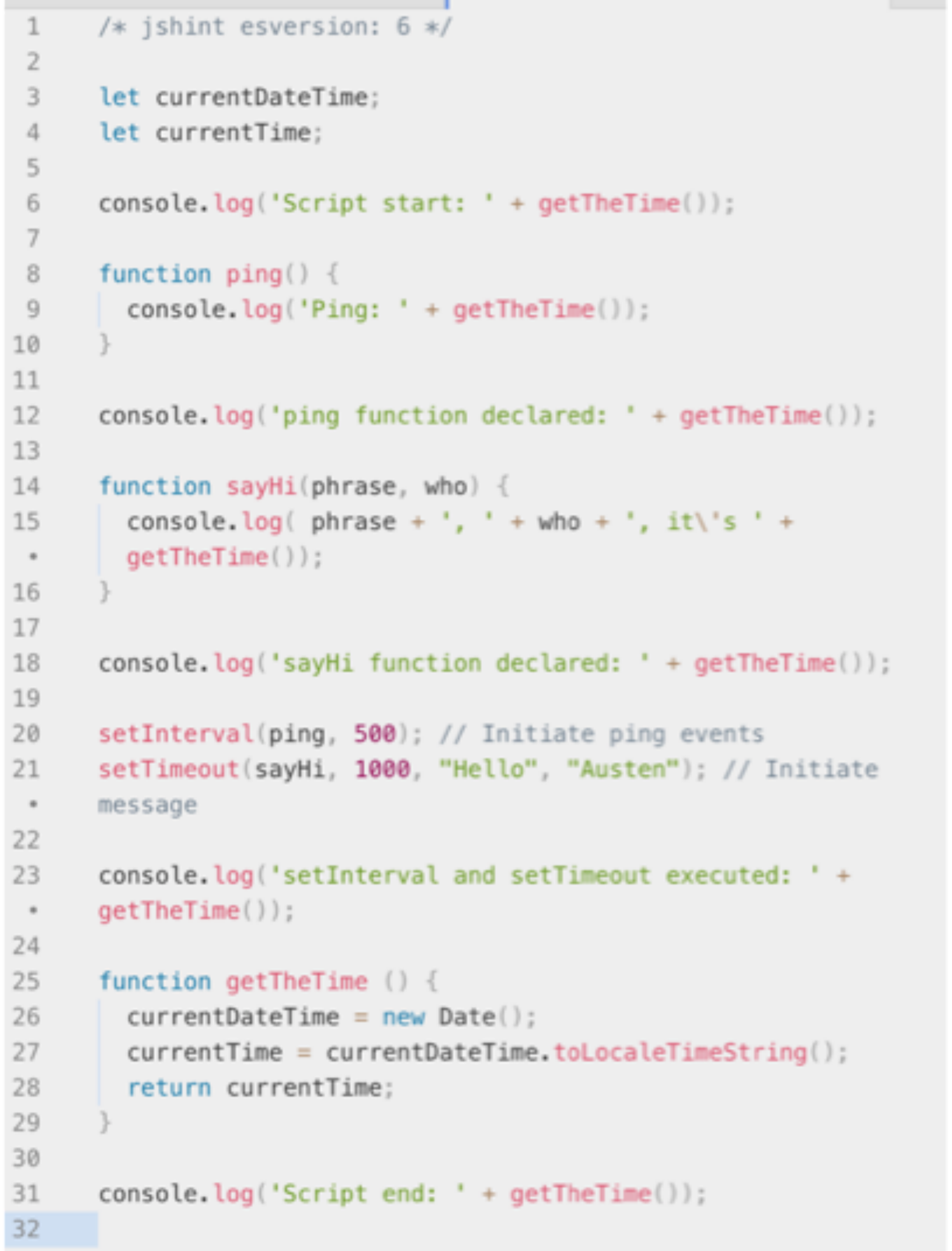


What the heck is the event loop anyway?

https://www.youtube.com/watch?v=8aGhZQkoFbQ



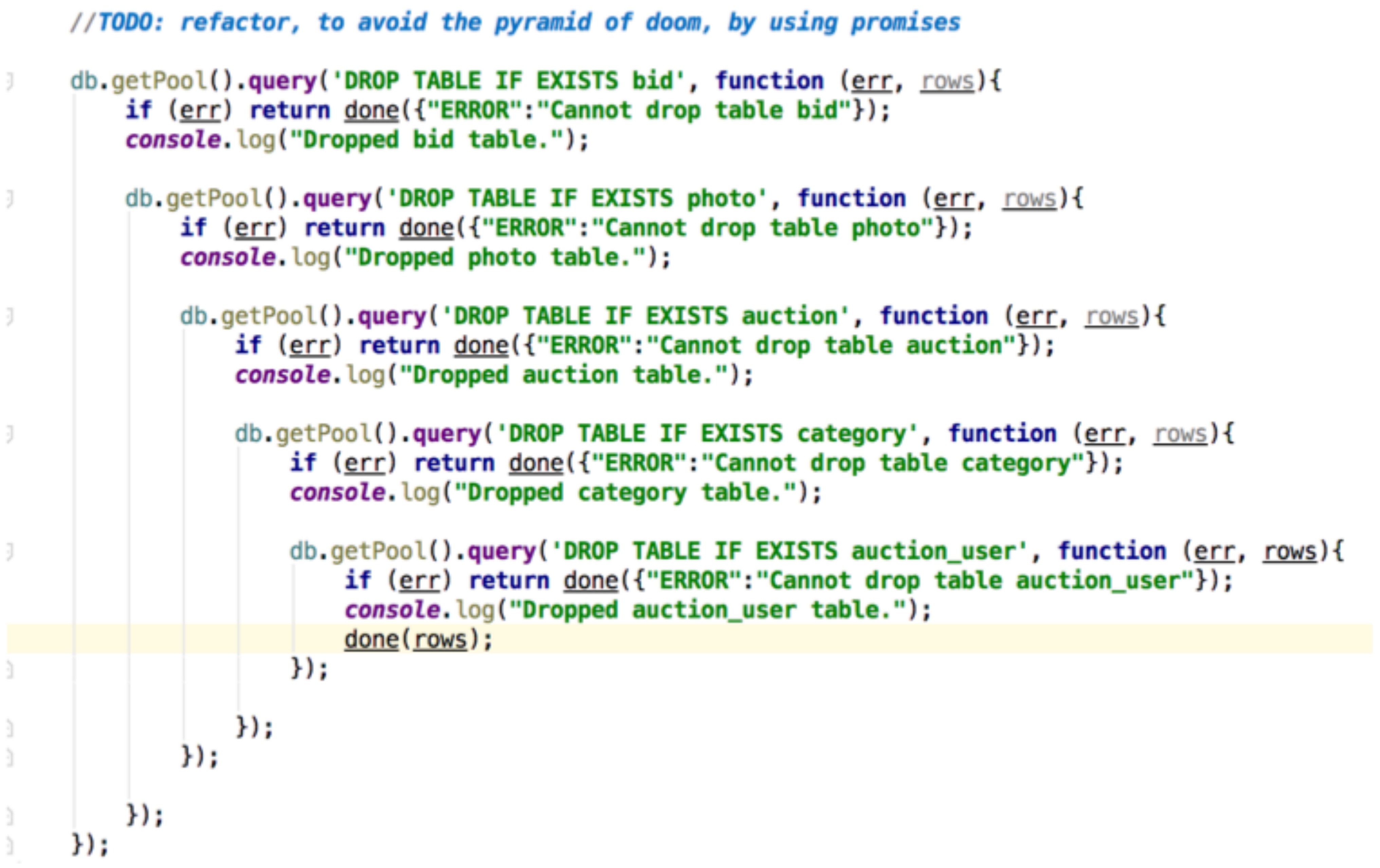
`setTime()` and `setInterval()` code examples



```
/* jshint esversion: 6 */

let currentDateTime;
let currentTime;

console.log('Script start: ' + getTheTime());

function ping() {
  console.log('Ping: ' + getTheTime());
}

console.log('ping function declared: ' + getTheTime());

function sayHi(phrase, who) {
  console.log( phrase + ', ' + who + ', it\'s ' +
  getTheTime());
}

console.log('sayHi function declared: ' + getTheTime());

setInterval(ping, 500); // Initiate ping events
setTimeout(sayHi, 1000, "Hello", "Austen"); // Initiate
message

console.log('setInterval and setTimeout executed: ' +
getTheTime());

function getTheTime () {
  currentDateTime = new Date();
  currentTime = currentDateTime.toLocaleTimeString();
  return currentTime;
}

console.log('Script end: ' + getTheTime());

```

Callback Hell and The Pyramid of Doom

By Asynchronous Code



```
//TODO: refactor, to avoid the pyramid of doom, by using promises

db.getPool().query('DROP TABLE IF EXISTS bid', function (err, rows){
    if (err) return done({"ERROR":"Cannot drop table bid"});
    console.log("Dropped bid table.");

    db.getPool().query('DROP TABLE IF EXISTS photo', function (err, rows){
        if (err) return done({"ERROR":"Cannot drop table photo"});
        console.log("Dropped photo table.");

        db.getPool().query('DROP TABLE IF EXISTS auction', function (err, rows){
            if (err) return done({"ERROR":"Cannot drop table auction"});
            console.log("Dropped auction table.");

            db.getPool().query('DROP TABLE IF EXISTS category', function (err, rows){
                if (err) return done({"ERROR":"Cannot drop table category"});
                console.log("Dropped category table.");

                db.getPool().query('DROP TABLE IF EXISTS auction_user', function (err, rows){
                    if (err) return done({"ERROR":"Cannot drop table auction_user"});
                    console.log("Dropped auction_user table.");
                    done(rows);
                });

            });
        });

    });
});
```

Good practices

- Give anonymous functions a name
  - For example:
    `function (parameter) {…};` becomes
    `function aName (parameter) {…};`
  - Improves readability
  - Give indication of intent of function
  - Helps with debugging stack traces: a named function is easier to identify in the stack trace
- Remove unnecessary callbacks
- Separate conditional code from flow control (where possible)

API calls and Callback Hell

Consider that:

- An API call from the client may *under*-fetch data…
  (the API is not designed to provide all and only the data the client needs)
- … so the client will need to make subsequent API calls.
- For example:
  - First, an API call to get a list of student IDs in order to select one ID, then
  - an API call to get the list of courses studied by that student, and then
  - another API call to get further information of specific courses
- This produces a nested (conditional) set of API calls
  - For each call, the client must test whether the call was successful or not

Promises

https://developers.google.com/web/fundamentals/primers/promises

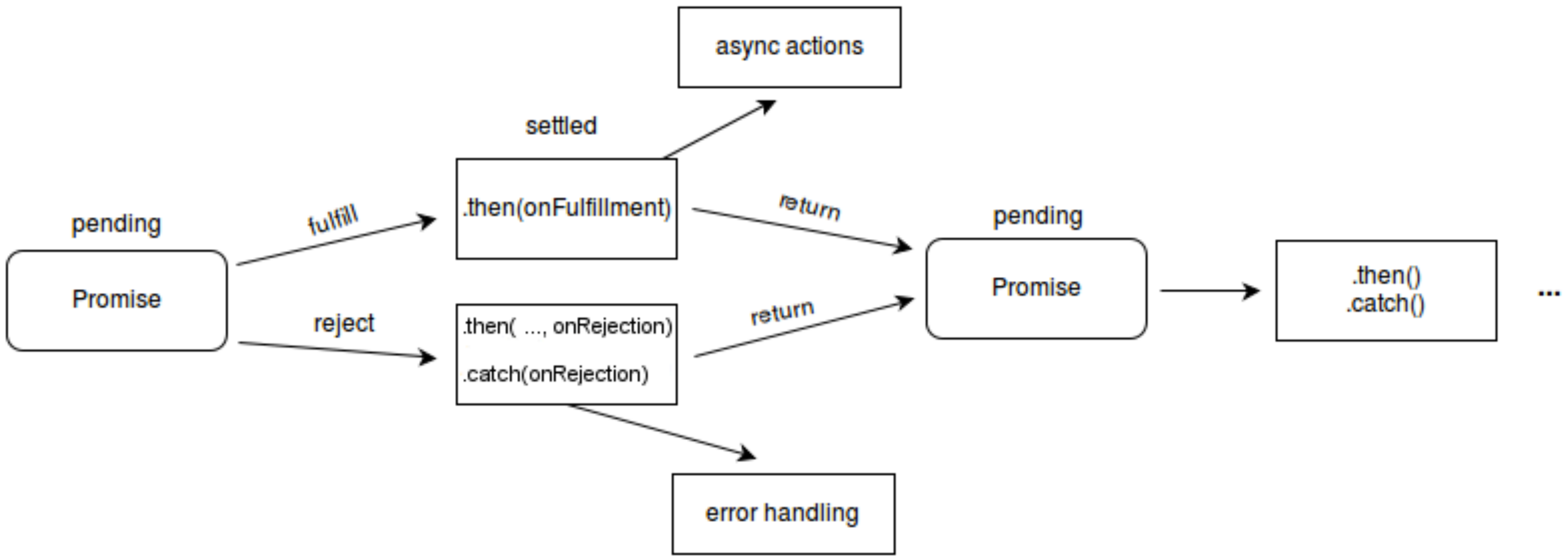
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

Promise

- The Promise object is used for deferred and asynchronous computations.
- Promises allow you to use synchronous and asynchronous operations with each other
- A Promise represents an operation that hasn't completed yet, but is expected in the future.
  - *pending*: initial state, not fulfilled or rejected.

  Or resolved as either:
  - *fulfilled*: meaning that the operation completed successfully.
  - *rejected*: meaning that the operation failed.

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise



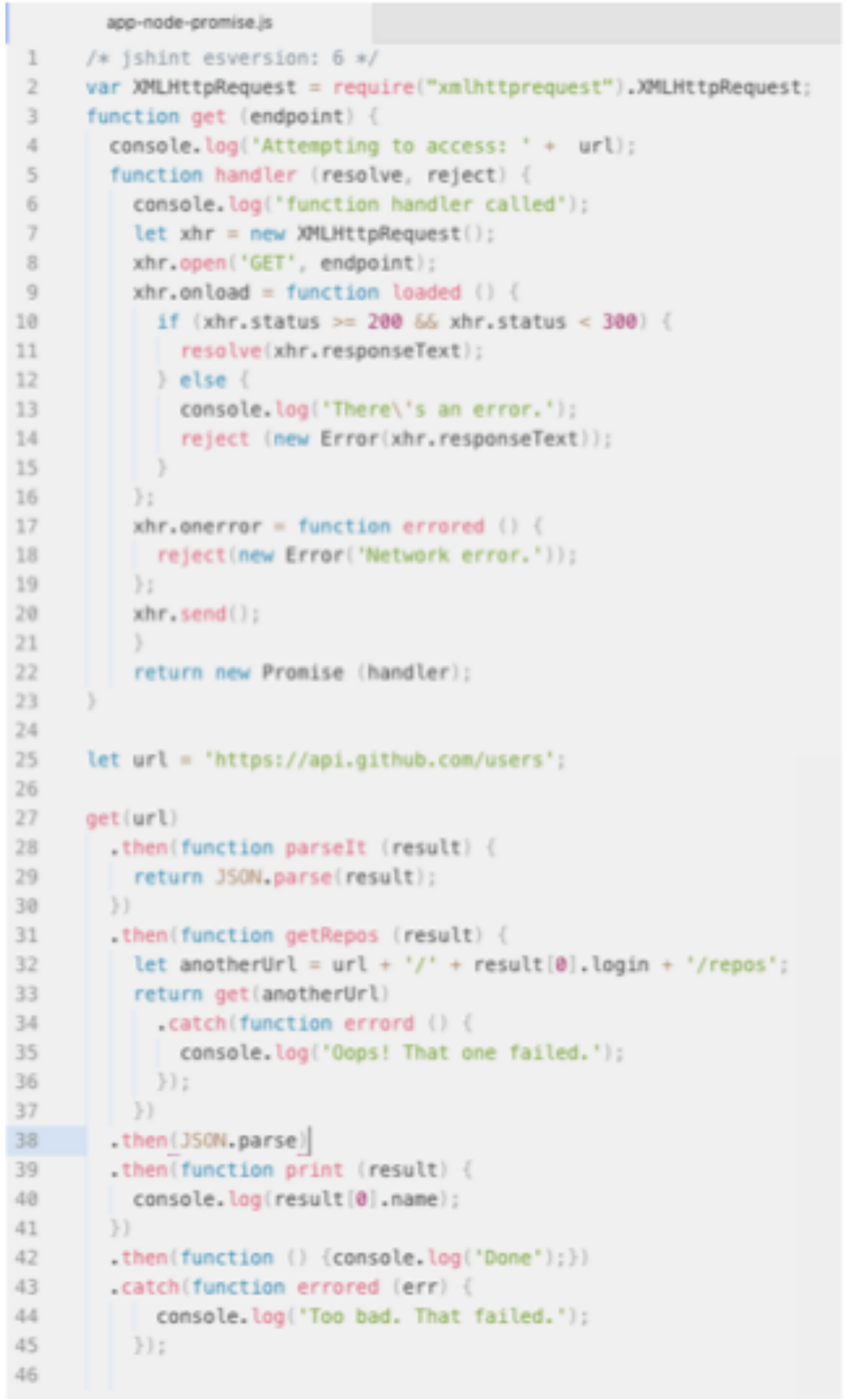
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise#Example_using_new_XMLHttpRequest()

Chaining Promise

- Each Promise is first pending, and then (eventually) either fulfilled OR rejected
- Chaining Promises allows you to chain dependent asynchronous operations, where each asynchronous operation is itself a Promise
- Each Promise represents the completion of another asynchronous step in the chain.
- To chain Promises, each Promise returns another Promise
    - Technically, each `then()` returns a Promise
- Chain Promises together using `.then()`
- Can have multiple `.then()`s
- Handles rejected state/s with `.catch()` (or `.then(null, callback)`)

Code example of Promises

- Using XMLHttpRequest on Node.js



app-node-promise.js

```javascript
/* jshint esversion: 6 */
var XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;
function get (endpoint) {
  console.log('Attempting to access: ' +  url);
  function handler (resolve, reject) {
    console.log("function handler called");
    let xhr = new XMLHttpRequest();
    xhr.open('GET', endpoint);
    xhr.onload = function loaded () {
      if (xhr.status >= 200 && xhr.status < 300) {
        resolve(xhr.responseText);
      } else {
        console.log('There\'s an error.');
        reject (new Error(xhr.responseText));
      }
    };
    xhr.onerror = function errored () {
      reject(new Error('Network error.'));
    };
    xhr.send();
    }
    return new Promise (handler);
}

let url = 'https://api.github.com/users';

get(url)
  .then(function parseIt (result) {
    return JSON.parse(result);
  })
  .then(function getRepos (result) {
    let anotherUrl = url + '/' + result[0].login + '/repos';
    return get(anotherUrl)
      .catch(function errord () {
        console.log('Oops! That one failed.');
      });
    })
  .then(JSON.parse)
  .then(function print (result) {
    console.log(result[0].name);
  })
  .then(function () {console.log('Done');})
  .catch(function errored (err) {
      console.log("Too bad. That failed.");
    });

```

ASYNC / AWAIT

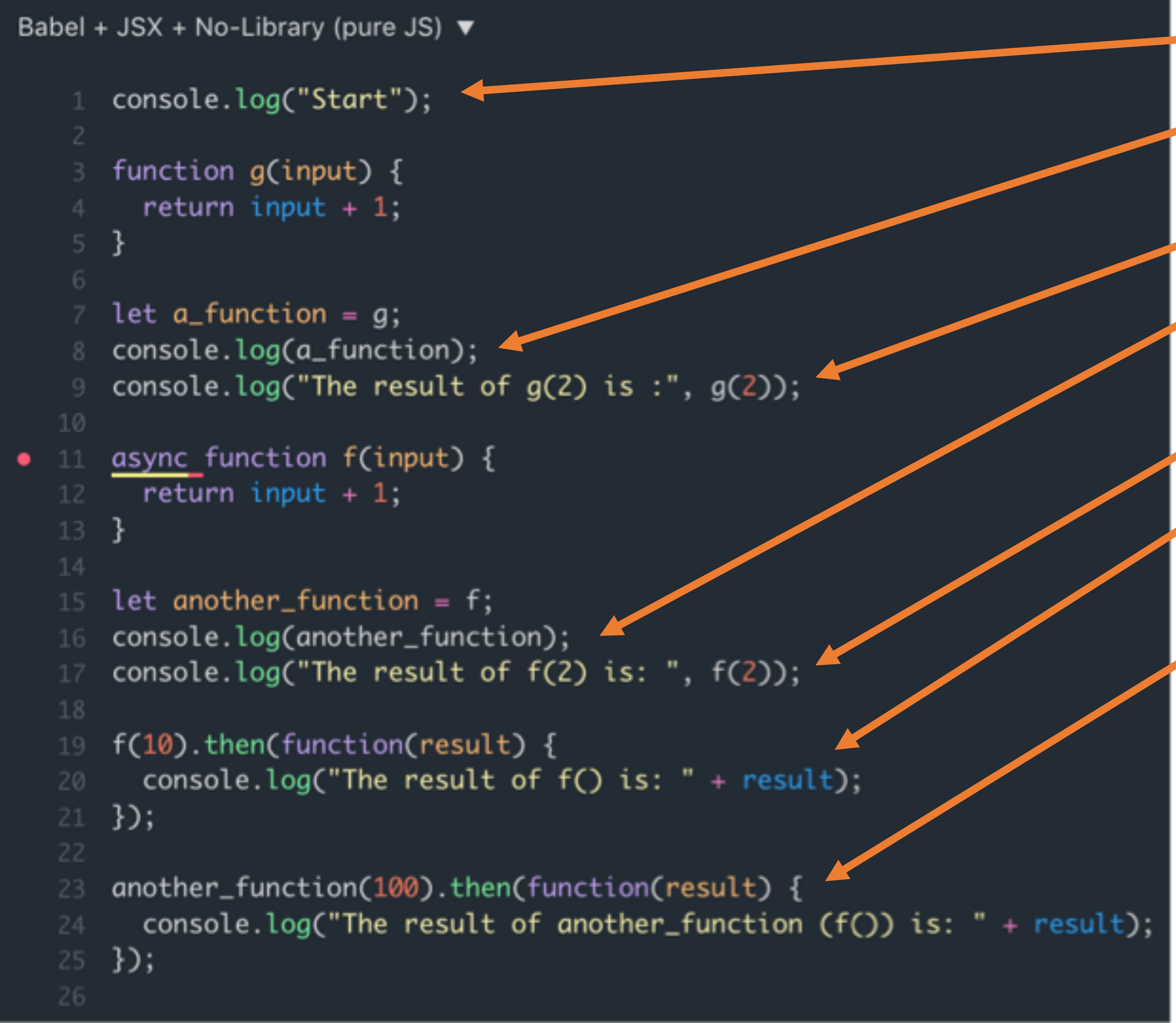
https://cdn-images-1.medium.com/max/2000/1*uJNC6fNLng8O0r9QvDgNDw.jpeg

`async`: Ensures a function returns a Promise (A kind of Promise wrapper)

```
async function f() {
  return 1;
}
f().then( () => { console.log(result); } );
```

Note:

async doesn’t execute the function immediately



Babel + JSX + No-Library (pure JS) ▼

```
console.log("Start");

function g(input) {
  return input + 1;
}

let a_function = g;
console.log(a_function);
console.log("The result of g(2) is :", g(2));

async function f(input) {
  return input + 1;
}

let another_function = f;
console.log(another_function);
console.log("The result of f(2) is: ", f(2));

f(10).then(function(result) {
  console.log("The result of f() is: " + result);
});

another_function(100).then(function(result) {
  console.log("The result of another_function (f()) is: " + result);
});
```

```
Start

function g(input) {
return input + 1;
}

The result of g(2) is : 3

function f(_x) {
return _ref.apply(this, arguments);
}

The result of f(2) is: [object Promise]

The result of f() is: 11

The result of another_function (f()) is: 101
```

https://jsfiddle.net/dnxquzym/15/

`await`: forces JavaScript wait for the Promise to resolve

- `await` is only legal inside an `async function`…
- … and async functions are Promises that commit to a future resolution…
- … so other code can continue to run

Module dependencies

module.export / require()

Modular JavaScript files

- CommonJS: **one** specification for managing module dependencies
  - Others exist e.g. RequireJS, ES2015 AMD (Async Module Definition)
- Node.js adopted CommonJS
  - To use CommonJS on front-end, you'll need to use Browserify (or similar)
- A module is defined by a single JavaScript file
- Use `module.exports.*` or `exports.*` (but not both) to expose your module's public interface
- Values assigned to `module.exports` are the module's public interface
  - A value can be lots of things e.g. string, object, function, array
  - You want to expose something? Add it to `module.exports`
- Import the module using `require()`

Creating modules & reusing existing module

You can of course create your own modules

```
  myModule.js
    module.exports...
```

And then reuse that module :

```
  myOtherModule.js
    var something = require('../../myModule.js');
```

Dependency management

- npm is (obviously) a package manager for node
- npm is designed to be node-specific
- `npm install` installs packages suitable for the CommonJS-like dependency management used by Node i.e. the `exports/requires` approach

Creating modules & reusing existing module

You can reuse existing modules provided by the node ecosystem

First, install the existing module through npm

```
  > npm install aModule
```

And then reuse that module :

```
  myOtherModule.js
    var something = require(aModule);
```

Note the differences in parameters for node and home-grown modules

Example from the lab sheets

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

- There’s a JavaScript file, called `express.js`, somewhere (hopefully) in your project
  - Do `npm install express` to get the file/module into your project

- The `express.js` file exposes a public interface using `module.exports()` (or maybe `exports()`)

SENG365 Web Computing Architecture: #2 JavaScript cont. and Asynchronous behavior

Ben Adams
Course Coordinator & Lecturer
benjamin.adams@canterbury.ac.nz
310, Erskine Building