Node.js, Express.js

Web Development and Security (ZEIT3119)

Week 9

Dr. Reza Rafeh

Revision

- MVC (Model-View-Controller)
  - A pattern for implementing user interfaces, data, and controlling logic
  - MVC emphasizes a separation between the software's business logic and display.
- Laravel
  - An open-source PHP web framework designed for creating web applications that follow the Model–View–Controller (MVC) architectural pattern
  - Composer is the dependency manager for Laravel much like npm for Node
  - Artisan is the command line interface included with Laravel
  - Eloquent is an Object-Relational Mapping (ORM) that allows you to query & manipulate data using an Object-Oriented programming language
- Postman
  - An API Platform for developers to design, build, test and iterate their APIs.

Outline

- Node.js
  - Client Requests
  - npm
  - File Handling
  - Connect with Database
  - Asynchronous Functions
- Express.js
  - URL Queries
  - Form Data
  - Connect with Database
  - HTTPS Module

Node.js

- Node.js is an open-source, cross-platform runtime environment for executing JavaScript code on the server.
- Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.
- It was initially released in 2009 by Ryan Dahl, and has since become one of the most popular tools for building server-side applications and APIs.
- Node.js is fast and lightweight, making it an excellent choice for building scalable, high-performance applications.
- It uses an event-driven, non-blocking I/O model, which allows it to handle a large number of concurrent connections without blocking the event loop.
- It also has a large and active community of developers, who contribute to its ongoing development and maintenance.

Use Cases for Node.js

- Node.js is well-suited for building real-time, data-intensive applications such as chat applications, online gaming platforms, and streaming services.
- It is also commonly used for building web applications and APIs, as well as for running scripts and automating tasks on the server-side.
- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

Getting started with Node.js

- To get started with node.js, you'll need to install it on your computer and set up a development environment.

https://nodejs.org/en/download

- You can then use a package manager like npm to install and manage dependencies, and a framework like Express.js to build web applications and APIs.
- There are also many online resources available for learning node.js, including documentation, tutorials, and community forums.

https://www.w3schools.com/nodejs

A Simple Node.js Example

```
console.log('Welcome to ZEIT3119');

console.log('This is an example of using console on server');
```

C:\Windows\system32\cmd.exe

```
D:\USB\UNSW\S1-2023\Lectures\Lecture 9 Materials>node console.js
Welcome to ZEIT3119
This is an example of using conole on server

D:\USB\UNSW\S1-2023\Lectures\Lecture 9 Materials>
```

First Node.js Example

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Welcome to ZEIT3119!');
}).listen(8080);
```

A node.js module

Creates a web server

200 means all is good

Listen to this port for requests

C:\Windows\system32\cmd.exe - node first.js

```
D:\USB\UNSW\S1-2023\Lectures\Lecture 9 Materials>node first.js
```

localhost:8080/

localhost:8080

Welcome to ZEIT3119!

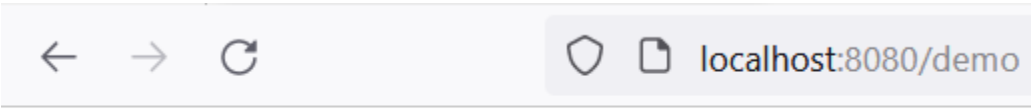
UNSW CANBERRA

Client Request

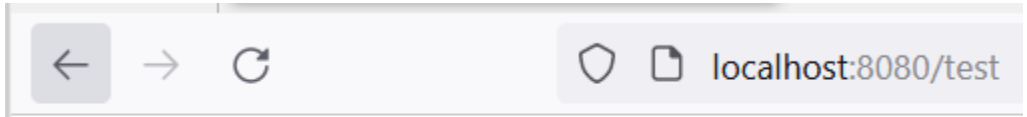
Client Request

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

URL part of Client Request



/demo



/test

URL Parameters

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  res.write("  "+JSON.stringify(q));
  res.write("  "+q.id);
  res.write("  "+q.name);
  res.end("  "+q.course);
}).listen(8080);
```

localhost:8080/?id=38271&name=Reza&course=ZEIT3119

{"id":"38271","name":"Reza","course":"ZEIT3119"} 38271 Reza ZEIT3119

Check Response in Postman

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type':
    'text/html'});
  var q = url.parse(req.url, true).query;
  res.end(q);
}).listen(8080);
```

GET http://localhost:8080/?id=38271&name=Reza&course=ZEIT3119

Params • Authorization Headers (8) Body • Pre-request Script Tests

Query Params

	Key	Value
☑	id	38271
☑	name	Reza
☑	course	ZEIT3119
☐	price	8.99
☐	description	Beef, Tomato, Sauce
☐	picture	uploads/4nVoQfqOerD3
	Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize HTML

```
1  {"id":"38271","name":"Reza","course":"ZEIT3119"}
```

File Handling

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
    fs.readFile('first.js', function(err, data) {
          res.end(data);
    });
}).listen(8080);
```

localhost:8080

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Welcome to ZEIT3119!');
}).listen(8080);
```

npm

- npm is a package manager for Node.js packages, or modules
- www.npmjs.com hosts thousands of free packages to download and use
- The npm program is installed on your computer when you install Node.js

```
npm install formidable
```

File Upload

```
var http = require('http');
var formidable = require('formidable');
var fs = require('fs');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      var oldpath = files.filetoupload.filepath;
      var newpath = 'd:/laragon/www/uploads/' + files.filetoupload.originalFilename;
      fs.copyFile(oldpath, newpath, function (err) {
        if (err) throw err;
        res.write('File uploaded and copied!');
        res.end();
      });
  });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

Parsing the uploaded file

Create an upload form

Node.js MySQL

```
npm install mysql
```

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "lecture8"
});

con.connect(function(err) {
    if (err) throw err;
    con.query("SELECT * FROM institutions", function (err, result, fields) {
      if (err) throw err;
      console.log(result);
    });
  })
```

```
D:\USB\UNSW\S1-2023\Lectures\Lecture 9 Materials>node connectdb.js
[
  RowDataPacket {
    id: 1000,
    name: 'Stanford University',
    region: 'California',
    country: 'United States of America',
    created_at: 2023-04-30T03:04:35.000Z,
    updated_at: 2023-04-30T03:04:35.000Z
  },
  RowDataPacket {
    id: 2000,
    name: 'Harvard University',
    region: 'Massachusetts',
    country: 'United States of America',
    created_at: 2023-04-30T03:04:35.000Z,
    updated_at: 2023-04-30T03:04:35.000Z
  },
  RowDataPacket {
    id: 3000,
    name: 'University of Oxford',
    region: 'Oxford',
    country: 'United Kingdom',
    created_at: 2023-04-30T03:04:35.000Z,
    updated_at: 2023-04-30T03:04:35.000Z
  }
]
```

Using Variables in SQL Queries

```
var ID = 2000;
con.connect(function(err) {
    if (err) throw err;
    con.query("SELECT * FROM institutions where id="+ID, function (err, result, fields) {
      if (err) throw err;
      console.log(result);
    });
  })
```

Variable

Synchronous and Asynchronous Models

Synchronous Model

- The waiter comes to the customer, asks for his/her order, sends it to the kitchen, and then waits till the order is complete before bringing the food to the customer.
- This model blocks the waiter from doing any other thing until the food is ready and, as such, will be pretty much handicapping to scale.

Asynchronous Model

- With the asynchronous model, the waiter will come to the customer, take his order, send it to the kitchen, doesn't wait until the food is done, but return and take more orders from other customers.
- Once the chef completes an order, the waiter will receive a signal to come to send the order to the customer who placed it. This will be repeated until the chef processes all the orders in the kitchen. This model is called non-blocking because it does not stop the waiter from taking more orders from the customers.

- Ref: https://www.knowledgehut.com/blog/web-development/mastering-node-js-await

Example of Synchronous Code

```
console.log("Get order 1");

console.log("Send order 1 to kitchen");

console.log("Wait for food prep...");

console.log("Return order to customer 1");

console.log("Get order 2");
```

```
D:\USB\UNSW\S1-2023\Lectures\Lecture 9 Materials\DBExpress2\asyncnodejs>node sync.js
Get order 1
Send order 1 to kitchen
Wait for food prep...
Return order to customer 1
Get order 2
```

Example of Asynchronous Code

```
console.log("Get order 1");

console.log("Send order 1 to kitchen");

setTimeout(() => {

 console.log("Wait for food prep, then service customer 1");

 console.log("Return order to customer 1");

}, 2000);

console.log("Get order 2");
```

```
D:\USB\UNSW\S1-2023\Lectures\Lecture 9 Materials\DBExpress2\asyncnodejs>node async.js
Get order 1
Send order 1 to kitchen
Get order 2
Wait for food prep, then service customer 1
Return order to customer 1
```

Asynchronous Programming in JS

In JavaScript, there are three design patterns for dealing with asynchronous programming:

- callbacks
- promises
- async/await (just a syntactical sugar of promises)

Callback

```
const placeOrder = (customerId, callback) => {
    console.log("Preparing dish...")
    setTimeout(() => {
      console.log("Dish Prepared...")
      callback({customerId: customerId, customerOrder: 'Pizza'})
    }, 2000);
  }
  placeOrder(1, (order) => console.log("Order", order))
```

Promises

Promises have four states:

- **fulfilled** - The action succeeded
- **rejected** - The action failed
- **pending** - Action yet to be fulfilled or rejected
- **settled** - Action fulfilled or rejected

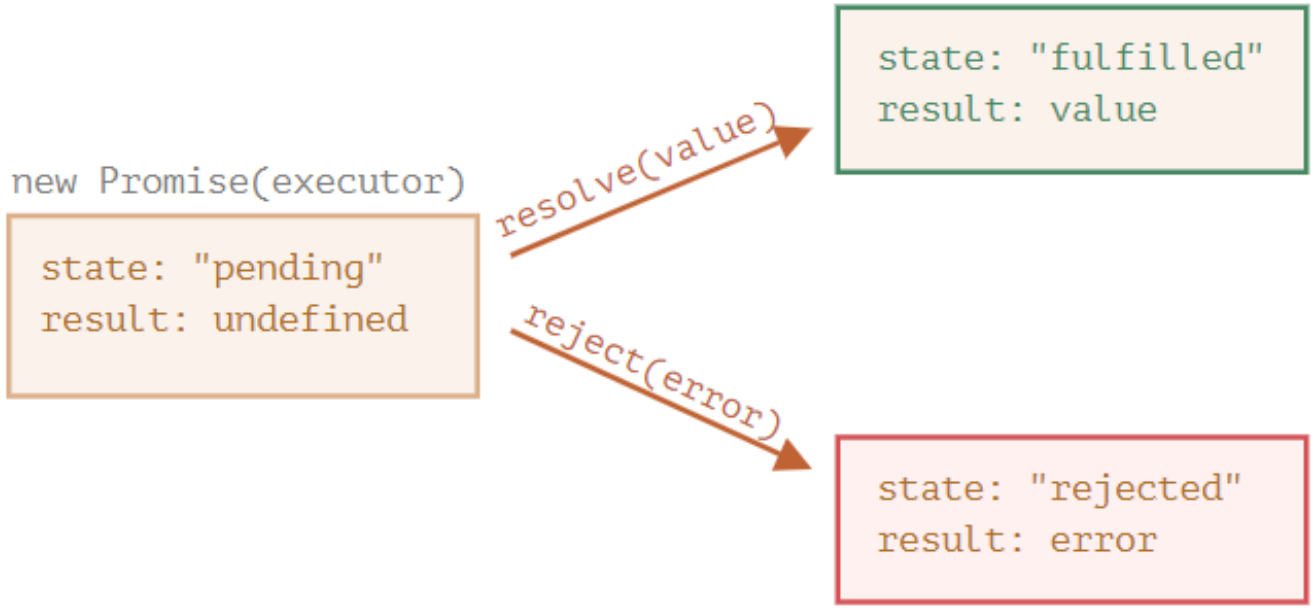
Promises (Cont.)

```
const meetCustomer = (id) => {
      return new Promise((resolve, reject) => {
        setTimeout(() => {
          console.log(`Waiter approached customer at table #${id}...`);
          resolve({ customerId: id });
        }, 2000);
      });
    }
const getOrder = (id) => {
          return new Promise((resolve, reject) => {
            setTimeout(() => {
              console.log(`Order Received for customer at table #${id}...`);
              resolve({ customerId: id, customerOrder: "Pizza" });
            }, 2000);
          });
        }
```

Promises (Cont.)

```
const notifyWaiter = (id) => {
              return new Promise((resolve, reject) => {
                setTimeout(() => {
                  console.log(`Order for customer at table #${id} processed....`);
                  resolve({ customerId: id, customerOrder: "Pizza" });
                  // reject(new Error("Error occured with waiter"));
                }, 2000);
              });
            }
const serveCustomer = (id) => {
              return new Promise((resolve, reject) => {
                setTimeout(() => {
                  console.log(`Customer with order number #${id} served...`);
                  resolve({ customerId: id, customerOrder: "Pizza" });
                }, 2000);
              });
            }
```

```
# Replacing Callback with Promises
meetCustomer(1)
  .then((order) => getOrder(order.customerId))
  .then((order) => notifyWaiter(order.customerId))
  .then((order) => serveCustomer(order.customerId))
  .catch((err) => console.log("Error: ", err.message));
```



Async/await

```
const runRestaurant = async (customerId) => {
              const customer = await meetCustomer(customerId)
              const order = await getOrder(customer.customerId)
              await notifyWaiter(order.customerId)
              await serveCustomer(order.customerId)
             console. log(`Order of customer fulfilled...`)
             }
runRestaurant(1)
  .then(() => console.log(`Order of customer fulfilled...`))
  .catch((error) => console.log(error))
There is no need to use the await modifier and the throw new
Error().
```

Express.js

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
- Express is an open source framework developed by TJ Holowaychuk and maintained by the Node.js foundation.
- It supports building server-side web applications, APIs, and websites.

Install Express.js

- You need to have node and npm. To ensure they are both installed:

  node --version

  npm –version

- To create a new project, you need a package.json file.

  npm init

Then, install express:

  npm install --save express

- The --save flag can be replaced by the -S flag. This flag ensures that Express is added as a dependency to our package.json file.

- To restart the server as soon as making a change in any of files, install nodemon:

  npm install -g nodemon

The package installed globallly (not just in the folder)

First Express.js app

```
var express = require('express');
var app = express();
   app.get('/', function(req, res){
   res.send("Welcome to ZEIT3119");
});
   app.listen(3000);
```

route

Call back

localhost:3000

Welcome to ZEIT3119

GET http://localhost:3000

Params • Authorization Headers (8) Bo

Query Params

	Key
☐	description
☐	picture
	Key

Body Cookies Headers (7) Test Results

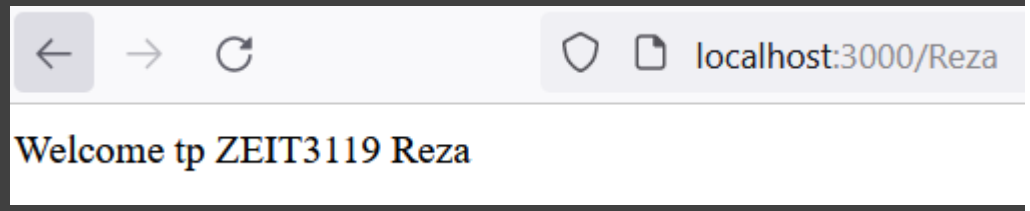
Pretty Raw Preview Visualize

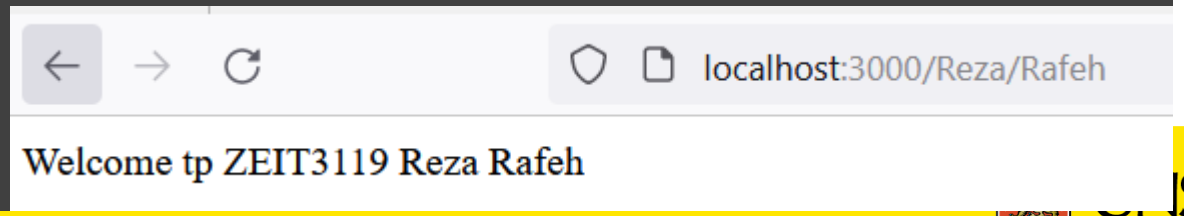
1 Welcome to ZEIT3119

UNSW CANBERRA

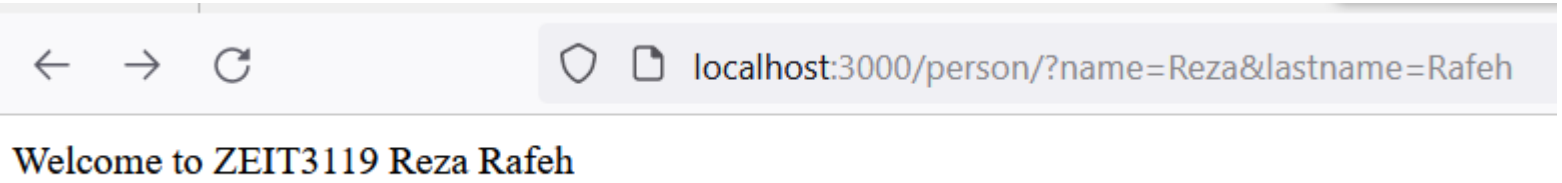
URL Binding - params

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

var app = express();

    app.get('/', function(req, res){

   res.send("Welcome to ZEIT3119");

});

    app.get('/:name', function(req, res){

   res.send('Welcome to ZEIT3119 ' + req.params.name);

});

    app.get('/:name/:lastname', function(req, res){

   res.send('Welcome to ZEIT3119 ' + req.params.name +' '+req.params.lastname);

});

app.listen(3000);
```





URL Query




```
app.get('/person', function(req, res){
   const name = req.query.name;
   const lastname = req.query.lastname;
   res.send("Welcome to ZEIT3119" + name +' '+lastname);
});
```

Note: If you add this to the end of the previous app, it doesn't work. It must be added before this route:

```
app.get('/:name', function(req, res){
   res.send('Welcome to ZEIT3119 ' +
req.params.name);
});
```

Form Data

- Install this package

  npm install express body-parser multer

for parsing application/json

for parsing application/xwww-form-urlencoded

for parsing multipart/form-data

```
var express = require('express');
var bodyParser = require('body-
parser');
var multer = require('multer');
var upload = multer();
var app = express();
app.use(bodyParser.json());

app.use(bodyParser.urlencoded({
extended: true }));

app.use(upload.array());
app.use(express.static('public'));

app.post('/', function(req, res){
   console.log(req.body);
   res.send("recieved your request!");
});
app.listen(3000);
```

Form Data

POST http://localhost:3000/

Params • Authorization Headers (8) Body • Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

	Key	Value
☑	id	500
☑	name	RMIT
☑	region	VIC
☑	country	Australia
	Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize HTML

```
1  recieved your request!
```

```
[Object: null prototype] {
  id: '500',
  name: 'RMIT',
  region: 'VIC',
  country: 'Australia'
}
```

```
var express = require('express');
var bodyParser = require('body-
parser');
var multer = require('multer');
var upload = multer();
var app = express();
app.use(bodyParser.json());

app.use(bodyParser.urlencoded({
extended: true }));

app.use(upload.array());
app.use(express.static('public'));

app.post('/', function(req, res){
   console.log(req.body);
   res.send("recieved your request!");
});
app.listen(3000);
```

Querying a Database

```
var express = require('express');
var app = express();
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'lecture8’
})
connection.connect()
connection.query('SELECT * from institutions',
    (err, rows, fields) => {
  if (err) throw err
  for (var i=0 ; i< rows.length; i++)
    console.log('The solution is: ', rows[i]);
})
connection.end()
```

```
D:\USB\UNSW\S1-2023\Lectures\Lecture 9 Materials\DBExpress>nodemon index.js
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
The solution is:  RowDataPacket {
  id: 1000,
  name: 'Stanford University',
  region: 'California',
  country: 'United States of America',
  created_at: 2023-04-30T03:04:35.000Z,
  updated_at: 2023-04-30T03:04:35.000Z
}
The solution is:  RowDataPacket {
  id: 2000,
  name: 'Harvard University',
  region: 'Massachusetts',
  country: 'United States of America',
  created_at: 2023-04-30T03:04:35.000Z,
  updated_at: 2023-04-30T03:04:35.000Z
}
The solution is:  RowDataPacket {
  id: 3000,
  name: 'University of Oxford',
  region: 'Oxford',
  country: 'United Kingdom',
  created_at: 2023-04-30T03:04:35.000Z,
  updated_at: 2023-04-30T03:04:35.000Z
}
[nodemon] clean exit - waiting for changes before restart
```

Export Modules

- The **module.exports** in Node.js is used to export any literal, function or object as a module.
- It is used to include JavaScript file into node.js applications.
- The **module** is similar to variable that is used to represent the current module and **exports** is an object that is exposed as a module.
- For example, we store the database config in config.js, then use it in index.js

```
const config = {
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'lecture8'
  };

  module.exports = config;
```

```
const config = require('./config');

const connection = mysql.createConnection(config);
```

APIs for CRUD Operations

```
app.get('/', function(req, res){
var result = [];
connection.query('SELECT * from
   institutions', (err, rows, fields) => {
  if (err) throw err
  for (var i=0 ; i< rows.length; i++){
    result[i] = rows[i];
  }
  res.send(result);
})
```

GET http://localhost:3000

Params • Authorization Headers (8) Body • Pre-request Scrip

Query Params

	Key	Value
☐	description	Beef, To
☐	picture	uploads
	Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON


```
1  [
2      {
3          "id": 1000,
4          "name": "Stanford University",
5          "region": "California",
6          "country": "United States of America",
7          "created_at": "2023-04-30T03:04:35.000Z",
8          "updated_at": "2023-04-30T03:04:35.000Z"
9      },
10     {
11         "id": 2000,
```

Show

```
app.get('/:id', function(req, res){
    var result = [];
    connection.query(`SELECT * from institutions
where id="${req.params.id}"`,
      (err, rows, fields) => {
        if (err) throw err
        for (var i=0 ; i< rows.length; i++){
        result[i] = rows[i];
        }
        res.send(result);
    })
        });
```

GET http://localhost:3000/2000

Params • Authorization Headers (8) Body • Pre-request Scr

Query Params

	Key	Value
☐	descriptionid	Beef,
☐	picture	uploa
	Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 2000,
        "name": "Harvard University",
        "region": "Massachusetts",
        "country": "United States of America",
        "created_at": "2023-04-30T03:04:35.000Z",
        "updated_at": "2023-04-30T03:04:35.000Z"
    }
]
```

Post Data

```
app.post('/', function(req, res){
        const id = req.body.id;
        const name = req.body.name;
        const region = req.body.region;
        const country = req.body.country;
        const myQuery = `INSERT INTO institutions (id, name, region,
    country)
        VALUES("${id}","${name}","${region}","${country}")`;
        console.log(myQuery);
        connection.query(myQuery, (err, rows, fields) => {
          if (err) throw err;
          else{
                res.statusCode = 200;
                res.end( 'Success');
          }
        })
```

POST http://localhost:3000/

Params • Authorization Headers (8) Body • Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

	Key	Value
☑	id	500
☑	name	RMIT
☑	region	VIC
☑	country	Australia
	Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

1 Success

lecture8.institutions: 4 rows total (approximately)

id	name	region	country	created_at	updated_at
500	RMIT	VIC	Australia	(NULL)	(NULL)
1,000	Stanford University	California	United States of America	2023-04-30 13:04:35	2023-04-30 13:04:35
2,000	Harvard University	Massachusetts	United States of America	2023-04-30 13:04:35	2023-04-30 13:04:35
3,000	University of Oxford	Oxford	United Kingdom	2023-04-30 13:04:35	2023-04-30 13:04:35

Delete

```
app.delete('/:id', function(req, res){
        connection.query(`DELETE from institutions where id="${req.params.id}"`, (err,
rows, fields) => {
            if (err) throw err
            else res.send('Success');
        })
            });
```

Host: 127.0.0.1 | Database: lecture8 | Table: institutions | Data | enrollment.sql | online_attendance.sql

lecture8.institutions: 2 rows total (approximately)

id	name	region	country	created_at	updated_at
1,000	Stanford University	California	United States of America	2023-04-30 13:04:35	2023-04-30 13:04:35
3,000	University of Oxford	Oxford	United Kingdom	2023-04-30 13:04:35	2023-04-30 13:04:35

DELETE http://localhost:3000/2000

Params • Authorization Headers (8) Body • P

Query Params

	Key
☐	descriptionid
☐	picture
	Key

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize HTML

1 Success

Error Handling and Status Code

```
app.post('/', function(req, res){
        if (!req.body.id || !req.body.name || !req.body.region ||!req.body.country){
            res.statusCode = 400;
                res.end( 'Invalid form data');
            }
        else {
                const id = req.body.id;
                const name = req.body.name;
                const region = req.body.region;
                const country = req.body.country;
                const myQuery = `INSERT INTO institutions (id, name, region, country)
                            VALUES("${id}","${name}","${region}","${country}")`;
                console.log(myQuery);
                connection.query(myQuery, (err, rows, fields) => {
                if (err) {
                        res.statusCode = 500;
                        res.end( 'Error in SQL query');
                    }
                else{
                        res.statusCode = 200;
                        res.end( 'Success');
                }
                })
            }
```

400 (Bad Request)

500 (Server Internal Error)

200 (ok)

SignIn

```
const crypto = require('crypto');
  function hashPassword(password) {
    const hash = crypto.createHash('sha256').update(password).digest('hex');
    return hash;
  }

app.post('/signin', async(req, res) => {
          const { email, password } = req.body;
          console.log(req.body);
          hashedPassword = hashPassword(password);
          await console.log(hashedPassword);
          // Check if username and password match
          const query = `SELECT * FROM users WHERE email = '${email}' AND password =
'${hashedPassword}'`;
          connection.query(query, (err, results) => {
            if (err) throw err;
            if (results.length > 0) {
              res.status(200).send('Sign in successful');
            } else {
              res.status(401).send('Username or password incorrect');
            }
          });
        });
```

Signup

```
app.post('/signup', (req, res) => {
          const { name, email, password } = req.body;
          hashedPassword = hashPassword(password);
          // Check if username already exists
          const checkQuery = `SELECT * FROM users WHERE email = '${email}'`;
          connection.query(checkQuery, (err, results) => {
            if (err) throw err;
            if (results.length > 0) {
              res.status(409).send('Username already exists');
            } else {
              // Insert new user into database
              const insertQuery = `INSERT INTO users (name, email, password) VALUES
('${name}','${email}', '${hashedPassword}')`;
              connection.query(insertQuery, (err, result) => {
                if (err) throw err;
                res.status(201).send('User created successfully');
              });
            }
          });
        });
```

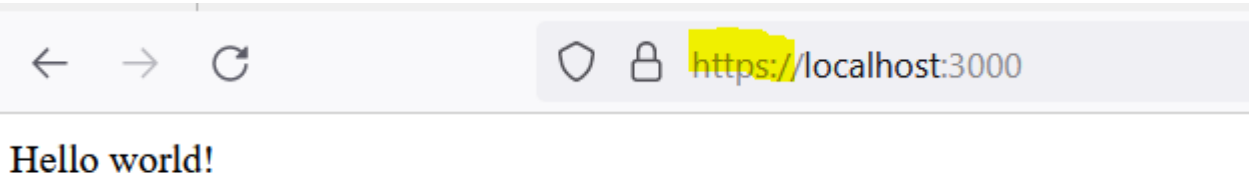
HTTPS

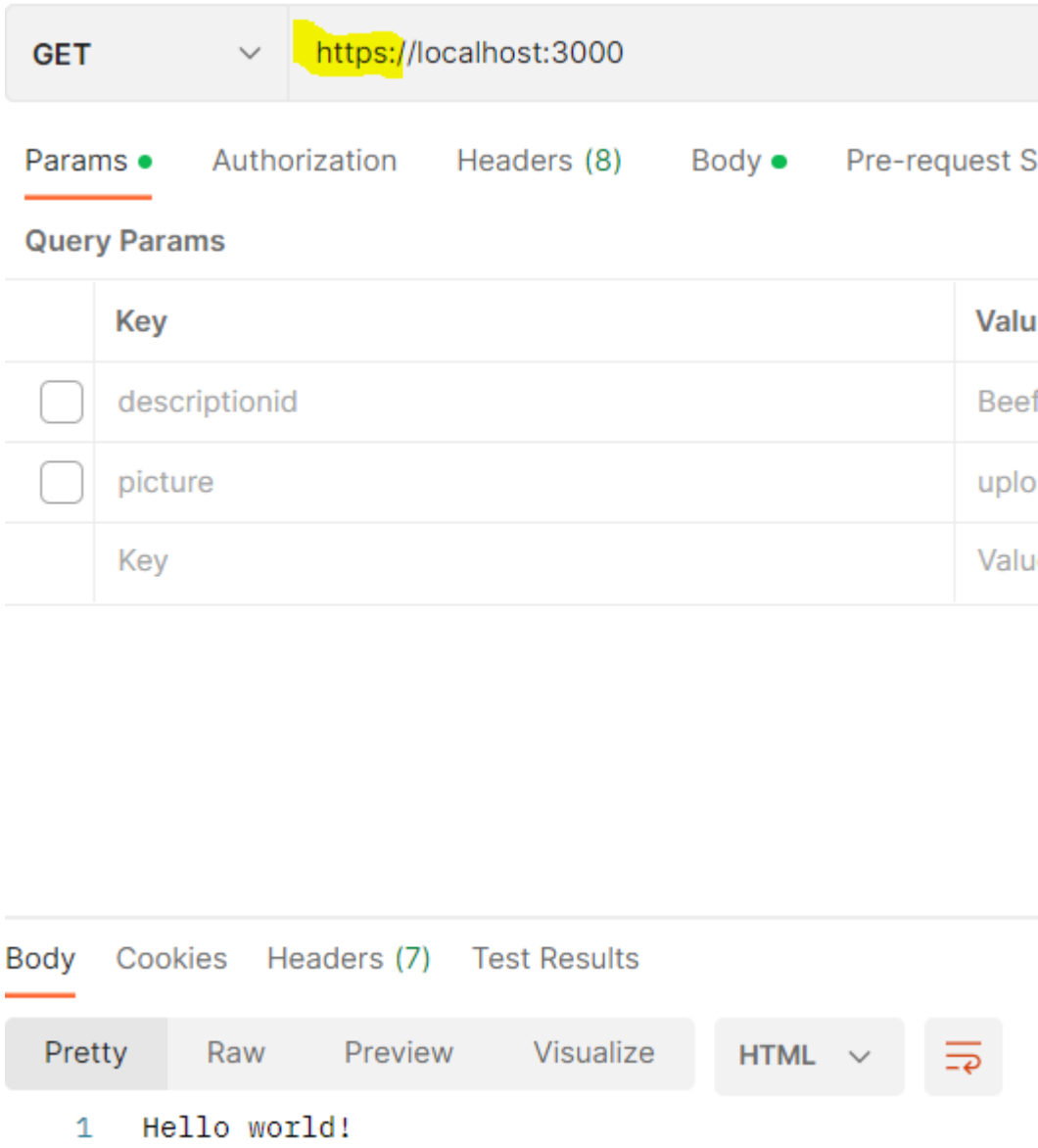
- HTTPS is the secure version of HTTP (Hypertext Transfer Protocol).
- It uses SSL/TLS (Secure Sockets Layer/Transport Layer Security) to encrypt data sent between a client and a server, making it more secure.
- SSL/TLS certificates are digital certificates that authenticate the identity of a website and encrypt data sent between a client and a server.
- They are issued by Certificate Authorities (CAs) and come in different types and levels of validation, depending on the needs of the website.

HTTPS Module

```
const fs = require('fs');
const key = fs.readFileSync('./CA/localhost/localhost.decrypted.key');
const cert = fs.readFileSync('./CA/localhost/localhost.crt');

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

app.get('/', (req, res, next) => {
  res.status(200).send('Hello world!');
});

const https = require('https');
const server = https.createServer({ key, cert }, app);

const port = 3000;
server.listen(port, () => {
  console.log(`Server is listening on https://localhost:${port}`);
});
```

HTTPS Module (Cont.)





Create SSL Certificate for Localhost

Follow this link to create SSL certificate for your server and make your localhost secure:

https://www.section.io/engineering-education/how-to-get-ssl-https-for-localhost/

Final Note

- **Please attend the labs this week and discuss Project 2 requirements with your lab demonstrators**