**Note: You should be able to copy and paste between the code here and your code in Visual Studio Code.**

Note: Remember write your code on the C: drive not on the OneDrive. Use Github to store the code. If you have difficulty with github please ask for help. It is important that you master its use.

**https://github.com/ulestrange/firstExpressApp**

# Section 1:

## Task 1

Create a folder for your project call this folder **firstapp.**

Open this folder in visual studio code and open a terminal.

Type **git init** to set up git for this application.

What happens?

Add a file called .gitignore with the following:

# OS X

.DS\_Store\*

Icon?

.\_\*

# Windows

Thumbs.db

ehthumbs.db

Desktop.ini

# Linux

.directory

\*~

# npm

node\_modules

package-lock.json

\*.log

\*.gz

# Coveralls

coverage

# Benchmarking

benchmarks/graphs

Type**npm init.**

There will be a dialog: you can accept all the defaults.

It will ask if this is O.K: just say Yes.

You have just created a **package.json** file, open it and look at the contents. We won't be editing this file directly.

Type**npm i express@4.18.2**

Notice your package.json file has changed. It should now include a dependency to express.

{

  "name": "firstexpressapp",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "",

  "license": "ISC",

  "dependencies": {

    "express": "^4.18.2"

  }

}

Create a new file called **index.js**

const express = require('express')

const app = express()

const port = 3000

app.get('/', (req, res) => res.send('Hello World from Una!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))

Run this app (Use **node index** in the terminal) – what does it do?

Does it work?

This is your first application built using Express. ☺

Note: When you type node index in the terminal you are starting a node web server, this is listening at localhost:3000 it will continue running until you press ctrl-c.

**-🡪 Start of Rant**

Some students write code and forget to run it and test that it does what they expect.

You haven’t finished this worksheet unless you start up a web browser and go to localhost:3000.

Ideally you would then “play” a bit. Change the message that gets sent, change the port number etc.

In each case we need to save our changes, use ctrl-c to stop the web server and node index to restart it.

If you cannot get this to work please ask for help.

**End of Rant 🡨----**

## Task 2

Add a second “route” to your program and test that it works by going to localhost:3000/bananas

app.get('/bananas', (req, res) =>

  res.send('hello world, this is bananas'));

Add some silly extra routes of your own.

Example:

app.get('/fruit/apples', (req, res) =>

  res.send('hello world, this is apples '));

# Section 2

================================================================================

The next set of tasks are to demonstrate how we could implement a simple API.

The API will have the following endpoints:

GET /contacts

GET /contacts/{id}

DELELTE /contacts/{id}

POST /contacts

We will assume for the moment that any contacts to be stored are simple json objects with three properties: name, phonenumber and email.

**Task 1: POST**

First create an array which we will use to store the contacts.

let contacts = [];

*For the moment we will just keep a list of contacts in an array. We will change this later to get data from a persistent store. Because we are just using an array every time we stop and restart node we will lose all the data. This will make testing a bit harder, but we can work with it for now.*

Next add the POST route.

app.post('/contacts', (req, res) => {

    const contact = req.body;

    console.log(contact) // to check what was received

    contacts.push(contact);

    res.send ('contact has been added to the database');

    console.log(`contact name is ${contact.name} number of contacts is ${contacts.length}`);

});

To test this we need to send a post request with some data to localhost:3000/contacts.

We will use the VS Code extension Rest Client.

Install the VS Code Extension **Rest Client.**

Then create a new folder called tests and a file called contacts.http within this folder.

Add the following code to the contact.http file.

###

### POST Request for a contact

## Expected result is ........

POST http://localhost:3000/contacts HTTP/1.1

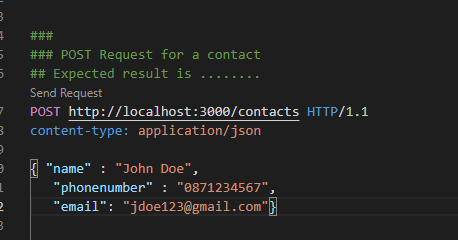
content-type: application/json

{ "name" : "John Doe",

   "phonenumber" : "0871234567",

   "email": "jdoe123@gmail.com"}

To run this you hit the grey Send Request.



If everything was working you should receive back a HTTP response with status 200 OK and the message “contact has been added to the database” but if you check your terminal you will see that an error has occurred.

This error is because the app does not yet know how to parse the JSON it received in the body of the request. We need to add this functionality explicitly to the application. This is done by adding **middleware** to decode the request body when json is received. It needs to be added before the route handlers.

app.use(express.json());

Check what happens when this line is added to your code.

What happens if you move it to below the POST request handler?

**Task 4: GET all the contacts.**

Use Rest Client to make requests to add a few contacts to the array.

Then add the following code to get a list of all contacts.

app.get('/contacts', (req, res) => {

    res.send(contacts);

})

This can be tested easily in the browser using the url **localhost:3000/contacts** or you can use Rest Client

###

### GET Request for all contact

## Expected result is ........

GET http://localhost:3000/contacts HTTP/1.1

**Task 5: GET a specific contact**

Now we want to retrieve a specific contact. For this demo we will just use the index into the array as an ID for the contact. This will keep things simple.

We need to use a **route parameter** in Express

  app.get('/contacts/:id', (req,res) => {

     let id = req.params.id;

      res.json(contacts[id]);

  })

This can be tested using the URL **localhost:3000/contacts/0** to retrieve the first contact etc.

Add code to the contacts.http file to test this.

What happens if we look for a contact that doesn’t exist?

**Task 5: DELETE a specific contact.**

  app.delete('/contact/:id',(req, res) =>

 {

    let id = req.params.id;

    console.log(`removing contact ${contact[id].name}`)

    contact.splice(req.params.id, 1);

    res.send(contact);

  })

This deletes the contact from the list, writes a message to the console and then sends the new list as a response.

Add code to the contact.http file to test this.

**Conclusion**

The above tasks give a very quick demonstration of how we can use Express to write an API.

There are a few things we need to change to make this a more realistic example.

* At the moment all the code is in one file so we need to consider how we can split it up for maintenance purposes.
* We need to work with contact identifiers rather than just an index into an array.
* We also need to change our “handler” functions so that they use a database rather than a simple array.

Working code is available here, you need to look at the section 2 branch.

[**https://github.com/ulestrange/firstExpressApp.git**](https://github.com/ulestrange/firstExpressApp.git)