Accessing the database from Node

Welcome to this lab activity

In this lab activity, you will explore how to access databases from your Node web application. You will write Node.js code to connect to your database and retrieve data, displaying the data on a web page.

Task 1: Make a copy of the previous Node.js application

Let's start with your previous Node.js lab code.

1. Make a copy of the folder `10_post_form_data` from your previous lab and name it `15_access_database`.
2. Open the new folder in Visual Studio Code.
3. Run the code and confirm that you can see the home page, about page, search page and register page and that the forms work (refer to lab 10 POST – collecting form data, if you need a reminder of how to do this).

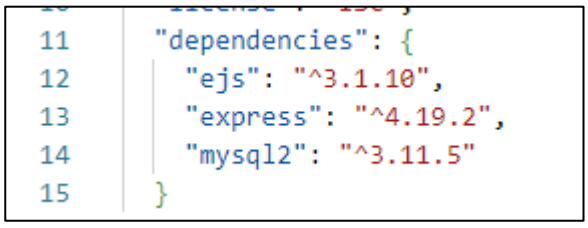
Task 2: Add the MySQL node module

In order to access your MySQL database from your Node.js application, you will need to install another module called mysql2.

4. Install mysql2 in your application by running the following command in the Terminal pane of Visual Studio Code:

```
npm install mysql2
```

If you check your `package.json` file, you will see that `mysql2` has been set up as a dependency, along with Express and EJS:



Task 3: Connect to your database

The next step is to add some code to your `index.js` file to connect it to your MySQL database.

5. Add the following code to `index.js` before the `const port = 8081` line:

```
const mysql = require ("mysql2");
```

This imports the `mysql2` module.

6. Add the following code to `index.js` before your code that loads the route handlers:

```
// Define the database connection
const db = mysql.createConnection ({
    host: 'localhost',
    user: 'root',
    password: 'YOURROOTPASSWORD',
    database: 'myBookshop'
})

// Connect to the database
db.connect((err) => {
    if (err) {
        throw err
    }
    console.log('Connected to database')
})
global.db = db
```

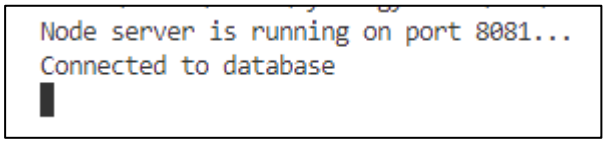
Replace `YOURROOTPASSWORD` with your actual root password (that you entered when you set up MySQL).

Looking deeper

Using you root user and password like this has some significant security implications. Can you think of what they are? We will explore more secure ways to access our database later in this module.

This code defines a database connection, containing all the details that you need to connect your application to the database. It then uses this database connection to actually connect to the database. If the connection fails for some reason, you will see an error. Can you think of some reasons why the connection might fail?

7. Run your application to check that the database connection is successful. You should see a confirmation in the terminal:



If it fails, check that all the details in the database connection are correct and check that your database is running.

Task 4: Query data from your database

Now it's time to retrieve some data from the books table in the database.

8. In main.js, add the following route handler:

```
router.get("/list", function(req, res, next) {
    // Query database to get all the books
    let sqlquery = "SELECT * FROM books";

    // Execute sql query
    db.query(sqlquery, (err, result) => {
        if (err) {
            next(err);
        } else {
            res.send(result);
        }
    });
});
```

Task 5: Test your app

Let's run the app to see if we can see a list of books!

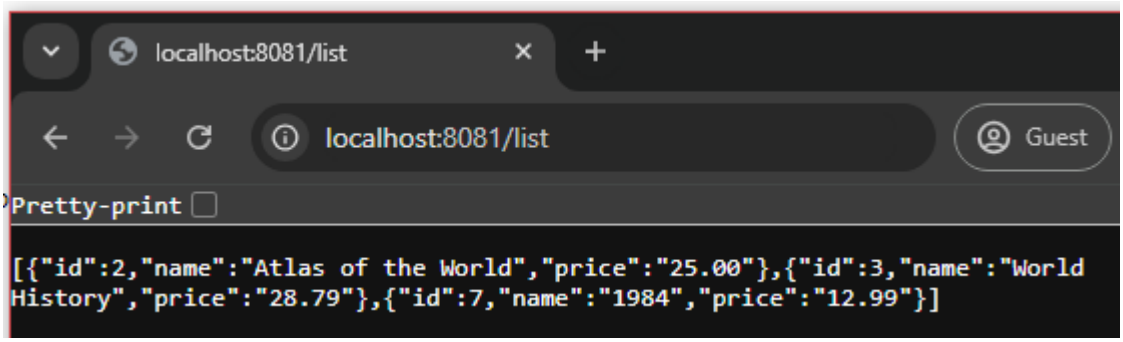
9. Run the `index.js` file with the following Terminal command:

```
node index.js
```

The above command will start a web server running on port 8081.

10. Browse to the list page at http://localhost:8081/list.

You should see your books appear:



Task 6: Explore further

When tackling these lab activities, it's always good to stretch yourself by doing some research and attempting some changes on your own.

Can you add another route `/bargainbooks`, that shows books just like the `/list` route, but only books that cost less than £20?

End of lab

Congratulations on completing this lab.

In the next lab activity, you will use the EJS templating engine to make your application dynamic.