Working on this project was exciting for me because I was able to demonstrate the knowledge acquired from the module practically and I was able to achieve the following:

- New MySQL query

```
db.query(
  "CREATE TABLE IF NOT EXISTS devices (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), type VARCHAR(255)
  (err, result) => {
    if (err) {
      console.log("An error occurred while creating the table", err);
      // throw err;
    }
    console.log("Table devices created");
  }
);
```

To ensure the database is set up before responding to requests at the start of the app (index.js) a table is created only if it doesn't exist to avoid crashing the server as shown in the snippet above.

- Optimising Network Request

```
db.query(sqlquery, (err, result) => {
  if (err) {
    console.log("An error occurred while selecting data", err);
    message = "An error occurred";
    res.redirect("/");
    clearMessage();
  }
  console.log("Data selected successfully", result);
  const id = req.query.id;
  let device = result.find((device) => device.id == id);
  //This will return the device with the id that was passed in the query
  console.log("Device found", device);
  res.render("device-status", { devices: result, device: device });
});
```

The code snippet above shows how the javascript find method was used to select a particular device from all devices using its unique id instead of querying the database again.

- Applied DRY principle learnt in the SDD module using ejs partial for the navbar.

```
<!-- This will include the ejs nav partial -->
<%- include('partials/nav') %>
```

- I encountered an issue with names for querying the database according to the requirement:

**Control device server-side functionality:**

- A POST action which collects the form data and passes it to the server
- A POST endpoint which uses the device name to update the device with the new settings

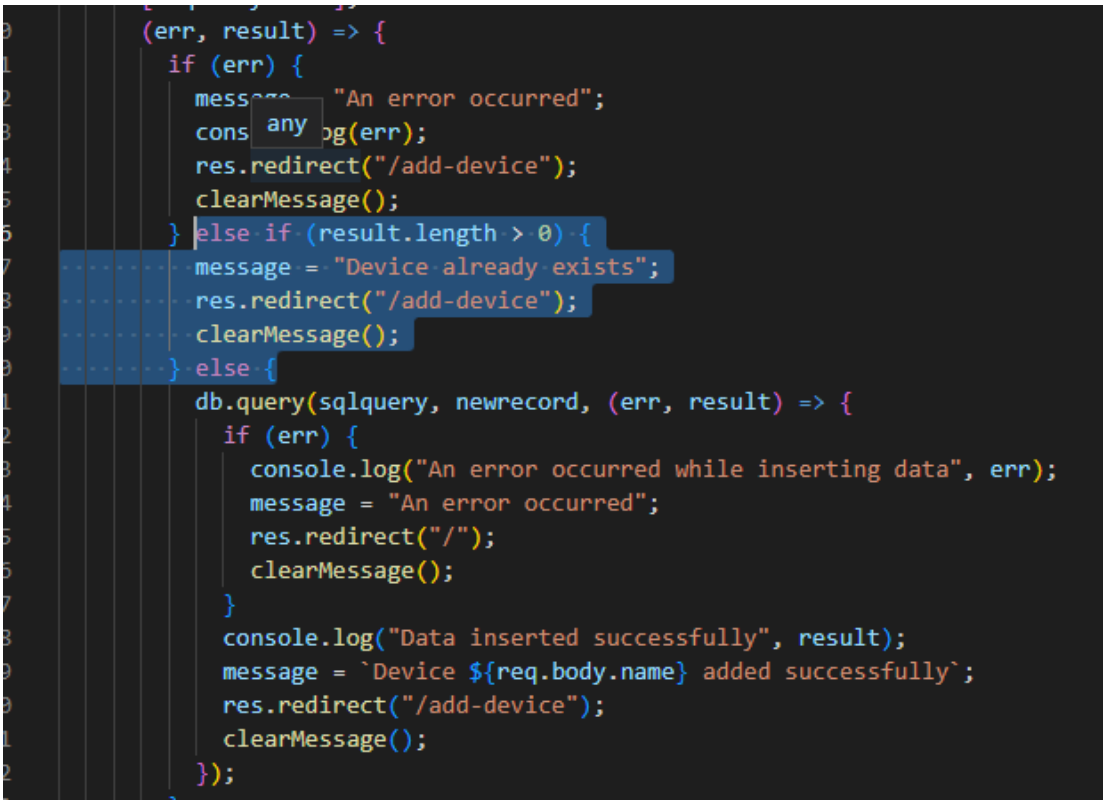
**Delete device page**

- A device selector input which allows the user to select a device from a list of created devices
- A delete button
- Extension: Improve the design by asking for the user's confirmation with a confirm modal before the delete operation.

**Delete device - server-side functionality**

- A POST action which collects the form data and passes it to the server
- A POST endpoint which uses the device name to delete the device and related data from the database

The issue is those devices with the same name when modified update together but I overcame it by restricting duplicate names as highlighted in the snippet below:



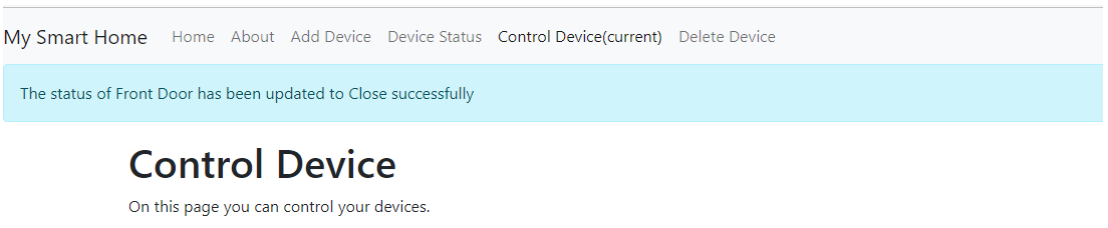
All extensions were implemented as follows:

- Success feedback

```
<!-- This will include the ejs nav partial -->
<%- include('partials/nav') %>
    <!-- This will display success message whenever there's a response from the server with ejs syntax. -->
    <% if (message) { %>
        <div class="alert alert-info" role="alert">
            <%= message %>
        </div>
        <% } %>
```

```
let message; // global variable to store the message

//This function will set message to  undefined after 3 seconds when called
//This will ensure that message from previous page is not displayed on
//newly navigated page
function clearMessage() {
  setTimeout(() => {
    message = undefined;
  }, 1000);
}
```

```
app.post("/control-device", function (req, res) {
  //This post request will be used to update the status of the device using the name of the device
  let sqlquery = "UPDATE devices SET status = ? WHERE name = ?";
  let newrecord = [req.body.status, req.body.name];
  console.log("New record", newrecord);
  db.query(sqlquery, newrecord, (err, result) => {
    if (err) {
      console.log("An error occurred while updating data", err);
      message = "An error occurred";
      res.redirect("/");
      clearMessage();
    }
    console.log("Data updated successfully", result);
    message = `The status of ${req.body.name} has been updated to ${req.body.status} successfully`;
    res.redirect("/control-device"); //For hot reloading
    clearMessage();
  });
```




As shown above, a message variable was passed to each page with a post request to display a message after the action.

- Hot reloading was implemented by redirecting after a successful action, and the redirected endpoint fetch the latest changes as shown below:

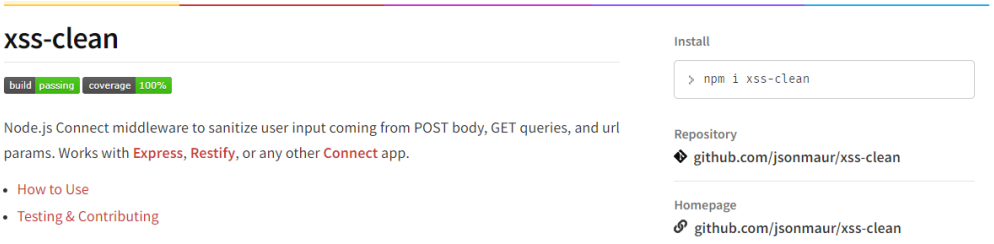
```
db.query(sqlquery, newrecord, (err, result) => {
  if (err) {
    console.log("An error occurred while updating data", err);
    message = "An error occurred";
    res.redirect("/");
    clearMessage();
  }
  console.log("Data updated successfully", result);
  message = `The status of ${req.body.name} has been updated to ${req.body.status} successfully`;
  res.redirect("/control-device"); //For hot reloading
  clearMessage();
});
```

```
app.get("/control-device", function (req, res) {
  let sqlquery = "SELECT * FROM devices";
  db.query(sqlquery, (err, result) => {
    if (err) {
      console.log("An error occurred while selecting data", err);
      message = "An error occurred";
      res.redirect("/");
      clearMessage();
    }
    console.log("Data selected successfully", result);
    const id = req.query.id;
    let device = result.find((device) => device.id == id);
    //This will return the device with the id that was passed in the query
    console.log("Device found", device);
    res.render("control-device", {
      devices: result,
      device: device,
      message,
    });
  });
});
```

- Non-applicable fields were disabled using Other as device type with the conditional statement in the snippet below:

```
function chooseDevice(value, selectedValue) {
  const onoff = ["Light", "Fan", "Cooker", "Fridge", "Washing Machine"];
  const openclose = ["Door", "Window", "Garage", "Dishwasher"];
  const updown = ["Thermostat", "Television", "Air Conditioner", "Oven"];

  if (onoff.includes(value)) {···
  } else if (updown.includes(value)) {···
  } else if (openclose.includes(value)) {
    document.getElementById("statusType").innerHTML = `
        <select class="form-control" name="status" id="deviceStatus">
            <option value="Open">Open</option>
            <option value="Close">Close</option>
        </select>
        `;
    //pick the selected value
    document.getElementById("deviceStatus").value = selectedValue;
  } else { //For Non-appliicable fields
    document.getElementById("statusType").innerHTML = `
        <input type="text" class="form-control" name="status" placeholder="Not Applicable" value="${selectedValue}" >
        `;

  }
}
```

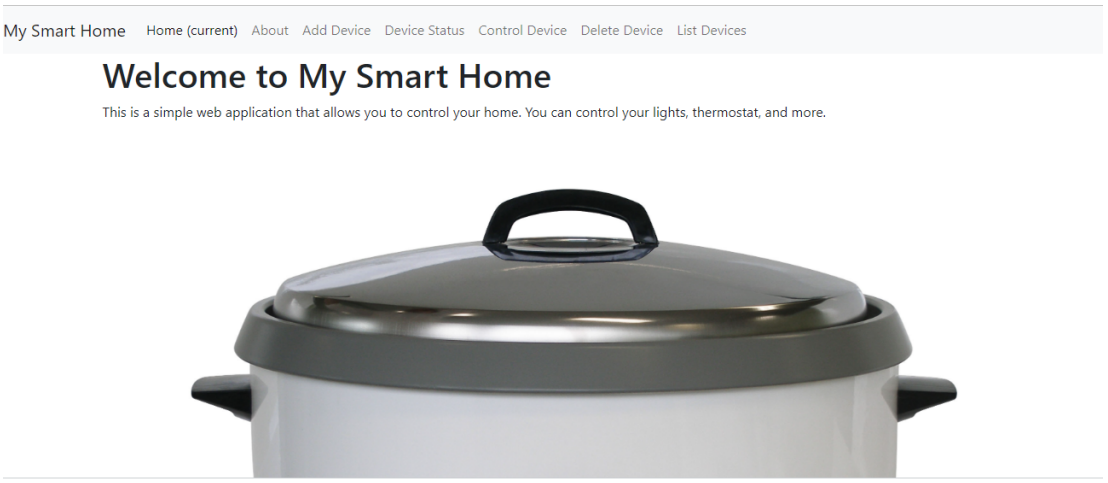
- With research, a library for sanitising data against XSS attacks was discovered

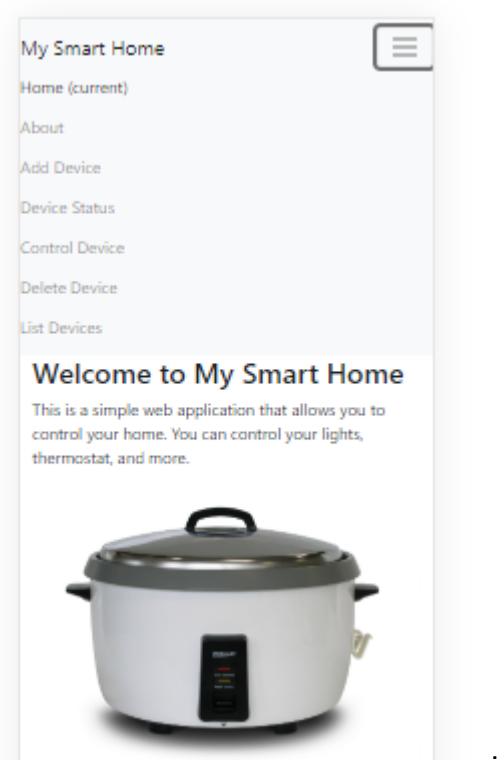



And it was added before the route as follows

```
var xss = require('xss-clean')

const port = 8089;

console.log("process.env.host", process.env.host);

const db = mysql.createConnection({
  host: process.env.host || "localhost",
  user: process.env.user || "root",
  password: process.env.password || "root",
  database: process.env.database || "mySmartHome",
  port: process.env.port || "3306",
});
// connect to database
db.connect((err) => {···
});
global.db = db;
app.use(bodyParser.urlencoded({ extended: true }));
//serve the static files from the public folder
app.use(express.static("public"));

app.use(xss()); //prevent Cross-Site Scripting attacks
```

- Bootstrap was used for frontend styling






**Database design**

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | int | NO | PRI | NULL | auto_increment |
| name | varchar(255) | YES | | NULL | |
| type | varchar(255) | YES | | NULL | |
| status | varchar(255) | YES | | NULL | |

Id is the primary key and it automatically increases to avoid duplicates. Name, type and status are variable characters and they are allowed to be null by default.

| id | name | type | status |
|---|---|---|---|
| 12 | Cooker | Cooker | Off |
| 13 | Front Door | Door | Close |
| 14 | Kitchen Thermo | Thermostat | 29 |

For more advanced cases, a device-type table can be created to convey the possible status of each device.