Fundamentals of Software Development for Electronics

Prepared By: Dr. Rony Ibrahim

Course outline

- Session 1: Importance of Software Engineering in Electronics
- Session 2: Basic Concepts and Terminologies
- Session 3: Software Development Life Cycle (SDLC)
- Session 4: Integrated Development Environments (IDEs) and Basic Git Workflow
- Session 5 and 6: JavaScript & Introduction to TypeScript
- Session 7 and 8: Building Web Applications with ReactJS
- Session 9: CSS Basics and Responsive Design Principles
- Session 10: Basics of Flutter / React Native / Ionic
- Session 11: Project Presentation & Final Review

JavaScript & Introduction to TypeScript

Objectives

- Deepen understanding of advanced JavaScript concepts.
- Learn how TypeScript enhances JavaScript through static typing.
- Apply JavaScript and TypeScript in the context of Raspberry Pi projects.
- Work with libraries that enable JavaScript interaction with hardware.

ES6+ Features for Electronics Applications

- Arrow Functions: Simplified syntax for writing functions.
- Template Literals: Use for creating dynamic strings when communicating with devices.
- Destructuring: Use with sensor data to extract values easily.
- Example:

```
const { temp, humidity } = getSensorData();

console.log(`Temperature: ${temp}, Humidity: ${humidity}`);
```

Promises and Asynchronous Code

Importance of Asynchronous Programming:

Dealing with sensor data and real-time responses.

Example: Reading Sensor Data from Raspberry Pi:

```
function readSensorData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Sensor data read!");
    }, 1000);
  });
}

readSensorData().then((response) => console.log(response));
```

Demo - Building a JavaScript Control Panel

Set up Raspberry Pi for GPIO Control: Needs

- Raspberry Pi (with Raspbian OS installed)
- 2 LEDs (e.g., red and green)
- 2 resistors (330Ω or 220Ω)
- Breadboard
- Jumper wires

GPIO Pin Connection:

- Red LED: Connect the positive terminal to GPIO pin 17.
- Green LED: Connect the positive terminal to GPIO pin 27.
- Both LEDs' negative terminals should be connected to ground (GND) through the resistors.

Demo - Building a JavaScript Control Panel

Enable SSH and Set Up Wi-Fi:

- Ensure the Raspberry Pi is connected to the same Wi-Fi network as your Windows laptop.
- Enable SSH on the Raspberry Pi so you can connect to it from your laptop.

Remote Access from Windows:

- Install a terminal application like PuTTY on your Windows laptop to SSH into the Raspberry Pi. Or you can use SSH connection from VSCODE.
- SSH into the Raspberry Pi by using its IP address:

```
ssh pi@<raspberry_pi_ip_address>
```

Install Node.js and Required Libraries

On the Raspberry Pi, run the following commands to install Node.js

```
curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash -
sudo apt install -y nodejs
```

- Use the onoff library to control the GPIO pins via JavaScript

```
npm i onoff
```

- Install Express.js to create the web interface

```
npm install express
```

Create the Project Structure

Create a new project folder for your control panel on the Raspberry Pi:

```
mkdir led-control-panel
cd led-control-panel
```

- Initialize the Project - Initialize a new Node.js project

```
npm init -y
```

- Set up the Server - Create a new file called server.js

```
touch server.js
nano server.js
```

Write the JavaScript Code for Controlling the LEDs

In the server.js file, write the following code to control the LEDs and serve the control panel web page.

```
// Import required modules
const express = require('express');
const Gpio = require('onoff').Gpio;
const path = require('path');

// Initialize GPIO pins
const redLed = new Gpio(17, 'out');
const greenLed = new Gpio(27, 'out');

// Initialize Express app
const app = express();
app.use(express.static('public'));  // Serve static files
```

Write the JavaScript Code for Controlling the LEDs

```
// Route to turn the red LED on or off
app.get('/led/red/:state', (req, res) => {
try{
  const state = req.params.state === 'on' ? 1 : 0;
  redLed.writeSync(state);
  res.status(200).send(`Red LED is ${req.params.state}`);
}catch(err){
      res.status(500).send(`Something went wrong`, err);
}
});

// Route to turn the green LED on or off
app.get('/led/green/:state', (req, res) => {
  const state = req.params.state === 'on' ? 1 : 0;
  greenLed.writeSync(state);
  res.send(`Green LED is ${req.params.state}`);
});

// Serve the control panel web page
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
```

Write the JavaScript Code for Controlling the LEDs

```
// Start the server
app.listen(3000, () => {
  console.log('Server running on port 3000');
});

// Cleanup on exit
process.on('SIGINT', () => {
  redLed.unexport();
  greenLed.unexport();
  process.exit();
});
```

Create the Web Interface (HTML)

Create the public Directory - Inside your led-control-panel folder, create a new folder called public to hold the web page files:

```
mkdir public
```

Inside the public folder, create a file called index.html:

```
nano public/index.html
```

Write the HTML for the Control Panel: Add the following code to index.html to create a simple control panel:

Create the Web Interface (HTML)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>LED Control Panel</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin-top: 50px;
    }
    button {
      padding: 10px 20px;
      margin: 10px;
      font-size: 16px;
    }
  </style>
</head>
```

Create the Web Interface (HTML)

```
<body>

  <h1>LED Control Panel</h1>

  <div>
    <h2>Red LED</h2>
    <button onclick="controlRedLed('on')">Turn On</button>
    <button onclick="controlRedLed('off')">Turn Off</button>
  </div>

  <div>
    <h2>Green LED</h2>
    <button onclick="controlGreenLed('on')">Turn On</button>
    <button onclick="controlGreenLed('off')">Turn Off</button>
  </div>

  <script>
    function controlRedLed(state) {
      fetch(`/led/red/${state}`).then(response => response.text()).then(data => {
        alert(data);
      });
    }

    function controlGreenLed(state) {
      fetch(`/led/green/${state}`).then(response => response.text()).then(data => {
        alert(data);
      });
    }
  </script>

</body>
</html>
```

Run the Application

On your Raspberry Pi, run the following command to start the server:

```
node server.js
PS: you can use nodemon || pm2
```

On your Windows laptop, open a browser and navigate to:

```
http://<raspberry_pi_ip_address>:3000
```

Test and Verify

Control the LEDs:

- Click the "Turn On" and "Turn Off" buttons for both LEDs. The respective LEDs connected to the Raspberry Pi should turn on and off.
- The browser will display alerts confirming the LED state.

Cleanup:

- When done, press ***Ctrl + C*** in the terminal to stop the server. The GPIO pins will be cleaned up automatically to avoid damage to the hardware.

Exercise Enhancement

- Add a feature to read sensor data (e.g., temperature) from the Raspberry Pi and display it on the web interface.

- Implement a real-time control panel using WebSockets to allow for instant updates on the LED state without refreshing the page.