Coding Standards

Chanseop Carter Shin edited this page on Jun 24 · 4 revisions

Coding Standards/Guidelines

Overview

Our project focuses on developing a software solution for the Raspberry Pi platform, aimed at automating the retrieval of essential information from IoT devices connected via USB or UART. The software gathers details such as firmware version, chip model, and voltage usage, compiling this data into an XML format for storage or transmission for further analysis. The goal is to deliver a robust, scalable, and maintainable system that efficiently manages data retrieval and storage, ensuring high performance and reliability.

Development Tools

For development, we will use:

- **Python**: Leveraging its extensive libraries and simplicity for IoT applications.
- **React**: For frontend development to create a seamless and intuitive user interface.
- **Node.js**: For backend development to manage data storage, user authentication, and other essential features.

Components Library

- **Python**: Use `pyusb` for USB communication and `xml.etree.ElementTree` for XML data formatting and parsing.
- **React**: Use Material UI Core to create all of the components.
- **Node.js**: Use Express.js for server-side logic and API endpoints.

Database and Backend

For data storage, we will utilize:

- **Local storage on the Raspberry Pi:** Leveraging the file system for XML data storage.
- **Database**: We have chosen to use MongoDB Atlas as our database which is a noSQL database. This is necessary due to the information we store from the raspberry pi.

File and Folder Naming Conventions

File and folder names should be in snake_case wherever possible to ensure compatibility across different filesystems and avoid issues with case sensitivity, especially when using `git` .

For example, on Windows systems `this_is_a_file.py` and `thisIsAFile.py` are considered equivalent, but those would be considered distinct names on non-Windows systems (macOS and Linux variants).

Python Code

1. **General Naming Conventions:**
   - **Variables and Functions:** Use snake_case for variable and function names.

   > Correct

   ```python
   def get_device_info():
       device_name = "Raspberry Pi"
   ```

   > Incorrect

   ```python
   def getDeviceInfo():
       deviceName = "Raspberry Pi"
   ```

   - **Classes:** Use CamelCase for class names.

   > Correct

   ```python
   class DeviceManager:
       pass
   ```

   > Incorrect

   ```python
   class device_manager:
       pass
   ```

2. **Error Handling:**
   - Use exceptions to handle errors and special conditions. Do not use return codes or other error indicators.

> Correct

```
def read_data_from_device(device):
    if not device.is_connected():
        raise ConnectionError("Device not connected")
    return device.read_data()
```

> Incorrect

```
def read_data_from_device(device):
    if not device is_connected():
        return None
    return device.read_data()
```

3. **Constants:**
   - Constants should be defined at the top of the file and use all uppercase letters with underscores separating words.

> Correct

```
MAX_VOLTAGE = 5.0
```

> Incorrect

```
maxVoltage = 5.0
```

4. **Function and Variable Names:**
   - Names should be descriptive and unambiguous. Avoid single-letter names except for loop counters.

React Code

1. **Component Naming Conventions:**
   - **Component Files and Names:** Use PascalCase for React components and their filenames.

> Correct

```
// Component file: MyComponent.jsx
function MyComponent() {
    return <div>My Component</div>;
}
```

> Incorrect

```
// Component file: myComponent.jsx
function myComponent() {
    return <div>My Component</div>;
}
```

2. **JSX Syntax:**

   - Use self-closing tags for elements without children.

> Correct

```
<input type="text" />
```

> Incorrect

```
<input type="text"></input>
```

3. **CSS and Styling:**

   - Use CSS modules or styled-components for styling React components. Class names should be in kebab-case.

> Correct

```
import styles from './MyComponent.module.css';

function MyComponent() {
    return <div className={styles.myComponent}>My Component</div>;
}
```

> Incorrect

```
import './MyComponent.css';

function MyComponent() {
    return <div className="MyComponent">My Component</div>;
}
```

Node.js Code

1. **General Naming Conventions:**

   - **Variables and Functions:** Use camelCase for variable and function names.

> Correct

```
function getDeviceInfo() {
    const deviceName = "Raspberry Pi";
}
```

> Incorrect

```
function get_device_info() {
    const device_name = "Raspberry Pi";
}
```

- **Classes:** Use PascalCase for class names.

> Correct

```
class DeviceManager {
    // ...
}
```

> Incorrect

```
class device_manager {
    // ...
}
```

2. **Error Handling:**

- Use exceptions to handle errors and special conditions. Do not use return codes or other error indicators.

> Correct

```
function readDataFromDevice(device) {
    if (!device.isConnected()) {
        throw new Error("Device not connected");
    }
    return device.readData();
}
```

> Incorrect

```
function readDataFromDevice(device) {
    if (!device.isConnected()) {
        return null;
    }
```

```
        return device.readData();
    }
```

3. **Constants:**

   - Constants should be defined at the top of the file and use all uppercase letters with underscores separating words.

   > Correct

   ```
   const MAX_VOLTAGE = 5.0;
   ```

   > Incorrect

   ```
   const maxVoltage = 5.0;
   ```

Git Usage and Branching Model

Follow the conventional Gitflow workflow branching model which includes but is not restricted to the following branches:

- Main
- Develop
- Feature branches
- Hotfix
- Production

Ensure Commit Comments are Descriptive

Example: For a commit that refactors or improves the readability of code:

> Correct

"Refactor the conditional logic in handle_device_connection function for clarity"

> Incorrect

"Updated function"

Conclusion

By adhering to these coding standards, we aim to ensure consistency, readability, and maintainability across our Raspberry Pi IoT project. Consistent coding practices also facilitate easier collaboration among team members and contribute to the overall quality of our application.

+ Add a custom footer

Pages 11

Find a page...

- User Manual
- User Stories

+ Add a custom sidebar

Clone this wiki locally

```
https://github.com/COS301-SE-2024/IoT-DIRfram.wiki.git
```