IBM Quantum Proximity Gateway - Deployment Manual

This a deployment manual for the Quantum Proximity Gateway project, in this manual you can find instructions on Building, Deploying and Executing all the different components of the project.

Table of Contents

Summary

The goal of this project was to be able to walk up to any hot-desking computer and, simply by being near the computer, automatically get logged in and have your profile preferences loaded up onto the device. In the end, this project ended up combining a wide variety of functionalities together, from Bluetooth Low Energy signals for communication between an ESP32 and a Raspberry Pi, to USB HID keyboard simulation with a Raspberry Pi Pico, as well as using an LLM such as IBM Granite to seamlessly allow the user to let the computer know which preferences need to be changes (and saved), amongst other technologies.

This README file will cover what each of the 5 main parts of the project generally are for, as well as how to set them up (they will have links to the READMEs in their respective repos, but also collapsible copies of those READMEs).

Setup/Installation

Since all of the submodules in this project have different requirements, we have decided to write what are essentially separate READMEs for each of them (each of which can be accessed from this file itself).

Below is the command to clone this entire repository (and its submodules) together. However, note that some of the repositories (such as the `rpi-code` repo) need to be cloned and used on a different hardware device (i.e. the Raspberry Pi 5).

To clone all of the code for this project, run the following command:

```
git clone --recurse-submodules https://github.com/quantum-proximity-gatew
ay/main.git
```

Server

Purpose

The server is where we handle all of the requests that the other components of the project make, as well as store our data. In our case, this means handling how to register an ESP32 device with a user (and their account details), as well as storing a user’s accessibility preferences which can be retrieved. This also handles the majority of the encryption aspect of the project, ensuring that we use a quantum-secure encryption standard such as CRYSTALS-Kyber.

Setup

The server has already been deployed and is currently being hosted on IBM Cloud, but if you want to run it locally anyways, refer to server/README.md for more information.

Alternatively, you can open the following collapsible section to view the contents of this submodule’s README. We do, however, recommend looking at the linked README instead, as this one can quickly get quite cluttered.

README

Server README

Table of Contents

This server is already deployed, so the instructions below are just in case you want to run the server locally (or change the code yourself).

There are 2 ways of running this server locally:

1. Via **Docker** - the server will run locally, but you will not be able to change the code. This method is extremely easy, is unlikely to have failing dependency problems, and also runs much quicker since liboqs doesn't need to be re-compiled.
2. Via **Python** as a litestar application - the server will run locally and you will need to change the code.

Method 1 - Docker

Requirements

- Docker

Usage

Firstly, ensure docker is running and active. Then, run the following command:

```
docker run -p 8000:8000 raghav2005/qpg-server
```

If it is easier, `make docker` can also be run instead of that command (it runs the same thing).

Method 2 - Python

Requirements

- Pipenv

Installation

Navigate to the `backend/` directory.

Then, to install the dependencies, run:

```
pipenv install -r requirements.txt && pipenv install
```

Usage

You can either run the server directly, without spawning a new shell for pipenv, or you can activate the environment and run the server.

For the first option, run the following command:

```
pipenv run python -m litestar run --host 0.0.0.0 --port 8000
```

For the second, run the following commands in order:

```
pipenv shell
litestar run --host 0.0.0.0 --port 8000
```

> NOTE: You can check if the server is running by trying to access http://localhost:8000 in a browser. If you see {"status":"success"} on the screen, the server is running.

Misc.

- Instead of using the commands listed above individually, you can run `make docker`, `make install`, or `make run` from the root directory of this repository to run the server.

License

This project is licensed under the terms of the MIT license. Refer to LICENSE for more information.

Registration Website

Purpose

This website is how we register the ESP32 with a user (and their associated username and password). It scans the USB ports of the current device and asks the user to select a peripheral. The user will be able to enter their username and password, and on clicking submit, teh ESP32 should be registered to the user.

Setup

The registration website has already been deployed and is currently accessible here, but if you want to run it locally anyways, refer to registration-site/README.md for more information.

As above, you can open the following collapsible section to view the contents of this submodule's README.

README

Registration Website README

Table of Contents

This website is already deployed and is accessible here, so the instructions below are just in case you want to run the registration website locally (or change the code yourself).

Requirements

- NodeJS

Installation

To install the dependencies, run the following command:

```
npm install
```

Usage

To run the website, run the following command:

```
npm run dev
```

Misc.

- If you want to change the server URL, it is located in `.env.local`.
- Instead of using the commands listed above individually, you can run `make install` or `make run` to install the necessary packages and run the website.

License

This project is licensed under the terms of the MIT license. Refer to LICENSE for more information.

ESP32 Code

Purpose

This is the code needed on the ESP32 device which ensures that its MAC address is transmitted as and when necessary, and also handles the Bluetooth Low Energy (BLE) communication with the Raspberry Pi as the user approaches the computer to be logged in into.

Setup

If you want to flash the code to another ESP32, refer to esp32-code/README.md for more information.

As above, you can open the following collapsible section to view the contents of this submodule's README.

README

ESP32 Code README

Table of Contents

This code is already setup on the ESP32s, but we have included the instructions below just in case you want to change the code yourself and test things out.

Requirements

- PlatformIO

Setup

The very first thing that needs to be done is to connect the ESP32 to the device which has this folder/code.

Next, navigate to the `BLE-Broadcasting/` directory in the terminal.

Usage

Once the Setup steps have been completed, run the following command:

```
pio run --target upload
```

Then, you just need to wait until the *=== [SUCCESS] ===* message shows up. This should usually take approximately 30 seconds to complete.

Troubleshooting

To see if the ESP32 code was flashed to the device correctly, run the following command:

```
pio device monitor
```

The MAC address of the device should be constantly outputted to the screen every couple of seconds. If this is not the case, then you may need to unplug and replug the ESP32 back into the computer, then re-flash the code to it.

To exit this *Serial Monitor*, type *Ctrl* and *C* on the keyboard.

Misc.

- Instead of using the commands listed above individually, you can run `make run` or `make monitor` to flash to the ESP32 and run the serial monitor.

License

This project is licensed under the terms of the MIT license. Refer to LICENSE for more information.

Raspberry Pi Code

Purpose

This code checks for BLE signals from registered ESP32 devices and verifies their credentials, alongside constantly checking the distance from the computer. Once the distance is small enough, it uses the camera module attached to the Raspberry Pi and checks for the registered user, and if they are found the Raspberry Pi 5 communicates with the Raspberry Pi Pico via UART which acts as a USB HID (Human-Interface Device) keyboard connected to the computer to be logged in into, and types out the user's username and password.

Setup

If you want to change/run the Raspberry Pi 5 or Raspberry Pi Pico code, refer to rpi-code/README.md for more information.

As above, you can open the following collapsible section to view the contents of this submodule's README.

README

Raspberry Pi Code README

Table of Contents

This code is already setup on the Raspberry Pi 5s and Raspberry Pi Picos, so only the Usage section needs to be looked at. However, we have also included additional instructions below just in case you want to change the code yourself and test things out.

The code is split into 3 main parts:

1. `main/` - this includes all the code used on the Raspberry Pi 5s.

2. `pico/` - this includes all the code used on the Raspberry Pi Picos.
3. `web-app/` - this includes the code for the simple web application which displays the current active users (ESP32s) the Raspberry Pi 5 is searching for, and their distance to the Raspberry Pi 5.

Raspberry Pi 5

There are 2 methods to setup and run this code. The first is much easier to setup and is therefore recommended.

Method 1 - PyInstaller

Requirements

- Pyinstaller (Python)

Building

To build the application, run the following command after navigating to `main/`:

```
pyinstaller pyinstaller.spec
```

Usage

From the same directory, run the following command to run the script:

```
./dist/proximity_gateway/proximity_gateway
```

Method 2

Requirements

- Pip (Python)

Installation

To install all the dependencies, run the following command from the `main/` directory:

```
pip install -r requirements.txt
```

Usage

To run the script from the same directory, run the following command:

```
python main.py
```

Raspberry Pi Pico

Requirements

- CircuitPython
- Thonny

Setup

Open `pico-hid.py` with Thonny and save the file to the Pico with the filename `code.py`. This is enough to get it so that whenever the Pico gets power, it will automatically run the contents of the file.

Web App

Requirements

- NodeJS

Installation

First, navigate to the `web-app/` directory. Then, run the following command to install the necessary packages:

```
npm install
```

Usage

From the same directory as Installation, run the following command to run the website:

```
npm run dev
```

Troubleshooting

Liboqs

Refer to the open quantum safe liboqs get started page.

Misc.

TODO

License

This project is licensed under the terms of the MIT license. Refer to LICENSE for more information.

Desktop Application

Purpose

This application opens as soon as a user is logged in, and immediately fetches their preferences from the server, executing commands as necessary to have these preferences loaded. This can be things like increasing zoom or high contrast mode, or even opening particular applications on startup. The application also has a chatbot, which the user can talk to, and the chatbot will automatically be able to determine how to create and execute a command so that the user's needs are met (and saved for the next time they login into any device).

Setup

This application has already been built and can be downloaded from the Releases page. If you want to modify or run the desktop application yourself, refer to desktop-app/README.md for more information.

As above, you can open the following collapsible section to view the contents of this submodule's README.

README

Desktop Application README

Table of Contents

Building

Refer to the Releases page to download and build the application.

> NOTE: For proper command execution and application startup, you must run it on a GNOME-based GUI (any flavour of Linux e.g. Kali-Linux).

Running

- Ensure ollama is running and active
- Run the built application
  - This can be done either by finding the app and double clicking on it OR
  - By opening the terminal, navigating to the directory with the file and running `./<name of the file>`

In case you want to change the code, below are the instructions to install and use the application without the built versions.

Requirements

- NodeJS
- Cargo
- Ollama

Installation

Navigate to the `QPG-Application/` directory.

Then, to install the dependencies, run:

```
npm install
```

Usage

Firstly, ensure that ollama is running. If it is not, run:

```
ollama serve
```

> NOTE: You can check if ollama is running by trying to access http://localhost:11434 in a browser.

To build the application for production, run:

```
npm run tauri build
```

The built application can be found in `QPG-Application/src-tauri/target/release/`.

Development

To start the application in development mode instead, run:

```
npm run tauri dev
```

Misc.

- If you want to change the ollama and server URLs, they are located in `QPG-Application/.env.example`.
- Instead of using the commands listed above individually, you can run `make dev` or `make build` from the root directory of this project to install the necessary packages and run/build the project.

License

This project is licensed under the terms of the MIT license. Refer to LICENSE for more information.

User Manual

The user manual for this project can be found here.

License

This project is licensed under the terms of the MIT license. Refer to LICENSE for more information.

Contributors

- Raghav Awasthi
- Marwan Yassini Chairi El Kamel
- Abdulhamid Abayomi
- Abdul Muhaymin Abdul Hafiz