Backend API

Source code for the backend of the **IBM FISE Concierge**, providing support for both the app and the companion web app,
accessed via a restful API on `localhost`.

The backend can be run on a separate server-like device in a fully-federated manner. To accommodate this, the backend
sends out UDP broadcast messages to all devices in the local network allowing devices to discover the IP address of the
machine running the backend to connect to the RESTful API.

Deployment

The backend is containerised to make for simple deployment.

Docker

To start the application in a Docker container, you must first build the image with:

```
docker build --tag backend .
```

And then to start the application on port `8100`:

```
docker run --rm -it -p 8100:8100 backend backend
```

Without Docker

If you would prefer to not use Docker, you will need JDK 11 and Maven installed.

To start the application, ensuring no service is currently using port `8100`, run the following command in a terminal in
this directory (where the `pom.xml` is located):

```
mvn clean; mvn org.springframework.boot:spring-boot-maven-plugin:run
```

An executable `jar` file can be used instead, built and executed manually using:

```
mvn clean package
```

You can also start the application using a Python script with the following command:

```
python3 tools/run.py
```

This requires python to be installed.

or simplicity, a shell script is included to run the backend as a RESTful API:

```
./backend
```

Depending on your system configuration, you may need to give the script execution permissions which can only be done with administrator privileges (`sudo`):

```
sudo chmod +x backend
```

Command Line Arguments

There are different execution options for the API, summarised below.

Note that these can only be enabled by passing them as command line arguments to the Python `run` script or running the
application as a `jar`. Command line arguments are unavailable with Java if not executing using a `jar` file (using
the `java -jar ...` command).

The following arguments are all optional.

- A desired port can be specified using `-port=PORT` (or one of `--port`, `-p` and `--p`). Note that this must be an
  integer and is set to `8100` by default.

- Logging information to the console can be enabled with the `-l` (or `--l`) flag - logging is disabled by default.

- The application can be built into a single jar and executed using the `-jar` (or `--jar`) flag.
  - This option is only available using the Python `run` script.

When using the `jar` file (with `java -jar ...`):

- The `-l` flag is supplied **without** the leading `-` character(s) - i.e. as `l`.

- The `-port` flag is **always** supplied with a single `-` and the same goes for `-p`.

Running `python3 tools/run.py -h` gives you a similar breakdown of possible command line arguments and their aliases.

- The `-h` flag is not available outside of the Python script.