Overview--

Group project created and designed by:

	Name	Student Id
1	Will Lehmann	A1889855
2	Edward Chipperfield	Axxxxxxx
3	Aditeya	Axxxxxxx
4	Sadman	Axxxxxxx

Table of contents

Installation of required packages

First step to running the server and client is to install the required dependencies.

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

Server / Starting Servers

- On start up, the server will generate a new RSA keypair and save it in `/storage`. After this, all instances of the server will use the persistance of the RSA key pairs for following runs of the server unless they are deleted, then a new keypair will be generated.
- Further, as there was no specific requirements from the SOCP regarding logging, our servers log the `uuidv4s` of clients connecting to the console for ease of use.

Single Server

Start a single server server (default port 8765):

```
python -m backend.run_mesh
```

- By defualt, this server listens on 0.0.0.0:8765

Multiple Servers

To run a federated mesh, start each server on a different port and set peers:

Server 1:

```
python -m backend.run_mesh
```

- Default server runnning on 0.0.0.0:8765

Server 2:

```
set SOCP_PORT=8766
python -m backend.run_mesh
```

- Secondary server runnning on 0.0.0.0:8766

Client / Starting Client

Using the CLi client config

Each client needs a unique config file (e.g., `client/alice.json`, `client/bob.json`).
The client will auto-generate this file if it does not exist.

Running the CLI Client

To start a client:

```
python -m client.cli_client client/alice.json
```

```
python -m client.cli_client client/bob.json
```

- Each client config will have a unique user ID and keypair.
- You can run as many clients as you want, each with its own config.

Commands accessable to the client.

After the client is connected, sends its welcome, and its advertisement and is acknowledged it will get access to the following commands:

- `/list`
  List all users currently connected to the local server.
- `/tell <user> <text>`
  Send a direct message to a user (by UUID).
- `/all <text>`
  Send a public message to all users on all connected servers.
- `/file <user|public> <path>`
  Send a file to a user or to the public channel.
- `/quit`
  Disconnect the client.

SOCP Compliance

This implementation is **SOCP-compliant**:

- All server-to-server and user-to-server frames are signed and verified.
- User and server IDs are UUIDv4.
- Deduplication is enforced for all relayed messages.
- Presence and user advertisements are propagated mesh-wide.
- Heartbeats and peer reaping are implemented.
- All state is kept in memory (no persistent user/group directories).
- Error codes and envelope structure follow the SOCP spec.

Development Notes

- **No persistent user/group storage:**
  All user presence and keys are kept in memory. When a server restarts, all presence is lost.
- **Keypairs:**
  Server keypairs are stored in `/storage`. Client keypairs are stored in their config files.
- **Testing:**
  See `scripts/tests_smoketest.py` for a minimal automated test.

Troubleshooting

- **Multiple clients disconnect each other:**
  Ensure each client uses a unique config file (unique user ID).
- **Peer not connecting:**
  Check that the peer server is running and reachable.
- **Signature errors:**
  Ensure all keys are generated and loaded correctly.
