Through The Clouds: An Introduction to Cloud Computing

Day 3: Projects

Instructors: Anurag Khandelwal, Ramla Ijaz, Garrett Sager
TA: Joaquin Soto

Yale

Today’s Agenda

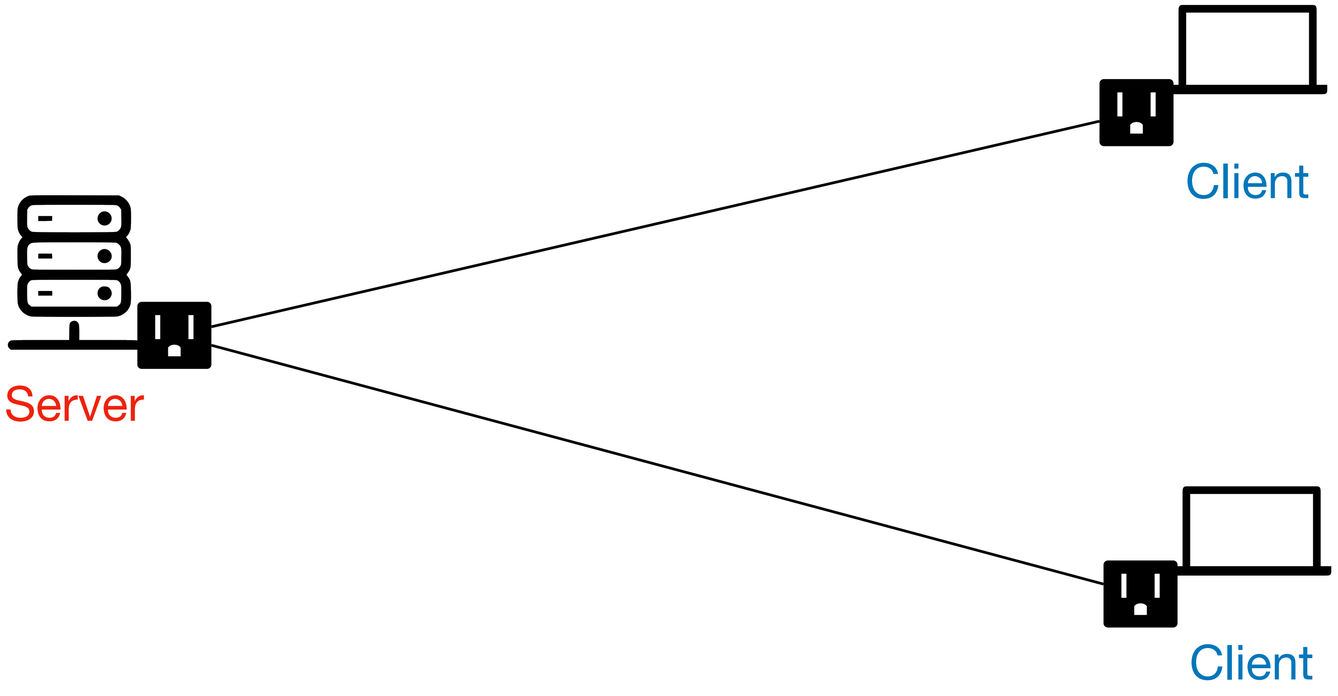
Today’s Agenda

- Keep our hands dirty!
  - [Continued] Simple Ping-Pong Applications
  - Client Registry
  - Instant Message

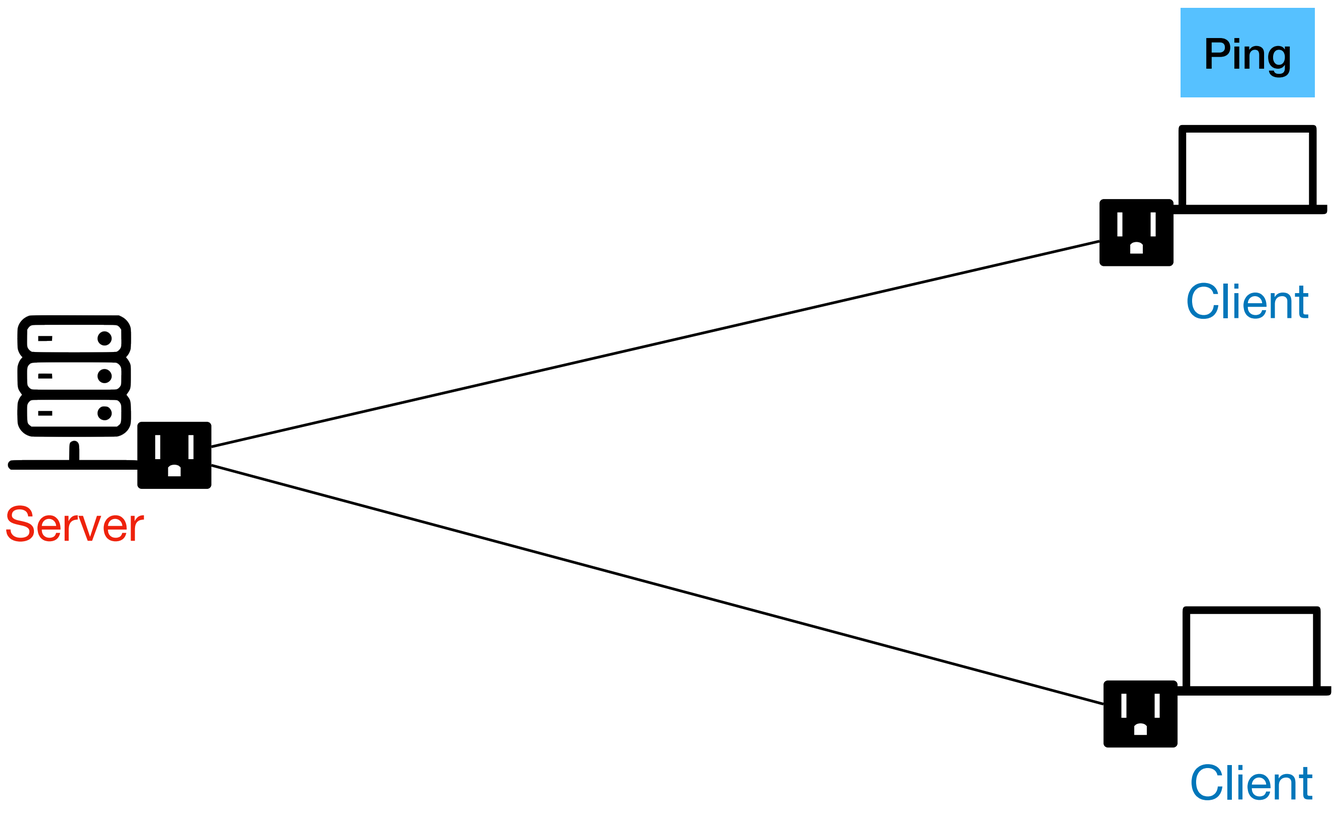
Building Your Chat Application in 4 Simple Steps

- Ping pong 1: Single Server, Single Client
- **Ping pong 2: Single Server, Two Clients**
- Client Registry & Login
- Instant Messaging!

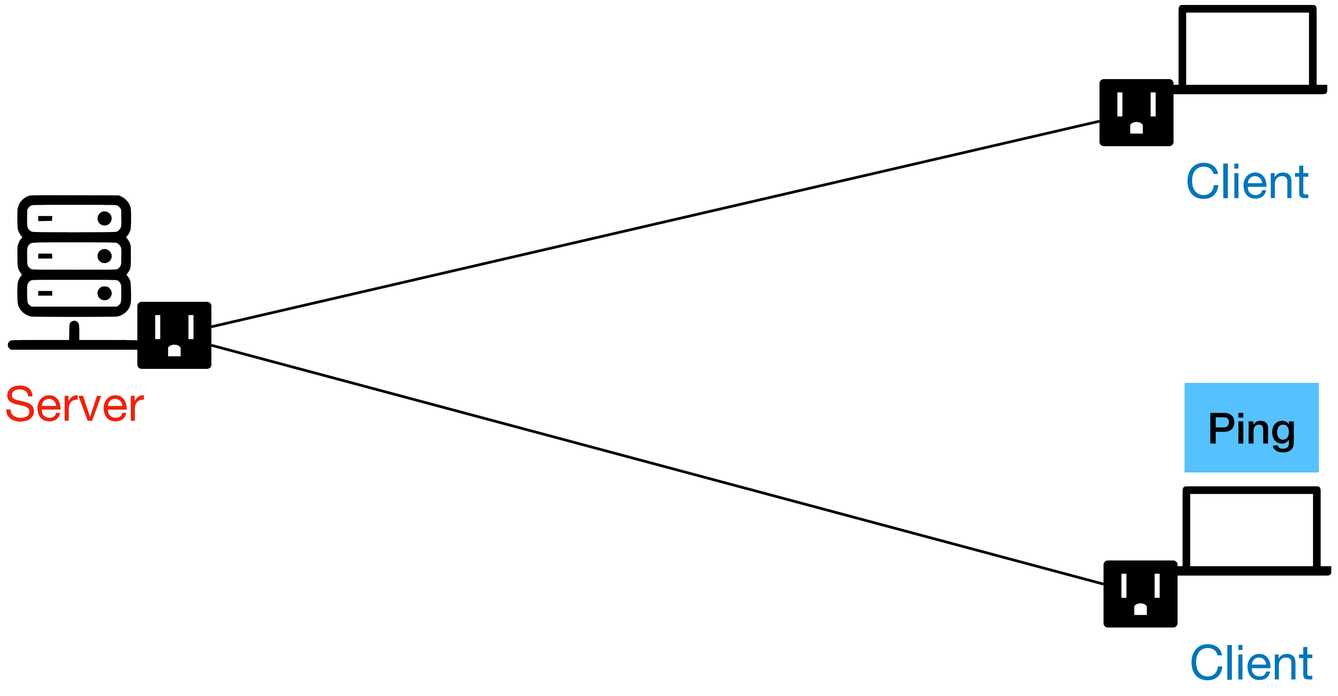
A simple ping-pong application



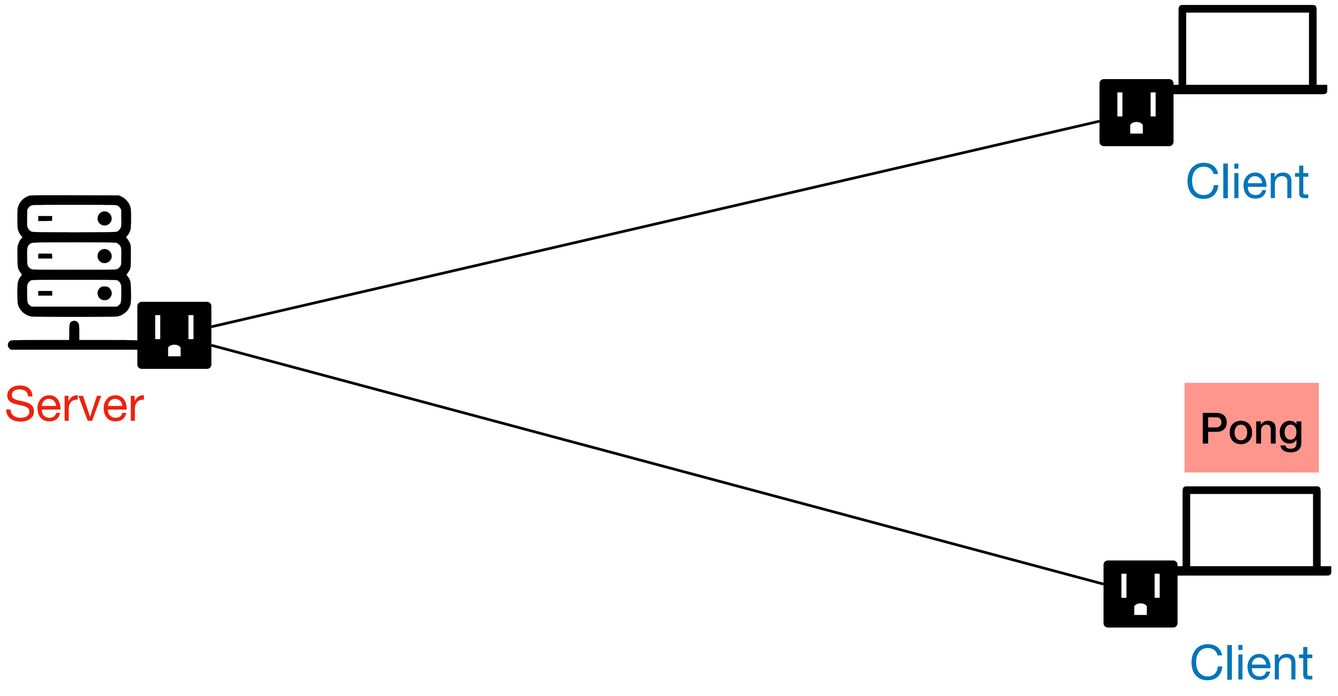
A simple ping-pong application



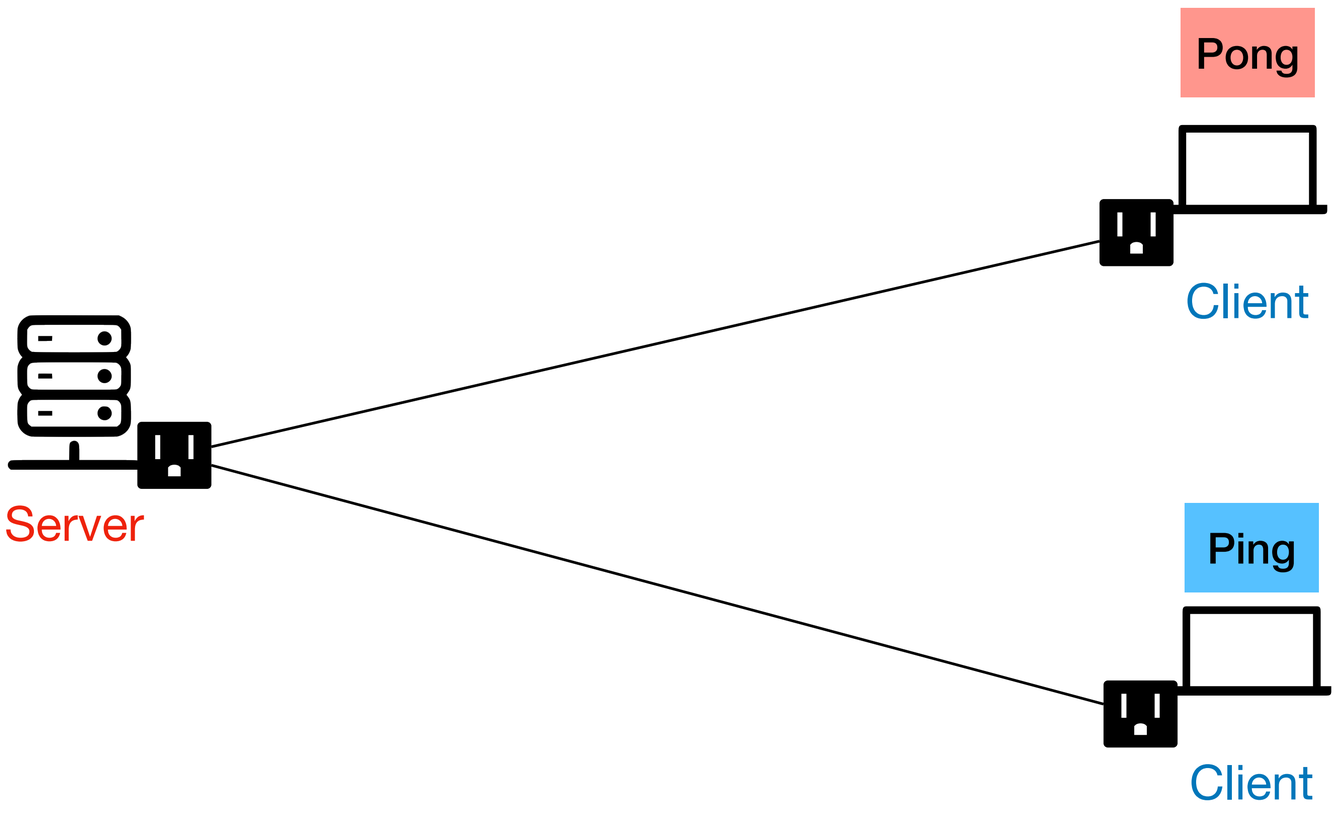
A simple ping-pong application



A simple ping-pong application



A simple ping-pong application



How do you do it?

How do you do it?

- The functions you have to edit:
- In client_logic.py:

```
async def client1_logic(client):
  # your code here

async def client2_logic(client):
  # your code here
```

- In server_logic.py:

```
async def server_logic(server, msg):
  # your code here
```

How do you do it?

- The functions you have to edit:
- In client_logic.py:

```
async def client1_logic(client):
  # your code here

async def client2_logic(client):
  # your code here
```

- In server_logic.py:

```
async def server_logic(server, msg):
  # your code here
```

Same as before

How do you do it?

- The functions you have to edit:
- In client_logic.py:

```
async def client1_logic(client):
  # your code here

async def client2_logic(client):
  # your code here
```

- In server_logic.py:

```
async def server_logic(server, msg):
  # your code here
```

Same as before

```
await server.forward_message(msg)
```

How do you test it?

How do you test it?

- Open **three** terminals
- Go to the directory where the code resides (`pingpong2`) on all terminals

  `cd /path/to/pingpong2` (on Linux & MacOS)
  `cd \path\to\pingpong2` (on Windows)

- On one terminal, run the server:

  ```
  python3 server.py
  ```

- On the other two terminals, run the clients:

  ```
  python3 client1.py
  python3 client2.py
  ```

Demo & Coding

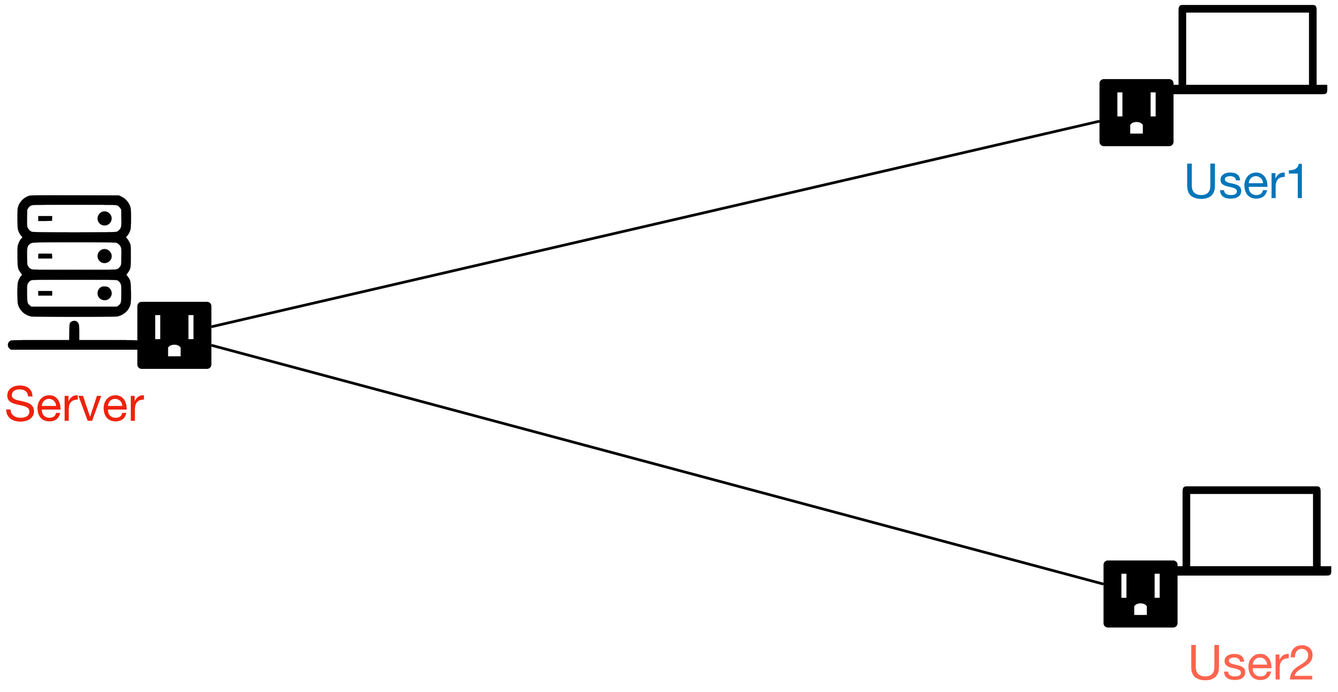
Building Your Chat Application in 4 Simple Steps

- Ping pong 1: Single Server, Single Client
- Ping pong 2: Single Server, Two Clients
- **Client Registry & Login**
- Instant Messaging!

User registration and login

Registered Users

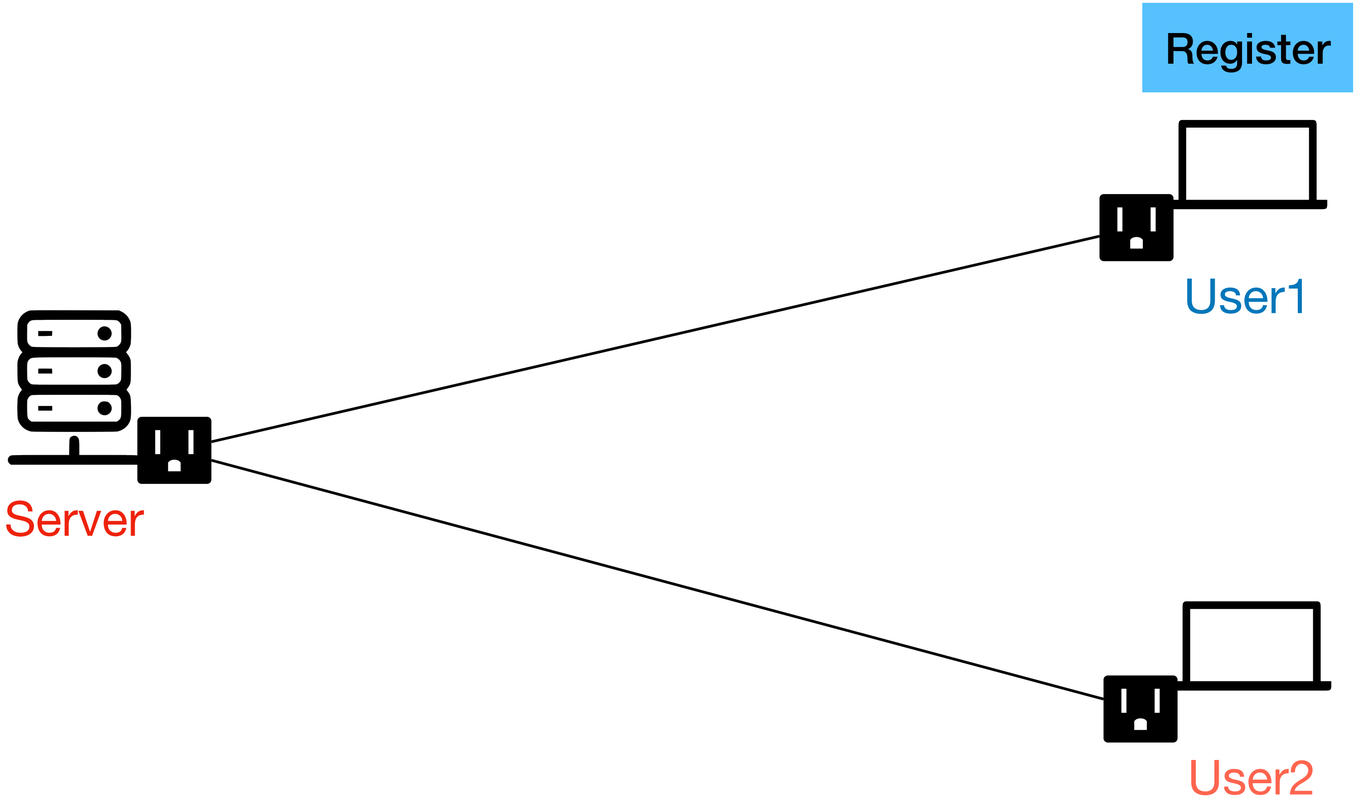
Logged-in Users



User registration and login

Registered Users

Logged-in Users

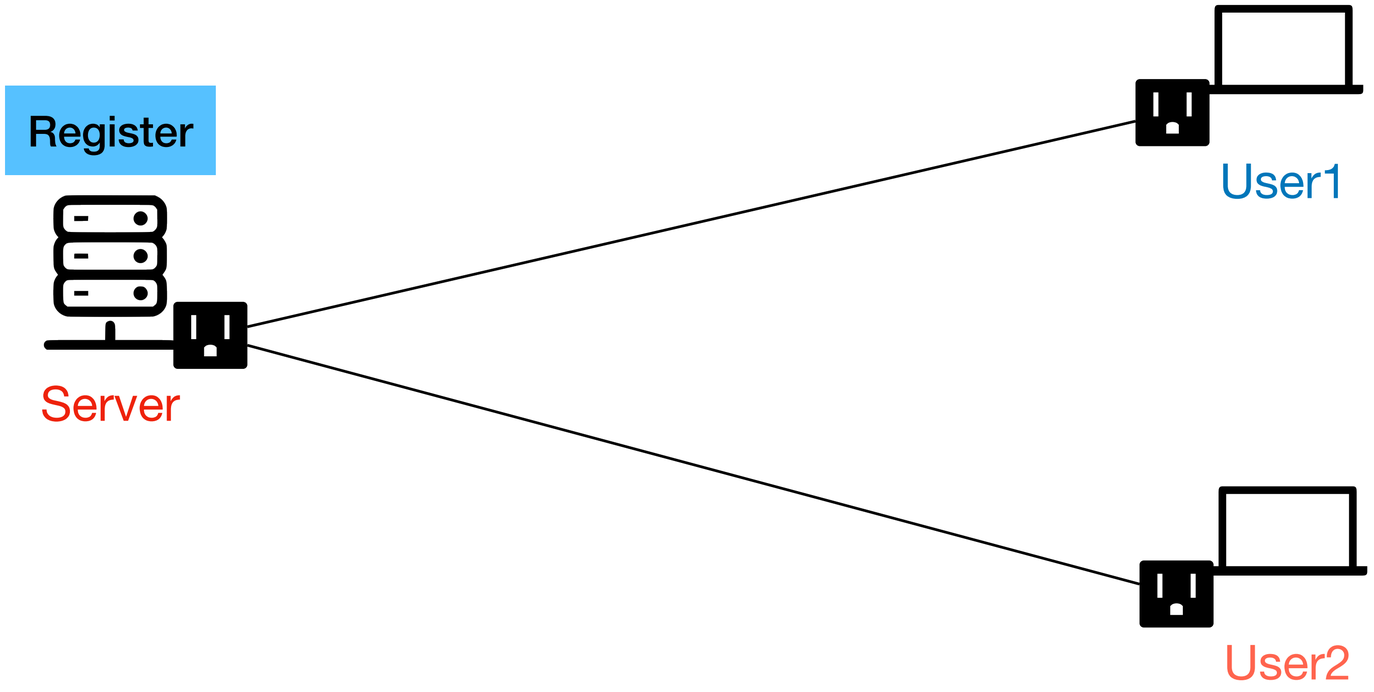


User registration and login

Registered Users

User1

Logged-in Users

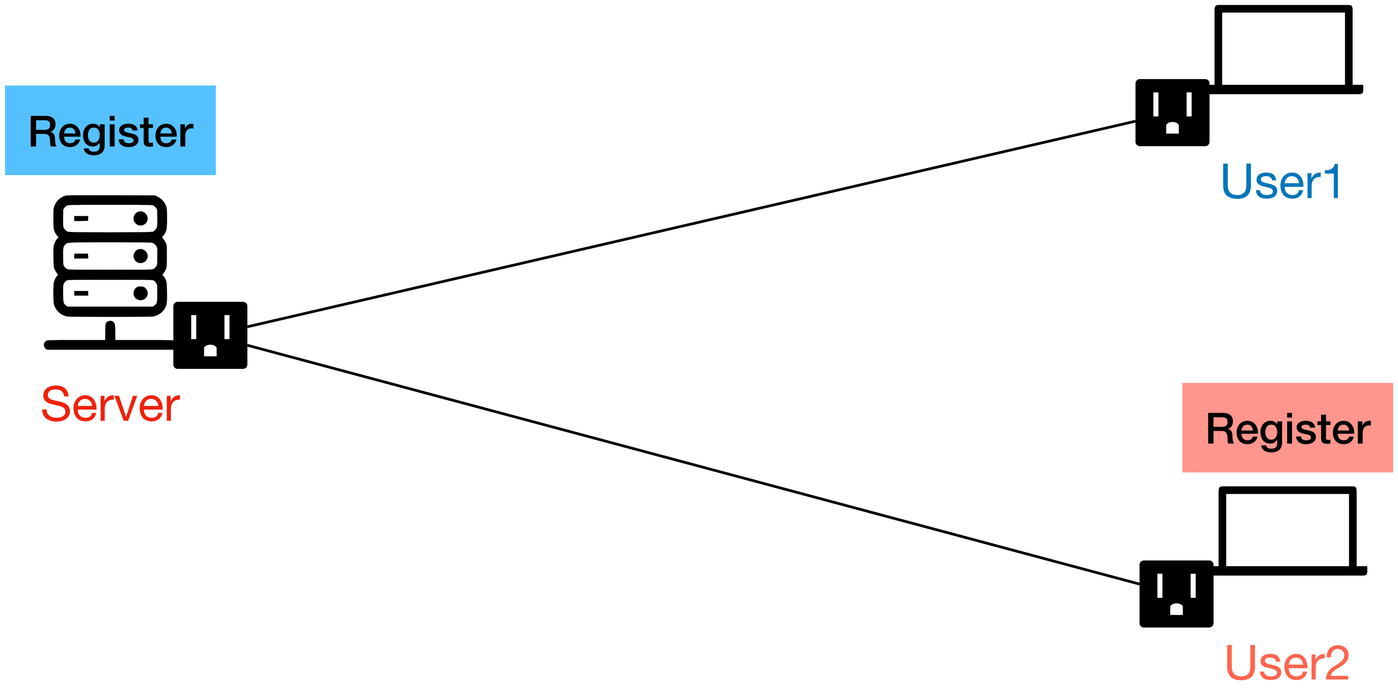


User registration and login

Registered Users

User1

Logged-in Users



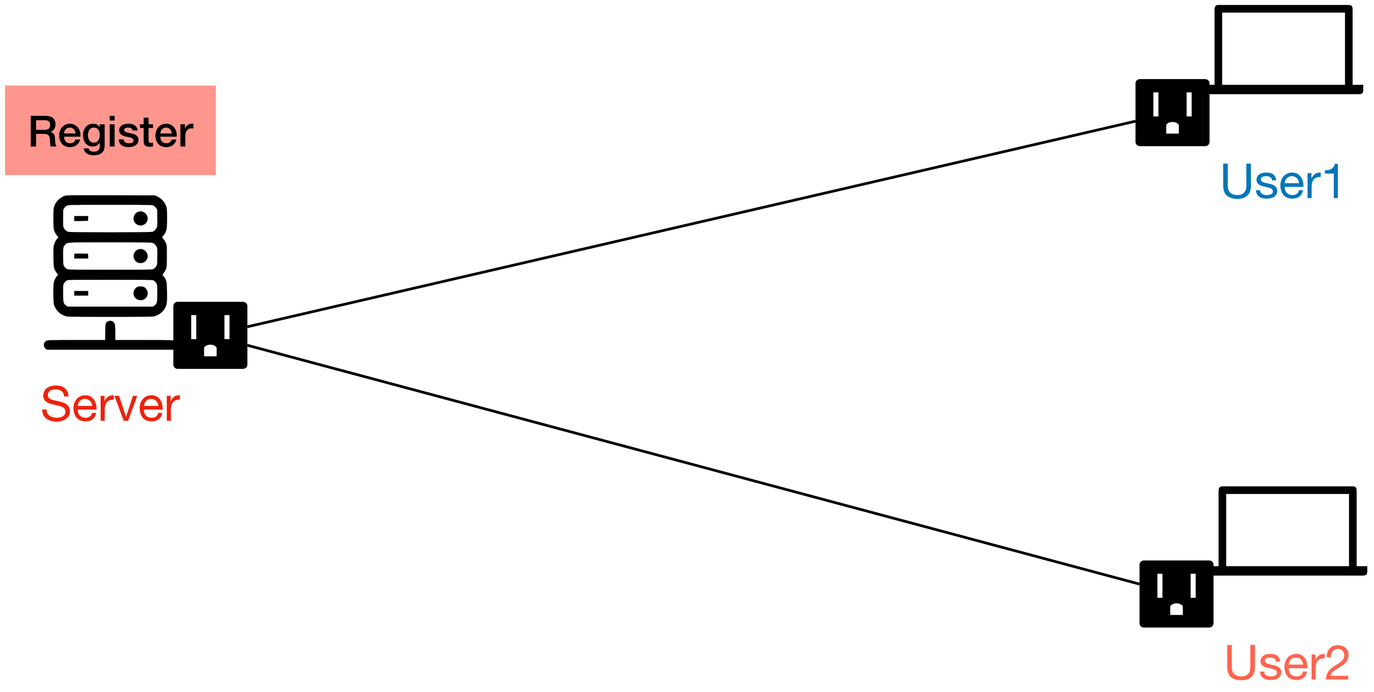
User registration and login

Registered Users

User1

User2

Logged-in Users



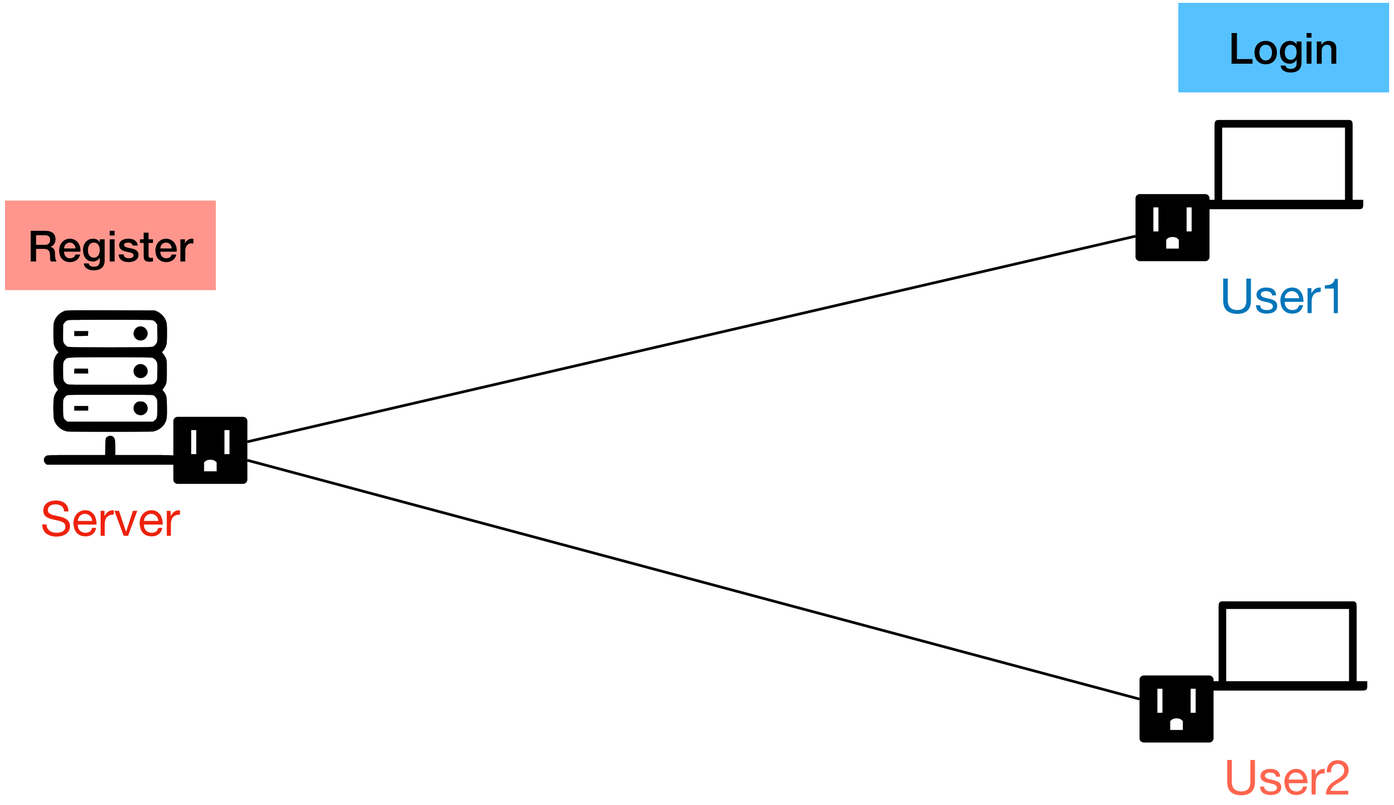
User registration and login

Registered Users

User1

User2

Logged-in Users



User registration and login

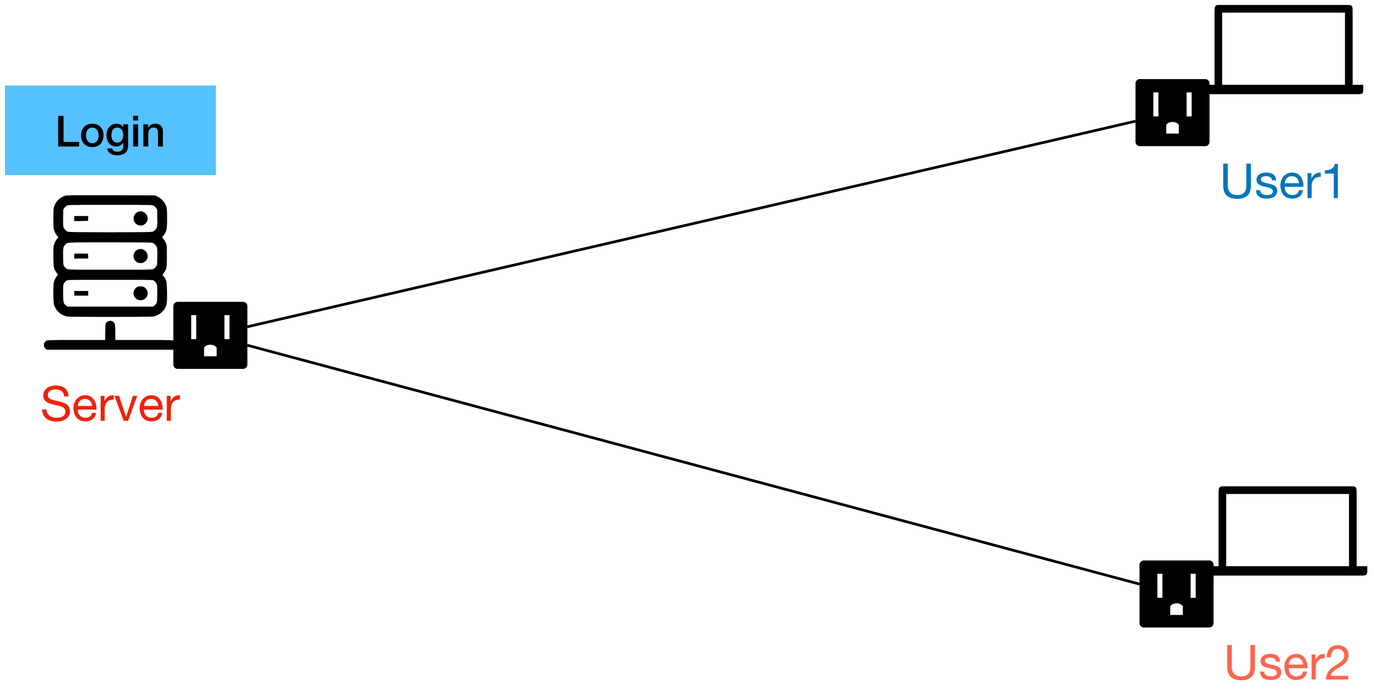
Registered Users

User1

User2

Logged-in Users

User1



User registration and login

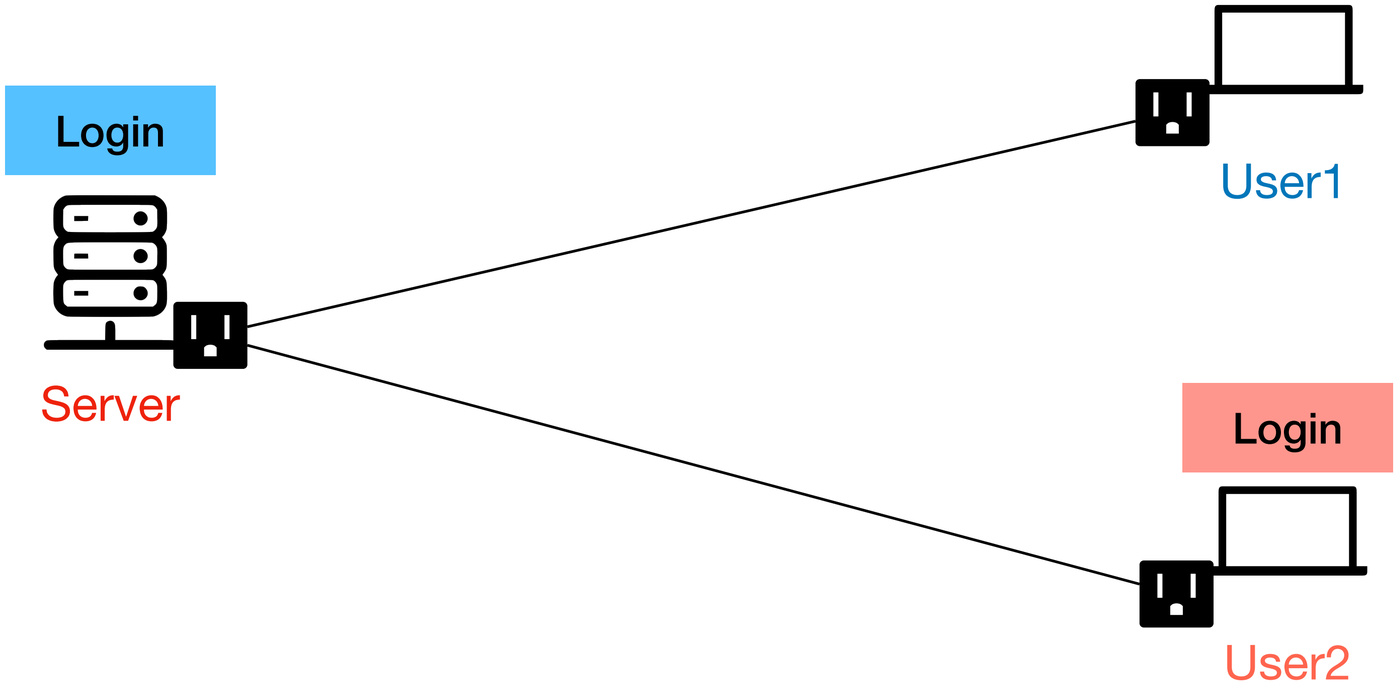
Registered Users

User1

User2

Logged-in Users

User1



User registration and login

Registered Users

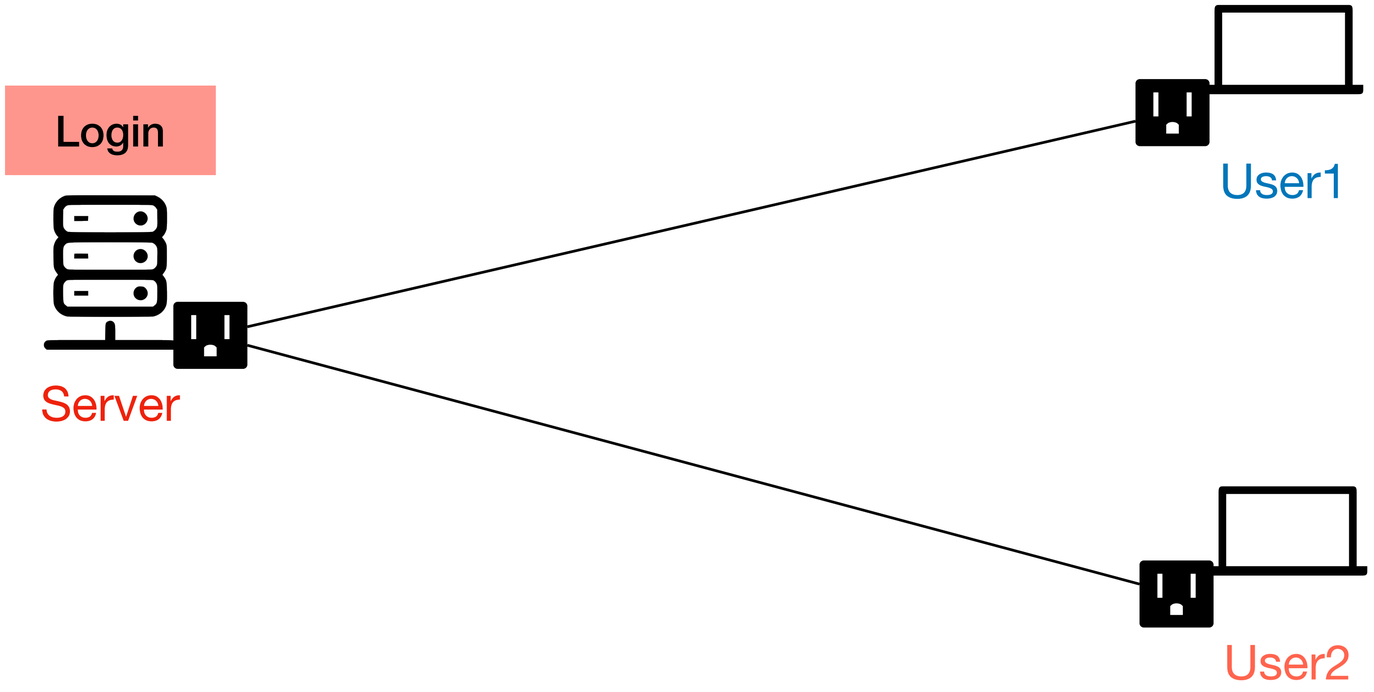
User1

User2

Logged-in Users

User1

User2



How do you do it?

How do you do it?

- The functions you have to edit:
- In client_logic.py:

```
async def register_user(client, username):
  # your code here

async def login(client, username):
  # your code here

async def request_user_list(client):
  # your code here
```

How do you do it?

- The functions you have to edit:
- In client_logic.py:

```
async def register_user(client, username):
  # your code here

async def login(client, username):
  # your code here

async def request_user_list(client):
  # your code here
```

Send a message to the server based on the request type

How do you do it?

How do you do it?

- For client_logic, use the following helper functions:
  - `await client.register(self, username)`
  - `await client.login(self, username)`
  - `await client.request_registry(self)`

How do you do it?

How do you do it?

- The functions you have to edit:
- In server_logic.py:

```
async def register_client(server, username):
  # your code here

async def login_client(server, username):
  # your code here

async def send_registry_to_client(server):
  # your code here
```

How do you do it?

- The functions you have to edit:
- In server_logic.py:

```
async def register_client(server, username):
  # your code here

async def login_client(server, username):
  # your code here

async def send_registry_to_client(server):
  # your code here
```

1. Check if the client has already registered or if the username is taken
2. If it has, then send a registration_failed message
3. If not, register the client and send a registration_successful message

How do you do it?

- The functions you have to edit:
- In server_logic.py:

```
async def register_client(server, username):
  # your code here

async def login_client(server, username):
  # your code here

async def send_registry_to_client(server):
  # your code here
```

1. Check if the client has already registered or if the username is taken
2. If it has, then send a registration_failed message
3. If not, register the client and send a registration_successful message

1. Check if the client has already registered and its username matches what the server has on record
2. If it has, login the client and send a login_successful message
3. If not, then send a login_failed message

How do you do it?

- The functions you have to edit:
- In server_logic.py:

```
async def register_client(server, username):
    # your code here

async def login_client(server, username):
    # your code here

async def send_registry_to_client(server):
    # your code here
```

1. Check if the client has already registered or if the username is taken
2. If it has, then send a registration_failed message
3. If not, register the client and send a registration_successful message

1. Check if the client has already registered and its username matches what the server has on record
2. If it has, login the client and send a login_successful message
3. If not, then send a login_failed message

1. Check if the client has already logged in
2. If it has, then send the registry
3. If not, send a request denied message

How do you do it?

How do you do it?

- For registration, a few helpful functions to help you out:
  - `server.registered()`: is the client registered?
  - `server.username_exists(username)`: does the username already exist?
  - `server.register_user(username)`: adds the user to the list of registered users
  - `await server.registration_successful(username)` / `server.registration_failed(username)`: responds to client to indicate success/failure

How do you do it?

How do you do it?

- For login, a few helpful functions to help you out:
  - `server.username_matches_record(username)`: is the client registered with this username?
  - `server.login_client(username)`: adds the username to the list of logged in users
  - `await server.login_successful(username)` / `server.login_failed(username)`: responds to client to indicate success/failure

How do you do it?

How do you do it?

- For sending registry to the client, a few helpful functions to help you out:
  - `server.logged_in()`: is the client logged in with the server?
  - `await server.send_registry()` / `server.request_denied()`: responds to client with registry on success/a request denied message on failure.

How do you test it?

How do you test it?

- Open **three** terminals
- Go to the directory where the code resides (`client_registry`) on all terminals

  `cd /path/to/client_registry` (on Linux & MacOS)
  `cd \path\to\client_registry` (on Windows)

- On one terminal, run the server:

  ```
  python3 server.py
  ```

- On the other two terminals, run two clients:

  ```
  python3 client.py
  ```

Demo & Coding