# Lab1

Lorin Speybrouck

## Preparation

`Problems during preparation:` I was unable to install the required packages before also updating pip, installing rust and installing Visual Studio build tools

## Tasks

### Task 1: Registration

`Result:` see **registration_succes.mp4** and **registration_existing.mp4**

`Code`

```
# routes.py
# This method creates a RFIDuser with the entered information and adds
it to users if it does not exist in there yet, it then returns the user
to the index page. When successful it also returns the name and a succes
alert. When unsuccessful it says that a user like this already exists
@router.post("/register")
def register(request: Request, data: Registration = Depends(),
alert:Optional[str] = None):
    temp = RFIDuser(data.firstname, data.lastname, data.email,
data.rfidid)
    if temp in users:
        return templates.TemplateResponse("index.html", {
            "request": request,
            "alert": "User with this RFID already exists"
        })
    users.append(temp)
    return templates.TemplateResponse("index.html",{
            "request": request,
            "name": data.firstname,
            "alert": "Succusfully registered!"
    })
```

```html
<!-- register.html -->
<div class="jumbotron">
    <h1 class="display-3">Registration</h1>
</div>

<div class="row marketing">
    <div class="col-lg-12">
        <form method="POST" action="register">
            <input style="margin-bottom: 5px;" type="text" class="form-
control" name="firstname" id="firstname"
                    placeholder="firstname">
            <input style="margin-bottom: 5px;" type="text" class="form-
control" name="lastname" id="lastname"
                    placeholder="lastname">
            <input style="margin-bottom: 5px;" type="text" class="form-
control" name="email" id="email"
                    placeholder="email">
            <input style="margin-bottom: 5px;" type="text" class="form-
control" name="rfidid" id="rfidid"
                    placeholder="rfidid">
            <button class="btn btn-primary">Register</button>
        </form>
    </div>
</div>
```

## Task 2: IoTRPi exercise

`Result:` see **registration_succes.mp4**

`Code`

```python
# MyRPI.py
# Uses the GPIO method to setup read and write the LED_PIN
LED_PIN = 4

class MyRPi(metaclass=RaspberryPi):
    """
    Class that is responsible for accessing and setting the pins on the
RPi
    """
    def __init__(self):
        """
        Initiate the LED_PIN as OUTPUT and initial value of LOW
        """
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(LED_PIN, GPIO.OUT, initial=GPIO.LOW)

    def get_status(self):
        """
        Method to retrieve the status of the actuator
        :return: status as boolean (HIGH / LOW)
        """
        return GPIO.input(LED_PIN) == GPIO.HIGH

    def set_status(self, status):
        """
        Method to set the status of the actuator
        :param status: boolean (HIGH / LOW)
        """
        if status:
            GPIO.output(LED_PIN, GPIO.HIGH)
        else:
            GPIO.output(LED_PIN, GPIO.LOW)
```

```python
# routes.py
# The show_control_led GET method renturn the control_led.html page with
the light status gotten from the MyRPI class
@router.get("/control_led")
def show_control_led(request: Request):
    light_status = rpi.get_status()
    return templates.TemplateResponse("control_led.html", {
        "request": request,
        "lightStatus": light_status
    })

# The show_control_led POST methodtoffle the status using the MyRPI
class and return the ontrol_led.html page with the new status
@router.post("/toggle_led/{status}")
def toggle_led(request: Request, status: str):
    print("Turn light", status)
    new_status = True if status == "on" else False
    rpi.set_status(new_status)
    light_status = rpi.get_status()

    return templates.TemplateResponse("control_led.html", {
        "request": request,
        "lightStatus": light_status
    })
```

```html
<!-- control_led.html -->
<div class="jumbotron">
    <h1 class="display-3">Control Room</h1>
        <p class="lead">Turn the LED <span
style="color:forestgreen">ON</span> / <span style="color:red">OFF</span>
by using the buttons below.</p>
</div>

<div class="row marketing">
    <div class="col-lg-12 text-center">
        {% if lightStatus %}
                <img src="/static/img/light_on.png" alt="Light on"/>
        {% else  %}
            <img src="/static/img/light_off.png" alt="Light off"/>
        {% endif %}
    </div>
    <div class="col-lg-6 text-center">
        <form action="/toggle_led/on" method="post">
            <button name="LED" value="ON" class="btn btn-success"
type="submit">Turn the light ON</button>
        </form>
    </div>
    <div class="col-lg-6 text-center">
        <form action="/toggle_led/off" method="post">
            <button name="LED" value="ON" class="btn btn-danger"
type="submit">Turn the light OFF</button>
        </form>
    </div>
</div>

<div class="row marketing">
    <div class="col-lg-12 text-center">
    </div>
</div>
```

## Task 3: Visualize the RFID users on a new route

`Result:` see **extra task.mp4**

```python
# routes.py
# The get_users GET method returns the users.html page with the existing
users
@router.get("/users")
def get_users(request: Request):
    return templates.TemplateResponse("users.html", {
        "request": request,
        "users": users
    })
```

```
<!-- users.html -->
<!-- Thet user.html page displays the information of the users in a
table using the for loop -->
<!-- Download functionality is also added using JavaScript -->
<div class="jumbotron">
    <h1 class="display-3">Users</h1>
</div>

<table class="table table-striped table-bordered" id="userTable">
    <thead class="thead-dark">
        <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>RFID ID</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody>
        {% for user in users %}
        <tr>
            <td></td>
            <td></td>
            <td></td>
            <td></td>
        </tr>
        {% endfor %}
    </tbody>
</table>

<button id="downloadBtn" class="btn btn-primary mb-3"
type="button">Export users as CSV</button>

<script>
    document.getElementById('downloadBtn').addEventListener('click',
function() {
        let table = document.getElementById('userTable');
        let rows = table.querySelectorAll('tr');
        let csvContent = "data:text/csv;charset=utf-8,";

        rows.forEach(row => {
            let cols = row.querySelectorAll('th, td');
            let rowData = [];
            cols.forEach(col => rowData.push(col.innerText));
            csvContent += rowData.join(",") + "\n";
        });

        let encodedUri = encodeURI(csvContent);
        let link = document.createElement('a');
        link.setAttribute('href', encodedUri);
        link.setAttribute('download', 'user_list.csv');
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    });
</script>
```

## Question 1

> How would you implement task 2 in such a way that a user would not have to refresh the page to see the current status of the LED if switched by another client?

Real time updates can be accomplished using polling or websockets

- Polling: here the page has a Javascript script that periodically asks the server for the state of the LED. This would be the easiest way of implementing but would be inefficient
- Websockets: this is a standardized way for a server to send data to a client without the client specifically asking for that data. This would be a far more efficient way of implementing this feature
    - This works by the client asking for a websocket connection with the server. The server that remembers all connected clients, and when the LED status updates it sends a update to each of them

## Question 2

> Describe what happens and why when someone posts invalid data, such as a login with blank username, or a typo in the field name.

When invalid data is posted to the server, validation checks occur. When these fail the server returns a "Bad Request" error and the URL show is updated with information about the fault. For example: http://iot07.local:8000/register?alert=Bad+Request%3A+email%3A+Field+required