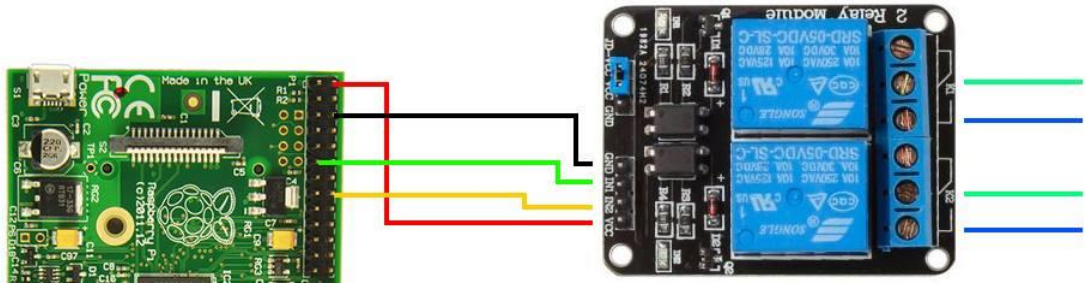
Project Repo

https://github.com/DanielDelchev/HomeAutomation

THE BASIC IDEA:

The purpose of the project is to allow a user to control appliances through a browser (for example turning on and off lighting or setting up an appliance to turn on itself (by connecting to the grid at a user specified time point and turning itself off at another time point (by disconnecting itself from the grid))).Basically, the structure behind the idea is the following: there is a raspberry pi to which a relay is connected. The relay is a part of an electrical circuit to which the appliance which will be controlled is also a part. When a signal is passed from the client to the server, the servers calls a corresponding method to raise or lower a leaver key and connect or disconnect the electrical circuit. The raspberry hosts a server for a website which will be used for an interface to the methods implemented for the usage of the relay. Since the resource (the relay) is only 1 currently, the server will not be multithreading (each user logged in controls the same relay considering the program will be used in a single household). A client can connect to the site through a browser assuming the device the user is using and the raspberry are in the same network. The raspberry on which the website is hosted can be connected to the internet via a router, or it can be connected with a LAN cable to the same local network as the device from which the site is accessed. When a user visists the site he is pompt to log in with a username and passwod there are two types of users – user and admin. Admins can create/delete users.Users and Admins and their encrypted passwords (hashcode) is kept on the severside on json files.



RUNNING THE PROJECT:

The server itself CANNOT be run on a regular machine since the RPi.GPIO module used as an interface to the pins of the raspberry (those pins are connected to the relay an send low voltage signals to the relay causing it to raise or lower its leaver keys) cannot be used on a regular computer/laptop since they do not have such hardware. To run the program (server side) on a raspberry, it must have python3.5 or higher and the following modules installed:

RPi.GPIO>=0.6.2 , bottle>=0.12.9 , Beaker>=1.8.1 , bottle-beaker>=0.1.3 , bottle-cork>=0.12.0

Also the raspberry has to be connected with a relay (as shown in the picture above (don't know if the picture will be displayed in pdf format)).The relay should be in the same electrical circuit as the appliance we would like to control. Since the site is not currently deployed, the raspberry and the

client machine should be connected either via a router or via a LAN cable. The site on the server side is started with the command ”sudo python3 web_control.py”. The site is run on the address specified in the web_control.py file. Currently the address is with IP 192.168.137.1 on port 8888 which I use because I intend to connect my laptop and the raspberry locally with LAN cable for the demo presentation. Of course the IP or port in the source code can be change to another. Neither the frontend , nor the backend code require compilation since python is a script language.

ABOUT THE BACKEND CODE, MODULES IMPORTED, BASIC IDEAS:

The pin_controls.py module in the project provides interface to the functions of RPi.GPIO module which controls the pins of the raspberry, though which the relays states are changed. The PIN1 and PIN2 constants are corresponding numbers of the pins on the board of the raspberry. Since there are different numerical systems for the relays the init function sets the system to BCM mode to which PIN1 and PIN2 correspond accordingly. The getState function returns the state of the relay whose pin is passed as an argument to the function returns 0 if the relay ‘s lever key is down (circuit is closed) or 1 otherwise (circuit) is not closed. The clean function resets all the pins of the raspberry to their initial state, this function is usually code before serve shutdown or in case of exceptions.The switchState function switches the state of the relay the pin parameter is the pin whose state is to be changed and op is the state to be changed to. There is a small delay (sleep) before the function returns so as to wait the actual physical fall/rise of the leaver of the relay.

The web_control.py module in the project is the main backend of the project. Amongst other things it uses the following imported modules :

Bottle – Bottle is a fast, simple and lightweight WSGI micro web-framework for Python.It is distributed as a single file module and has no dependencies other than the python standard library.Provides easily manageable routing of views and has got a Built-in HTTP development server and supports any any WSGI capable HTTP server.

https://github.com/bottlepy/bottle/blob/master/bottle.py

Cork - Cork provides a simple set of methods to implement Authentication and Authorization in web applications based on Bottle. Methods from this module are used in the project to implement the base login/logout operation, write down/ delete users from server data (json files are used to store the information for users (passwords are kept hashed)).

https://github.com/FedericoCeratto/bottle-cork/blob/master/cork/cork.py

beaker-middleware – used for sessions and cookies

https://github.com/bbangert/beaker/blob/master/beaker/session.py

logging - This module defines functions and classes which implement a flexible event logging (logs for messages and errors etc.) system for applications and libraries.No actual logs are kept by the

program but without a Logger object a user can still browse the pages of the site even after logging out.

https://docs.python.org/3.5/library/logging.html

The rest imported modules are more or less standard.

@authorize(role , fail_redirect) decorator is used on functions rendering templates/views which require logged in session role is the minimum role a user must have to access that part of the site, fail_redirct specifies the action to be performed in case of failure. @route('URL') decorator specifies which URL redirects to this function route stands for both post and get methods, although @post('URL') and @get('URL') decorators also exists in bottle. The @bottle.view('template.tpl') decorator specifies which template(view) to be displayed upon calling this function usually through a route URL, the template could also be rendered by returning it at the end of the method, but in cases in which a json is returned to the html, it is easier to render the template with the decorator.There are two types of users currently – user and admin, although the cork module allows the creation of versatile roles with different level of access. The postd() method requests the fields from forms, the post_get method requests stings from fields in html with the corresponding name tag as its first argument .There are a few methods whose purpose is to simply route URLS to their corresponding templates occasionally passing some data from the backend to the htmls to be displayed, these templates are discussed later.

There are two monitor semaphors (Lock objects) the first one is used for the relays in some cases. For example locking them while a check for their states is being performed and unlocking them after that. The other monitor is used for the list of appointment objects which are basically passively waiting threads and information about them. When a request for the relay to connect the circuit at time X and disconnect it at time Y a passive thread is created and put into a list, keeping information about those threads. There are also a few methods for switching the state of the relays manually, or requesting an automated switch at time X or removing such request. The methods corresponding to the URLS '/server_time' and '/epoch_time' return json objects containing the current time of the server in standard time or in unix epoch time. There is a constantly running daemon thread which every 6 seconds or so checks if there is an Appointment which must be removed from the list of Appointments from one of the relays. For example if a request for relay1 to connect the circuit at time X and disconnect it at time Y has expired (it is now passed time Y) the thread will remove that Appointment object with had contained the passive thread for the operation.

ABOUT THE FRONTEND CODE, LIBRARIES USED, BASIC IDEAS:

The front end consists of 8 templates which are basically html files with some javascript code and hardly any css. Bootstrap.min.js and jquerry.js are used. Most of the communication between the front end and back end is done via json objects and jquerry.ajax, with most of it not done asynchronically .A few of the data transfers are performed with bottle.py native methods or native python code in the templates.

The login.tpl consists only of a simple log-in form which on submission passes its content to the "/login" URL which is handled by the login() and login_form() functions in the backend.

The sorry_page.tpl is displayed when a client with role user tries to visit the admin exclusive page.

The admin_page.tpl consists of two forms – one for adding a new user with some username, role and passwords, and one for deleting an existing user. On submission the content is parsed into json and is sent to the corresponding url which is the attribute action of the form. Also the currently registered users and their roles are displayed (templates can have native python code with special syntax starting with % and ending with %end the list of users and roles is passed to the html as json objects upon request for rendering the template from the backend. There are also a button for logging out and a button redirecting to home page.

The make.tpl consists mainly of two forms – one for each relay. The request consist of timedate when the relay will be turned on and timedate when it will be turned off. On submission a javascript function is called which performs validation and sends the parameters for the request as a json object to the backend with jQuery.ajax. the request is not asynchronical. Apart from the forms there are a few buttons which redirect to other pages or call methods which delete all Appointments for a specific relay. On loading the template a timer function is called immediately which displays the server time and the client time each second.

Show_relay1.tpl also has a function called upon loading it calls two function – one displaying the time of the server and the time of the client each second , and the other one assigning a json string received from the backend consisting of all Appointments scheduled for the relay along with their UUID code. There is a form in which one of those UUID codes can be copied and submitted to the backend which will cancell that Appointment, there are also a few redirecting buttons and again there is a button for deleting all Appointments for the relay.

Show_relay2.tpl is virtually the same but for the other relay.

The home_page.tpl again has a function called and repeated every second after loading the template. That reoccurring function updates the server timer (long polling is used –here and in every other template displaying server time) and the client time, and checks the state of the relays to see if the image for them should be changed. The initState(relay, data) function changes the image and description of ‘relay’ according to wheter data is ‘off’ or ‘on’ . getState() function polls the backend for the state of both relays and calls initState for both of them. change_state(relay) is called when a button for one of the relays is pressed, it sends a request to the backend to change the state of the relay and then calls initState to change the image and description of the relay in the webpage on the client side.The stop() function is called when the shut_down button is pressed. It refreshes the images of the buttons and calls a function from the backend which returns the initial state of the pins on the board of the raspberry and then shuts down the server (the purpose of this button being served is to give the users a sort of emergency switch shut down in case of a short circuit or other unexpected emergency event). Like the rest of the templates (views) this one also has a couple of redirect buttons.