Web-based RESTful Communications

Contents

Overview and Objectives

This project is about the basics of web servers and clients and will give you a chance to learn the principles of RESTful communication, which is widely used in many contexts including cloud and edge computing. The Internet and World Wide Web are at the centre of modern communications. **Representational state transfer** (**REST**) is an abstraction of the World Wide Web architecture. It is an architectural style applied to the development of web services. RESTful services have been increasingly popular on the web due to their simplicity.

The goal of the project is to develop a client-server Machine-to-Machine communication architecture. This architecture can be used for a variety of purposes, which explains its popularity and widespread use. A specific option is provided below but you can easily develop your own scenario and have more fun in your project!

Project Scoping

This is an open-ended project and scoping the project is your responsibility. In other words, this is your project, and you decide how to achieve the objectives described above.

Please keep in mind that you have only 5 weeks. Therefore, there is no time to waste, and you need to be very realistic about what you can achieve in 5 weeks. There is no need for stress, nobody is expecting you to deliver a capstone project in 5 weeks!

You will be well-supported during your project. Firstly, **you have to talk to the lecturer in the beginning about your project plan and scope and get feedback**. This is a requirement and is not optional! Secondly, your demonstrator will be there to help as much as possible. Thirdly, keep your lecturer in the loop regarding how the project is progressing and get feedback to ensure that you are on the right track.

Directions and Resources

The project focuses on web programming and communications at the application layer. This is a very broad and deep topic. Fortunately, powerful Python libraries empower you to do a lot of without much effort.

As a resource, the provided "Web Servers and Clients" practice mini workshop is entirely optional but you may find it useful as part of developing your project. The practice workshop itself will NOT be directly part of your project; it is something you can do [optionally] before your project to gain relevant information. Here are other online sources that may be useful.

Generic Information

- https://www.codecademy.com/article/what-is-rest
- https://blog.hubspot.com/website/what-is-rest-api
- https://www.techtarget.com/searchapparchitecture/tip/The-deep-rooted-relationship-between-REST-and-microservices
- https://www.mygreatlearning.com/blog/what-is-rest-api-in-python/
- https://realpython.com/api-integration-in-python/
- https://requests.readthedocs.io/en/latest/

You can come up with tens of different scenarios using this communications architecture. Although your project is only limited by your creativity, here is one possible application as an example. **You are expected to create a variation even if you use this specific scenario**, so please take the description below only as a starting point!

It is a good idea to use Wireshark for analysing the communication between your local clients and servers, which would enrich your project and report!

Example Scenario: Demand Management and Smart Grids

The electricity grid that powers our homes, businesses and industry is expected to work at all times including summer peaks in Australia when the demand for electricity skyrockets due to air conditioning. The whole power grid (poles, wires, transformers, generators, etc.) has to be scaled up to meet this peak demand to prevent blackouts. If you have not already noticed from your electricity bill, building, and maintaining this infrastructure are very costly. Even worse, most of the time the electricity demand is far below the summer peaks. In other words, the entire infrastructure remains idle for most of the time, and we pay dearly for something we use only a few days per year.

Big industrial sites such as factories have been making agreements with power companies to decrease their demand at peak times (e.g. stopping production for a few hours) in return for a nice reduction in their electricity prices. This is called demand management or response. The practice has not been as prevalent in commercial sectors (e.g. shopping centres) and almost no residential demand response has been implemented in Australia, yet.

The funny thing is residential demand response is very feasible from a communication and computing technology point of view. A demand aggregator or power company can agree with individual houses to implement automated demand management, thus saving power bills while causing minimum inconvenience thanks to smart technologies. Such an improvement naturally falls under the umbrella buzzword "smart grid".

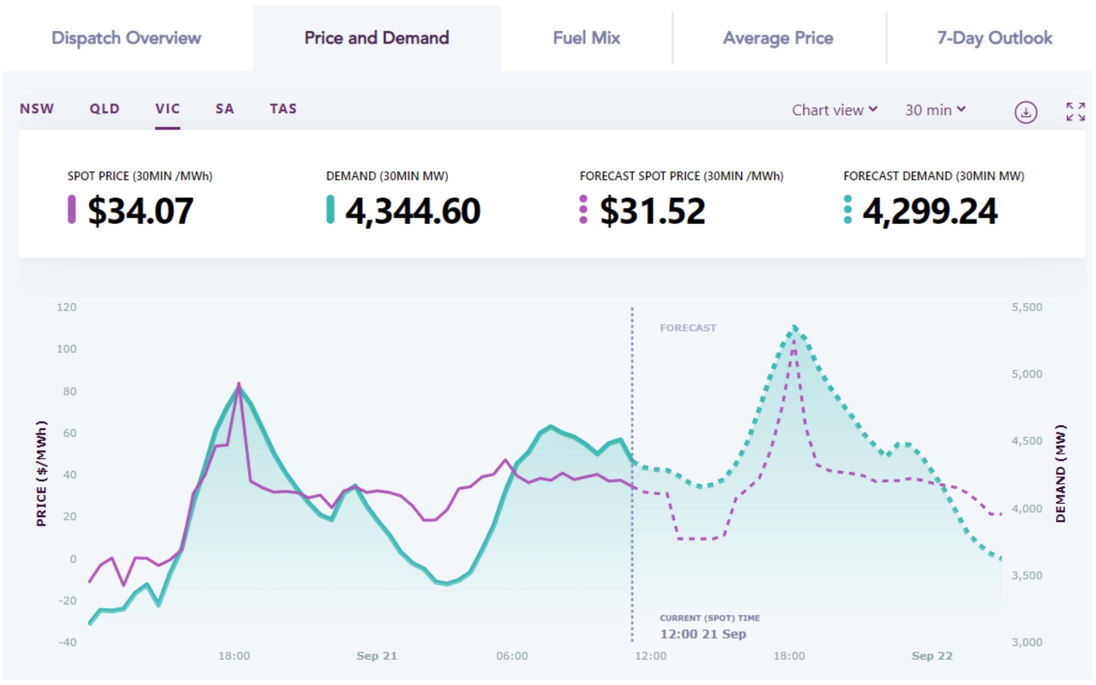
For example, consider a programmable thermostat, such as the one in Figure 1, controlling the heating/cooling in a house (e.g. it may have a minicomputer built-in similar to RaspberryPi which is capable of running Linux and Python). The thermostat adjusts the electricity consumption by changing the set-point of the heating/cooling system based on various factors including the electricity prices which generally peak when the demand goes high.

Figure 1: Nest thermostat

In Australia, the Australian Energy Market Operator (AEMO, http://www.aemo.com.au/) openly publishes <u>wholesale</u> electricity prices (search keywords: AEMO Price and Demand Graphs).
See also https://www.aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data

These can be used as a proxy for the demand as shown in Figure 2.

The main idea is to make the demand responsive to price peaks by creating a feedback loop using (wholesale) prices. In classical power systems (and currently) demand is a given and untouchable. However, if the peak demand can be reduced somehow using modern ICT, this would reduce price peaks, and unnecessary investments, and improve network efficiency!



SPOT PRICE (30MIN /MWh) $34.07

DEMAND (30MIN MW) 4,344.60

FORECAST SPOT PRICE (30MIN /MWh) $31.52

FORECAST DEMAND (30MIN MW) 4,299.24

Figure 2: Wholesale electricity prices and total demand in VIC, Australia, on 21-09-2020, published by AEMO. https://aemo.com.au/Energy-systems/Electricity/National-Electricity-Market-NEM/Data-NEM/Data-Dashboard-NEM

***Note that**, there is much more to a smart grid than demand management. The topic is very broad and deep, and outside the scope of our subject other than its communication aspects.*

As part of your project, you can design and implement a multiple-client multi-server M2M communication system between thermostats (clients) controlling the temperature in houses and centralised servers at the premises of a utility company which gives pricing information to the client as guidance.

As one example the client may only have access to local temperature data (simulate real-time availability, do not use future values) and the server may only have access to pricing data (again simulate real-time availability, do not use future values). The server does not know what the temperature is at the house of the client and the client does not have direct access to wholesale electricity prices. You can consider multi-client multi-server versions of this. For example, one server provides weather data, another one market data and clients get information from both and combine them.

The client can for example:

- Be at the centre of the simulation and keep the state (e.g. time) since the server is stateless!
- Read current outside temperature data from a CSV file or (SQL) database (which may be populated by sensor measurement in real life, for simplicity you can fill the database for a fixed duration of time with data beforehand)
- Request price data from a server.
- Request weather forecasts from another server.
- Decide on a temperature *Setpoint* based on outside temperature and wholesale electricity prices. Feel free to use your imagination on this part, e.g. the setpoint could be a function of humidity, pressure, or similar parameters and adjusted for the comfort of house occupants.

The server can for example:

- Read wholesale price data from the given CSV file or (SQL) database.
- Serve this data to the client. JSON is often used to format data.
- Collect information from the client.
- Serve ads depending on temperature (tea for cold days, cold drinks for hot days!)

Reliable Communications

As next step, you should implement a reliable communication scheme between your clients and servers at application level by implementing ARQ with (n)ack assuming the server-client communication is imperfect. Test your implementation by intentionally introducing lost or distorted packets.

Note: The temperature file was downloaded from the Australian Bureau of Meteorology:
http://www.bom.gov.au/vic/observations/melbourne.shtml
http://www.bom.gov.au/products/IDV60901/IDV60901.94866.shtml#other_formats
(search keywords: BOM Latest Weather Observations for the Melbourne Area)