Design

Note that although there is machine learning algorithm in my requirement, I am only addressing the requirements for 1a in this stage.

Application design

Design pattern

Before I can start with the design, I must first explore what a design pattern is and how they might be useful in the design of this API and in the projects I may work on in the future.

A design pattern is defined as “general, reusable solution to a commonly occurring problem within a given context in software design”. In our case, the designing of a API is a commonly occurring problem, and REST is the one of design pattern that is made to solve this problem.

REST stands for “Representational state transfer”. According to IBM, it is “a set of rules that define how applications or devices can connect to and communicate with each other” which consist of 5 principles: (IBM, n.d.) (Fielding, 2000)

1. Uniform interface
    - All request for the same resource should look uniform
    - The same piece of data should belong to only 1 URL
2. Client-server decoupling
    - Client and server application should be independent of each other
    - The client should only knows the URI of the resource and the server should not modify the client application
3. Statelessness
    - Server application aren’t allowed to store any data related to a client request
    - Each request must contain all information necessary
4. Catchability
    - A response should be labelled implicitly or explicitly as cacheable or non-cacheable
    - Client have the right to reuse the cache of response data later.
5. Layered system architecture
    - The communication between the client and server can go though a few layers.
    - Neither should be able to tell if it is communicating with an intermediate.
6. Code on demand
    - Response can contain executable code in certain cases.

Comparing these standards to my requirement, we can see that it is suitable for my API and that it provided guidance on how to implement some of the features.

In req 2.27 and 6.2.2, I wanted response to be accessible by the client again later, this perfectly matched with the catchability a RESTful API. There is also requirement that I might forego or modify based on REST. For example, to keep my API stateless, there should not be data related to a client request stored on server. However, in my requirements, item 7.2 suggested that data from a client request should be stored. This violates statelessness principle, it might not be a suitable feature if I am to adhere to the REST principle.

Design on the architecture

This section is mostly based on teaching material and section 5.2 REST Architectural Elements in field's dissertation. The terminology I am using, such as resource and representations are as defined in the dissertation.

Upon reading the dissertation, it became clear that I must understand what role my API should play in a wider context, before I can proceed with the design. In it's simplest form, I wanted the API to be an interface where users can access data stored on a server. As suggested in User story 9, I also envision my API to be used in other developer's code or apps. Therefore, the data format supplied to my client must be supported and can be rendered in external apps.

An example I have been looking at in the design stage is Google Drive, where authorised users can upload data onto a server and access them though an URL (req 1, 2, 5, 7) . In addition, I wanted my API to return some computed value if requested (req 4.2, 4.3). And perform computation based on set algorithm(req 4.4,4.5,4.6).

Resources, representation and data format

As defined by my requirement, the essential function of my API is to provide data entry as requested to my client (req 2). The client should have the ability to request all data associated to a specific date or location. Therefore the resources that can be requested as such are : "entryID", "temperature", "salinity", "density_anomaly", "pressure", "date", "longitude", "Latitude", "bottom_depth".

The representation of these resources will be numerical, most format are capable of representing numerical data. XML and JSON are the 2 most common format. JSON contain a key and value component and is perfect for application like this. As for XML (Extensible Markup Language), data are stored within "node", which is like a bracket for values. It will also be able to handle results we wanted to output. However, due to the existence of node tags, the text length is much greater than JSON. This might cause the file size outputted to be larger.

Considering that we would want the clients to be able to download resources, file size is a concern and JSON might be a better option. It would be ideal if the client can choose from a range of data format, but if there needs to be an default, JSON will be a good option.

Considering our "should have" features (req 4.2), graphs would also be an resource. The representation supplied to client can be in various format. However, the format that supports such output is much less common. We would be looking at mainly GraphSON and GraphML which are based on the popular JSON and SML format. Like JSON and XML, GraphSON tends to be less wordy then GraphML and is supported better with other programming languages. There for it would be more suitable than GraphML if we expect the output to be supplied to another application.

Resource Identifier

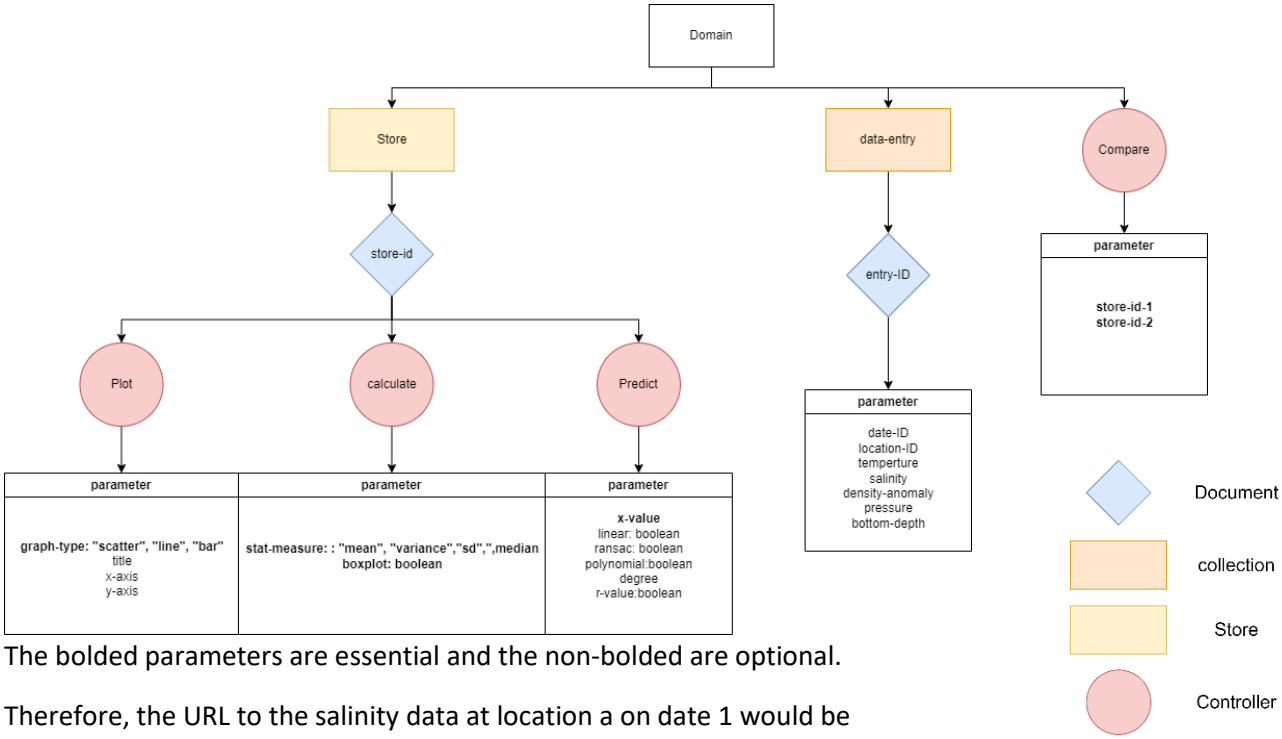
Each resource in our web API can be located with a URL (Uniform Resource Identifier). The URL will contain parameter so it act as a request to our API.

Following the guidance on REST API tutorial, consistency is key in creating URLs. First we will look at the archtype defined in the resources and how that fit into our API. The website defined archtype into documents, collection, store and controller.

In our API, document would be a single data entry. For example, in req 2.2, data can be requested by specifying the location and date ID. The single data entry requested would be a document. Collection would be a directory of resource, in our API, this would be the resource identified as in

req 2.2.4.1, the collection of data of a specific location. A store is a client-managed list of resource. This would be a archetype used if req 2.2.7 is to be fulfilled so that clients can save a customised set of data. At last controllers are defined as actions or procedures. In my API, plotting graph, as specified in req 4.2 could be under such archetype.

I have created a diagram to illustrate my hierarchy



The bolded parameters are essential and the non-bolded are optional.

Therefore, the URL to the salinity data at location a on date 1 would be

http://domain.com/data-entry?location-id=a&date-id=1&type=salinity

where “?location-ID=a&date-ID=1&” can be replaced by “/entry-id?”

And if the client were to request a prediction, an example URL would be

http://domain.com/store/store-id/prediction?x-value=20&linear=true

To compare 2 stored dataset, the url would be

http://domain.com/compare?store-id-1=id1&store-id-2=id2

Http Method

There is 4 verb available with a HTTP protocol. They can be used on each url to achieve different effect. In my design, there is 6 kind of url, 2 of which is for documents and 4 of which is for controller actions.

	Get	Post	Put	Delete
…/data-entry/entry-id	×	(×)	×	
…/store/store-id	×	×	×	×
…/store/store-id/plot	×		×	
…/store/store-id/calculate	×		×	
…/store/store-id/predict	×		×	
…/compare	×		×	

The method Get is applicable to all the url, it allows data to be accessed and for controllers, it allow the current state to be read. The verb Post is used to create new entry, therefore it is only available on the user-created stored. In some of the could have requirement, users can be allowed to add new data entry to the database, therefore, it is possible for users to use the Post method on the data-entry URLs.

As for Put method, it is used to update an entry, therefore, it is used in the controller action urls. This is because parameters need to be sent to the server for these actions to work. This is also applicable to the data URLs so that they can be added to user's store. The verb delete have the most limited usage. It is only available in the store URLs, this is because we do not want users to be able to delete record and there is no use to delete anything for the controllers.

Reference list

Fielding, R. (2000). *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*. [online] Uci.edu. Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

Gupta, L. (2018). *REST API - URL Naming Conventions*. [online] REST API Tutorial. Available at: https://restfulapi.net/resource-naming/ [Accessed 30 Dec. 2022].

IBM (n.d.). *What is a REST API? | IBM*. [online] www.ibm.com. Available at: https://www.ibm.com/topics/rest-apis.

Spotify (n.d.). *Web API Reference | Spotify for Developers*. [online] developer.spotify.com. Available at: https://developer.spotify.com/documentation/web-api/reference/#/.