# Coding Exercise No 1

## Misc assumptions / simplifications

* No permission / User authN, authZ => task / tasks list owner is the user who creates the object
* In terms of update, only replace or append text fields are considered to avoid complex merging issues (only incremental change considered in changeset). Replace only available to the owner
* Merging service should be available on server side to manage and raise conflict

## APIs design

### 2.1 CRUD tasks / tasks list API on client side

See REST API specs:

<http://127.0.0.1:8000/docs#/>

### 2.2 Tasks list sync API on server side

API specs between client and server:

1. Subscribe to tasks lists
2. Push incremental updates from server to client (for subscribed tasks lists)
3. Push incremental update from client when online
4. Initialize / refresh tasks lists from server to client (on startup or following an offline period)

REST API can be used for 1, 3 and 4 with an openapi callback approach for 2

A pub-sub oriented protocol can also be leveraged such as MQTT

## Data model

A close up of a map

Description automatically generated

## Synchronization design

Event based synchronization, basic principles:

* When client online (connected to the server), data modification / events are
  + submitted to the server
  + validated and merge on a centralized repo
  + based on a specific repo version, incremented on each change
  + sent back to the client for local update (local client cache updated and persisted)
  + all data modifications related to the subscribed tasks list are published to clients and a client local cache is continuously updated and persisted
* When client offline, data modification / events are:
  + buffered in a local queue
  + The client local cache is:
    - still continuously updated from the changes performed locally applied on top of the last version tasks lists version received from the server
    - the cache is not persisted as long as there is no new version received from the server
* When the client is back online:
  + Buffered changes are sent to the server for integration
  + Once received back by the client, they are applied on an updated tasks list version of the tasks lists (following the same integration order as the one followed on the server side for synchronization)

Merge conflicts might be raised on server side on changes list integration so there will be a need to manage those conflict by the user eventually when required and to ensure no changes list is lost.

The diagrams below give additional details to describe the eventual consistency across multiple clients

A close up of a map

Description automatically generated

## Client development

See <https://github.com/Nicomel/mutl>

## AWS deployment model for the service

![A screenshot of a cell phone

Description automatically generated]()