# RESTFUL API Design

Bar Inventory

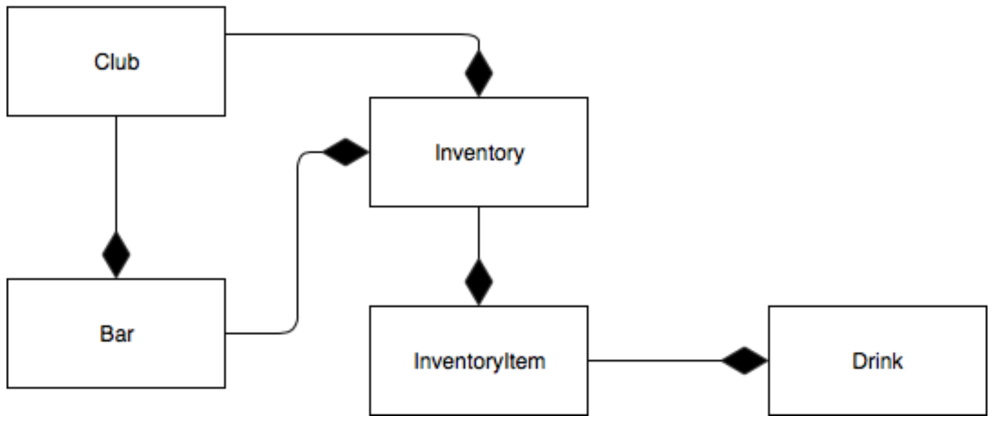
## User Stories

When thinking of user stories to be used in this assignment in terms of use of an API I found that some were included in the last assignment for the purposes of scalability and portability of objects. These user stories are below:

- User is able to access the data from anywhere with a internet connection.
- User does not have to store any information locally as it is stored on a database and retrieved easily.
- Different users can access data and can upload information to the server via the API.

## Endpoints

Here is the entity relationships in the classes.



All classes are implemented with the use of composition. Bars and Clubs have inventory which is used to track the amounts of drink at the specific locations for stocking purposes.

A Club has 0 to M Bars. Both clubs and bars have an inventory. Inventory is made up of inventory items which hold a drink type and the current and required stock for the given location it is under.

When designing the endpoints I thought mainly how to store data properly on the server. When keeping in mind the database I reduced what was 5 classes down to 3 tables in the database.

Here are the main endpoints that were made for the API:

**/locations**
**/drinks**
**/inventory**

---

## Thoughts behind the endpoints

The Drink class has its own table as it only contains information about the drinks. Further analyzing the classes I found it was impractical to have a table that stores clubs and another that stores bars. Instead I decided to view both as locations. This was what inspired me to create the endpoint for locations which stores information on the location and whether or not they are the club or not.

Currently in the system there is only the ability to have 1 club but later on this can be changed for other clubs to be stored on the same tables.

The last endpoint was inspired through the condensing of the Inventory class and InventoryItem class. The inventory class did not have any information to be stored on the database as it is just a way to organize the data in an Object Oriented format within the application. The inventory endpoint is used to return a mapping of drinks to locations along with slight information of how much the stock is at for said drinks.

Each endpoint makes use of GET POST, DELETE and PUT protocols. Although the GET and POST protocols do not require an ID the other two protocols do to be able to update a specific item in the database.

## CRUD

| Endpoint | GET | POST | PUT | DELETE |
|---|---|---|---|---|
| **/drink** | Returns a list of drinks | Inserts a new drink in the database and returns a drink object with the newly formed ID | N/A | N/A |
| **/drink/<id>** | Returns drink with the specified id | N/A | Returns updated drink with the specified id | Returns empty string on successful deletion |
| **/location** | Returns a list of locations | Inserts a new location in the database and returns a location object with the newly formed ID | N/A | N/A |
| **/location/<id>** | Returns location with the specified id | N/A | Returns updated location with the specified id | Returns empty string on successful deletion |
| **/inventory** | Returns a list of inventory | Inserts a new inventory item in the database and returns an inventory object with the newly formed ID | N/A | N/A |
| **/inventory/<id>** | Returns inventory item with the specified id | N/A | Returns updated inventory item with the specified id | Returns empty string on successful deletion |
| **/inventory/remove** | N/A | N/A | Takes an array of inventory items to subtract specific values from the current stock. Returns the updated items | N/A |
| **/inventory/add** | N/A | N/A | Takes an array of inventory items to add specific values to the current stock. Returns the updated items | N/A |