Major Project Summary Document- Team 4

Purpose of This Document

This document provides a summary of the documentation of the entire final project. To view sprint by sprint details, use the Major Project Comprehensive Report - Team 4 document for a more in depth breakdown of this final project.

Problem Statement

Clubbing in times of COVID! YelpHelp is a company that specializes in recommending bars based on quality and occupancy. YelpHelp approaches you to help design a system that will allow them to recommend a bar that is both cool (to be seen!), as well as secure (not above a given occupancy or density factor). Obviously, everyone can't go to the highest-rated bar, as it would be unsafe. And no one wants to go to the safest bar, i.e, which by definition is the bar where no one else goes! In other words, if there are too many people it's unsafe; if there are too few people at the bar, it's uncool. YelpHelp!

Dependability Implementation: **Security**

Project Management Concept: **Team Work**

Advanced SWE Topic Implementation: **Software Reuse**

Requirements Engineering

Non-Functional Requirements

ID	*Requirement and Description*
NF1	Capability to handle large number of requests (300+)
NF2	Capacity to store large amounts of bar data (100+)
NF3	User friendly I/O interface to interact with data
NF4	Insert data into database within 1 seconds of request
NF5	Remove data from database within 1 seconds of request
NF6	Get list of relevant data from database within 1 second of request
NF7	Allow only authenticated users the ability to add and remove bars
NF8	Recommend a bar within 1 second of request
NF9	Update current traffic of bar within 1 seconds of recommending a bar to a user

Functional Requirements

ID	*Requirement and Description*	*Backlog ID*
F1	Handle user request for adding a bar to the database	
F2	Handle user request for removing a bar from the database	
F3	Handle user request to query bar database and return all relevant data	
F4	Return relevant confirmation/error for each user request	
F5	Implement authenticated users through a log-in system	
F6	Store sensitive data in a properly secured format	
F7	Given user input recommend a bar for them to go to	B2
F8	Keep track of what bars each user has been recommended to	B3
F9	Update wowFactor or capacity of a bar entry	B4

Assumptions

Add Bar

- Traffic is not greater than capacity
- The primary keys for bars is names and address
- The primary keys cannot be updated once inserted
- Bars can only be added using this endpoint, they cannot be deleted

Delete Bar

- The primary keys (name and address) must be provided to delete the bar

Get Bar

- Bars are returned as an array of JSON objects

Bar Selection

- Once a bar has been recommended, the individual's information and bar information is stored
- The primary keys of the client are name and phone number
- The users table is cleared every night
- Only one request is allowed per user per night
- Users must provide the following input: name, phone number, minimum wow factor, maximum capacity, preference
- The user must specify a preference (either capacity or wow factor). In the case of a tie, whichever bar has a higher value of the preference attribute the person selects, that bar will be recommended

View Users

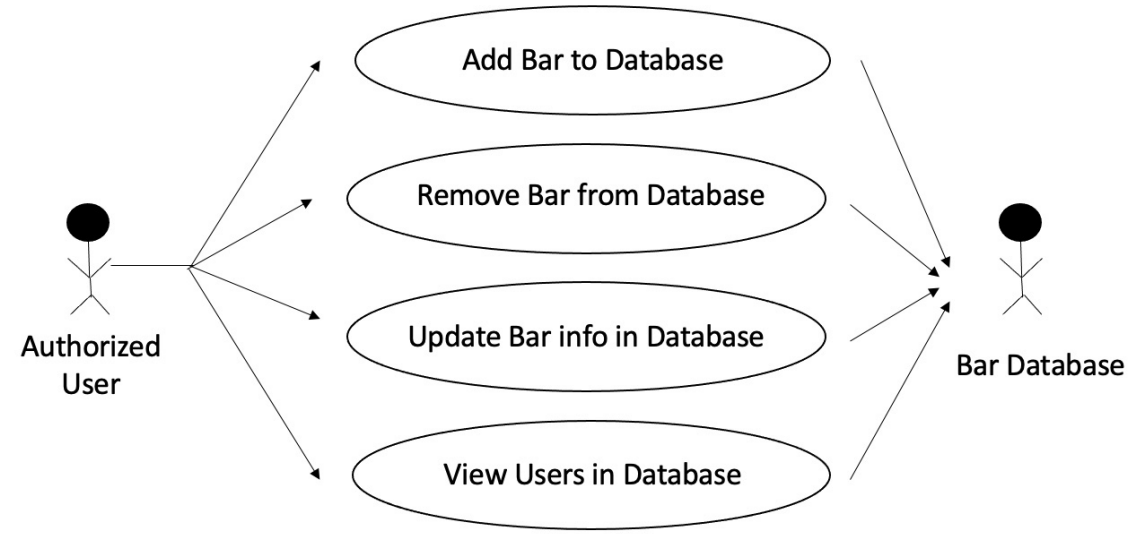
- Users are returned as an array of JSON objects

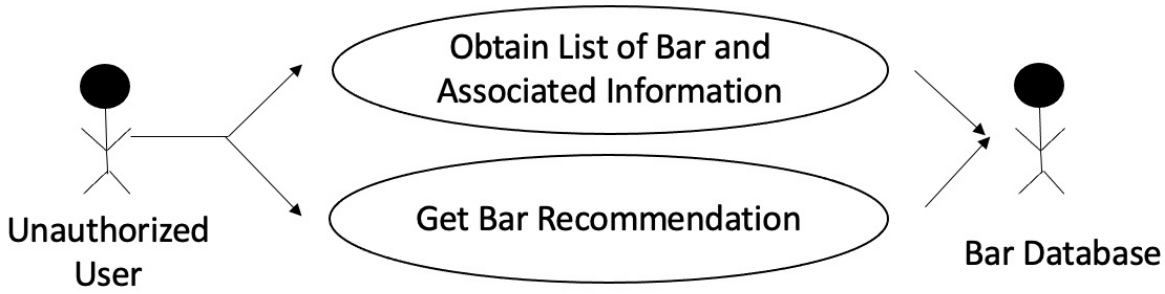
Update Bar Data

- The primary keys (bar name and address) cannot be updated once entered
- The current traffic will never be greater than the capacity

System Modeling

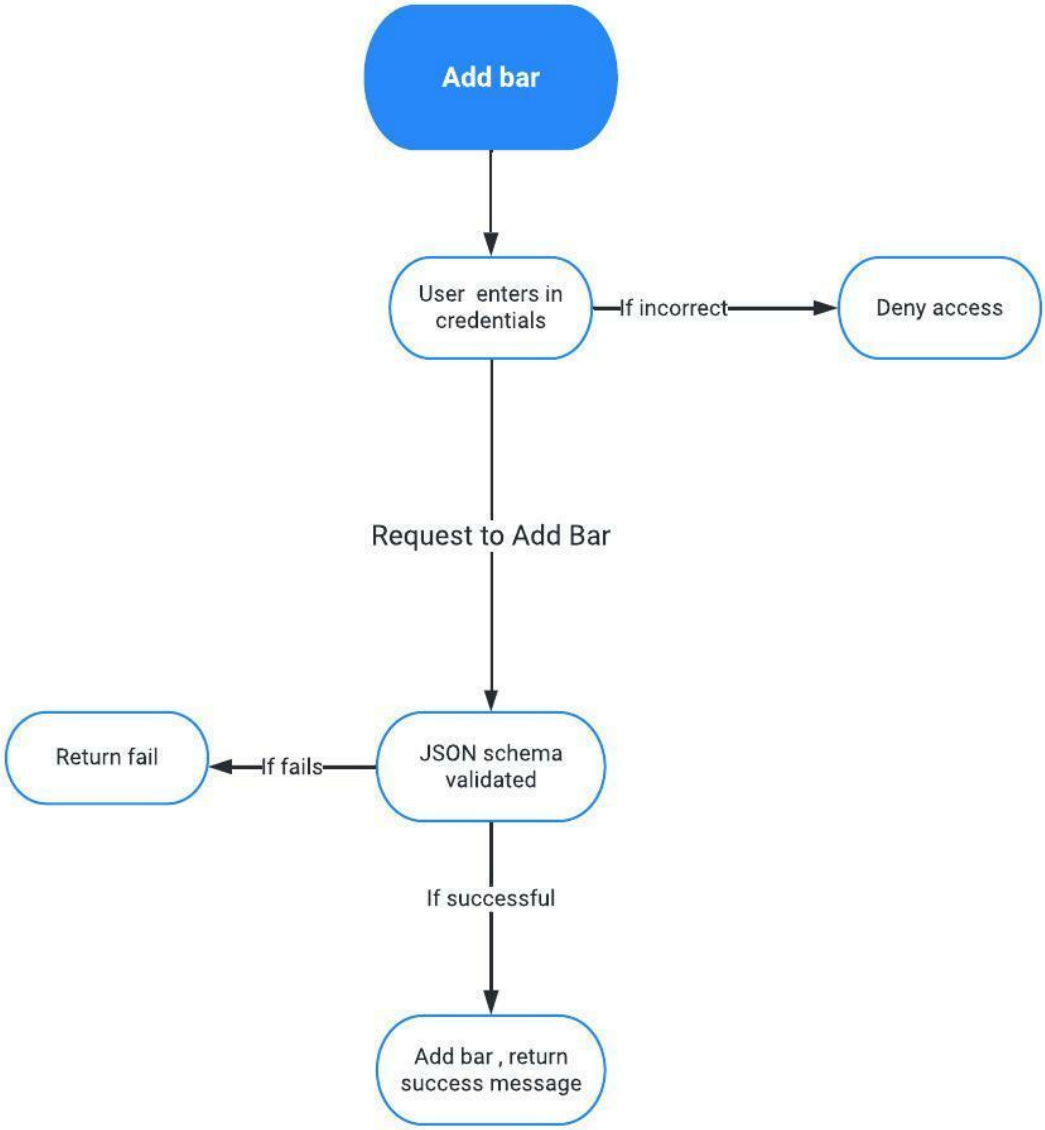
Use Cases



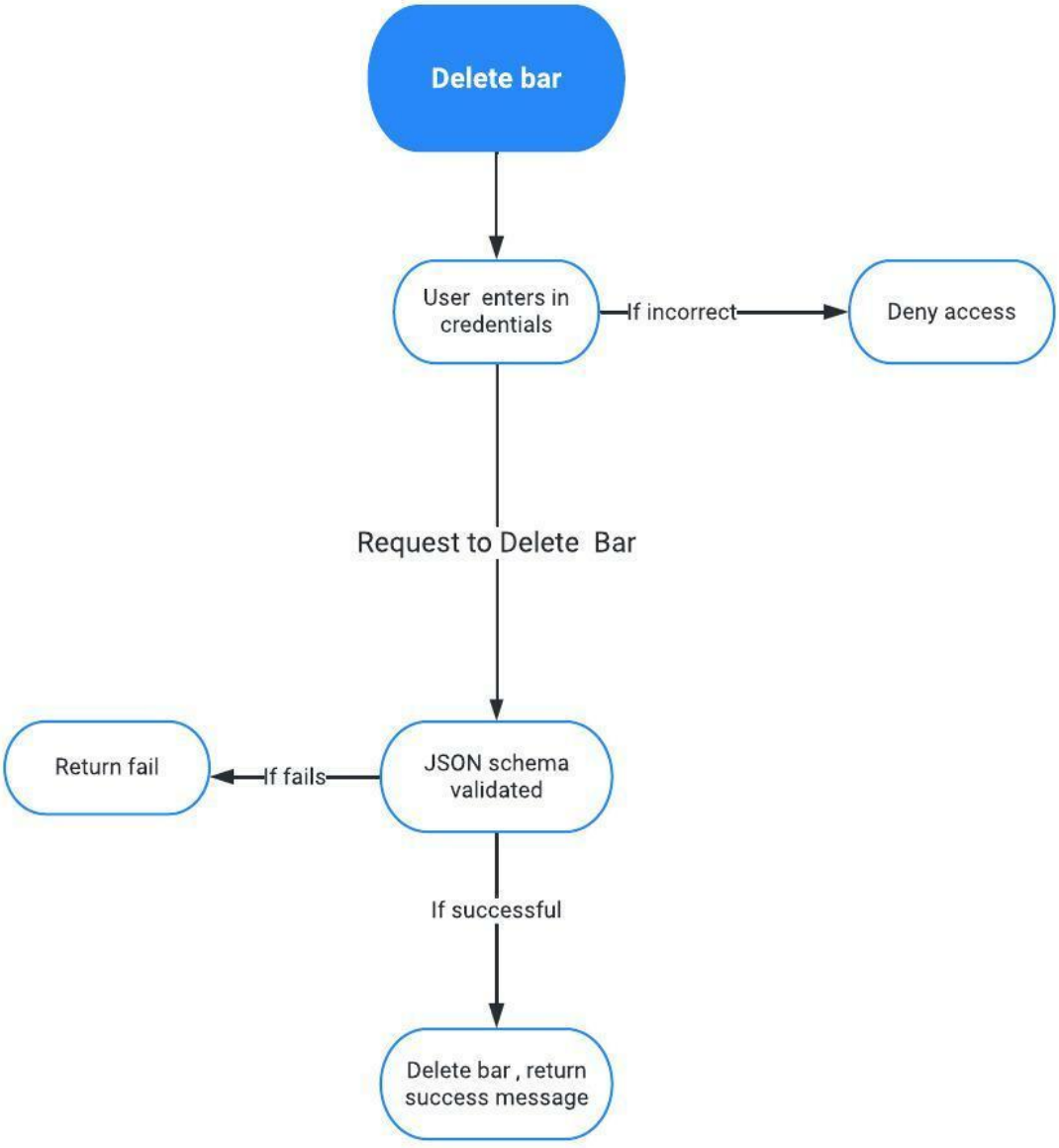


Conceptual Model

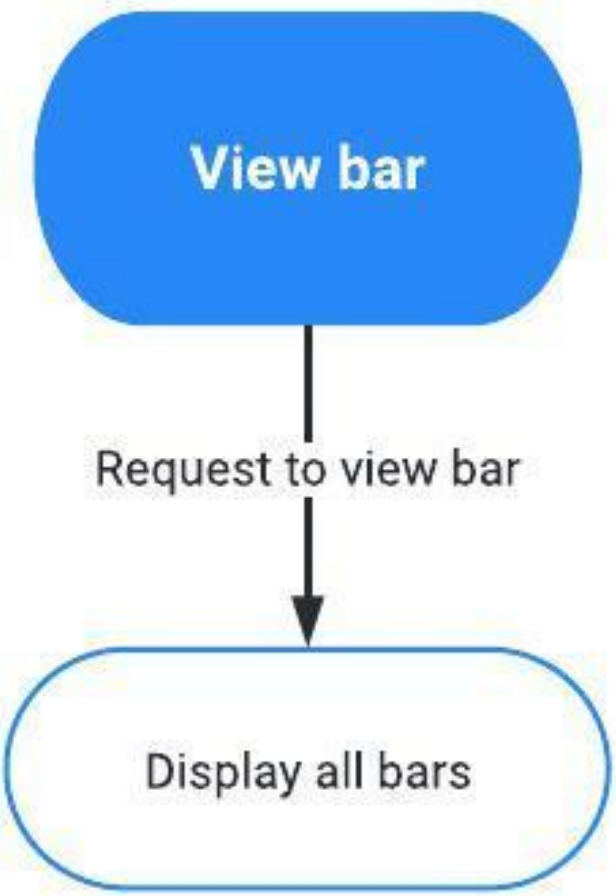
Add Bar



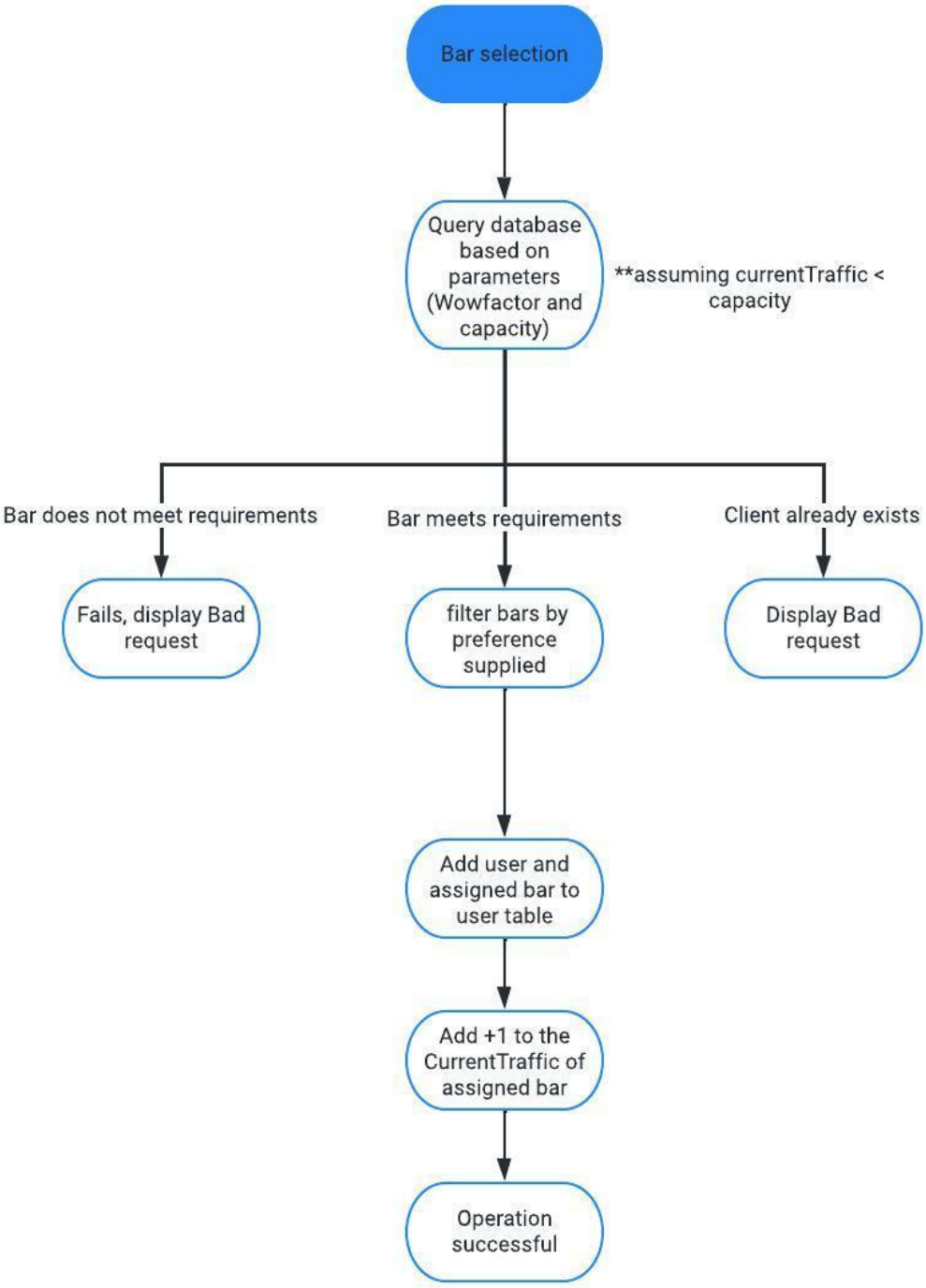
Delete Bar


Delete bar
User enters in credentials
If incorrect
Deny access
Request to Delete Bar
Return fail
If fails
JSON schema validated
If successful
Delete bar , return success message

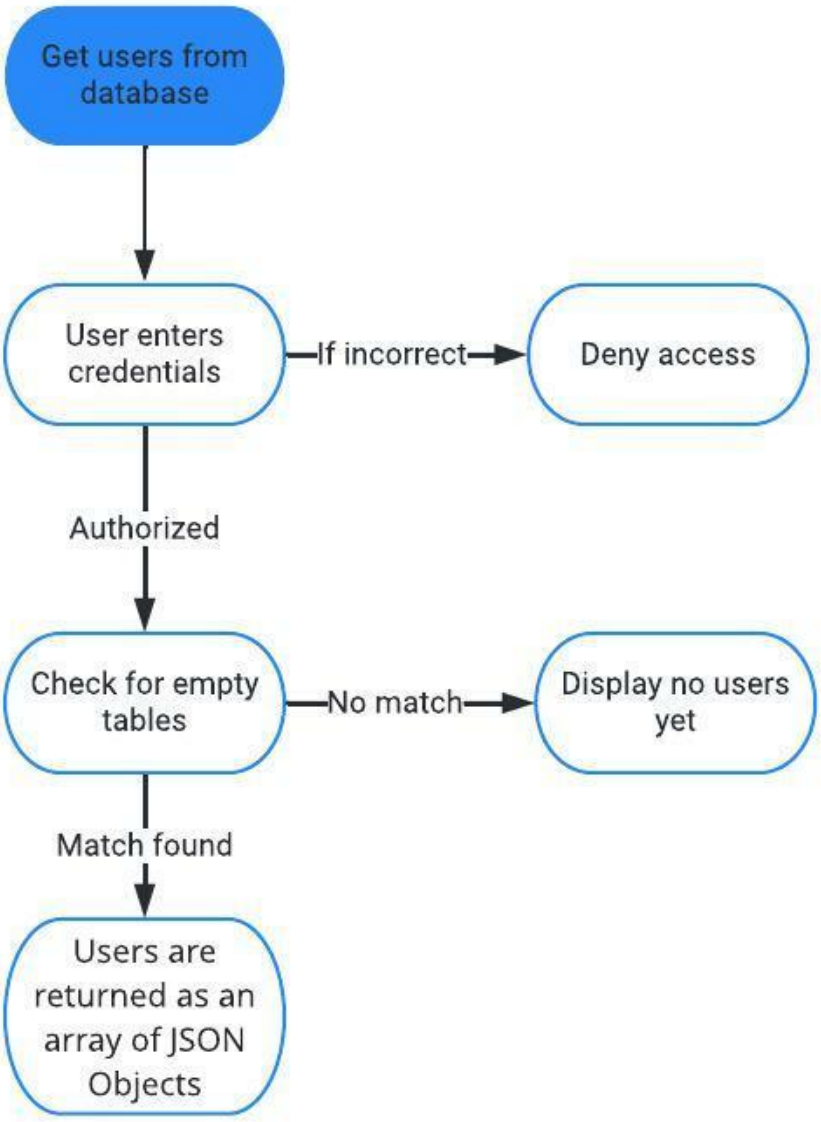
Get Bar



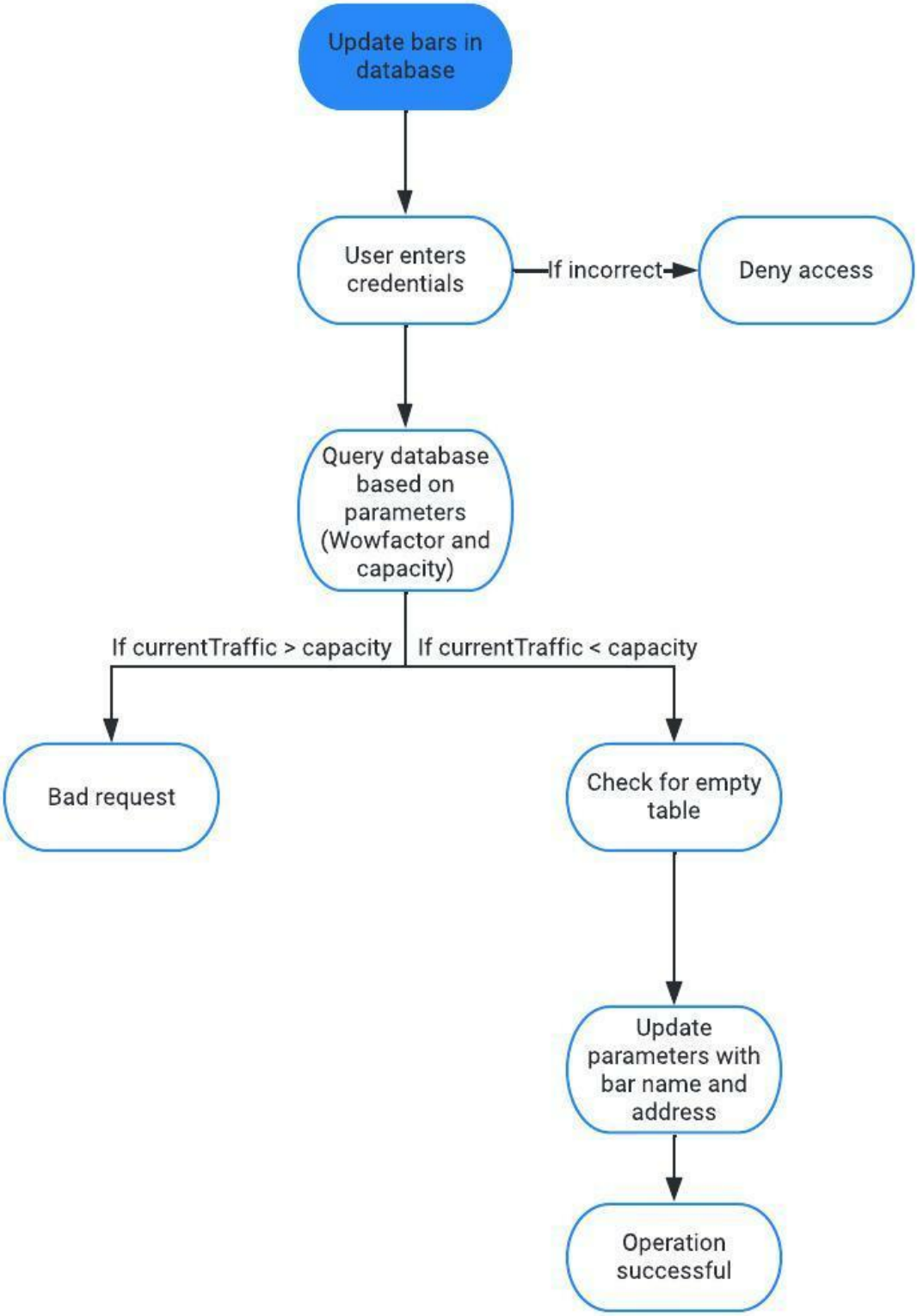
Bar Selection



View Users

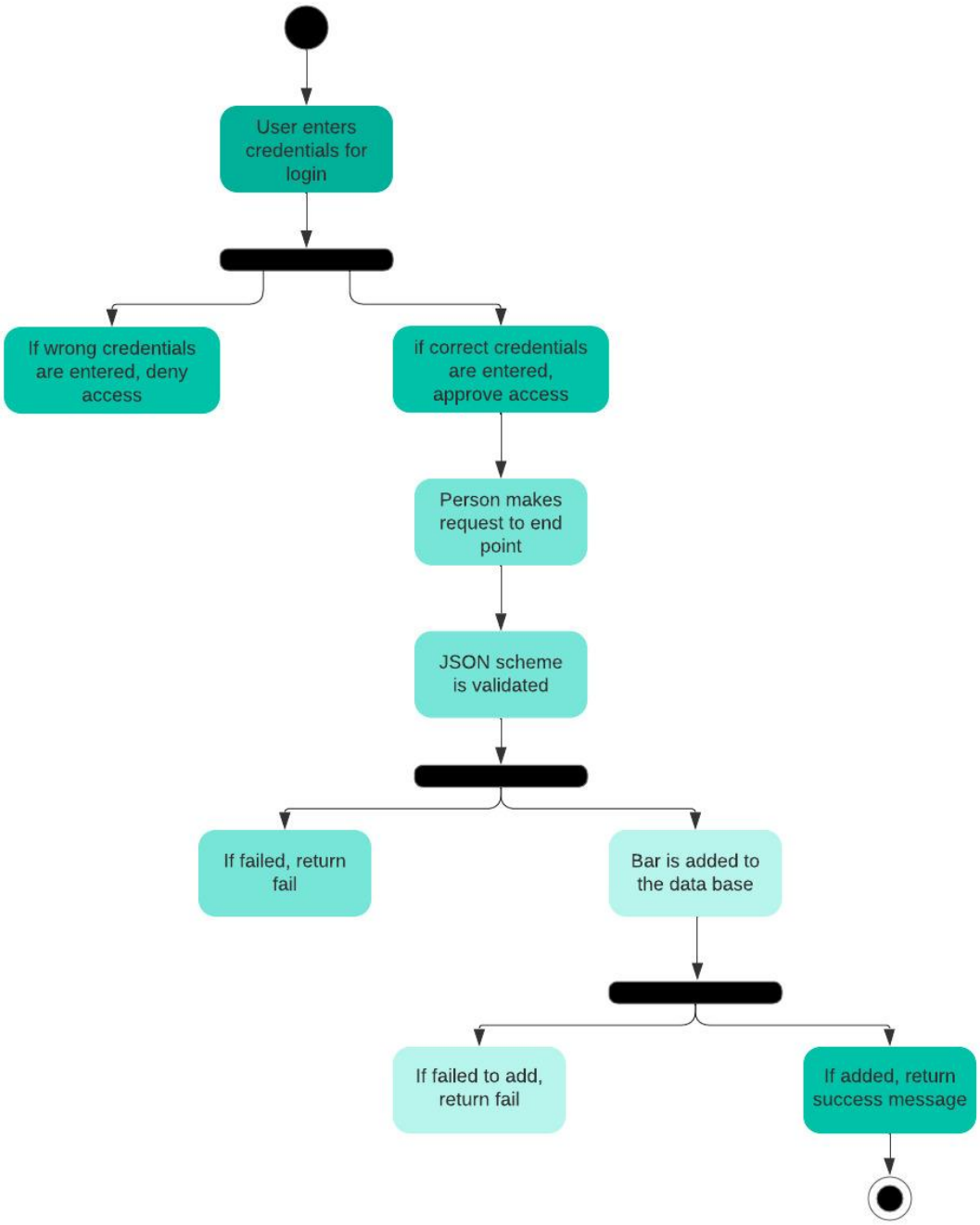

Get users from database
User enters credentials
If incorrect
Deny access
Authorized
Check for empty tables
No match
Display no users yet
Match found
Users are returned as an array of JSON Objects

Update Bar Data

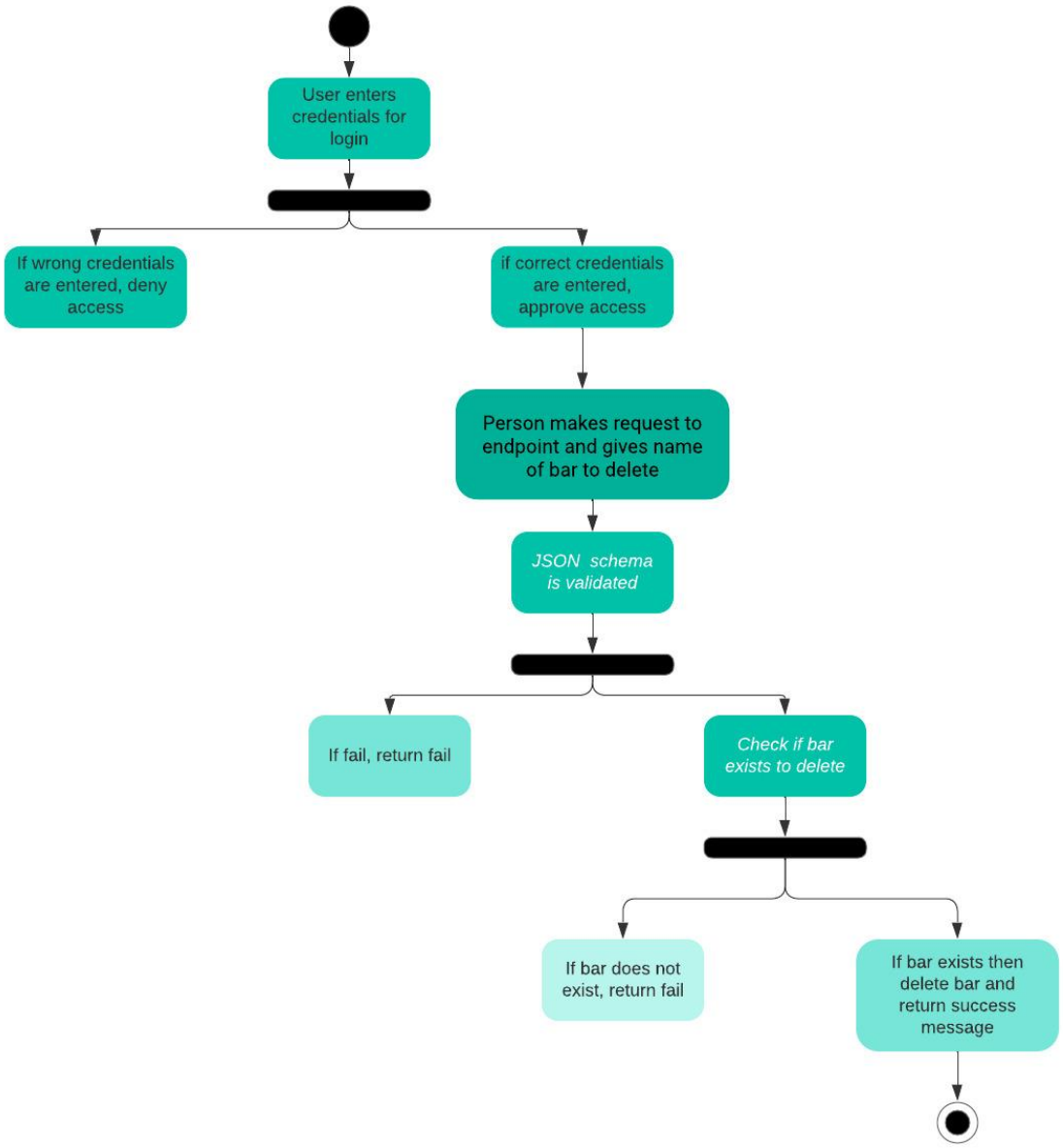


Activity Diagram (Logical View)

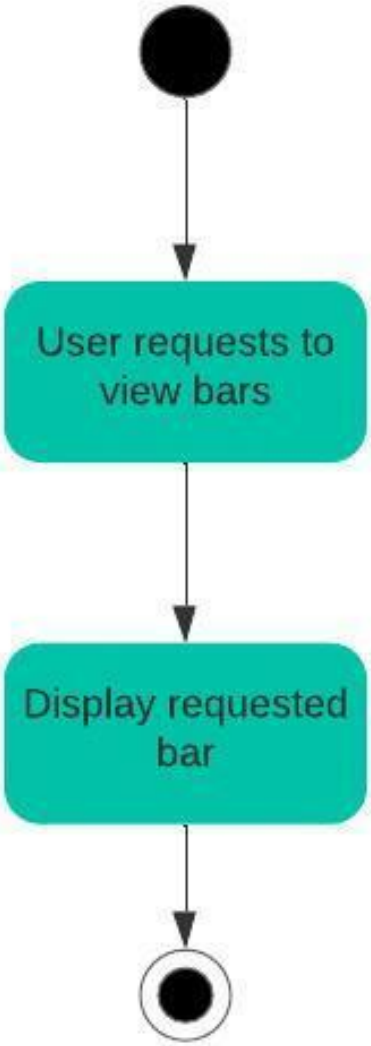
Add Bar



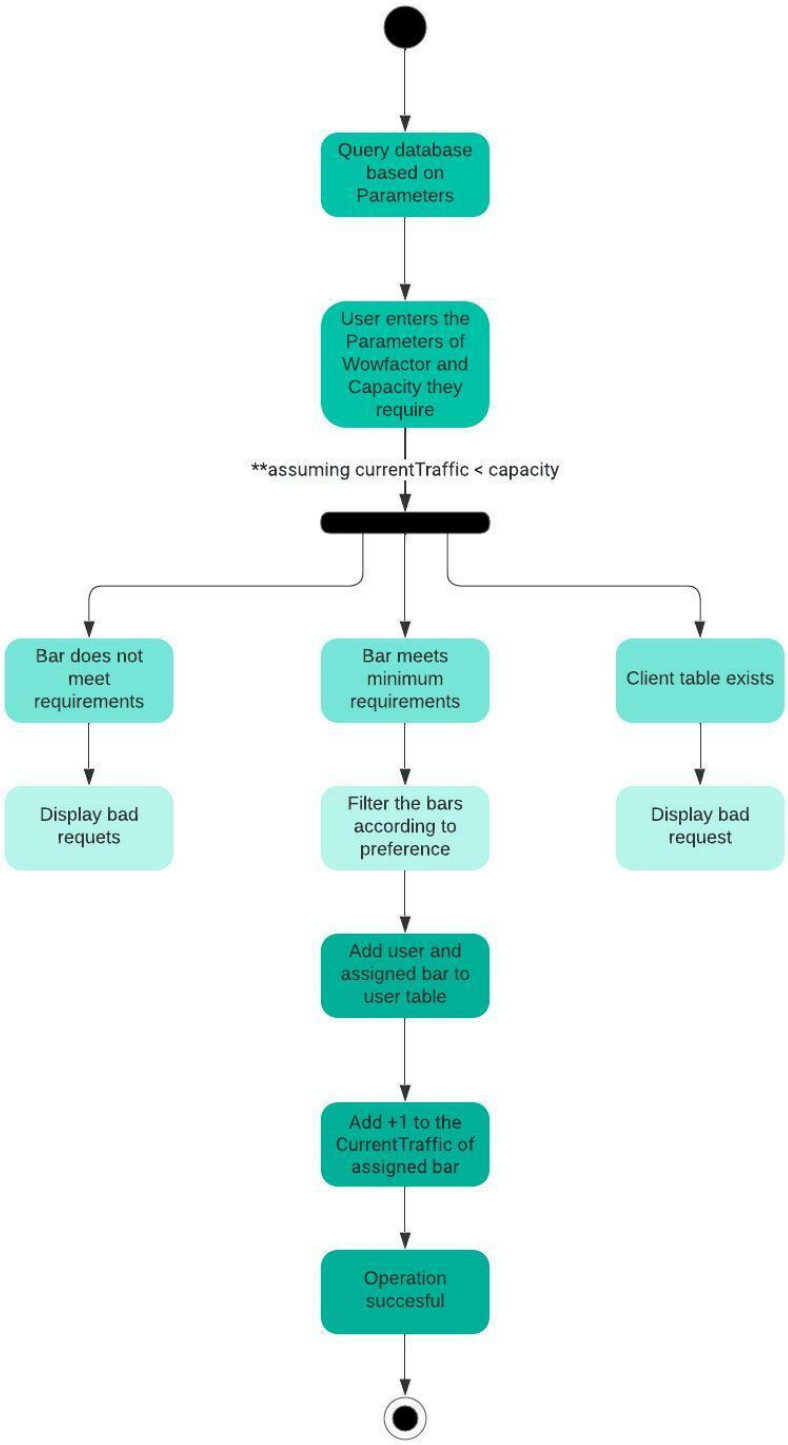
Delete Bar


User enters credentials for login
If wrong credentials are entered, deny access
if correct credentials are entered, approve access
Person makes request to endpoint and gives name of bar to delete
JSON schema is validated
If fail, return fail
Check if bar exists to delete
If bar does not exist, return fail
If bar exists then delete bar and return success message

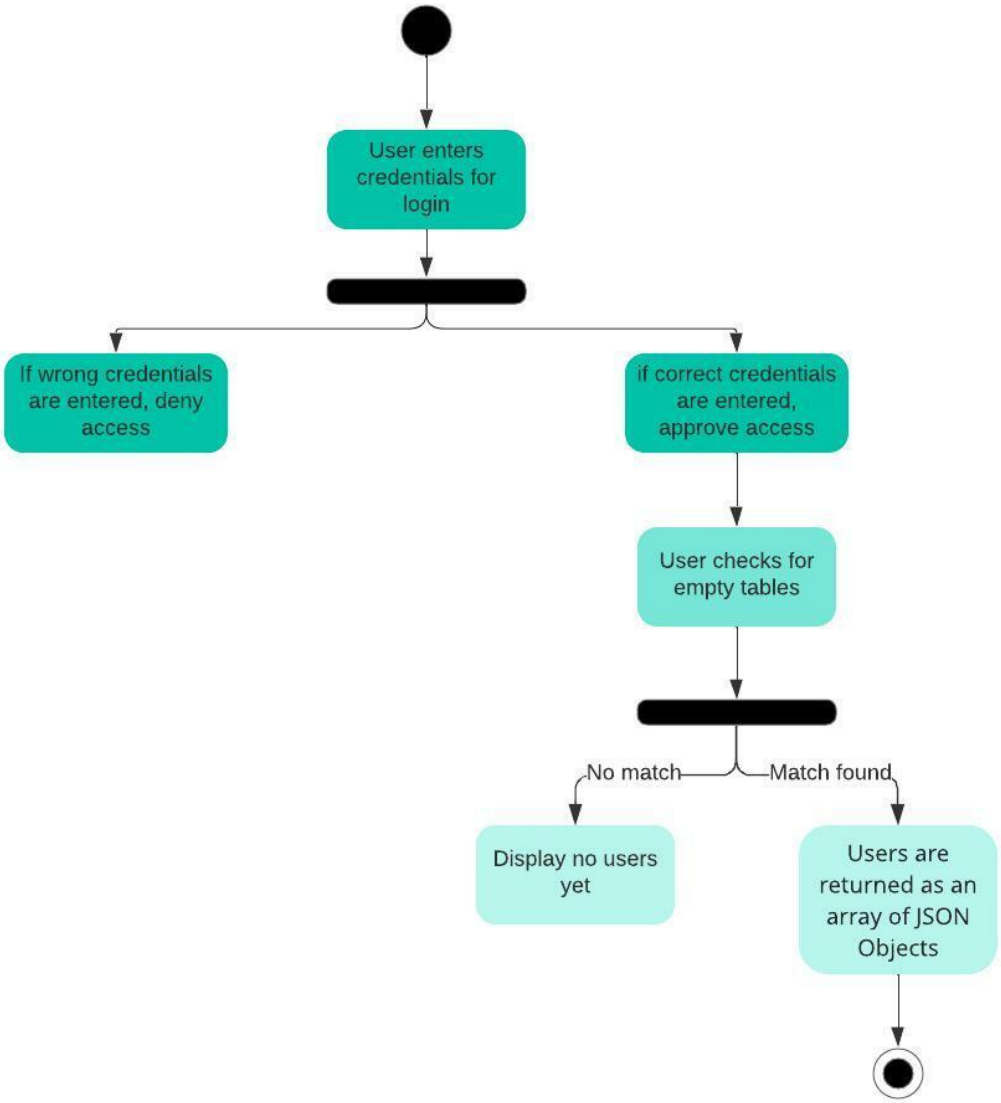
Get Bar



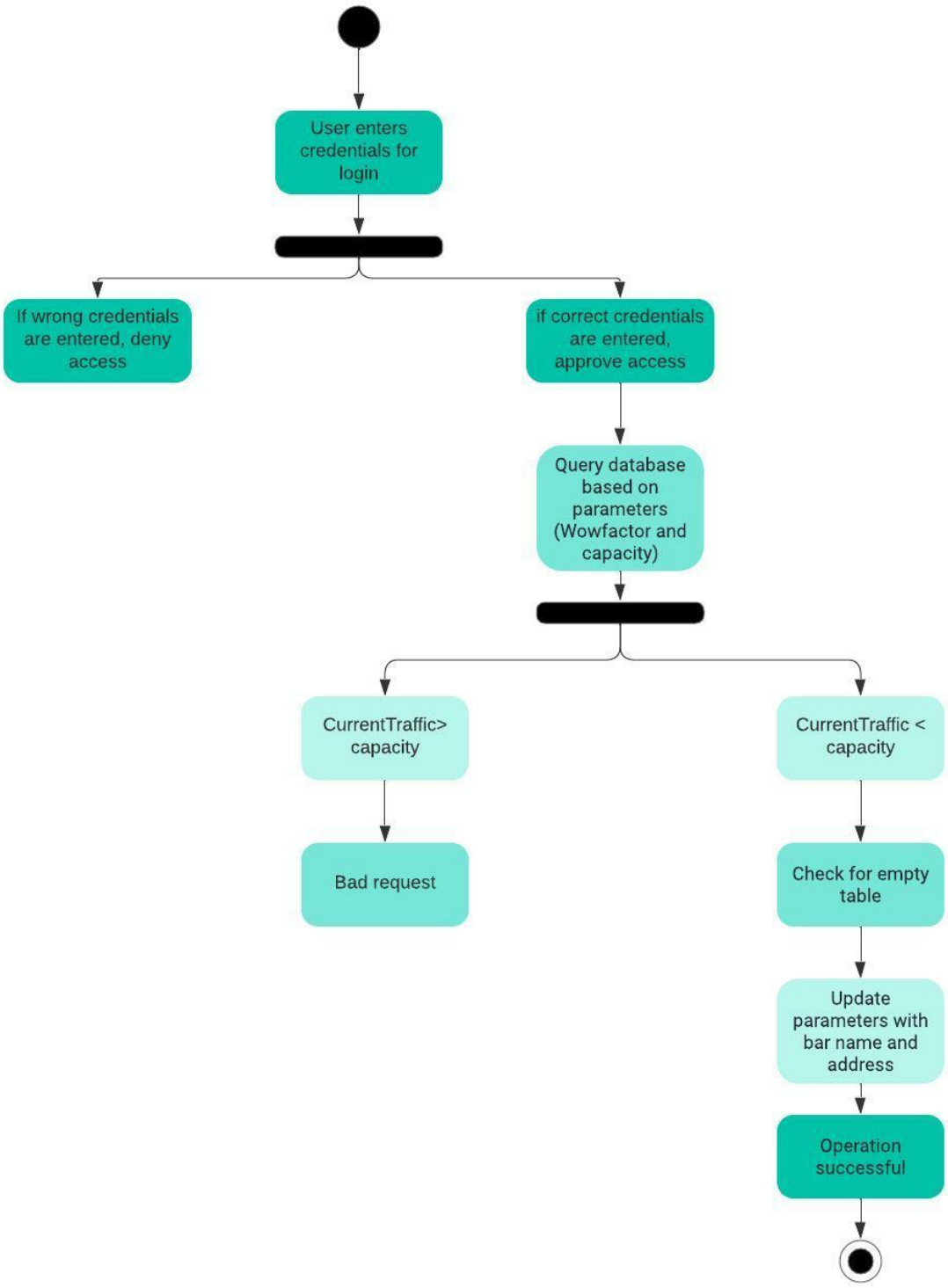
Bar Selection



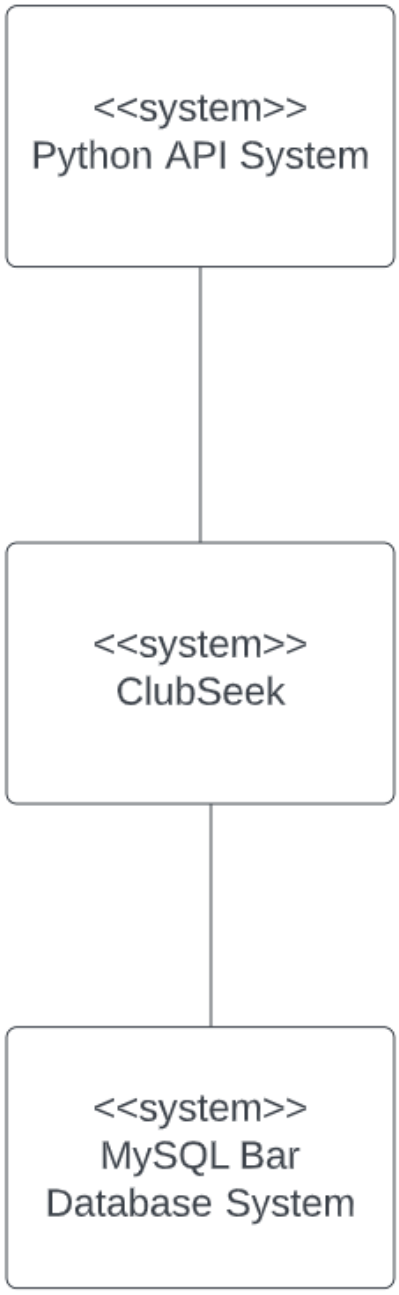
View Users


User enters credentials for login
If wrong credentials are entered, deny access
if correct credentials are entered, approve access
User checks for empty tables
No match
Match found
Display no users yet
Users are returned as an array of JSON Objects

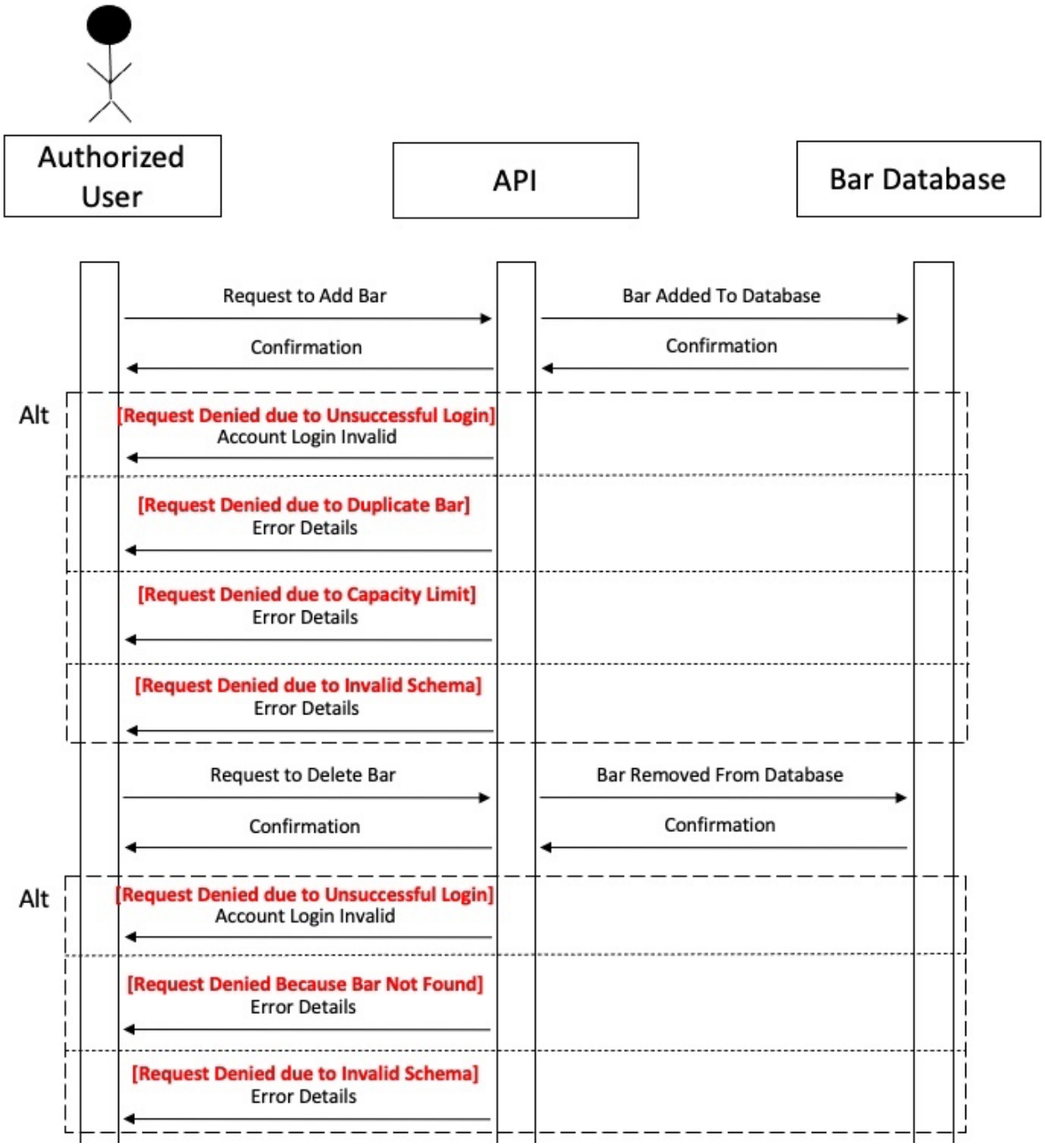
Update Bar Data

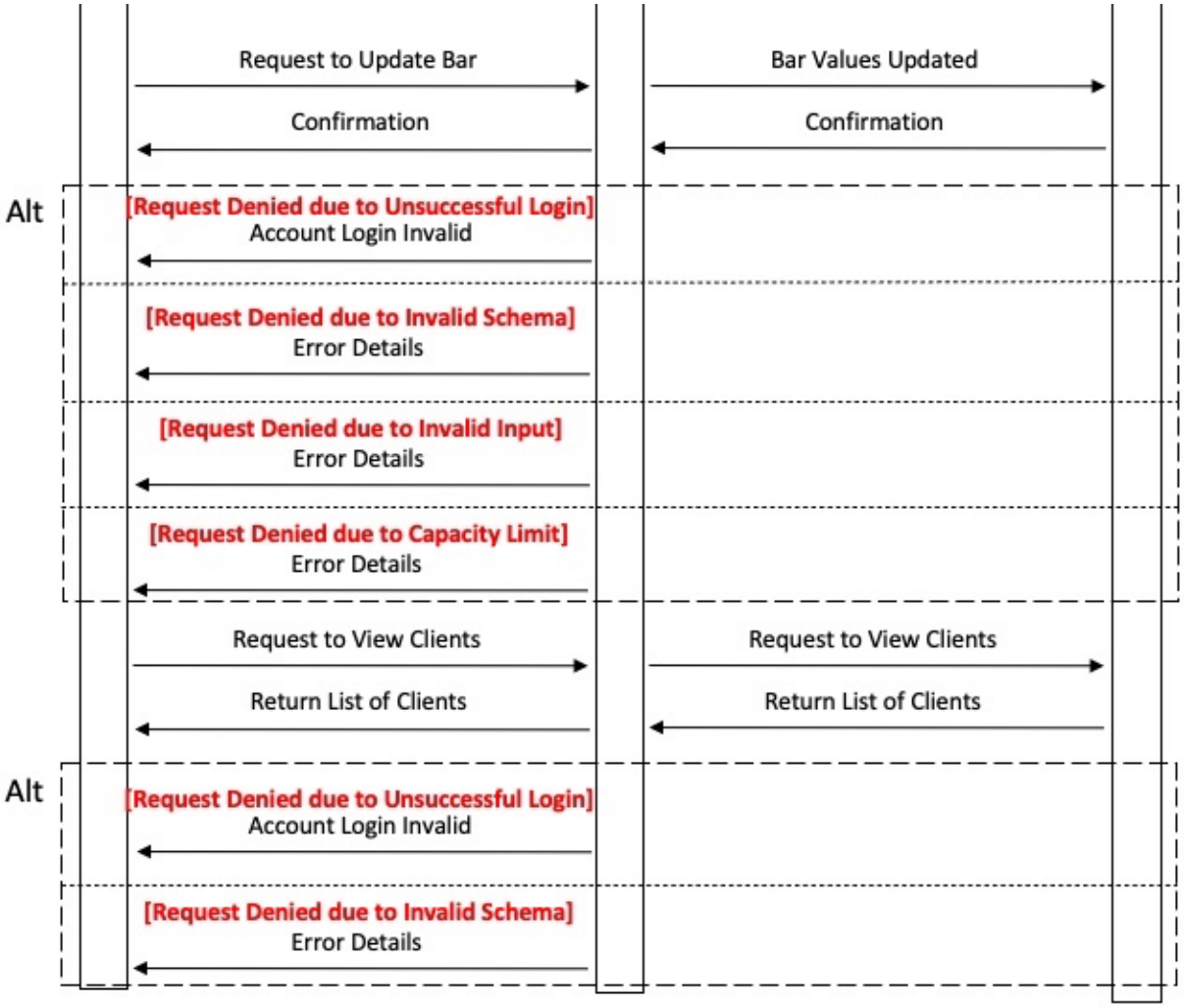


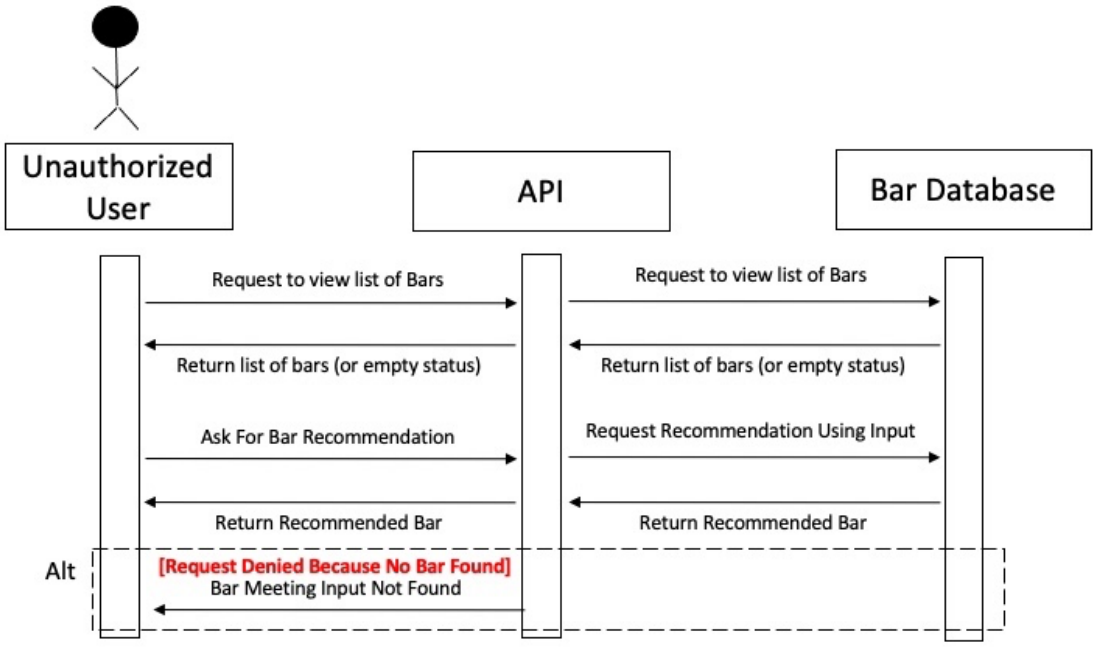
Context Model



Sequence Diagram (Process View)

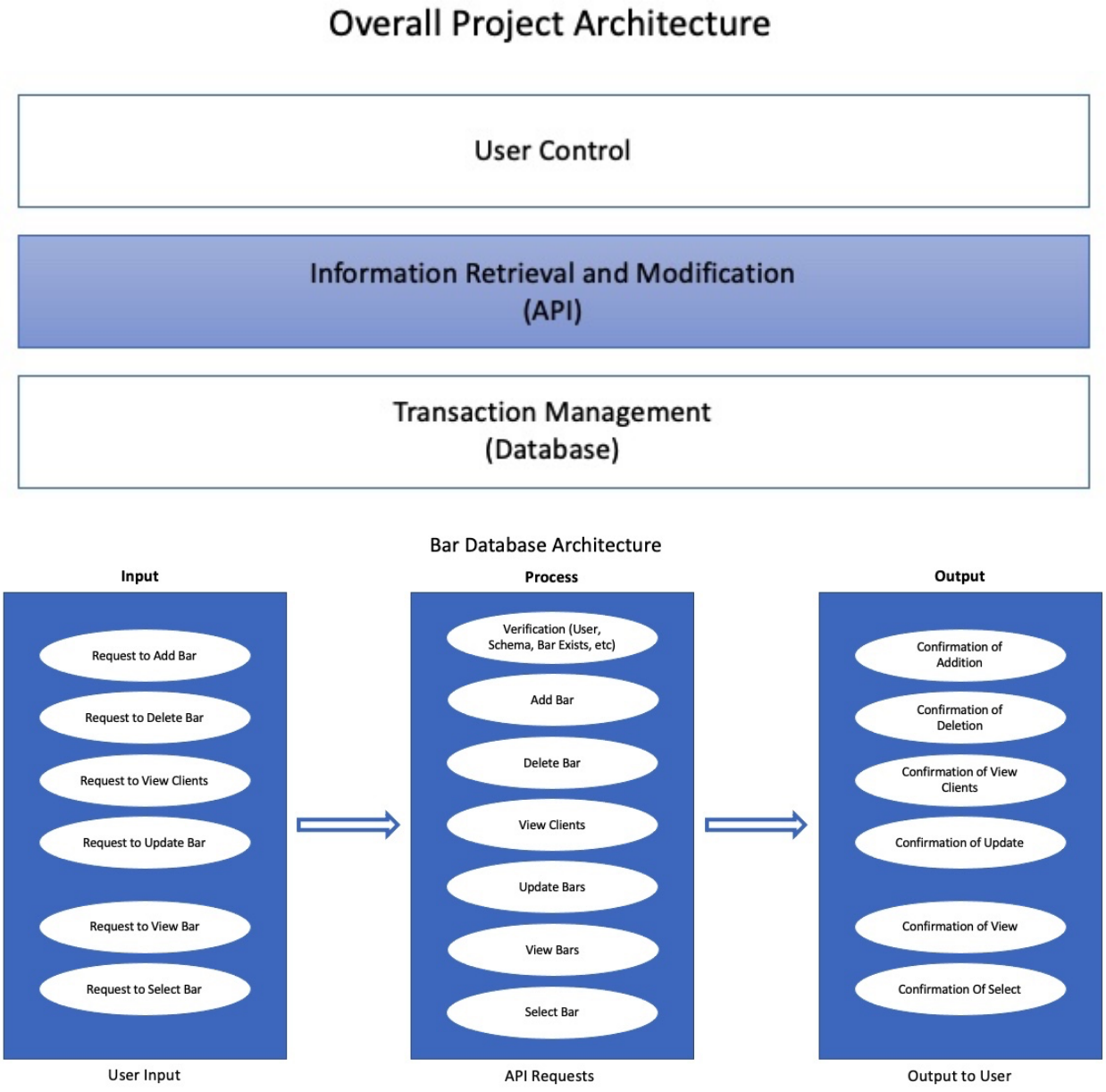



Request to Update Bar
Bar Values Updated
Confirmation
Confirmation
Alt
[Request Denied due to Unsuccessful Login]
Account Login Invalid
[Request Denied due to Invalid Schema]
Error Details
[Request Denied due to Invalid Input]
Error Details
[Request Denied due to Capacity Limit]
Error Details
Request to View Clients
Request to View Clients
Return List of Clients
Return List of Clients
Alt
[Request Denied due to Unsuccessful Login]
Account Login Invalid
[Request Denied due to Invalid Schema]
Error Details

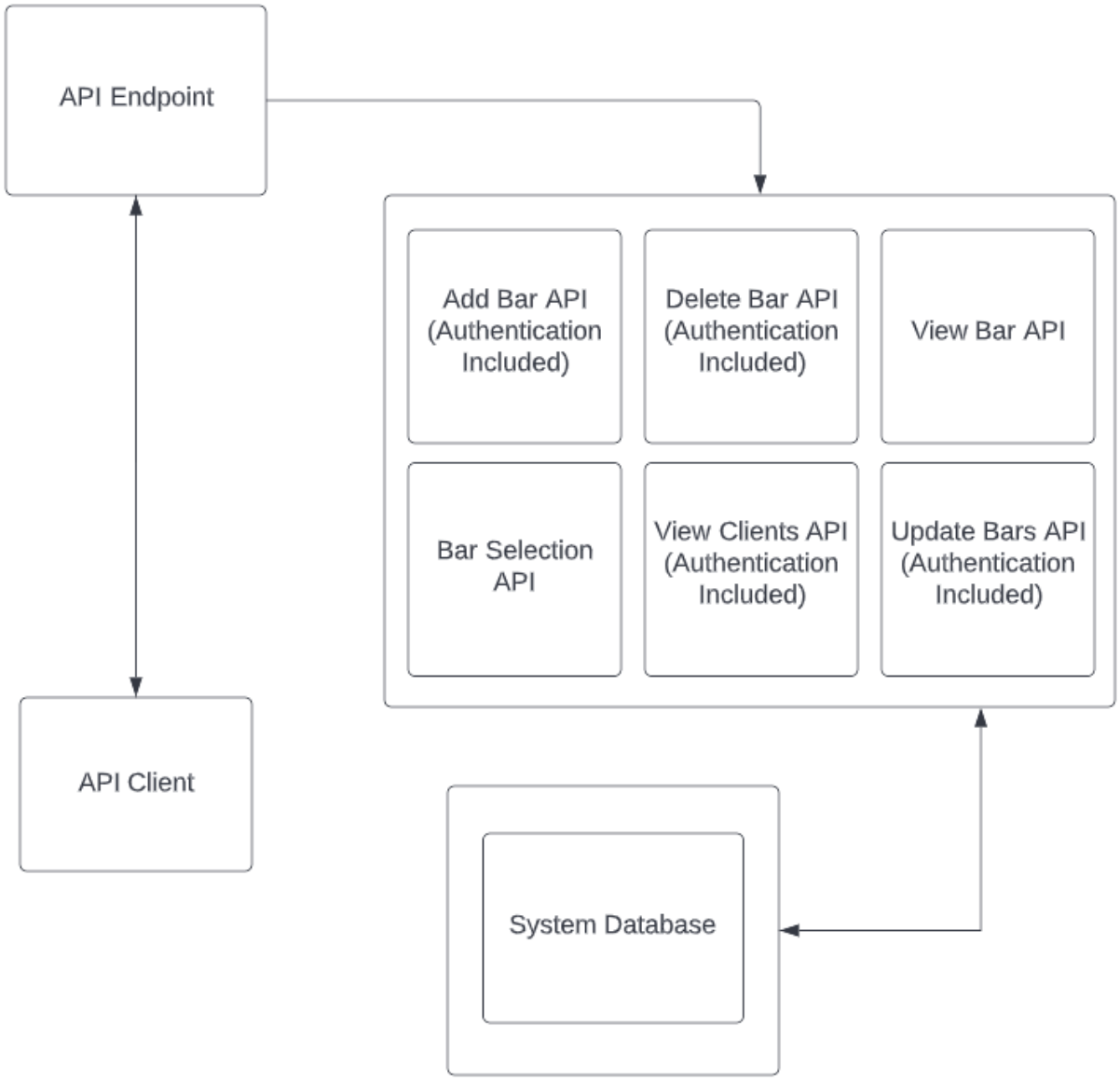

Unauthorized User
API
Bar Database
Request to view list of Bars
Request to view list of Bars
Return list of bars (or empty status)
Return list of bars (or empty status)
Ask For Bar Recommendation
Request Recommendation Using Input
Return Recommended Bar
Return Recommended Bar
Alt
[Request Denied Because No Bar Found]
Bar Meeting Input Not Found

Architectural Modeling:

Process Model



Component Diagram



Specification

Add Bar

Function	Adds bar to the database
Description	Adds a bar to the database. Includes the following fields: name, wow factor, capacity, current traffic, and address
Inputs	User credentials
Source	Existing database of the bar
Outputs	Operation Successful A bar has traffic that exceeds capacity A bar already exists in the database Authentication credits missing or invalid
Destination	API file
Action:	Takes user input and credentials and inputs name, wow factor, capacity, current traffic, and address of the bar
Requires	User input
Precondition	User credentials, bar should exist
Postcondition	Displays added bar
Side effects	None

Delete Bar

Function	Deletes bar from database
Description	Deletes bar name, wow factor, capacity, current traffic, and address from database
Inputs	User credentials
Source	Existing bars in database
Outputs	Operation Successful There is no bar with the following name or address Authentication credits missing or invalid
Destination	API file
Action:	Removes a bar and all its relevant data from the database
Requires	User input
Precondition	User credentials, bar should exist
Postcondition	Deletes bar
Side effects	None

Get Bar

Function	Gets requested bar
Description	Displays the bar that is requested by the user
Inputs	Bar name and address
Source	Existing bar database
Outputs	Operation Successful No Clients or Bars to Return
Destination	API file
Action:	Returns an array of all the bars stored in the database in a JSON object
Requires	Bar name and address
Precondition	User input
Postcondition	Array of JSON objects
Side effects	None

Bar Selection

Function	Matches user to a bar based on parameters
Description	Uses parameters like wowfactor and capacity to match users with the bar they would like
Inputs	User name, phone number, wowfactor, capacity, preference
Source	Existing bar database
Outputs	Operation Successful No bars meet minimum requirements A client already exists at the users table
Destination	API file
Action:	This API finds the best bar for an individual based on capacity and wowfactor. In the case of a tie, whichever parameter was preferred, the bar with a higher value of the preferred parameter will be returned
Requires	User name, phone number, existing bar, capacity, wowfactor, preference
Precondition	currentTraffic < capacity (assumed)
Postcondition	Displaying the bar match
Side effects	None

View Users

Function	Gets users from the database
Description	Displays existing users in the database at tables
Inputs	User credentials
Source	Existing bar database
Outputs	Operation Successful No clients or bars to return Authentication credentials missing or invalid
Destination	API file
Action:	Checks for empty tables
Requires	User credentials, existing bar and users at tables
Precondition	Authorized
Postcondition	Displaying the users at tables
Side effects	None

Update Bar Data

Function	Updates information of the bars
Description	Based on parameters, user is able to check for empty tables and update the parameters with bar name and address
Inputs	User credentials, wowfactor, capacity
Source	Existing bar database
Outputs	Operation successful, bar updated Bad request
Destination	API file
Action:	If currentTraffic < capacity, checks for empty tables and user is able to update information If currentTraffic > capacity, bad request
Requires	User authorization, parameters
Precondition	currentTraffic < capacity
Postcondition	Bar is updated
Side effects	None

Software Testing

Unit Testing

Name	*Description*	*RQ #*
test_verify_password()	Verify that a correct username and password returns True	F5
test_verify_password_fail()	Verify that a incorrect username and password returns True	F5
test_bar_greatest_wow()	Verify that Bar with highest Wow Factor from a list of Bars	F7
test_bar_lowest_traffic()	Verify that Bar with lowest traffic from a list of Bars	F7

Component Testing

Name	*Description*	*RQ #*
test_Add_In_Database	This test tested whether the schema requests for adding an entry to the bar database was correct. It also showed that we were able to successfully view the contents of the database without interfacing with the Python API.	
test_Del_In_Database	This test tested whether the schema requests for deleting an entry to the bar database was correct, and that we were able to successfully delete entries from the bar database.	
test_update_bar_on_selection_speed	This test successfully proved that our system would meet the nonfunctional requirement of updating the currentTraffic of a bar entry after adding a user to it within 1 second.	NF9

System Testing

Name	*Description*	*RQ #*
test_bar_adding()	Test for proper functionality of the addBar API	F1 NF4
test_bar_updating_bar()	Test for proper functionality of the updateBar API	NF4
test_bar_bad_update()	Test that request is denied if currentTrafficChange>capacity	
test_bar_selection()	Test for proper functionality of the selectBar API	NF4 NF6 NF8 NF9
test_bar_selection_duplicate_Request()	Test that selectBar API only allows one user per night	
test_bar_selection_no_avail_Bars()	Test that selectBar does not recommend Bar if minimum requirements are not met	
test_view_clients()	Test for proper functionality of the viewUser API	NF6
test_view_clients_no_auth()	Test for proper functionality of Authentication in viewUser API	NF7
test_bar_deleting_bar_without_auth()	Test for proper functionality of Authentication in deleteBar API	NF7
test_bar_deleting_bar()	Test for proper functionality of the deleteBar API	NF5
test_bar_fail_deleting_bar()	Test for error message when a Bar matching the request does not exist in the request	
test_bar_add_300_bars()	Test that API and DB can handle 300 consecutive requests	NF2 NF4 NF1
test_view_300_bars()	Test that API and DB can handle large requests	NF6
test_bar_selection_with_many_bars()	Test for proper functionality of the selectBar API with multiple Bars	NF8 NF9

Project Backlog

ID	*Requirement and Description*	*Implemented?*
B1	Add a secondary database that keeps track of all queries and requests made to the database	
B2	Create an algorithm to recommend a bar to a user based on given parameters	Y
B3	Keep track of users and what bars they have been recommended	Y
B4	Update the current traffic of the bar that is recommended to the user	Y
B5	Deploy containers to Kubernetes for fault tolerance and high availability	
B6	Develop a front end to provide a UI before mass-market release	