## Client-Side Caching Strategy

Client-side caching was implemented to reduce the number of server calls and bandwidth usage for the client. The method creates a cache in the same folder as the Client.py file. Upon a request for either the dates or cars lists, the client checks the cache for the existence of one of these files. If the file does not exist, a request is sent to the server for the file. Upon receipt of the file, the information is first written to the cache and a timestamp is recorded, then presented to the client. If the file exists in the cache, its validity is checked using the timestamp to ensure it is not stale (less than 60 seconds old). If it is valid, it is read from the cache and presented to the user, if not a server request is made.

## Server-Side Leader Election Strategy

The server election process begins if a server detects that there is no leader alive. The server that detected the lack of leadership broadcasts an election message. Any live server that receives the message will reply with their server ID, which by default is the system PID. The server that started the election will receive all the responses and parse them to find the server with the highest ID. Once this is found, it broadcasts a new message with the leader announcement. All the live servers then parse the announcement and the server declared as the leader then sets its own leader status as True. All servers by default have their leader status set to False.

## Leader Status Check-in Strategy

To check whether the leader is alive, servers will periodically broadcast a heartbeat message. All the live servers will receive the message as the ID of the leader is unknown to the any of the servers. The leader broadcasts their response, letting all the servers know they are alive, but this message is only processed by the server that initiated the heartbeat. If the leader does not respond, they are assumed to be dead, and a new election is initiated by the server that sent the heartbeat.

## Start-up Instructions

### Client.py:

python client.py <multicast IP> <port number>

The first command line argument corresponds to the IP address of the server(s) to communicate with. The client sends its messages to this address and expects responses from this address. This is not the client’s current address or the address the client will bind to; the client will bind to the loopback address exclusively. The second argument is the port number. This corresponds to the port at the server’s address to which messages will be sent. Outgoing messages are directed to this port, but incoming messages may come from a different port at the same address. Note that the address and port for the client and server should match.

Ex: python client.py 235.1.1.1 62002

### Server.py:

python server.py <multicast IP> <port number> <ID \*optional\*>

The first command line argument corresponds to the IP address intended for use by this system. The servers will all listen at this address for incoming messages from clients and other servers. The second argument is the port number. This corresponds to the port at this address that the servers are listening on. All incoming messages should be directed to this port. Any replies to clients will happen from a different port dynamically generated at the time of response and then will be closed when the reply has been sent. The third argument is an optional numerical ID that can be assigned to the server to force a specific leader. Note that the address and port for the client and server should match.

Ex: python server.py 235.1.1.1 62002