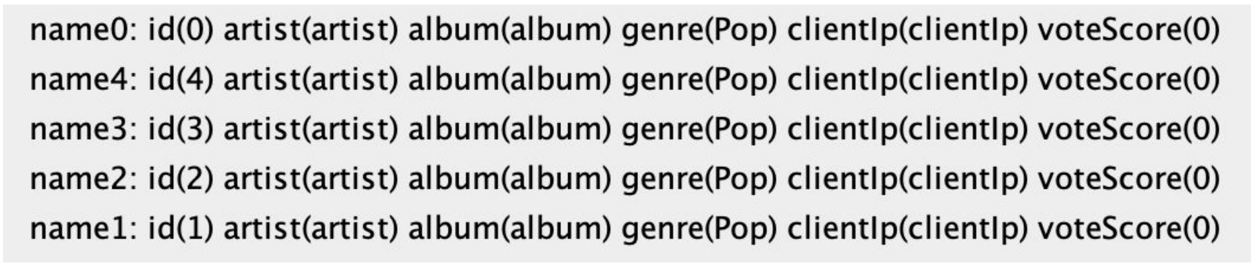
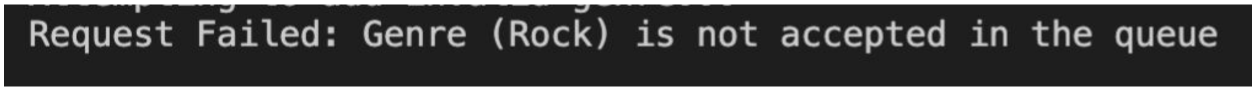
TEST ONE: “Adding 5 songs to a queue with max 5”
Input: 5 songs called song0, song1, song2, song3, song 4
Expected output: a queue displaying 5 songs with equal total votes
Why: the queue is set to a max of 5, therefore 5 songs can be successfully added
Actual output:



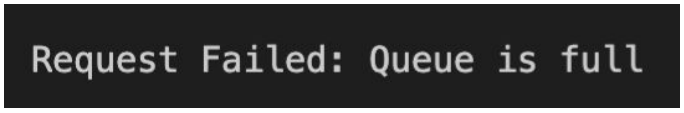
```
name0: id(0) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
name4: id(4) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
name3: id(3) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
name2: id(2) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
name1: id(1) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
```

TEST TWO: “Attempting to add invalid genre”
Input: a song with the genre “Rock” into a queue that only accepts “Pop”
Expected output: a notification that the song was not added to the queue
Why: In the add song method, there is a conditional check for genre type
Actual output:



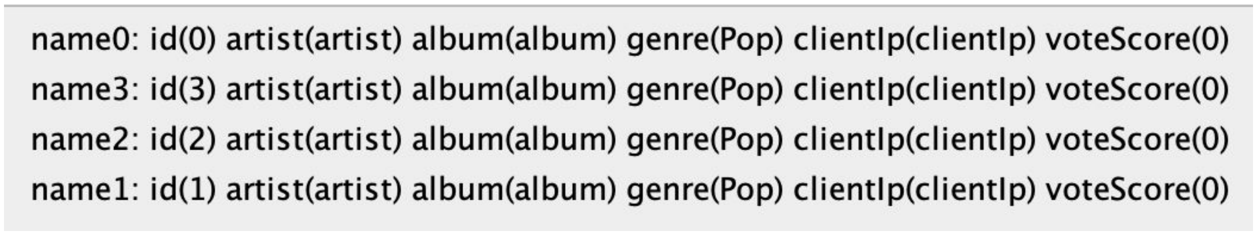
```
Request Failed: Genre (Rock) is not accepted in the queue
```

TEST THREE: “Adding a 6th song to the queue”
Input: a song request object
Expected output: a notification that the song was not added to the queue
Why: In the add song method, there is a conditional check for if the queue is full
Actual output:



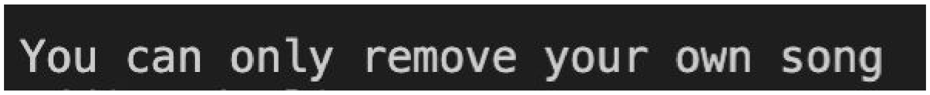
```
Request Failed: Queue is full
```

TEST FOUR: “Removing one song”
Input: song4 into the remove method
Expected output: a queue containing songs song0, song1, song2, song3 in the same order as before
Why: When a song is removed, the queue is updated to remove it visibly
Actual output:

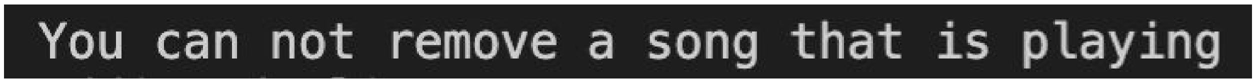


```
name0: id(0) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
name3: id(3) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
name2: id(2) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
name1: id(1) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(0)
```

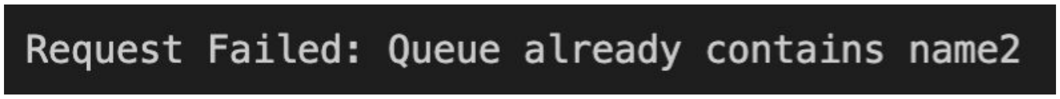
TEST FIVE: “Removing someone else’s song”
Input: a clientIp into the remove method that does not match the clientIp of the song
Expected output: a notification of failure to remove song
Why: When a song is removed, the clientIps are compared to make sure the user is only removing their own song
Actual output:


```
You can only remove your own song
```

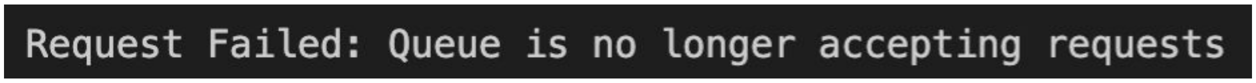
TEST SIX: “Removing a song that is playing”
Input: a song that is currently playing into the remove method
Expected output: a notification of failure to remove song
Why: When a song is removed, a conditional check is made to make sure the song is not playing
Actual output:


```
You can not remove a song that is playing
```

TEST SEVEN: “Adding a duplicate song”
Input: song2 into the add method
Expected output: a notification that the add failed
Why: Before adding the song to the queue, the add method checks if it already exists
Actual output:


```
Request Failed: Queue already contains name2
```

TEST EIGHT: “Adding a song when queue is no longer accepting requests”
Input: a SongRequest object into the add method
Expected output: a notification that add failed
Why: Before a song is added to the queue, the add method checks if the host is still accepting song requests
Actual output:


```
Request Failed: Queue is no longer accepting requests
```

TEST NINE: “Liking and disliking songs”
Input: liking song1 3 times, song2 and song3 once each, and disliking song0 once
Expected output: a queue that orders the songs based on their total vote score
Why: When a song is liked or disliked, its position in the queue is updated
Actual output:

```
name1: id(1) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(3)
name2: id(2) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(1)
name3: id(3) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(1)
name0: id(0) artist(artist) album(album) genre(Pop) clientIp(clientIp) voteScore(-1)
```