SOFTWARE TESTING REPORT

DigiQ

Overview

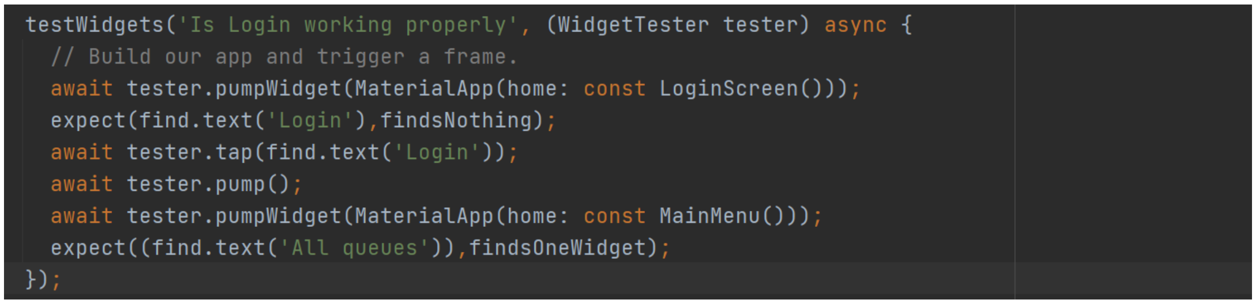
Create and Manage Queues in an efficient way. No need to stand in the queue anymore. Just let the technology do the work for you while you work on things that matters the most.

Testing

Unit Testing

We have tested 8 features using unit testing in our mobile app. It includes Model testing, UI testing (Working of a button click), API testing.

Login Feature

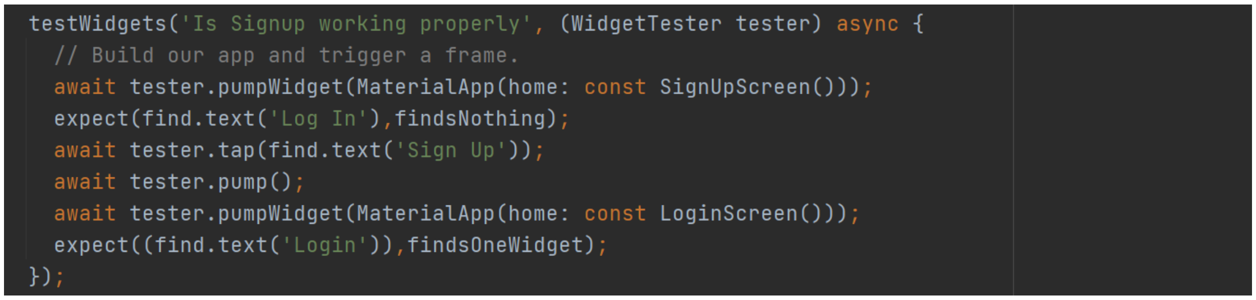


```
testWidgets('Is Login working properly', (WidgetTester tester) async {
  // Build our app and trigger a frame.
  await tester.pumpWidget(MaterialApp(home: const LoginScreen()));
  expect(find.text('Login'),findsNothing);
  await tester.tap(find.text('Login'));
  await tester.pump();
  await tester.pumpWidget(MaterialApp(home: const MainMenu()));
  expect((find.text('All queues')),findsOneWidget);
});
```

testing reports :

Test cases	Output
Wrong Password	Wrong credentials error
Valid registration	Redirects to All Queues page

Signup Feature



```
testWidgets('Is Signup working properly', (WidgetTester tester) async {
  // Build our app and trigger a frame.
  await tester.pumpWidget(MaterialApp(home: const SignUpScreen()));
  expect(find.text('Log In'),findsNothing);
  await tester.tap(find.text('Sign Up'));
  await tester.pump();
  await tester.pumpWidget(MaterialApp(home: const LoginScreen()));
  expect((find.text('Login')),findsOneWidget);
});
```

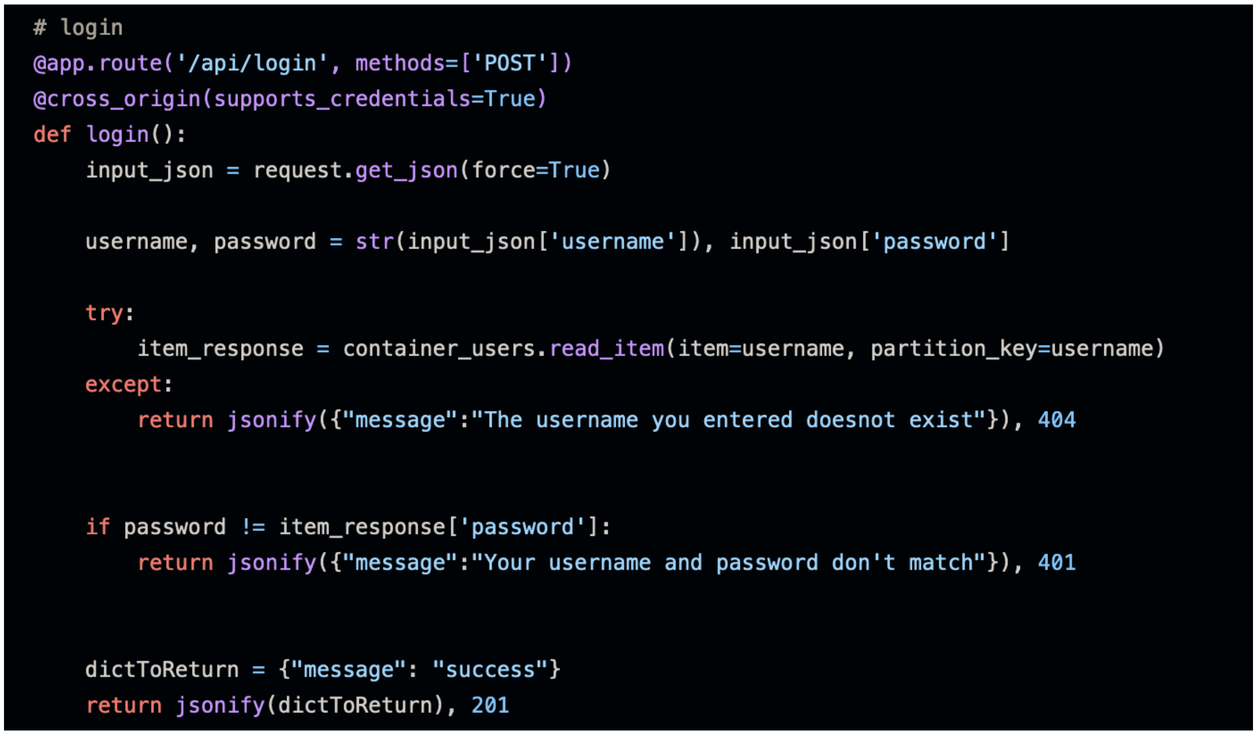
The above unit tests will test the signup and the login functionality of the app.
testing reports :

Test cases	Output
User already exist	The username already exist, select a new username
Valid registration	Redirects to Login Page

Backend Report

Login Feature

Here is the code snippet for the login details:



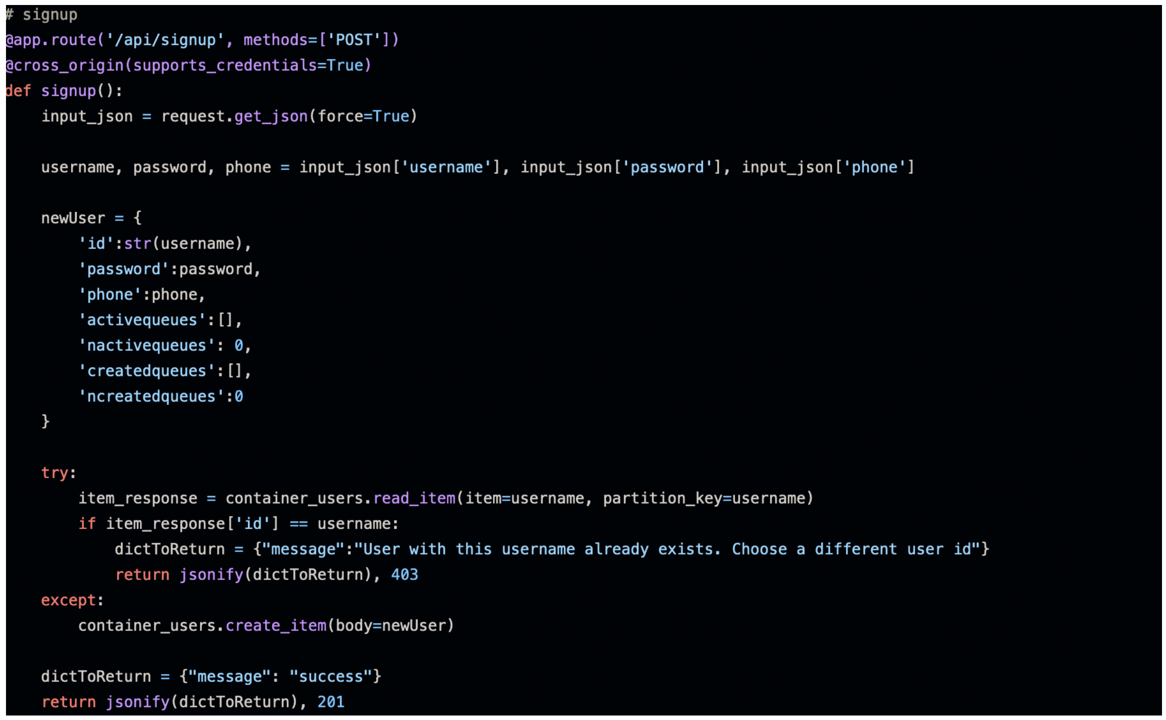
```python
# login
@app.route('/api/login', methods=['POST'])
@cross_origin(supports_credentials=True)
def login():
    input_json = request.get_json(force=True)

    username, password = str(input_json['username']), input_json['password']

    try:
        item_response = container_users.read_item(item=username, partition_key=username)
    except:
        return jsonify({"message":"The username you entered doesnot exist"}), 404


    if password != item_response['password']:
        return jsonify({"message":"Your username and password don't match"}), 401


    dictToReturn = {"message": "success"}
    return jsonify(dictToReturn), 201
```

testing reports:

Test Cases	Output
Unregistered username	Message: The username does not exist (404 error)
Wrong credentials	Message: Your username and password do not match (401 error)
Valid credentials	Message : success (201 success)

SIGN-UP FEATURE:

Here is the code snippet for signup



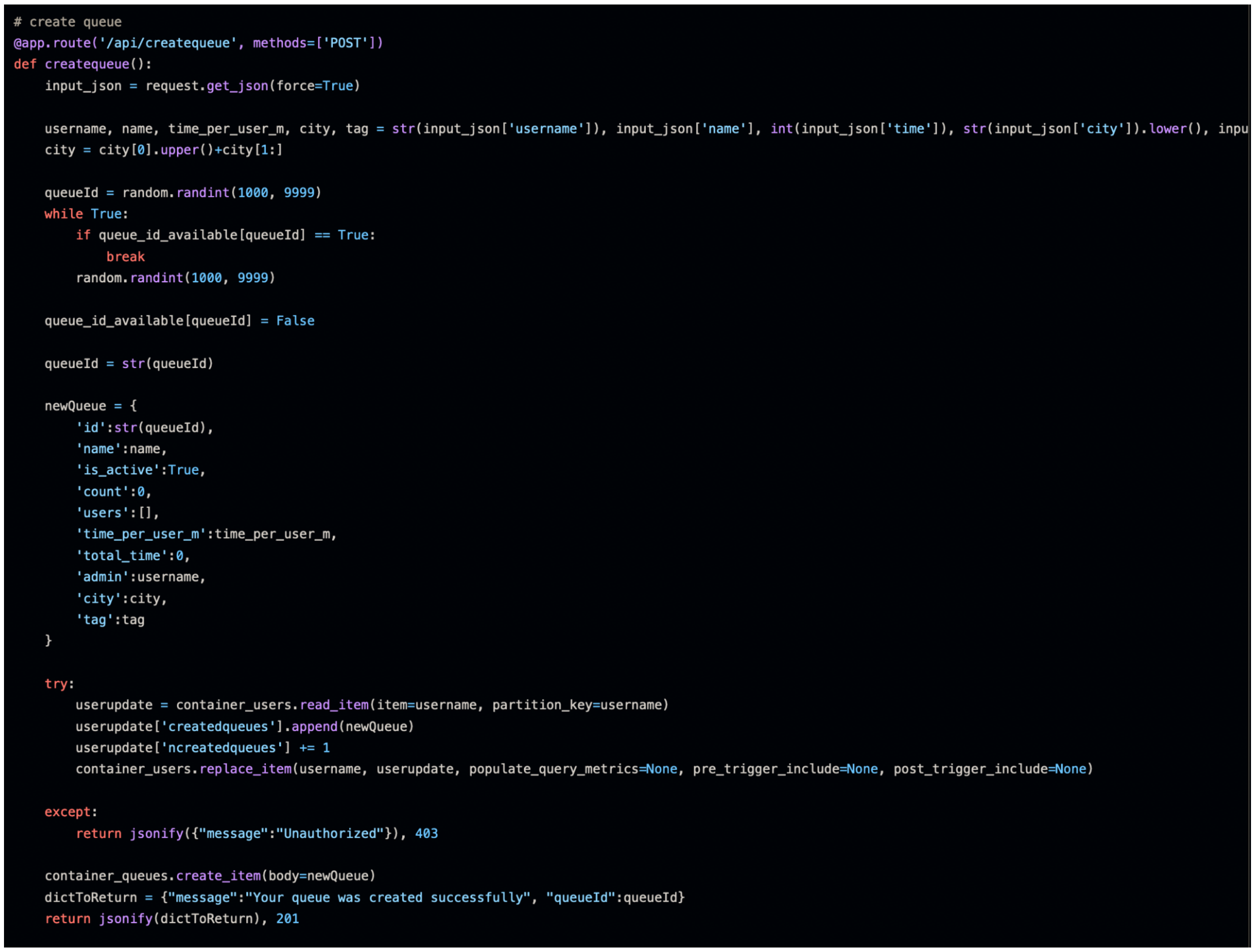
```python
# signup
@app.route('/api/signup', methods=['POST'])
@cross_origin(supports_credentials=True)
def signup():
    input_json = request.get_json(force=True)

    username, password, phone = input_json['username'], input_json['password'], input_json['phone']

    newUser = {
        'id':str(username),
        'password':password,
        'phone':phone,
        'activequeues':[],
        'nactivequeues': 0,
        'createdqueues':[],
        'ncreatedqueues':0
    }

    try:
        item_response = container_users.read_item(item=username, partition_key=username)
        if item_response['id'] == username:
            dictToReturn = {"message":"User with this username already exists. Choose a different user id"}
            return jsonify(dictToReturn), 403
    except:
        container_users.create_item(body=newUser)

    dictToReturn = {"message": "success"}
    return jsonify(dictToReturn), 201
```

testing reports :

Test cases	Output
User already exist	The username already exist, select a new username (403 error)
Valid registration	Success (201)

CREATE QUEUE:

Code snippet



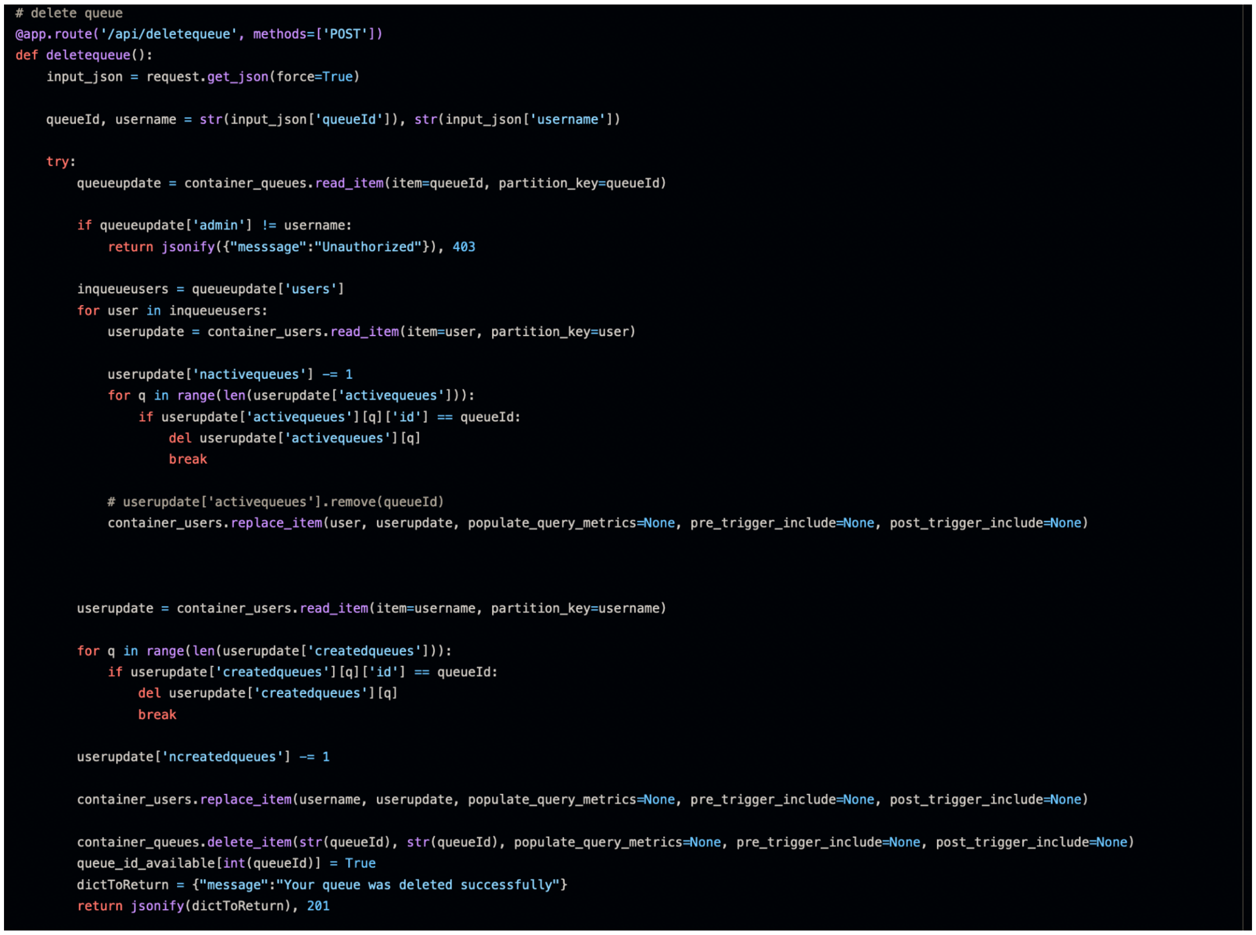
```
# create queue
@app.route('/api/createqueue', methods=['POST'])
def createqueue():
    input_json = request.get_json(force=True)

    username, name, time_per_user_m, city, tag = str(input_json['username']), input_json['name'], int(input_json['time']), str(input_json['city']).lower(), inpu
    city = city[0].upper()+city[1:]

    queueId = random.randint(1000, 9999)
    while True:
        if queue_id_available[queueId] == True:
            break
        random.randint(1000, 9999)

    queue_id_available[queueId] = False

    queueId = str(queueId)

    newQueue = {
        'id':str(queueId),
        'name':name,
        'is_active':True,
        'count':0,
        'users':[],
        'time_per_user_m':time_per_user_m,
        'total_time':0,
        'admin':username,
        'city':city,
        'tag':tag
    }

    try:
        userupdate = container_users.read_item(item=username, partition_key=username)
        userupdate['createdqueues'].append(newQueue)
        userupdate['ncreatedqueues'] += 1
        container_users.replace_item(username, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)

    except:
        return jsonify({"message":"Unauthorized"}), 403

    container_queues.create_item(body=newQueue)
    dictToReturn = {"message":"Your queue was created successfully", "queueId":queueId}
    return jsonify(dictToReturn), 201
```

testing report :

Test case	output
Unregistered user creating queue	Unauthorized
Logged in user creating queue	Queue being successfully created

DELETE QUEUE:

Code snippet



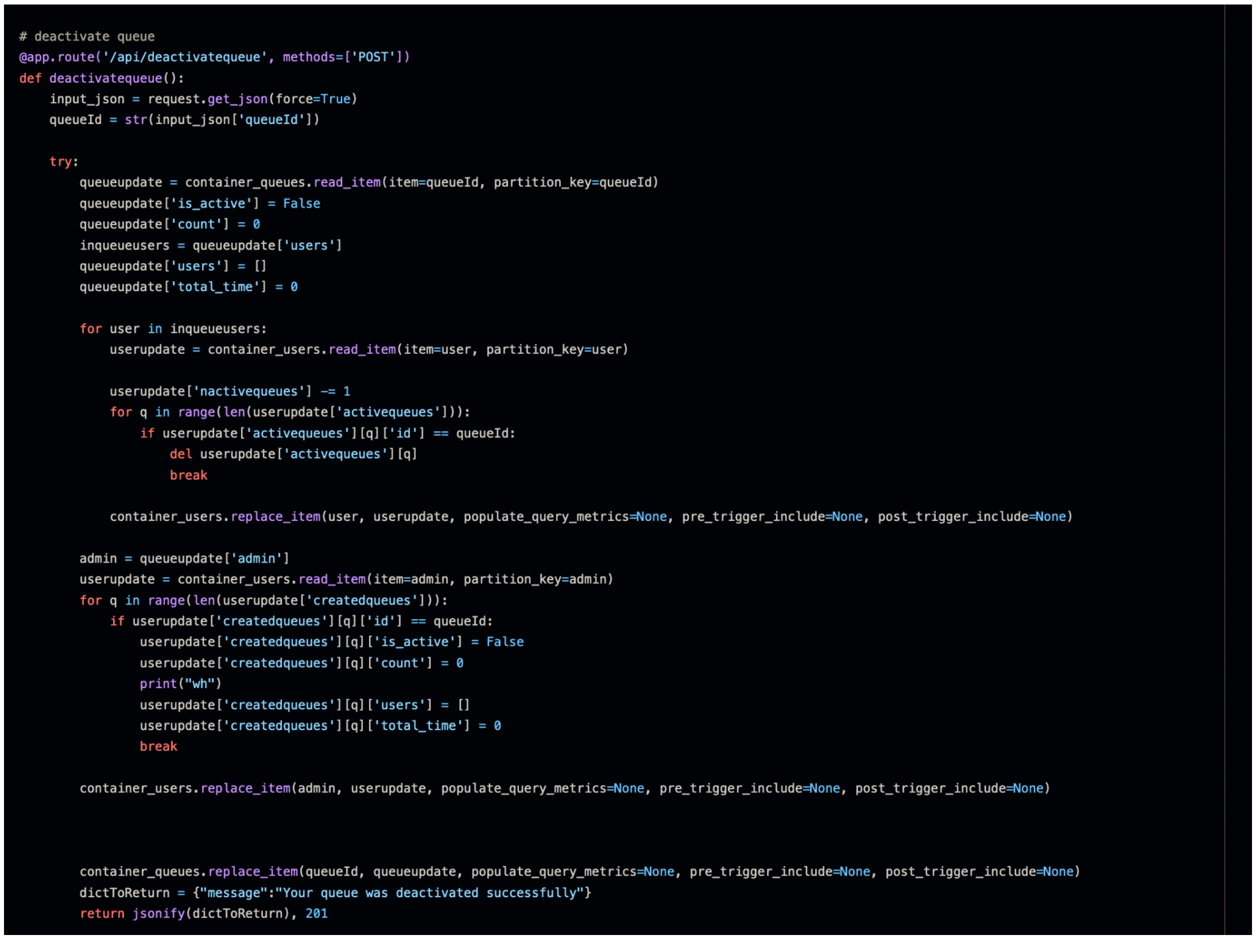
```
# delete queue
@app.route('/api/deletequeue', methods=['POST'])
def deletequeue():
    input_json = request.get_json(force=True)

    queueId, username = str(input_json['queueId']), str(input_json['username'])

    try:
        queueupdate = container_queues.read_item(item=queueId, partition_key=queueId)

        if queueupdate['admin'] != username:
            return jsonify({"messsage":"Unauthorized"}), 403

        inqueueusers = queueupdate['users']
        for user in inqueueusers:
            userupdate = container_users.read_item(item=user, partition_key=user)

            userupdate['nactivequeues'] -= 1
            for q in range(len(userupdate['activequeues'])):
                if userupdate['activequeues'][q]['id'] == queueId:
                    del userupdate['activequeues'][q]
                    break

            # userupdate['activequeues'].remove(queueId)
            container_users.replace_item(user, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)



        userupdate = container_users.read_item(item=username, partition_key=username)

        for q in range(len(userupdate['createdqueues'])):
            if userupdate['createdqueues'][q]['id'] == queueId:
                del userupdate['createdqueues'][q]
                break

        userupdate['ncreatedqueues'] -= 1

        container_users.replace_item(username, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)

        container_queues.delete_item(str(queueId), str(queueId), populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)
        queue_id_available[int(queueId)] = True
        dictToReturn = {"message":"Your queue was deleted successfully"}
        return jsonify(dictToReturn), 201
```

testing report:

Test case	output
Deleting a queue which does not exist	Queue does not exist
Queue deleting which exists	Queue deleted successfully

DEACTIVATE QUEUE:

Code snippet



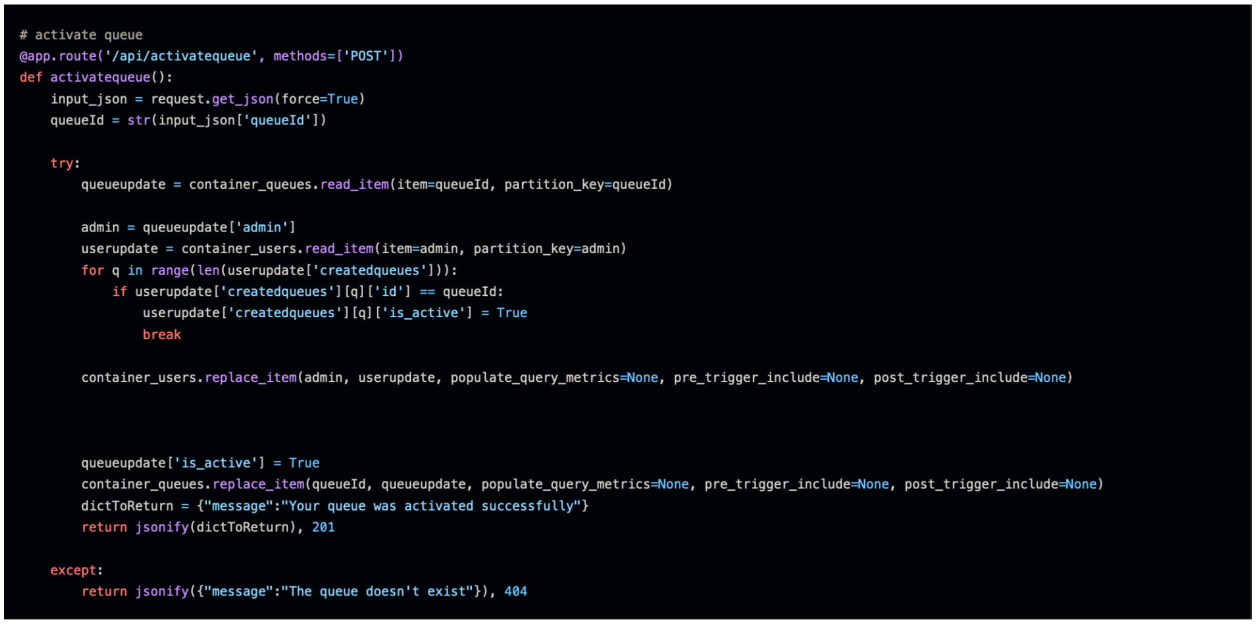
```
# deactivate queue
@app.route('/api/deactivatequeue', methods=['POST'])
def deactivatequeue():
    input_json = request.get_json(force=True)
    queueId = str(input_json['queueId'])

    try:
        queueupdate = container_queues.read_item(item=queueId, partition_key=queueId)
        queueupdate['is_active'] = False
        queueupdate['count'] = 0
        inqueueusers = queueupdate['users']
        queueupdate['users'] = []
        queueupdate['total_time'] = 0

        for user in inqueueusers:
            userupdate = container_users.read_item(item=user, partition_key=user)

            userupdate['nactivequeues'] -= 1
            for q in range(len(userupdate['activequeues'])):
                if userupdate['activequeues'][q]['id'] == queueId:
                    del userupdate['activequeues'][q]
                    break

            container_users.replace_item(user, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)

        admin = queueupdate['admin']
        userupdate = container_users.read_item(item=admin, partition_key=admin)
        for q in range(len(userupdate['createdqueues'])):
            if userupdate['createdqueues'][q]['id'] == queueId:
                userupdate['createdqueues'][q]['is_active'] = False
                userupdate['createdqueues'][q]['count'] = 0
                print("wh")
                userupdate['createdqueues'][q]['users'] = []
                userupdate['createdqueues'][q]['total_time'] = 0
                break

        container_users.replace_item(admin, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)



        container_queues.replace_item(queueId, queueupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)
        dictToReturn = {"message":"Your queue was deactivated successfully"}
        return jsonify(dictToReturn), 201
```

testing report

Test case	output
Deactivating a queue which does not exist	Queue does not exist
Queue deactivating which exists	Queue deactivated successfully

ACTIVATE QUEUE:

Code snippet



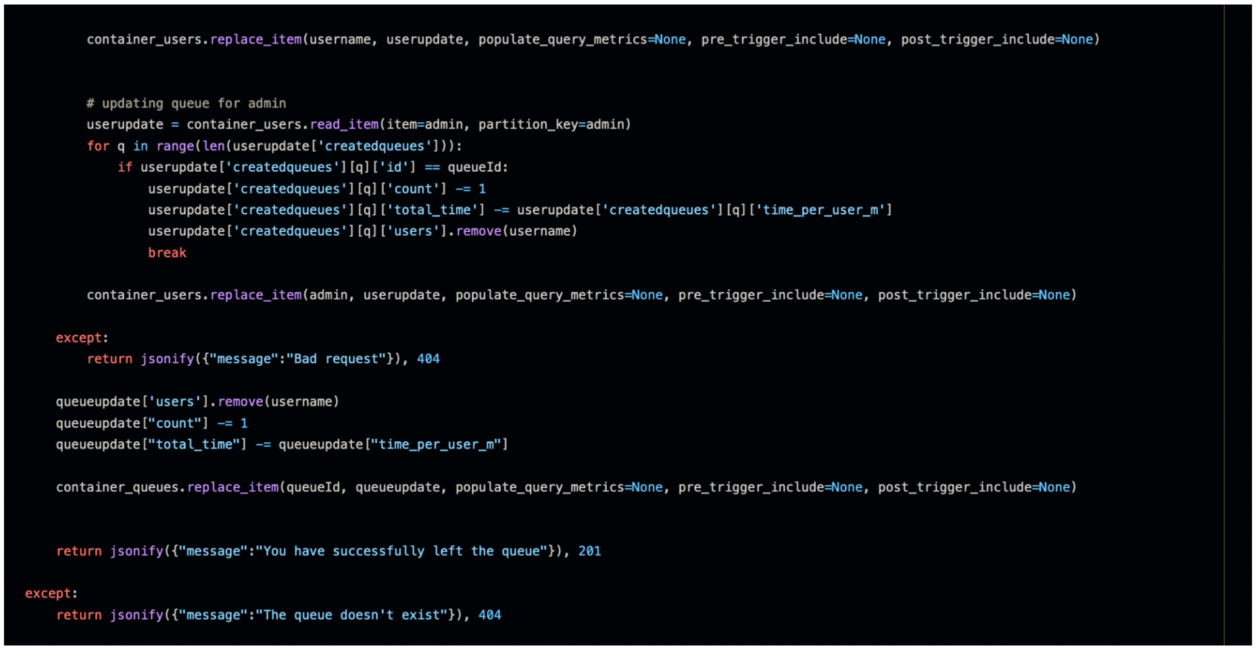
```python
# activate queue
@app.route('/api/activatequeue', methods=['POST'])
def activatequeue():
    input_json = request.get_json(force=True)
    queueId = str(input_json['queueId'])

    try:
        queueupdate = container_queues.read_item(item=queueId, partition_key=queueId)

        admin = queueupdate['admin']
        userupdate = container_users.read_item(item=admin, partition_key=admin)
        for q in range(len(userupdate['createdqueues'])):
            if userupdate['createdqueues'][q]['id'] == queueId:
                userupdate['createdqueues'][q]['is_active'] = True
                break

        container_users.replace_item(admin, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)



        queueupdate['is_active'] = True
        container_queues.replace_item(queueId, queueupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)
        dictToReturn = {"message":"Your queue was activated successfully"}
        return jsonify(dictToReturn), 201

    except:
        return jsonify({"message":"The queue doesn't exist"}), 404
```

testing report

Test case	output
Activating a queue which does not exist	Queue does not exist
Queue activating which exists	Queue activated successfully

JOIN QUEUE:

Code snippet



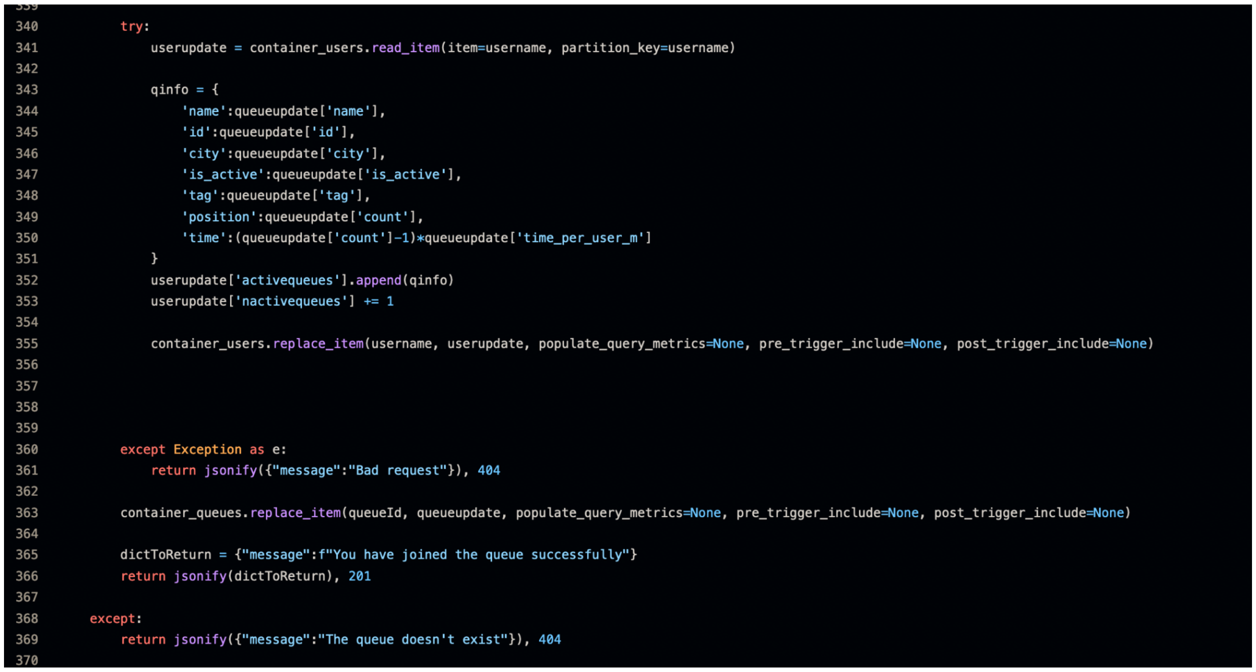
```
            container_users.replace_item(username, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)


            # updating queue for admin
            userupdate = container_users.read_item(item=admin, partition_key=admin)
            for q in range(len(userupdate['createdqueues'])):
                if userupdate['createdqueues'][q]['id'] == queueId:
                    userupdate['createdqueues'][q]['count'] -= 1
                    userupdate['createdqueues'][q]['total_time'] -= userupdate['createdqueues'][q]['time_per_user_m']
                    userupdate['createdqueues'][q]['users'].remove(username)
                    break

            container_users.replace_item(admin, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)

        except:
            return jsonify({"message":"Bad request"}), 404

        queueupdate['users'].remove(username)
        queueupdate["count"] -= 1
        queueupdate["total_time"] -= queueupdate["time_per_user_m"]

        container_queues.replace_item(queueId, queueupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)


        return jsonify({"message":"You have successfully left the queue"}), 201

    except:
        return jsonify({"message":"The queue doesn't exist"}), 404
```

testing report

Test case	output
Joining a queue which does not exist	Queue does not exist
Joining queue which exists	Queue joined successfully

LEAVE QUEUE:

Code snippet



```
        try:
            userupdate = container_users.read_item(item=username, partition_key=username)

            qinfo = {
                'name':queueupdate['name'],
                'id':queueupdate['id'],
                'city':queueupdate['city'],
                'is_active':queueupdate['is_active'],
                'tag':queueupdate['tag'],
                'position':queueupdate['count'],
                'time':(queueupdate['count']-1)*queueupdate['time_per_user_m']
            }
            userupdate['activequeues'].append(qinfo)
            userupdate['nactivequeues'] += 1

            container_users.replace_item(username, userupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)




        except Exception as e:
            return jsonify({"message":"Bad request"}), 404

        container_queues.replace_item(queueId, queueupdate, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None)

        dictToReturn = {"message":f"You have joined the queue successfully"}
        return jsonify(dictToReturn), 201

    except:
        return jsonify({"message":"The queue doesn't exist"}), 404
```

Test case	output
Leaving a queue which does not exist	Queue does not exist
Leaving queue which exists	Queue left successfully

Manual Testing

Apart from the unit tests we have also done manual testing of the app and the website.

1. We created a new user
2. Tested the Sign Up
3. Made a new queue under his/her ID.

4. Joined a random queue under his/her ID.
5. Added multiple members in his/her queue and tested features like deactivate/delete queue
6. Tested the logout feature.

Integration Testing

We ran all unit test modules. We found that all the modules are loosely coupled and don’t require separate integration testing.

To perform manual integration testing,we have individually tested all the modules and made sure that they are working properly in combination with each other.

Conclusion

After running the above test cases we fixed all detected irregularities between the unit tests that are integrated together. We made sure that none of the features are breaking and the system as a whole is working smoothly in all circumstances
.