Cloud Computing Project

Louai Zaiter
13715636
Birkbeck College, University of London

April 2022

Academic Declaration:
I have read and understood the sections of plagiarism in the College Policy on assessment offenses and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines, or meta-searching software.

Introduction

The main purpose of this project is to use the technologies studied during the module to implement and test a RESTful API that enables users to authenticate and bid for items for sale. This project includes implementing microservices that allow CRUD operations on items and auctions.

Project Structure:

```
.
|--- checkWinner.js
|--- index.js
|--- models
    |--- Auction.js
    |--- Bid.js
    |--- Item.js
    |--- User.js
|--- package.json
|--- package-lock.json
|--- routes
    |--- auctions.js
    |--- auth.js
|--- test.py
|--- validations
    |--- validation.js
```

```
|--- verifyToken.js

3 directories, 13 files
```

Routes and End-points:

The API allows authenticated users to add items then associate an auction to a specific item. The process of adding auction starts by verifying if an item with the given id is stored within the database. If the item is found, then the auction is created, otherwise, the client will receive a 400 response status. The second feature is allowing users to search for items in the sale. This process starts by searching for auctions with the status “true” meaning that they are not complete. The next step is allowing authenticated users to actually bid for items. This is done by first checking the auction id, if it exists, then checking if the user bidding for the item is not the owner of it. In case one of the previous conditions is not met, the client receives a 400 status response with a message clarifying the error. Finally, the bid could be created and, in a second step, added to the auction by pushing the bids to the existing ones within the auctions collection. The user has the right to view all sold items details and request to view the bidding history for a given item. The process of browsing the bidding mase for a certainly sold item starts by checking if the item id is given within the URL parameters is stored within the database then searches for the auction associated with that items and returns the bids stored within that auction.

Checking Winner

Thanks to the node-schedule npm package that enables the execution of a process whenever a certain date is met. The “checkWinner” function is invoked whenever a user adds an auction. First, it extracts the deadline of the new auction then it schedules the process of electing a winner whenever the auction deadline comes.

```
const checkWinner= async(auctionId)=>{
  /* get the active auction and
  extract its expiration date*/
  const auction=await Auction.findOne(
                  {'_id': auctionId})
  /*schedule the winner nomination
  whenever the auction expires */
  const job = schedule.scheduleJob(auction.timer,
  async function (){
    // get the auction after it has finished
    const newAuction= await Auction.findOne(
                          {'_id':auctionId})
    // associate a winner to that auction
    var highestBid=null;
    for (var bidIndex in newAuction.bids){
     var bid= await Bid.findOne(
                newAuction.bids[bidIndex]);
      if (bid.price > highestBid){
          highestBid=bid
      }
    }
    await Auction.findOneAndUpdate(
    {'_id': newAuction._id},
    {'price': highestBid.price,
    'winner': highestBid.user,
    'status': false});
});}
```

Authentication OAuth 2.0

The authentication service is duplicated from lab 4. The main functionalities implemented are allowing users to register and only store a hashed version of their password while performing some validation for the inputs. When the user login with the correct credentials, he/she receives an access token that allows him to access other endpoints.
Remark The secret token that is required for the JWT authentication is stored in a ".env" file within the main directory.

Testing the API

To test the API, I've started by used Postman as a client. I started by creating a user, login that user and store the access token. The next step was to add items and associate auctions to them. Then, I've checked the active auctions, retrieved one auction id, and added a bid to it. Finally, I browsed sold items, retrieved one item id, and checked the bidding history for that item. If an auction expiration date comes, the system automatically changes its status and edits the winner.
In a second step, I used python's "requests" package to test the API. Although I didn't use any assertion, I tried to keep the testing process the same as described above using Postman.
Remark You can find the testing algorithm within the root of the project; named as 'test.py'. Before you run it, please make sure that the server is running and the database is connected.
The user in the test below is already signed up, feel free to add new credentials and unlock the test.

Python Testing Code

```
import requests
import json

#Sign up a user (unlock this test by turning locker to true)
locker=False
if (locker==True):
    user={
    'username': 'louaiZaiter',
    'email': 'zlouai@gmail.com',
    'password': '12AZqswx!!',
    }
    r=requests.post("http://localhost:3000/api/user/register",
    json=user)
    print(r.status_code)
    print(r.json())


#Login a user and store the token and the user id
user={
    'email': 'zlouai@gmail.com',
    'password': '12AZqswx!!',
}
r=requests.post("http://localhost:3000/api/user/login",
json=user)
token=r.json()['auth-token']
user_id= r.json()['user_id']

#declare the authorization header for further tests
headers = {'auth-token': token}

#create an item
item = {
   'item':{ 'title': 'HAMHAM',
    'description': 'nothing',
    'expiration' :'2022-02-11T01:12:51.118+00:00',
    'owner': user_id,
    'condtion': 'New'}}
r=requests.post("http://localhost:3000/api/auctions/items",
headers = headers, json=item)
print(r.json())
item_id= r.json()['_id']

#create an auction associated to the previous item
from datetime import datetime, timezone,timedelta
date_time = datetime.now(timezone.utc)
date_time= date_time+ timedelta(seconds=5)
formatted = date_time.isoformat()
```

```
auction={
    'auction':{
    'price': 10,
    'status': True,
    'timer' : formatted,
    'user': user_id,
    'item': item_id
    }}
r=requests.post("http://localhost:3000/api/auctions",
headers = headers, json=auction)
print(r.json())
auction_id= r.json()['_id']

#get active auctions
r=requests.get("http://localhost:3000/api/auctions/active",
headers = headers)
print(r.json())

#bid for an active auction( add 2 bids to an auction)
bid_1={
    'bid':{
        'price': 10000,
        'user': '622dc8570e4910905d9455e3'
    }
}
bid_2={
    'bid':{
        'price': 7000,
        'user': '622dc8570e4910905d9455e3'
    }
}
r=requests.patch('http://localhost:3000/api/auctions/bid/'
+auction_id,
headers=headers, json=bid_1)


r=requests.patch('http://localhost:3000/api/auctions/bid/'
+auction_id,
headers=headers, json=bid_2)
print(r.json())

#Browse sold items
r=requests.get('http://localhost:3000/api/auctions/items/sold',
headers=headers)
print(r.json())
```

```
#Browse bidding history of a sold item
r=requests.get('http://localhost:3000/api/auctions/items/sold/'
+item_id,
headers=headers)
print(r.json())
```

Containerizing the Application

I considered deploying the application using docker within a GCP. The first step was to add, commit, and push the source code to github. Then, I created a virtual machine and cloned the project. The next step involved creating a Dockerfile, building the image, and running the container, as seen in the lab5. A final step was to authorize the VM to access the Mongo Atlas cluster. Using the VM's external IP address, I was able to perform http requests to the remote web server.
Remark: The python testing file within the source code folder uses the external ip address of the GCP VM.