1. NodeJS - IPL Auction

IPL Auction - NodeJS

You should create a back-end application using Node.js and MongoDB that manages an IPL auction.

- **ipl-auction** is the name of the database that you will be using in the application
- **admins** and **players** were the collections that reside inside the database **ipl-auction**.

Collections:

There are two files for collections namely **admin.js** and **players.js** that reside inside **src/mongoose/models**. The schema of the collections is as follows.

admin				
Sl. No.	**Name**	**Type**	**Validations**	**Required**
1	_id	ObjectID	Auto-generated	-
2	name	String	-	TRUE
3	password	String	-	TRUE
4	tokens	Array of objects	Each object should have the following properties -> token(JWT token)	TRUE
5	__v	Number	Auto-generated	-

players				
Sl. No.	**Name**	**Type**	**Validations**	**Required**
1	_id	ObjectID	Auto-generated	-
2	name	String	Should contain only alphabets, should contain atleast 3 characters	TRUE
3	age	Number	Min = 15	TRUE
4	type	String	Can have the values ‘Batsman’ or ‘Bowler’ or ‘All-rounder’	TRUE
5	bats	String	Should have the values of either ‘Right’ or ‘Left’ or ‘NA’	TRUE
6	bowls	String	Should have the values of either ‘Right’ or ‘Left’ or ‘NA’	TRUE
7	bowling_style	String	Can have the values ‘Fast’ or ‘Medium’ or ‘Spin’ or ‘Leg-spin’ or ‘Chinaman’	TRUE
8	bat_avg	Number	Default = 0	FALSE
9	bowl_avg	Number	Default = 0	FALSE
10	bat_strike_rate	Number	Default = 0	FALSE
11	bowl_strike_rate	Number	Default = 0	FALSE
12	catches	Number	Default = 0	FALSE
13	run_outs	Number	Default = 0	FALSE
14	thirtys	Number	Default = 0	FALSE
15	fifties	Number	Default = 0	FALSE
16	centuries	Number	Default = 0	FALSE
17	three_WH	Number	Default = 0	FALSE
18	five_WH	Number	Default = 0	FALSE
19	highest_runs	Number	Default = 0	FALSE
20	best_bowling	String	Default = ‘0/0’	FALSE
21	overseas	Boolean	Default = false	FALSE
22	displayed_count	Number	Default = 0	FALSE
23	unsold	Boolean	Default = true	FALSE
24	base_price	Number	Min = 1000000, Max = 20000000	FALSE
25	sold_price	Number	Default = 0	FALSE
26	bought_by	String	Default = “”	FALSE

27	bidded_by	String	Default = “”	FALSE
28	__v	Number	Auto generated	-

Middlewares:

- There is a single middleware file namely **adminAuth.js** that resides inside **src/middlewares**. The file should authenticate the endpoints with **JWT Token Authentication**.
- Use **_id of the admin** and **xeEo2M0ol8CeWr7Nw2g2GjH8QEUK4dyyKCHi4TYJK6znm5fuAHIIPHSQ5YvdVcLlnaxppN64xK6xbhRileWvIlzCEqrBMCiITD8z** as the secret token to generate JWT tokens for admin inside adminAuth.js.

If the authentication was not successful for any of the routes that should be authenticated, then you should send a response code **400** with the following response.

```
{
  "error": "Please Authenticate"
}
```

Routers:

There is a single file namely **admin.js** that resides inside **src/routers**. The endpoints marked in red should be authenticated using **adminAuth.js**. The endpoints and their functionalities are as follows.

admin.js:

1) ***/login*** **-> POST Method ->** This route should verify the credentials (name and password) of the admin and if valid, a JWT token should be generated and should be concatenated with the tokens array of the admin.

Sample data sent with the request:

```
{
  "name": "admin@ipl.com",
  "password": "Password@12"
}
```

- If the authentication was successful, then you should send a response code of **200**.
- If something went wrong and the login was unsuccessful, then you should send a response status code of **400**.

2) ***/addPlayer*** **-> POST Method ->** This route should validate the data that comes with the request body and if valid, you should create a new document inside the players collection.

Sample data sent with the request:

```
{
    "name": "player new",
    "age": 30,
    "type": "All-rounder",
    "bats": "Right",
    "bowls": "Right",
    "bowling_style": "Medium"
}
```

- If the data got saved successfully inside the database, then you should send a response code of **200**.
- If something goes wrong in adding the player to the database, then you should send a response code of **400**.

3) ***/viewPlayer/:id*** **-> GET Method ->** This route should fetch the details of the player from the players collection that has _id equal to the id that comes with the request URL.

- If the data is fetched successfully from the database, then you should send a response code of **200**.
- If something goes wrong and the data is not fetched successfully from the database, then you should send a response code of **400**.

4) ***/editPlayer/:id*** **-> PATCH Method ->** This route should update the details of the document inside the players collection that has _id equal to id that comes with the request URL with the data that comes along the request body.

- If the updation was successful, then you should send a response code of **200**.
- If something goes wrong and the updation is not successful, then you should send a response code of **400**.

5) ***/deletePlayer/:id*** **-> DELETE Method ->** This route should delete the document from the players collection that has the _id equal to the id that comes with the request URL.

- If the deletion was successful, then you should send a response code of **200**.
- If something goes wrong and the deletion is not successful, then you should send a response code of **400**.

6) ***/viewPlayers/:teamName*** **-> GET Method ->** This route should fetch the documents from the players collection which has the bought_by equal to the teamName that comes with the request URL.

- If the data is fetched successfully from the database, then you should send a response code of **200**.
- If something goes wrong and the data is not fetched successfully from the database, then you should send a response code of **400**.

7) ***/playerBought/:id*** **-> PATCH Method ->** This route should update the player from the players collection that has _id equal to the id that comes with the request URL. The properties to be updated are as follows.

- The unsold of the player should be updated to false.
- The bought_by of the player should be updated with the bidded_by of the player.
- If the updation is successful, then you should send a status code of **200.**
- If something goes wrong and the data is not updated, then you should send a status code of **400.**

8) ***/players/bid/:id*** **-> PATCH Method ->** This route should update the sold_price and bidded_by of the player as follows.

- You should update the bidded_by of the player with the teamName that comes with the request body.
- If the sold_price of the player is 0, then you should assign the base_price as the sold_price.
- Then depending upon the value of the sold_price you should update it.
- If 1000000 <= sold_price <10000000, then you should add 500000 to the sold_price.
- If 10000000 <=sold_price < 50000000, then you should add 1000000 to the sold_price.
- If 50000000 <= sold_price < 100000000, then you should add 2500000 to the sold_price.
- If 100000000 <= sold_price < 200000000, then you should add 5000000 to the sold_price.
- If 200000000 <= sold_price, then you should add 10000000 to the sold_price.

- If the bid was added successfully, then you should send a response code of **200**.
- If something goes wrong, then you should send a response code of **400**.

9) ***/displayPlayer/:count*** **-> GET Method ->** This route should fetch the data from the players collection that has the highest base_price, unsold as true, displayed_count equals the count that comes with the request URL and type equal to the type that comes with the request query parameter(type). Also, this count should increment the displayed_count of the player by 1.

Sample request: /displayPlayer/0?type=Bowler

This request should fetch the data with type bowler, has displayed_count as 0 and has the highest base_price among the unsold players.

- If the route was executed successfully, then you should send a response code of **200**.
- If something goes wrong, then you should send a response code of **400**.

MongoDB commands:

- You can open the mongo shell by running ***mongo*** from the terminal.
- You can view all the data from the database in MongoDB by running ***show dbs*** from the mongo shell.
- You can select the database by running **use ipl-auction**.
- You can view the names of collections by running ***show collections***.
- You can view the data inside a collection by running ***db.collection_name.find().***
- Enter ***ctrl+c*** to exit.

Steps to be followed:

- If you face any difficulties with respect to the dependencies, click on **Run-> Install** to re-install the dependencies (If you are not getting **starting database mongodb** at the end of the process click the Install button again).
- Once you are done writing your code, you can run the project by clicking **Run-> Run.**
- To run the test cases click **Run Tests** button at the right bottom of the online IDE.
- After you are done with your test click the **Submit Test** button at the bottom right corner of the online IDE.

Software Instructions

The question(s) requires **Node 8 LTS or above**.

- Download & Install Node.JS

Git Instructions

Use the following commands to work with this project

run Copy

```
node_modules/.bin/pm2 stop && node_modules/.bin/pm2 start -f src/index.js --watch;
```

test Copy

```
npm run test;
```

install Copy

```
npm install && sh dbinstall.sh;
```