Swish Analytics Engineering Project

This is the Swish Analytics Engineering Project. There are 2 parts to this project. In the first part you will develop a REST API to efficiently return a JSON object based on specific query parameters. The second part focuses on the system implementation of an application in production. Please deliver all of the code, diagrams and explanations for both parts in a zip. s

Part 1

One of the core products at Swish is our REST API. Use the Restify API Framework for your API. If you would like to use a different framework you may, however be sure to explain your reasoning. Your API should query against the provided JSON file of player props. For added functionality, your API should take in query parameters playerId (int) and eventId (int) and return the data that meets those requirements. Here is a sample request for Steph Curry /.../...?playerId=338365 and the response body below.


```
[
 {
  "id": 4336379,
  "season": 2017,
  "date": "2017-10-17",
  "event_id": 1947346,
  "team_id": 9,
  "team_abbr": "GS",
  "opp_id": 10,
  "opp_abbr": "Hou",
  "player_id": 338365,
  "name": "Stephen Curry",
  "primary_pos_abbr": "PG",
  "stat_type": "steals",
  "projection": 2.18,
  "line": 2
 }
]
```


To find the data that contains the requested parameters, build your own sort function and a search function using basic JS operations like for loops, while loops, and if statements. Do not use functions or methods like filter , reduce , map , or others. Your sort function should implement your choice of

selection sort, insertion sort, bubble sort, merge sort or quick sort. For your search function, implement a searching algorithm like linear or binary search. These 2 functions should work together to query the provided data set in original form and return the requested data. At the time of request, provide a way to dictate that the response object be sorted by playerId.

Your API should successfully handle requests like:

/.../...?playerId=338365

/.../...?eventId=1947132

/.../...?playerId=338365&eventId=1947132

/.../...?playerId=338365&eventId=1947132&sortBy=desc

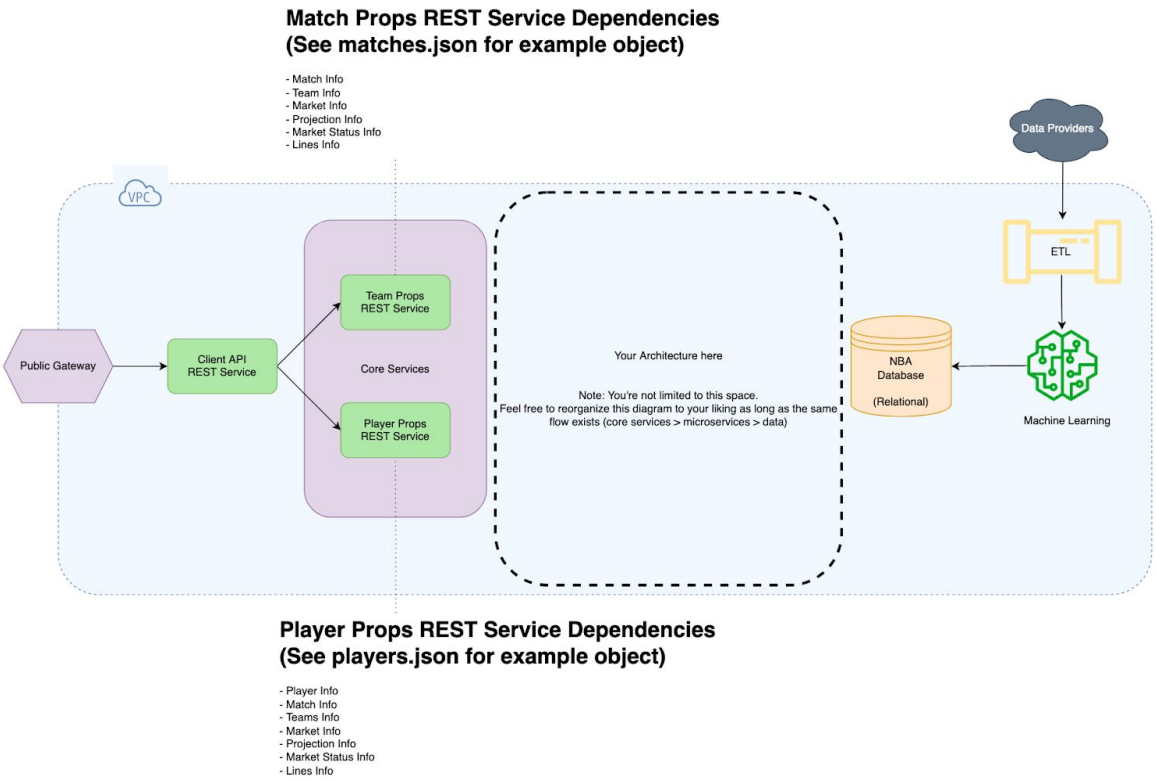
Return your code and a readme explaining the decisions you've made and why. Be sure to touch on the details of your sorting and searching functions.

Also, be sure to use npm to store your projects dependencies (npm init then npm install --save-prod restify) and expose a startup command. Do note, this is not a part of the assessment per se but rather allows you to have a clear point of entry to your project. If you need assistance with this please reach out.

Finally, provide at least 5 sample http requests you would like to use to test. We will test others, but this will be a great starting point.

Part 2

Understanding the underlying architecture and how services work together is key to a successful microservice strategy. In this portion of the test, you will design a microservice architecture that powers two core services. The diagram below is the starting point of your architecture. Provide documentation explaining what you did and why, as well as your considerations and general thought process. Also include your completed architectural diagram. Below are specific tasks to be completed.



Tasks:

1. Use the included diagram xml and upload it to http://diagrams.net/
    a. Start > decide later > File > Open from > Device… > select diagram.xml from the part-2 folder in your zip
    b. You will need the working diagram, so let us know if you have any issues . If you like, you can replicate this on your own, though it's not recommended.

2. Create the necessary microservices that efficiently provide the required dependencies for each Core service. For each microservice document its purpose, request parameters, and

response data. There's no need to dive too deep into the data beyond getting the point across to us. You will not be assessed on your knowledge of the data outside of the examples provided.

3. Develop a caching layer and strategy that each microservice will use. Document how it works.

   Questions to consider: Why did you choose this strategy & implementation? How is the data filled and managed? How might your strategy scale as demand increases and decreases?

4. Lastly, architect your diagram for a cache failure. Caches can fail physically or the data can become stale, both are considered as outages.

   Questions to consider: How are cache failures detected? Is the resolution manual or automatic? When this does happen how will your Core services remain operational providing the correct data?

Extra Credit:

Cost is a function of efficiency. A microservice architecture could be blazing fast but also extremely expensive, hurting the overall efficiency grade. Provide your thoughts on the relative cost of your architecture. Consider the following:

- What do you think are the most expensive areas of your architecture?
- Where could money be saved in your architecture and how?
- Where are the areas that might be costly, but are justified based on their importance?
- Assuming your architecture can service 100 requests per second before response times begin to slow, where would you scale up first to support 200 requests per second? (Make whatever assumptions you like, but let us know what they are)