* Is the service RESTful? If not, why not? If so, what key features make it RESTful?
  + Assuming that the Lab 09 template provided to us is RESTful, then our team’s RESTful web service based on that template is also RESTful.
  + I altered the template from the baseline to include a total of 4 DAOs and 4 resource classes. They are:
    - Player.java DAO, PlayerResource.java
    - Sport.java DAO, SportResource.java
    - Match.java DAO, MatchResource.java
    - Follow.java DAO, FollowResource.java
  + Based on the Guide, a RESTful web service has the features:
    - Use HTTP methods explicitly.
    - Be stateless.
    - Expose directory structure-like URI’s
    - Transfer XML, JavaScript Object Notation (JSON), or both.
  + I would say our web service satisfies all 4 of these criteria insofar as I can tell.
  + It is definitely transferring JSON data as that is how we retrieve the Sport data using Volley in order to populate the Sport Selection Activity.
  + We definitely expose and use URI’s as our endpoints:
    - <https://calvin-cs262-fall2018-pilot.appspot.com/knightranker/v1/players>
    - <https://calvin-cs262-fall2018-pilot.appspot.com/knightranker/v1/sports>
    - <https://calvin-cs262-fall2018-pilot.appspot.com/knightranker/v1/matches>
    - <https://calvin-cs262-fall2018-pilot.appspot.com/knightranker/v1/follows>
    - These retrieve lists of all the entries in each of our 4 relations.
  + Requests are stateless, they’re independent of each other and are added to a request queue using Volley
  + I am using GET, PUT, POST, and DELETE explicitly as they are intended. There are test activities that test each one of these CRUD operations for each of our 4 tables in the Knight-Ranker application.
    - GET retrieves data
    - POST creates a new tuple.
    - PUT modified a tuple.
    - DELETE deletes a tuple.
* What endpoints implement actions that are idempotent and/or nullipotent?
  + Nullipotent
    - All GET operations for Player, Sport, Match, and Follow Tables.
  + Idempotent
    - All GET, DELETE, and PUT operations for Player, Sport, Match, and Follow Tables.
  + POST operations for Player, Sport, Match, and Follow Tables are neither idempotent nor nullipotent.
* What evidence, if any, do you see of the commonly-cited impedance mismatch problem in object-relational information systems?
  + I actually encountered an impedance mismatch problem in the Sport DAO object when one of my team members was implemented a GET operation to retrieve data from the Sport table to populate our Sport Selection List Activity.
  + In the Sport DAO I had the constructor take in an ID field, type field, and then a name field, in that order. This order was reversed from what was in the Sport Table in our data base schema, which takes in an ID field, name field, and then a type field, in that order.
  + So, when we implemented the GET request in Sport Selection, retrieved the JSON data, and parsed it for the name field in order to populate the selection list, it was populating it with type names instead of sport names.
  + This is my understanding of what went wrong and the issue was promptly fixed. So, now I know order matters with the DAOs with the constructors.
* Would the web service built in the lab be susceptible to SQL-injection attacks?
  + I honestly don’t know enough about databases and web services to say either way.
  + According to this link, there’s definitely security guidelines to follow:
    - <https://www.owasp.org/index.php/REST_Security_Cheat_Sheet>
  + So, I suppose depending on whether or not these security features are implemented our web service could be susceptible to SQL injection attacks.
  + All I did was modify the provided template and add more methods to handle each of our tables, so if the template didn’t come with those security features then yes it would be susceptible to attacks.