**For the API Package:**

From a design perspective, I’m noticing the following:

* Most of the populate functions share some core code (GSON builder setup and fromJson call)
  + It would be good if there was a class that abstracted this away and just had a Singleton GsonBuilder for the lifetime of the program
    - That class could have a method which takes a response string & class object and return a generic that could be casted wherever the function was called
    - This would make most of the populate functions focus on the logic of the population and not library interactions
* The functionality of all \*\*\*\*\*API.java classes could/should be incorporated into their companion \*\*\*\*\*Data.java class.
* The extra\_info classes feel awkward and it’s unclear why they can’t be incorporated into the existing data objects. GSON will ignore extra data members if it doesn’t have something with the same key so you could easily populate these members, there, with an extra function call and remove these classes entirely.
* I would make an effort to make naming consistent with the model. Class names don’t really matter that much to GSON as much as the underlying structure and variable / member names.
* In an ideal world, we would use gson annotations to do custom deserialization and enforce our own variable naming too.

All that being said, there’s little we can do with the time we have available. If we have time at the end of the Sprint I would suggest making the first three bullet points a priority because it would make this directory a lot cleaner & easier to follow.

From a functional perspective:

* Everything seems to work. Without digging into the Taiga API and seeing what’s available under the hood it’s difficult for me to make a determination that data is collected as it should be so I’m generally trusting of the demonstration available in MainConnection.java
* I do think there are a lot of setters that might be unnecessary with GSON’s approach to reflection / instance creation and that could clean up the code further.

If we have the time to clean up this code at the end of the Sprint, I think that would be great. There are 20-some files in the directory and that could probably be cut in half without too much effort. The content of each file could be pared down if some common patterns are abstracted.

Maybe during that refactor the end result can be documented, too, as I’m sure that would go a long way in making it easier to read / follow.

All in all good work. This will definitely do the job.

**For the Integration:**

So, we’ve already fixed the timelines at each level of the model. The same basically needed to be done for collaborators (ie. each level shouldn’t only have collaborators associated with that level and no more). I think I did do this but haven’t really checked the logic extensively so when you look at this code again please take a look at that.

Otherwise, I think there are some holdover functions / data members in a lot of these classes from when you were figuring out how to integrate -- would be good to clean this up if we can but unnecessary for now.

We’ve also noted that Task’s claimant isn’t being set properly and that’s a problem!