1. GraphQL first design can aide the front-end team by only fetching the data that is needed. The front-end team does not need to worry about how the data needs to be fetched, they can simply focus on what data is needed for their clients.
2. The ability to pass multiple arguments with a query allows for more compact/streamlined code as you can retrieve all the data you need with a single query. In a REST API, you are only able to pass a single set of arguments (the query parameters and URL segment), meaning you would need several queries to return the needed data. With GraphQL, multiple arguments can be passed in a single query, allowing for a single query/response pair to capture all the needed data. (Code example?)
3. GraphQL was initially developed internally by Facebook in 2012 before being released to the public in 2015. GraphQL wanted to address a common problem with REST APIs and that is over-fetching and under-fetching. In a typical REST API, the user may need several GET requests and those requests may return some unneeded data as well. GraphQL addresses this by allowing clients to receive the specific data they need, in one request.
4. The *errors* entry will be present in the JSON response if any errors were encountered. The *errors* entry will contain a *message* key which contains a string description of the error so the developer can understand and correct it. The response will also contain a *locations* key which shows the line and column that the error occurs.
5. Introspection allows the developer to query GraphQL about the schema/system configuration. For example, one could use *{ \_\_schema { types { name }}}* to return the names of all the types that are available in the schema. It can also be used to return information about specific objects. For example, *{ \_\_type(name: “Barbarian”) { name fields { name type { name kind }}}}* would return all the fields that were defined for the *Barbarian* object.