Updating GraphQL in the Android Project

Update the PickingApp's GraphQL schema

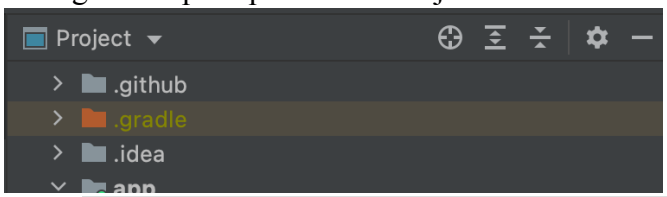
- Clone the InternalAPI repo and pull latest changes
  - https://github.com/morsco-reece/internal-api
- Go to the git changes for your feature
  - EXAMPLE:


```
{
  "name": "closeTask",
  "description": null,
  "args": [
    {
      "name": "input",
      "description": null,
      "type": {
        "kind": "NON_NULL",
        "name": null,
        "ofType": {
          "kind": "INPUT_OBJECT",
          "name": "CloseTaskInput",
          "ofType": null
        }
      },
      "defaultValue": null
    }
  ],
  "type": {
    "kind": "NON_NULL",
    "name": null,
    "ofType": {
      "kind": "OBJECT",
      "name": "CloseTaskResponse",
      "ofType": null
    }
  },
  "isDeprecated": false,
  "deprecationReason": null
},
```

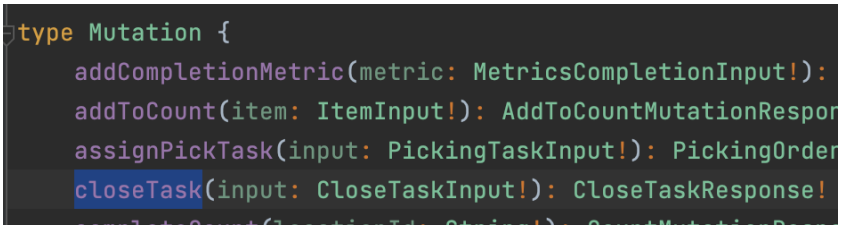

  - Make note of the “name” field. This will become the method name for the new query/mutation.
- Open and copy `graphql.schema.json` file contents.

Update Android Studio graphql schema.json file

- Change the top dropdown to “Project”



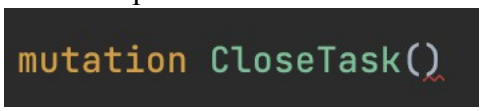
- Open `app/src/main/graphql/com/reece/pickingapp/schema.json`
  - Replace file contents with the copied changes from `graphql.schema.json`
- CLEAN and BUILD project
  - This is CRUCIAL so that GraphQL generates the types and inputs from the update.
- Look for the change
  - Easiest way is to open a current .qraphql file (like `PickingTasks.graphql`).
  - command click on an inner method (ie `pickingOrders`)
  - This opens the generated GraphQL readable file from the .json
  - Search for the new method “name” from the .json file



  - This shows the generated input(s) and response for the new query/mutation

Add the Mutation/Query

- Open / create .graphql file associated with the change.
  - i.e. If it has to do with Picking Tasks, put it in the `PickingTasks.graphql` file.
- Inside this file, create the GraphQL Class
  - *Prefix Type*: *query or mutation* - This is determined from the `type` in the GraphQL Schema file from the above step.
    - Example: the `closeTask()` method in above image is in the `Mutation` type
  - *Name*: Capitalized and camelcase. Use the method name.



  - *Arguments*: copy the arguments from the GraphQL Schema and paste them in
    - place $ in front of arg name(s)

Create the Method

NOTE: This is done INSIDE the class you just created in the above step.

- *Method Name*: should be the same as the "name" from the schema
- *Arguments*: arg names are the same as the class, but the value is the passed class arg
  - NOTE: Android Studio should autofill these

```
closeTask(input: $input)
```

- *Response*: Use the generated GraphQL method and command click on its response

```
type Mutation {
    addCompletionMetric(metric: MetricsCompletionInput!):
    addToCount(item: ItemInput!): AddToCountMutationRespor
    assignPickTask(input: PickingTaskInput!): PickingOrder
    closeTask(input: CloseTaskInput!): CloseTaskResponse!
```

  - I.e in image above you would Command click on `CloseTaskResponse`

```
type CloseTaskResponse {
    message: String!
    success: Boolean!
}
```

  - Use the arg *names* from the response

```
mutation CloseTask($input: CloseTaskInput!) {
    closeTask(input: $input) {
        message
        success
    }
}
```

```
26
27  type CloseTaskResponse {
28      message: String!
29      success: Boolean!
30  }
31
```

    - NOTE: you do not need to put the arg type or commas
- CLEAN and BUILD project

Repository Interface

- Open `PickingRepository`
- Create a corresponding method (it must be a `suspend` function)
  - Name it according to the GraphQL `type` (i.e. `query` or `mutation`)

```
suspend fun mutationCloseTask()
```

  - Pass in args (typing them in will auto-complete)

```
suspend fun mutationCloseTask(input: CloseTaskInput)
```

  - Add `response: Response<T>`

```
Response<CloseTaskMutation.Data>?
```

    - Type `<T>` should be `{GraphQL Class Name}{GraphQL Class Type}.Data`
  - Final result

```
suspend fun mutationCloseTask(input: CloseTaskInput): Response<CloseTaskMutation.Data>?
```

Repository Implementation

- Open `PickingRepositoryImpl`
- Create a corresponding override method
  - NOTE: Android Studio should auto complete this.

```
override suspend fun mutationCloseTask(input: CloseTaskInput): Response<CloseTaskMutation.Data>? {
    TODO( reason: "Not yet implemented")
}
```

- Add method body of return
  - Start with `return pickingApi.getApolloClient()?`
  - Add `mutate` or `query` depending on what the method is.

```
return pickingApi.getApolloClient()?.mutate()
```

  - Pass the GraphQL `mutation` or `query`
    - NOTE: this will be the same as the Response Type (without `.Data`)

```
return pickingApi.getApolloClient()?.mutate(CloseTaskMutation())
```

  - Pass in the method's args

```
return pickingApi.getApolloClient()?.mutate(CloseTaskMutation(input = input))
```

  - Add `.await()` at end of method

```
return pickingApi.getApolloClient()?.mutate(CloseTaskMutation(input = input))?.await()
```

    - NOTE: if you have errors in the method return make sure you have added appropriate safe (?) call (there are TWO needed: one after `getApolloClient()` and one after `mutate()`

You are now able to use this PickingRepository method in your ViewModels