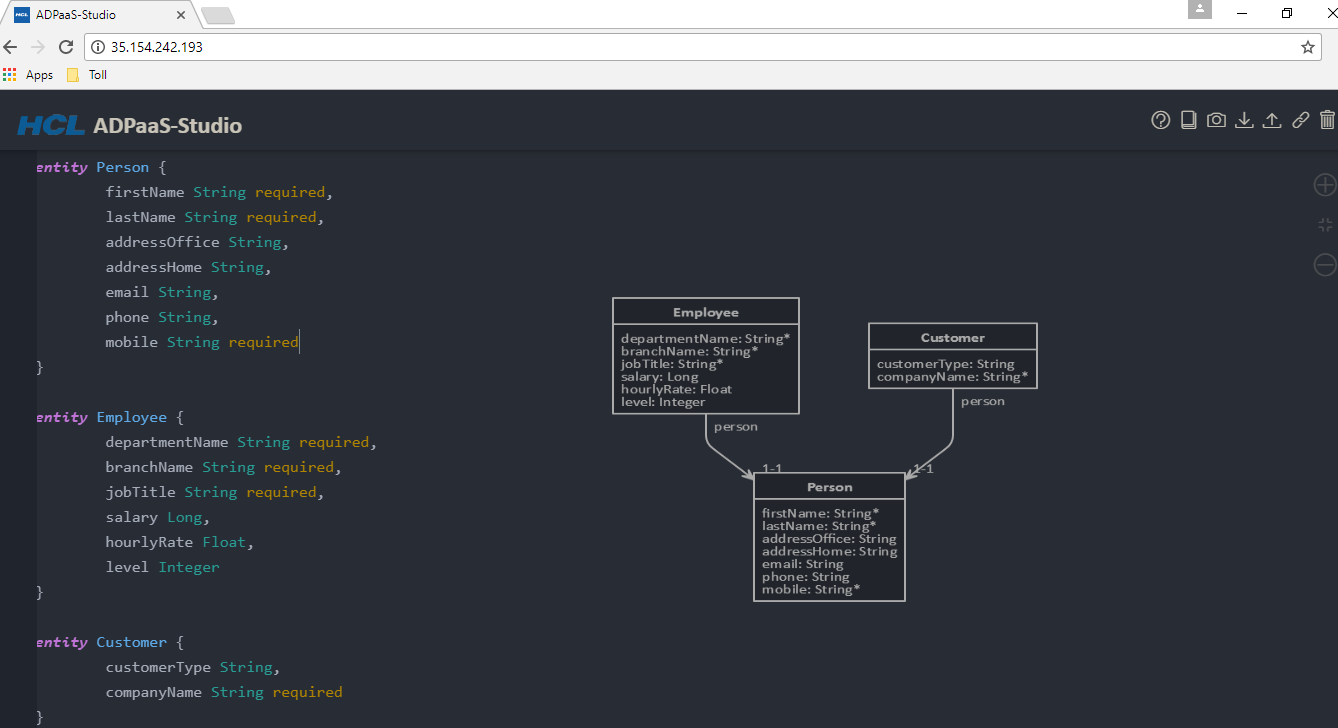
**Code Generation**

**process**

Created by: Suryaprakash Raghuwanshi

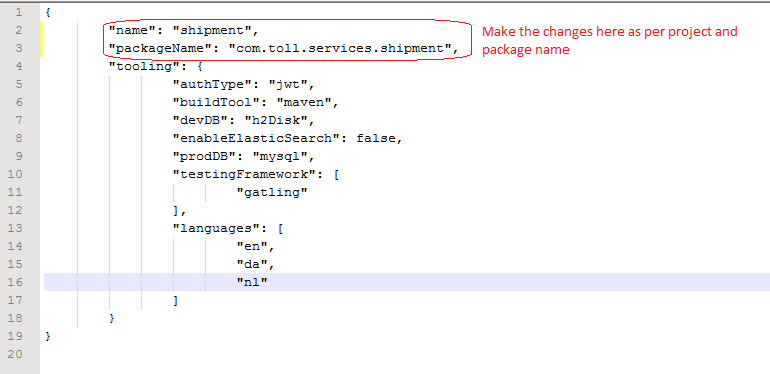
**Step1: Access to the ADPaaS URL’s as below**

* 1. Studio : <http://35.154.242.193/> This is hosted via AWS EC2 instance given for ADPaaS.

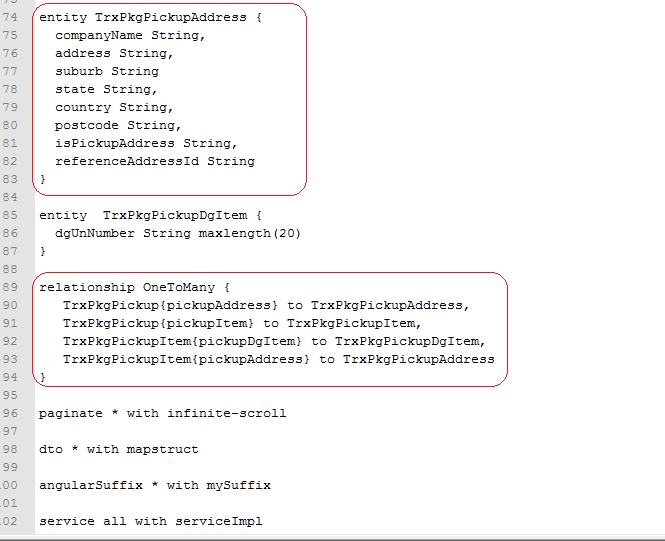


Step2: Moded your entities using AD-Studio.

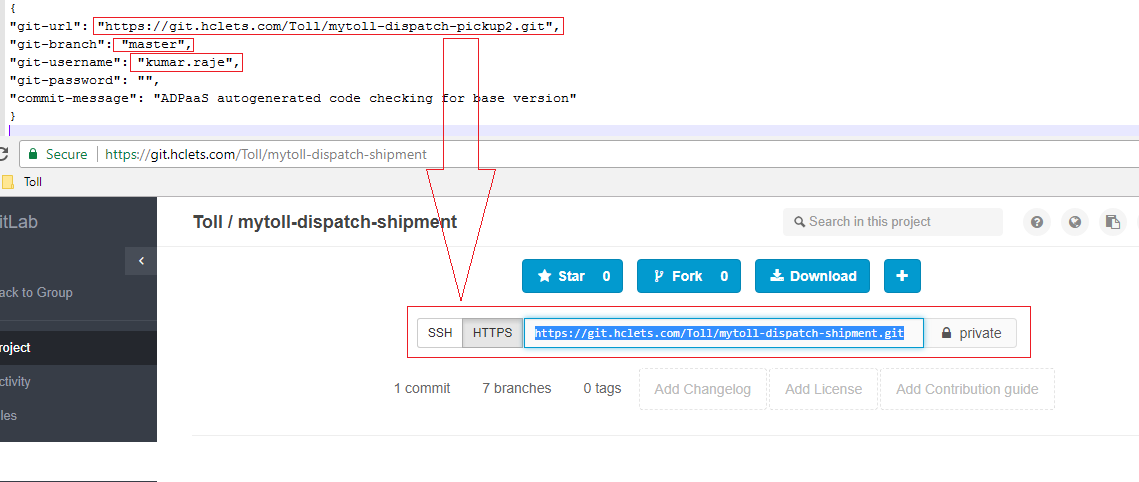
1. Get the entity model for your service.
2. Open the studio and start creating the. jh in studio. One sample model is placed in studio which can be referred as an example.
3. Some basic rules for writing the model in studio.
   * 1. Each table from DB mode become entity in JDL model.
     2. JDL model entity name should follow the “InitCaps” format. So each table name needs to be modified to InitCaps format and to be used as name of Entity in JDL model. Do not worry while generating JPA layer table name would be converted back to “\_”.
     3. Each column in table would become attribute of entity in JDL model and JDL attributes follow the CamelCase format.
     4. Take each column from table convert into the initCaps and add into JDL entity for that table. For datatypes supported please see appendix below.
     5. Following columns needs not to be added into JDL
        1. Primary keys – It would be auto generated by name of ID.
        2. Foreign Keys - It would be auto generated using format “{Foreign\_KEY\_TABLE\_Name}\_ID” which is Foreign keys table name appended by “\_ID”.
        3. Auditing Columns like CreatedDate, UpdatedDate, CreatedBy, UpdatedBy.
     6. Now create the relation between tables. Relations supported are OneToOne, OneToMany, ManyToMany.
     7. Please see the below example for various scenarios
        1. OneToOne - PrimaryTable{RelationName} To Secondary Table. E.g Customer{homeAddress} to Address
        2. OneToMany - PrimaryTable{RelationName} To Secondary Table. E.g Pickup{pickUpItem} to PickUpItems
     8. Each type of these relations has to be mentioned in their own group.
     9. Ensure that there are no errors in the studio.
4. **Step 3: Create configuration files for API.**
   1. Micro Service code generation.
      1. ***ms.json :*** Create a ms.json file for your microservice. Sample is attached. Team needs to update the name of microservice and java package.



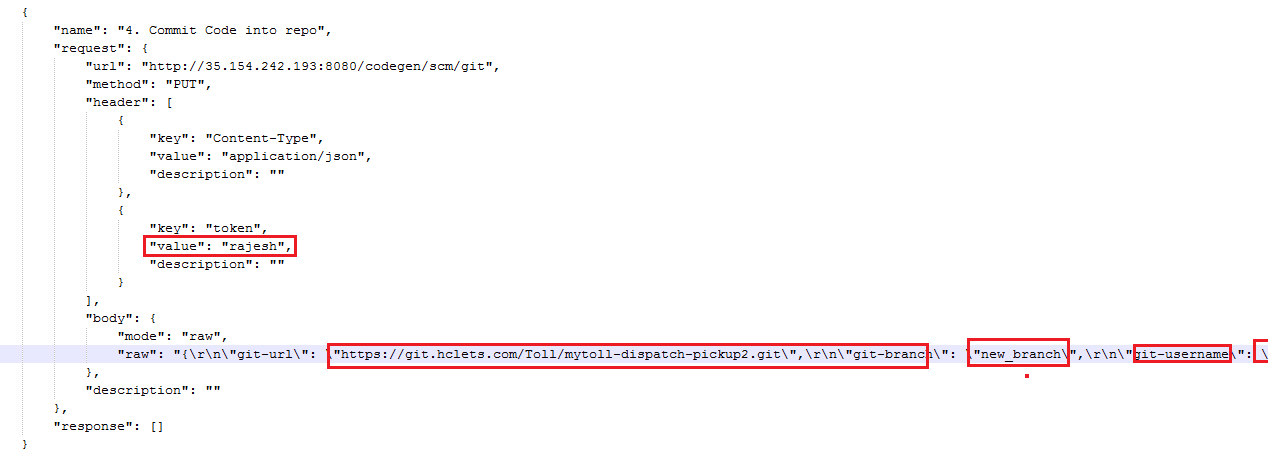
* + 1. ***entity.jh***: Use guidance provided in step2 to create the jh file.
  1. SCM Commit configurations



* + 1. ***Scm.json file***: Create a scm.json file to push the code into the GIT report.
    2. Use new branch to push the code into git. No need to pre create the branch.
    3. In the password, if you have @ then request might fail. So please ensure that you do not have @ in password.



iV. ***Toll\_ADPaaS\_API\_Collection.json*** file: used for postman, we needed to make below changes as per project requirements. (this file will generate four request)



Make bellow this change in ADPaaS file.

1. Value :<> any value you can provide here.
2. GIT URL: <>Pass the project specify https URL here.
3. Username:<> Pass git username here.
4. Password:<> Pass git Password here
5. **Invoke ADPaaS API to get the code generated**
   1. Attached is the PostMan collection which can be imported into Rest client. Import it into Rest
   2. Provides the details for various API.
   3. Team needs to update the header field with name “token” . It should be unique for service that you are generating.
   4. This “token” filed needs to be updated in all 4 requests in collection.
   5. Microservice POST is multipart file request. You need to selcted the correct file against correct attribute.
   6. Run the API one by one and at the end you should have the code in GIT.

