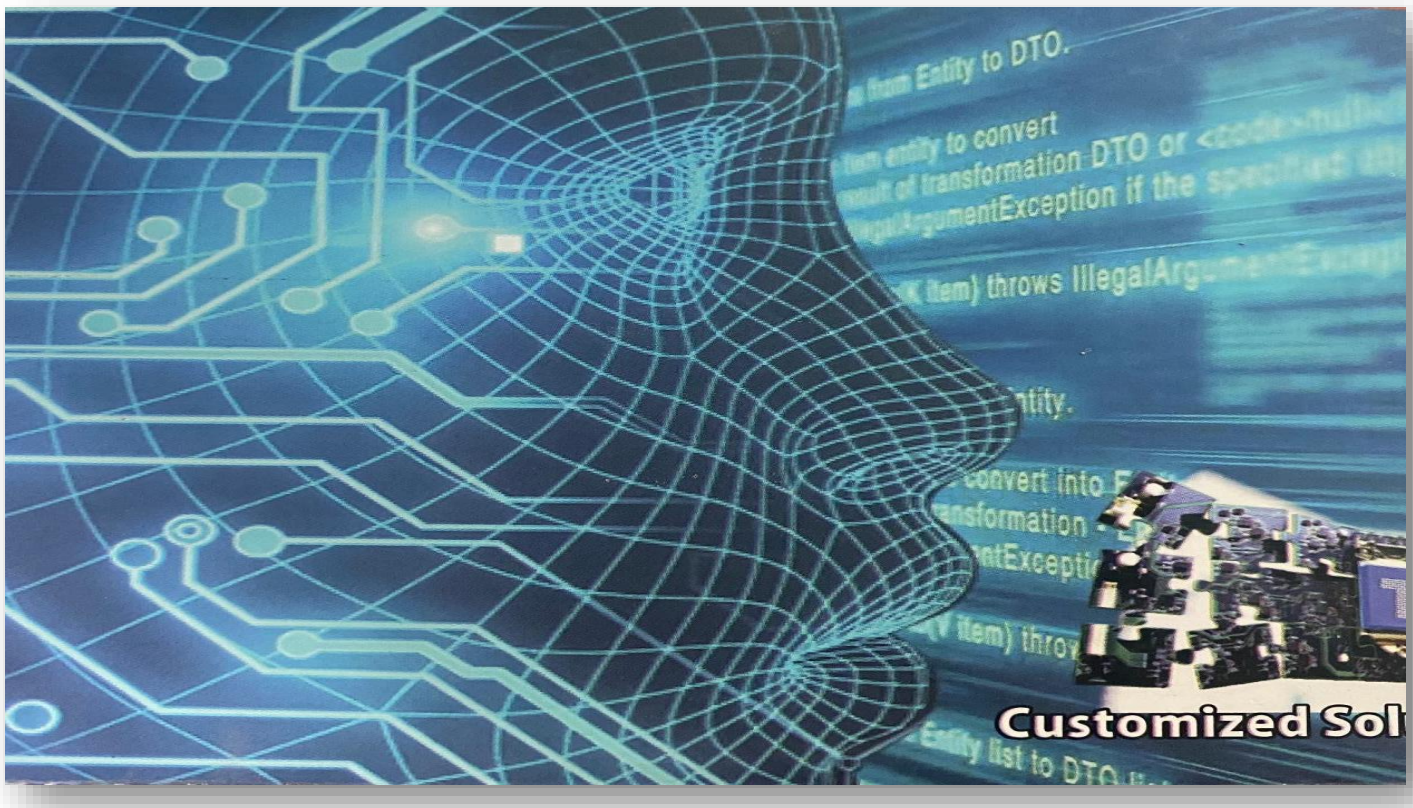
Development and Coding Conventions Guideline

TERESOL
Technological Research Solutions



Document Information

Category	Information
Document	Development and coding conventions guideline document
Document Version	2.2
Identifier	
Status	Developing
Author(s)	Dayyan Satti, Azhar Hussain, Sundus Zulfiqar
Reviewer (Initial)	Muhammad Bilal
Reviewer (Final)	Dr. Muqeem Sheri
Approver(s)	Dr. Faisal Khan
Publish Date	1/25/2022
Distribution	TeReSol Pvt Ltd (Software Department)

Document Revision History

Author	Date	Version	Description
Dr. Muqeem Sheri	25-Jan-2022	1.0	Made amended and changes to format with more additions of headings
Dr. Muqeem/ M. Bilal	26-Jan-2022	2.1	

1. Introduction:

This document has been introduced as an detailed guideline for the developers. Developers must adhere the coding conventions and templates defined in order to make sure the alignment within teams and making sure the similar patterns across different teams. This document shall also serve the purpose of single repository for coding & development requirements.

2. Document Convention:

[Optional]Template.java: anything in [] brackets is optional and must follow the Type-case as shown.

Example:

[Optional] can be replaced with empty string Template.java is valid.

[Optional] can be replaced with ServiceTempalte.java where optional is being replaced by Service.

<Required>Template.java: anything in <> brackets is required and must follow the Type-case as shown.

Example:

<Required> can not be replaced with empty string. Template.java is not valid.

<Required> is replaced with any service name. BatchTemplate.java is valid.

Important Notes:

*Examples given in the document are of Batch service.
XXX & YYY has to be validated from Dr. Faisal.

3. Core Services Template Code Structure

3.1. Repository Conventions:

Name conventions

git related conventions

3.2. Folder Structure

Note: All the folders inside are named in small alphabets.

1. The main folder name is the repository name and must be as core_api_<ServiceName>
2. Template for the structure can be downloaded from () and be customized and used for easy access.
3. Tree based folder structure with explanation.

- core_api_[ServiceName]
  - src
    - main
      - java
        - com/teresol/core/api/core_api_[ServiceName]
          - dto
          - exception
          - services
          - util
          - web
          - web-client

Note: All the folders inside are named in small alphabets.

1. dto
   1.1. Folder
      1.1.1. It contains DTOs required by the service. All Dtos must be only placed in the folder.
   1.2. File Names
      1.2.1. The filename for the Dto object must follow the following convention:
         1.2.1.1. <DomainName>[FunctionalDescription]Dto.java
            1.2.1.1.1. <DomainName> will be the name of the data object being maintained in this Dto
            1.2.1.1.2. [FunctionalDescription] will be a short description for specialized used of DomainName
            1.2.1.1.3. Examples:
               1.2.1.1.3.1. BatchDto.java
               1.2.1.1.3.2. BatchShortDetailDto.java

2. exceptions
   2.1. Folder
      2.1.1. This folder holds all custom exceptions that this service will use.
   2.2. File Names
      2.2.1. The filename for exception classes must follow the following convention:
         2.2.1.1. <ExceptionName>Exception.java
            2.2.1.1.1. <ExceptionName> is replaced with the name of exception.
         2.2.1.2. Example:
            2.2.1.2.1. ListNotEqualException.java

Note: Declare all the exception as checked exceptions by subclassing (). Please consult your team lead if there is a need to make an unchecked exception.

3. services
    3.1. Folder
        3.1.1. This folder has the java class in which the core logic for this service is implemented.
    3.2. File Names:
        3.2.1. The filename for service class must follow the following convention:
            3.2.1.1. Main<ServiceName>Service.java
                3.2.1.1.1. <ServiceName> is replaced with the name of service.
            3.2.1.2. Example:
                3.2.1.2.1. MainBatchService.java

4. util
    4.1. Folder
        4.1.1. This folder has the utility class in which the utility functions for this service are implemented.
    4.2. File Names:
        4.2.1. The filename for utlity class must follow the following convention:
            4.2.1.1. Utility.java

Note: There would only be one utility class. If another utility class is required, the team lead should be consulted first.

5. web
    5.1. Folder
        5.1.1. This folder has the main class that has all the exported http endpoints from this service.
    5.2. File Names:
        5.2.1. The filename for main class must follow the following convention:
            5.2.1.1. MainResource.java.
6. webclient
    6.1. Folder
        6.1.1. This folder contains the http client interface which is used to communicate to the any other service like other microservice or data access service through their http endpoints.
    6.2. File Names:
        6.2.1. The filename for main class must follow the following convention:
            6.2.1.1. Dataaccessapi<ServiceName>.java
                6.2.1.1.1. <ServiceName> is replaced with the name of service.
                6.2.1.1.2. Example:
                    6.2.1.1.2.1. DataaccessapiBatch.java
            6.2.1.2. Another file is included named as CustomRestClientBuilderListener.java.

4. Core Microservices Template Code Structure

4.1. Repository Conventions

Name conventions

git related conventions

4.2. Folder Structure

Note: All the folders inside are named in small alphabets.

1. The main folder name is the repository name and must be as core_ms_<XXX>
2. Template for the structure can be downloaded from () and be customized and used for easy access.
3. Tree based folder structure with explanation.

- core_ms_<XXX>
  - src
    - main
      - java
        - com/teresol/core/ms/core_ms_<XXX>
          - dto
          - exception
          - services
          - util
          - web
          - Web- client

Note: All the folder are named in small alphabets. All the classes follow the upper camel case for naming conventions.

1. dto
   1.1. Folder
      1.1.1. It contains DTOs required by the service. All Dtos must be only palced in the folder.
   1.2. File Names
      1.2.1. The filename for the Dto object must follow the following convention:
         1.2.1.1. <DomainName>[FunctionalDescription]Dto.java
            1.2.1.1.1. <DomainName> will be the name of the data object being maintained in this Dto
            1.2.1.1.2. [FunctionalDescription] will be a short description for specialized used of DomainName
            1.2.1.1.3. Examples:
               1.2.1.1.3.1. BatchDto.java
               1.2.1.1.3.2. BatchShortDetailDto.java

2. exceptions
   2.1. Folder
      2.1.1. This folder holds all custom exceptions that this service will use.
   2.2. File Names
      2.2.1. The filename for exception classes must follow the following convention:
         2.2.1.1. <ExceptionName>Exception.java
            2.2.1.1.1. <ExceptionName> is replaced with the name of exception.
         2.2.1.2. Example:
            2.2.1.2.1. ListNotEqualException.java

Note: Declare all the exception as checked exceptions by sub-classing (). Please consult your team lead if there is a need to make an unchecked exception.

3. services
    3.1. Folder
        3.1.1. This folder has the java class in which the core logic for this service is implemented.
    3.2. File Names:
        3.2.1. The filename for service class must follow the following convention:
            3.2.1.1. Main<XXX>Service.java
                3.2.1.1.1. <XXX> is replaced with the name of service.
            3.2.1.2. Example:
                3.2.1.2.1. MainBatchService.java
4. util
    4.1. Folder
        4.1.1. This folder has the utility class in which the utility functions for this service are implemented.
    4.2. File Names:
        4.2.1. The filename for utlity class must follow the following convention:
            4.2.1.1. Utility.java

Note: There would only be one utility class. If another utility class is required, the team lead should be consulted first.

5. web
    5.1. Folder
        5.1.1. This folder has the main class that has all the exported http endpoints from this service.
    5.2. File Names:
        5.2.1. The filename for main class must follow the following convention:
            5.2.1.1. MainResource.java.
6. webclient
    6.1. Folder
        6.1.1. This folder contains the http client interface which is used to communicate to the any other service like other microservice or data access service through their http endpoints.
    6.2. File Names:
        6.2.1. The filename for main class must follow the following convention:
            6.2.1.1. Dataaccessapi<XXX>.java
                6.2.1.1.1. <XXX> is replaced with the name of microservice.
                6.2.1.1.2. Example:
                    6.2.1.1.2.1. DataaccessapiBatch.java
            6.2.1.2. Another file is included named as CustomRestClientBuilderListener.java.

5. Data Access Services Template Code Structure

5.1. Repository Conventions

Name conventions

git related conventions

5.2. Folder Structure

Note: All the folders inside are named in small alphabets.

1. The main folder name is the repository name and must be as dataaccess_api_<ServiceName>
2. Template for the structure can be downloaded from () and be customized and used for easy access.
3. Tree based folder structure with explanation.

- dataaccess_api_[ServiceName]
    - src
        - main
            - java
                - com/teresol/dataaccess/api/dataaccess_api_[ServiceName]
                    - connection
                    - dto
                    - exception
                    - services
                    - querystores
                        - <TableName>queries
                    - util
                    - web

Note: All the folders inside are named in small alphabets.

1. connection
    1.1. Folder
        1.1.1. It contains the connection classes required by the service.
    1.2. File Names
        1.2.1. The filename for the connection provider class is as follows:
            1.2.1.1. <DBName>CredentialProvider.java
                1.2.1.1.1. <DBName> will be the name of the database required by the service.
                1.2.1.1.2. Examples:
                    1.2.1.1.2.1. Box1CredentialProvider.java
                    1.2.1.1.2.2. Box2CredentialProvider.java
            1.2.1.2. Another file is required named as PasswordDecrypter.java

2. Dto

2.1. Folder
2.1.1. It contains DTOs required by the service. All Dtos must be only palced in the folder.
2.2. File Names
2.2.1. The filename for the Dto object must follow the following convention:
2.2.1.1. <DomainName>[FunctionalDescription]Dto.java
2.2.1.1.1. <DomainName> will be the name of the data object being maintained in this Dto
2.2.1.1.2. [FunctionalDescription] will be a short description for specialsed used of DomainName
2.2.1.1.3. Examples:
2.2.1.1.3.1. BatchDto.java
2.2.1.1.3.2. BatchShortDetailDto.java

3. Exceptions

3.1. Folder
3.1.1. This folder holds all custom exceptions that this service will use.
3.2. File Names
3.2.1. The filename for exception classes must follow the following convention:
3.2.1.1. <ExceptionName>Exception.java
3.2.1.1.1. <ExceptionName> is replaced with the name of exception.
3.2.1.2. Example:
3.2.1.2.1. ListNotEqualException.java

Note: Declare all the exception as checked exceptions by sub-classing (). Please consult your team lead if there is a need to make an unchecked exception.

4. Services

4.1. Folder
4.1.1. This folder has the java class in which the data access logic for this service is implemented.
4.2. File Names:
4.2.1. The filename for service class must follow the following convention:
4.2.1.1. Main<ServiceName>Service.java
4.2.1.1.1. <ServiceName> is replaced with the name of service.
4.2.1.2. Example:
4.2.1.3. MainBatchService.java

5. Querystore

5.1. Folder
5.1.1. This folder contains sub-folders according to the names of the table of database.
5.1.1.1. Sub-folder
5.1.1.1.1. Folder
5.1.1.1.1.1. This follows the following naming convention:
5.1.1.1.1.1.1. <TableName>queries
5.1.1.1.1.1.1.1. <TableName> is replaced with the table name from database.
5.1.1.1.1.1.2. Example:
5.1.1.1.1.1.2.1. Batchtlqueries

5.1.1.1.2. File Names
5.1.1.1.2.1. The filenames for the class should be as follows:
5.1.1.1.2.1.1. <TableName>[FunctionalDescription]Query.java
5.1.1.1.2.1.1.1. <TableName> is replaced with the table name from database.
5.1.1.1.2.1.1.2. [FunctionalDescription] is replaced with the type of queries in the class.
5.1.1.1.2.1.1.3. Example:
5.1.1.1.2.1.1.3.1. BatchTlQuery.java
5.1.1.1.2.1.1.3.2. BatchTlUpdateQuery.java

6. util
   6.1. Folder
      6.1.1. This folder has the utility class in which the utility functions for this service are implemented.
   6.2. File Names:
      6.2.1. The filename for utlity class must follow the following convention:
         6.2.1.1. Utility.java

Note: There would only be one utility class. If another utility class is required, the team lead should be consulted first.

7. web
   7.1. Folder
      7.1.1. This folder has the main class that has all the exported http endpoints from this service.
   7.2. File Names:
      7.2.1. The filename for main class must follow the following convention:
         7.2.1.1. MainResource.java.

6. Data Access Microservices Template Code Structure

6.1. Repository Conventions

Name conventions

git related conventions

6.2. Folder Structure:

Note: All the folders inside are named in small alphabets.

1. The main folder name is the repository name and must be as dataaccess_ms_<YYY>
2. Template for the structure can be downloaded from () and be customized and used for easy access.
3. Tree based folder structure with explanation.

- dataaccess_ms_<YYY>
    - src
        - main
            - java
                - com/teresol/dataaccess/ms/dataaccess_ms_<YYY>
                    - connection
                    - dto
                    - exception
                    - services
                    - querystores
                        - <TableName>queries
                    - util
                    - web

Note: All the folders inside are named in small alphabets.

1. connection
    1.1. Folder
        1.1.1. It contains the connection classes required by the microservice.
    1.2. File Names
        1.2.1. The filename for the connection provider class is as follows:
            1.2.1.1. <DBName>CredentialProvider.java
                1.2.1.1.1. <DBName> will be the name of the database required by the microservice.
                1.2.1.1.2. Examples:
                    1.2.1.1.2.1. Box1CredentialProvider.java
                    1.2.1.1.2.2. Box2CredentialProvider.java
            1.2.1.2. Another file is required named as PasswordDecrypter.java

2. Dto

2.1. Folder
2.1.1. It contains DTOs required by the microservice. All Dtos must be only palced in the folder.
2.2. File Names
2.2.1. The filename for the Dto object must follow the following convention:
2.2.1.1. <DomainName>[FunctionalDescription]Dto.java
2.2.1.1.1. <DomainName> will be the name of the data object being maintained in this Dto
2.2.1.1.2. [FunctionalDescription] will be a short description for specialsed used of DomainName
2.2.1.1.3. Examples:
2.2.1.1.3.1. BatchDto.java
2.2.1.1.3.2. BatchShortDetailDto.java

3. Exceptions

3.1. Folder
3.1.1. This folder holds all custom exceptions that this microservice will use.
3.2. File Names
3.2.1. The filename for exception classes must follow the following convention:
3.2.1.1. <ExceptionName>Exception.java
3.2.1.1.1. <ExceptionName> is replaced with the name of exception.
3.2.1.2. Example:
3.2.1.2.1. ListNotEqualException.java

Note: Declare all the exception as checked exceptions by sub-classing (). Please consult your team lead if there is a need to make an unchecked exception.

4. Services

4.1. Folder
4.1.1. This folder has the java class in which the data access logic for this microservice is implemented.
4.2. File Names:
4.2.1. The filename for microservice class must follow the following convention:
4.2.1.1. Main<YYY>Service.java
4.2.1.1.1. <YYY> is replaced with the name of microservice.
4.2.1.2. Example:
4.2.1.2.1. MainBatchService.java

5. Querystore

5.1. Folder
5.1.1. This folder contains subfolders according to the names of the table of database.
5.1.1.1. Sub-folder
5.1.1.1.1. Folder
5.1.1.1.1.1. This follows the following naming convention:
5.1.1.1.1.1.1. <TableName>queries
5.1.1.1.1.1.1.1. <TableName> is replaced with the table name from database.
5.1.1.1.1.1.2. Example:
5.1.1.1.1.1.2.1. Batchtlqueries

5.1.1.1.2. File Names
5.1.1.1.2.1. The filenames for the class should be as follows:
5.1.1.1.2.1.1. <TableName>[FunctionalDescription]Query.java
5.1.1.1.2.1.1.1. <TableName> is replaced with the table name from database.
5.1.1.1.2.1.1.2. [FunctionalDescription] is replaced with the type of queries in the class.
5.1.1.1.2.1.1.3. Example:
5.1.1.1.2.1.1.3.1. BatchTlQuery.java
5.1.1.1.2.1.1.3.2. BatchTlUpdateQuery.java

6. Util

6.1. Folder
6.1.1. This folder has the utility class in which the utility functions for this microservice are implemented.
6.2. File Names:
6.2.1. The filename for utlity class must follow the following convention:
6.2.1.1. Utility.java

Note: There would only be one utility class. If another utility class is required, the team lead should be consulted first.

7. Web

7.1. Folder
7.1.1. This folder has the main class that has all the exported http endpoints from this service.
7.2. File Names:
7.2.1. The filename for main class must follow the following convention:
7.2.1.1. MainResource.java.

7. Development Environment Setup:

- IDE Dev- Vs code 1.63
- IDE DB-Beaver 21.3.2

7.1. Extension:

- SonarList
- OpenAPI(Swagger) Editor
- Dependency Analytics by Re Hat
- Language support for Java(TM) by Red Hat
- Doxygen Documentation Generator.
- Graphviz (installation from Terminal).
- Extension Pack for Java by Microsoft
- YAML by Red Hat

8. Code Commenting Rules:

8.1. Class Body Initialization:

```
/**
* This is main Class for calling BatchDA with seperate input and output parts of the query
* Service Names: BatchDetails
* Calling Classes: BatchDA.java & Batch.java
* @author     Dayyan Satti, Azhar Hussain
* @version    1.1
* @param  url  http://localhost:9081/batchDA/batchpost
* @return     Final Resultset in form of JsonArray.
*/
```

8.2. Library Detailed Description:

```
/**
* import libraries and classes:
* java.sql.SQLException: to throw SQL related exceptions being generated from the BatchDA.java
* java.util.List: Manage lists being passed as arguments
* javax.inject.Inject: to use injection of BatchCore class in this class
* javax.ws.rs.POST: to declare POST type request
* javax.ws.rs.Path: to declare of path of class or method
* javax.ws.rs.Produces: annotates that the method produces an output
* javax.ws.rs.core.MediaType: describes the type of output
* com.ibm.db2.cmx.internal.json4j.JSONArray: to use Json array functionality
* org.jboss.logging.Logger: to write logs when required
* BatchCore.BatchCore: to use an object for calling BatchCore method
* DTO.BatchListDTO: to use an object to recieve in dynamic method as parameter
* resource.BatchDA: to use batchDA object to call BatchDA method
*/
```

8.3. Method Detail Description:

```
/**Request type and path declaration and type of output produced for method*/

 @GET
 @Path("/da_batchDetail")
 @Produces(MediaType.APPLICATION_JSON)

 public JSONArray da_batchDetail(BatchDto bDto) throws SQLException

 {

  return
batchDA.fnBatchDetails(bDto.getOutputList(),bDto.getinputColumn(),bDto.getinputColu
mnValue());

 }
```

9. Coding Style Guide:

https://google.github.io/styleguide/jsguide.html

10. Doxygen-Code document

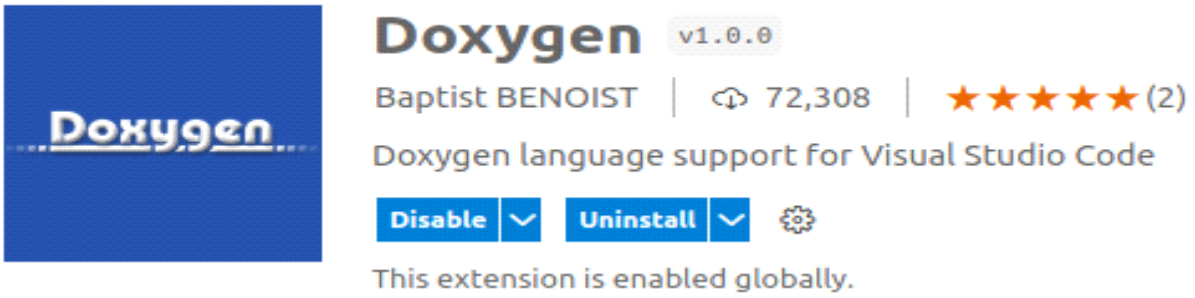
10.1. Introduction:

Doxygen is a documentation generator and static analysis tool for software source trees. When used as a documentation generator, Doxygen extracts information from specially-formatted comments within the code. It is used for analysis. Doxygen uses its parse tree to generate diagrams and charts of the code structure.

10.2. Installation guide:

There are two ways to install Doxygen:

10.2.1. Download Doxygen extension from vs code v1.0.0.



10.2.2. Download it manually by using this command.

10.2.2.1. To install Doxygen

sudo apt-get install doxygen

10.3. Doxy file generation

You can generate the doxyfile by using following steps:

10.3.1. To generate documentation of source code

i. doxygen -g config-file
ii. config-file (It's your file name)

10.3.2. To generate the documentation:

i. doxygen config-file

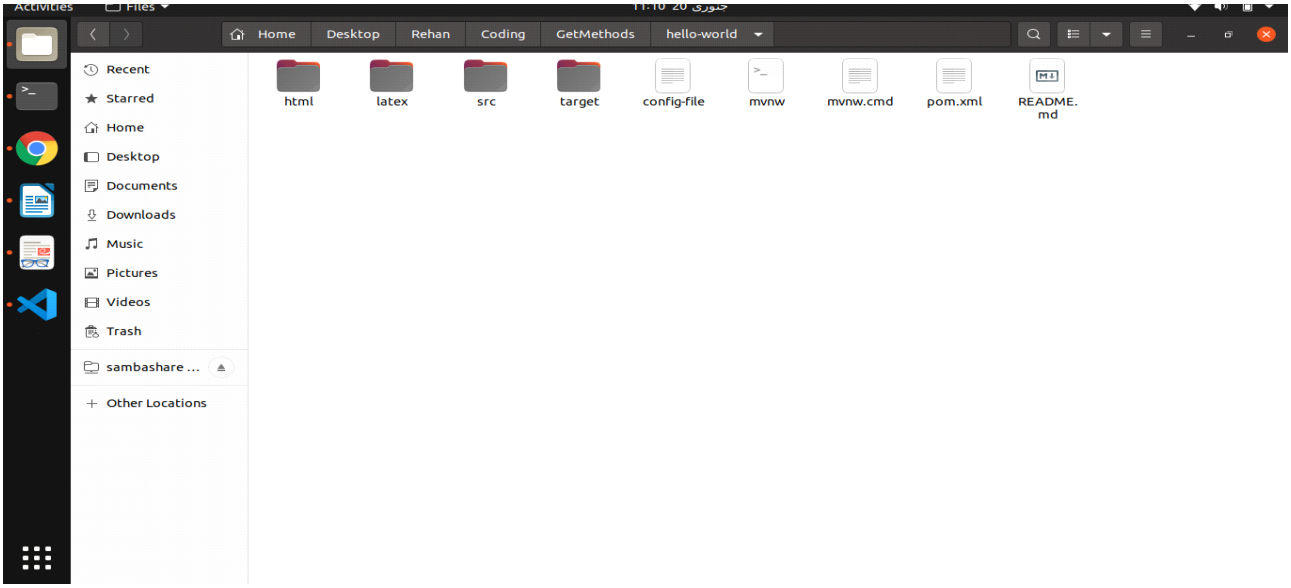
By running these commands it will make two folders and one file in your folder shown

```
pc@pc-HP-ProBook-650-G2:~/Desktop/20 Jan 2022/Batch-Core$ doxygen -g config-file

Configuration file 'config-file' created.

Now edit the configuration file and enter

  doxygen config-file

to generate the documentation for your project

pc@pc-HP-ProBook-650-G2:~/Desktop/20 Jan 2022/Batch-Core$
```

```
pc@pc-HP-ProBook-650-G2:~/Desktop/20 Jan 2022/Batch-Core$ doxygen config-file
Searching for include files...
Searching for example files...
Searching for images...
Searching for dot files...
Searching for msc files...
Searching for dia files...
Searching for files to exclude
Searching INPUT for files to process...
Searching for files in directory /home/pc/Desktop/20 Jan 2022/Batch-Core
Reading and parsing tag files
Parsing files
Reading /home/pc/Desktop/20 Jan 2022/Batch-Core/README.md...
Building group list...
Building directory list...
Building namespace list...
Building file list...
Building class list...
Computing nesting relations for classes...
Associating documentation with classes...
Building example list...
Searching for enumerations...
Searching for documented typedefs...
Searching for members imported via using declarations...
Searching for included using directives...
Searching for documented variables...
Building interface member list...
Building member list...
Searching for friends...
Searching for documented defines...
Generating search indices...
Generating example documentation...
Generating file sources...
Generating file documentation...
Generating page documentation...
Generating docs for page md_README...
Generating group documentation...
Generating class documentation...
Generating namespace index...
Generating graph info page...
Generating directory documentation...
Generating index page...
Generating page index...
Generating module index...
Generating namespace index...
Generating namespace member index...
Generating annotated compound index...
Generating alphabetical compound index...
Generating hierarchical class index...
Generating graphical class hierarchy...
Generating member index...
Generating file index...
Generating file member index...
Generating example index...
finalizing index lists...
writing tag file...
Running plantuml with JAVA...
Running dot...
lookup cache used 0/65536 hits=0 misses=0
finished...
```

```
pc@pc-HP-ProBook-650-G2:~/Desktop/20 Jan 2022/Batch-Core$ sudo apt install graphviz
[sudo] password for pc:
Reading package lists... Done
Building dependency tree
Reading state information... Done
graphviz is already the newest version (2.42.2-3build2).
0 upgraded, 0 newly installed, 0 to remove and 224 not upgraded.
pc@pc-HP-ProBook-650-G2:~/Desktop/20 Jan 2022/Batch-Core$
```

```
pc@pc-HP-ProBook-650-G2:~$ sudo apt-get install doxygen
[sudo] password for pc:
Reading package lists... Done
Building dependency tree
Reading state information... Done
doxygen is already the newest version (1.8.17-0ubuntu2).
0 upgraded, 0 newly installed, 0 to remove and 224 not upgraded.
pc@pc-HP-ProBook-650-G2:~$
```



10.4. Config-file:

A configuration file is a free-form ASCII text file with a structure that is similar to that of a Make-file, with the default name Doxyfile. It is parsed by doxygen. The file may contain tabs and newlines for formatting purposes.
By default it make only one JAVA. class file document.

- Click on a **config-file** and find **RECURSIVE**. Then change NO to "**YES".**
- You can edit your project-name, project-version, output language etc.

10.5. Execution command

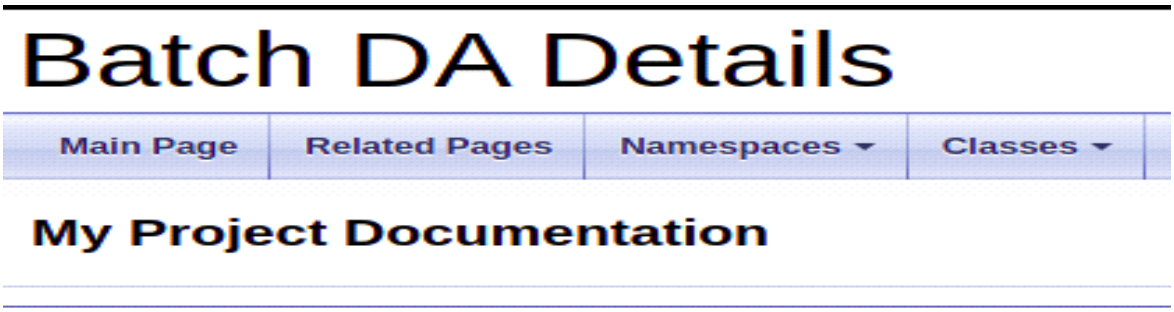
In terminal you can use this command.

- Doxygen **config-file**

This command is basically used for running doxygen config-file

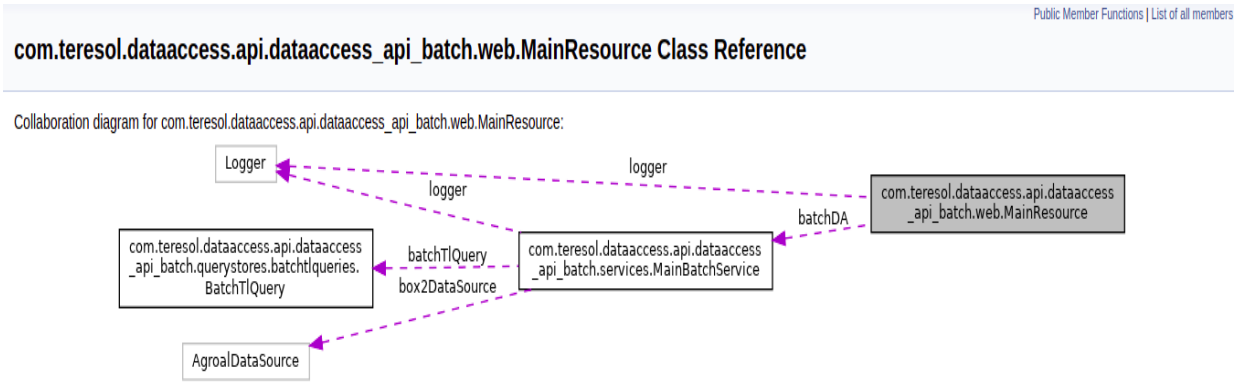
- **config-file**(It's your file name)
- After that open html folder and open index.html file.

Example:



10.6. Diagram generation

- Install Graphviz by using this command
  - sudo apt install **graphviz**
- For viewing doxygen diagram
  - Click **Classes** → I**ndex → Main Resource**



Note:

If you make any change in **Config-file** you must need to be save and again run doxygen file by using this command **doxygen config-file**. After that open html folder and click on index.html file.