**Git repository structure**

GatheService.<microservice\_project>

ex: GatheService.user, GatheService.community, etc

1 microservice = 1 project

by .gitignore in root folder, the IDE specific folder like “.idea” from jetbrains or “.settings” from eclipse should not be pushed to git repo.

**Maven project**

project

|- .idea *this is metadata for this project settings*

|- .mvn *maven jars*

|- src *all your codes are in here*

|- main

|- java

|- com.gathe.*microservice\_name*

*always try use one word for microservice\_name*

*if two word is needed, use hyphen ‘-‘ as separator*

|- controller

*classes which manage the API*

|- domain

*classes that represent a “THING” of our real world*

|- exception

*user defined exception*

|- repository

*classes that persisting domains to database*

|- impl

|- service

*classes that have the business logic*

|- impl

|- utility

*utility classes [optional]*

|- *Microservice*Application

*SpringBoot runner class*

*the class name will follow the project name*

*ex: project name = Community*

*class name = CommunityApplication*

|- resources *any static resources (image, sound, etc) put here*

|- test

*codes for testing,*

*package structure is exactly the same main package above*

|- pom.xml

*your maven settings (compilation, dependencies, etc)*

.idea folder should not be pushed to git repository

controller don’t need interface

domain should have class structure (ex: inheritence) if needed, but not for reusability

repository and service packages should have impl subpackage

all classes in repository and service must have interface. And the implementation of that interface must be put in the impl package

avoid inheritance for their classes whenever possible, use interface instead

utility classes can be both static and concrete classes

**Naming Convention**

package & file

com.gathe.community.controller

com.gathe.community.utility.activity

com.gathe.community.utility.map-locator

persistent.prop

server.prop

test-server.prop

class:

class SomeClassName {…}

class AnotherClassName {…}

class AnotherClassNameButLonger {…}

functions, methods, variables:

public doingSomething ( ) {…}

public static doingSomethingElse ( ) {…}

int counter;

boolean OK;

boolean finished;

don’t put “is” for boolean, it will be used by setters and getters of java POJO rules

Don’t put the char “I” at front of interface name

Interface should have readable name as it’s representing a Thing, not the implementation class

implementation for one interface should have Impl at the end

ex: interface = CommunityService

impl class = CommunityService**Impl**

every class and interface for *controller, repository, service, utility* should have tag name at the end

TheName**Controller** for controller

TheName**Exception** for exception

TheName**Repo** for repository

TheName**RepoStub** for repository with stub data

TheName**Service** for service

TheName**Util** for utility

**Interface and Class**

any function or method that can be called by other class should be put into the interface

the function or method don’t need to specify the access modifier (public) explicitly

the implementation must always have @Override

put explanation comment only in the interface

for classes without interface, put it in itself

use separator like —— *comment here* —————————————————

for each element / different concerns in one class

will be better for readability

it’s optional, but mandatory for domain classes

(constants, members, other methods, getters and setters mixed together in domain class has very bad readability)

for domain classes,

members should only have public-protected access

setters and getters should provide as the public access of the members

setters and getters naming convention follow java POJO rules

from top to bottom, the elements should be in this order:

constants, members, constructors, methods, setters and getters

for controller classes,

don’t use any interface

wrap / unwrap of domain (xml, json, to domain object, or vice versa) happens in this layer

anything API related happens here

for service and repository classes,

public methods come first at the top, followed by private class

Stub classes can be deleted after real implementation to DB is already implemented

for exception classes,

use custom exception for any failure of business logic

use inheritance for hierarchical exceptions

ex: NotFoundException

|- UserNotFoundException

|- CommunityNotFoundException

for classes with interface

always use interface as the type of variable

use concrete class only when needed / specific case

for Spring,

don’t use @Autowired, but wiring the dependency by Constructor param instead

@Autowired will create problem in Unit Testing

example:

\* \* \*

public class MemberServiceImpl {

@Autowired // don’t do this

MemberRepo memberRepo;

public MemberService (final MemberRepo memberRepo) { // do this

this.memberRepo = memberRepo;

}

}

\* \* \*

**Methods**

put standard java doc comment for every functions or methods

(refer to conventions explained above)

example:

\* \* \*

package com.gathe.community.service

public interface MemberService {

/\*\*

\* This method make the user as a new member of given community

\* @paramcommunity

\* @param user

\* @return true if user is successfully become member of the community

\* @return false if community or user is null

\* @return false if join failed because of *[other reason]*

\*/

boolean join (Community community, User user);

/\*\*

\* This method register the user to the given community, but yet becoming the member

\* @paramcommunity

\* @param user

\* @return true if user is successfully registered to the community

\* @return false if community or user is null

\* @return false if registration failed because of *[other reason]*

\*/

boolean register (Community community, User user);

}

package com.gathe.community.service.impl

public class MemberServiceImpl {

// - - interface methods - - - - - - - - - - - - - - - - - - - - - -

@Override

public boolean join (Community community, User user) {

// the code

}

@Override

public boolean register (Community community, User user) {

// the code

}

// - - private methods - - - - - - - - - - - - - - - - - - - - - - -

private void somethingElse () {

// the code

}

}

\* \* \*