**Coding Guidelines**

The following are proposed coding guidelines to ensure the quality of our software development team

**API Project Structure**

There are two common approaches to structuring a Java API application:

* Functional approach
* Java classes are organized into packages named after their functionality
* Classes with similar functionalities are grouped together in the same package
* Domain approach
* Java classes are organized into packages named after their domain
* Classes that deal with the same domain are grouped together in the same package
* While each of the approaches has its own benefits and disadvantageous, **we should adopt the hybrid approach that combines the best of both functional and also domain approaches**
* Hybrid approach
* Java classes are organized into packages according to their domain followed by functionality
* This approach provides clear visibility into both the domain and also the functionality of each Java classes

|  |  |  |
| --- | --- | --- |
| **Functional Approach** | **Domain Approach** | **Hybrid Approach** |
|  |  |  |

**Naming Convention**

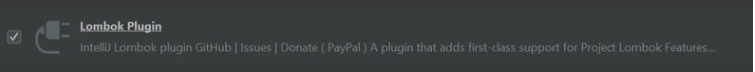
Names used for classes, variables, methods, and constants should be concise and consistent with the business context.

* Class name:
* **Use Pascal case convention: The first letter of every word is capitalized**
* Examples:
  + User
  + UserAddress
* Variables and method names
* **Use camel case convention: The first letter of the first word is lowercase while the first letter of subsequent words is capitalized**
* Examples:
  + firstName
  + getUserByFirstName
* Constant
* **Use capitalized snake case convention: Each word is separated by an underscore**
* Examples:
  + SUCCESS\_CODE
  + MAX\_CHARGE\_ATTEMPT
* Namespace
* Use reverse domain name for namespace naming
* Use lowercase with a dot separator
* Example:
  + com.amtrust.userapi.model.user.address
* API Endpoint Path Name
* **Use camel case convention: The first letter of the first word is lowercase while the first letter of subsequent words is capitalized**
* Endpoint path should only contain nouns
* Use the different mappings to indicate actions associated with each endpoint
  + GET, POST, PUT, DELETE
* All
* Ensure all constant, variable, method, and class names are spelled correctly

The aforementioned conventions are written for Java projects. For Python project, please refer to the guideline at: <https://www.python.org/dev/peps/pep-0008/>

**Code Verbosity**

To increase the maintainability and readability of our source code, we should adopt the following practices:

* Use annotation whenever possible
* Take advantage of Lombok annotation to remove cluttering source code with boilerplate codes. (i.e.: @Data annotation from [Lombok](https://projectlombok.org/))
* Besides adding the Lombok dependency to your project, the Lombok plugin must be enabled/installed on your IDE
  + IntelliJ
    - Enable the Lombok Plugin under Settings > Plugins
    - 
  + STS/Eclipse
    - Download Lombok jar and install
    - Instructions at <https://ahex.co/lombok-jar-installation/>
* Always provide comment but **avoid *paragraph***comment
* Remove commented codes
* Codes can be restored from source control if needed

**Dependency Management and Code Reusability**

We should simplify our dependency management and code reusability by introducing a base project that includes:

* common dependencies that are used across multiple projects
* base classes/interfaces to be inherited/implemented in different projects
* common constants

**Domain Driven Design Data Model**

Following the concept domain driven design, the data model should remain in the microservice that owns the domain (e.g.: User service owns the user data model) and all other services will access the required data through the respective microservice. This eliminates the duplication of data model across multiple projects and subsequently reduces the effort needed to maintain the data model

**Data Retrieval Optimization**

Each domain microservice should provide all necessary endpoints to expose the data required by other services:

* No greedy data fetching, each endpoint should only retrieve the necessary data from the database
* Avoid unnecessary query

**Unit Tests**

All major feature/logic should have its corresponding unit tests:

* Unit tests must include both positive and also negative test scenarios
* The QA team can refuse to begin testing if unit tests report is not available

**Logging**

Please refer to [Logging Guidelines](Logging%20Guidelines.docx)

**API Documentation**

Please refer to [API Documentation Guidelines](API%20Documentation%20Guidelines.docx)

**Code Monitoring**

Please refer to [Code Monitoring Guidelines](Code%20Monitoring%20Guidelines.docx)

**Security**

Sensitive information such as username and password should not be included in the source code in plain text. Instead, we should use an authentication repository to store our authentications.

**NEVER Delete**

No data should ever be deleted. We should perform a soft delete by using a special field to indicate that data is "**deleted**"