IT Jobs & Referrals – Workshop

Coding Guidelines for Backend Developers

- ✔ Need?
- ✔ Pull Request
- ✔ Code Review
- ✔ Coding guidelines:

Category	**Guidelines**

Naming Convention	✔ Class Names - UpperCamelCase
	✔ Method Names - lowerCamelCase
	✔ Variables - lowerCamelCase
	✔ Constants - UPPER_CASE_UNDERSCORE_SEPERATED
	✔ Variable Names
	  ✔ Names should be appropriate to the intention of how the variable is used and should be as descriptive as possible
	  ✔ Names should never be shorter than 3 characters unless it is an indexer
	✔ Shorthand names should not be used as they obfuscate the usage of the variable (e.g. btl instead of bottle)

Guidelines for methods	✔ Docs on all public methods, within reason
	  ✔ Method Explanation
	  ✔ Inputs
	  ✔ Outputs
	  ✔ Exceptions
	✔ Docs must be updated when code is updated
	✔ No Doc is required for getters and setters
	✔ Docs on all public Classes
	  ✔ Description of the class
	✔ Max Arguments limit to 7, after 7 create a payload
	✔ Avoid really long methods. ideally a method text should fit within a single screen view. Try to reduce the method size by creating helper methods and removing any copy/pasted or duplicated code.
	✔ Classes that have too many fields could be redesigned to have fewer fields, possibly through some nested object grouping of some of the information.
	✔ Method to do one and only one function
	✔ Plural naming convention for methods returning lists
	✔ Don't insert new methods in the middle of existing code.

	✔ Private methods should be in the end of the file. ✔ All non-private methods should validate the input parameters for null, empty and other required conditions before being used for any operations
Variables	✔ Variable declarations should be grouped together in the highest common code scope that makes sense ✔ Variable should be declared from outside loop. ✔ All local variables declared should have a default value assigned to them ✔ Remove unused parameters and variables
Statements	✔ Limit 1 statement per line ✔ Remove duplicate statements ✔ Tertiary|Ternary Statements ✔ Only simple statements should be used ✔ Complex or compound Tertiary statements should be avoided Braces should be present even if only one line of code exist (not applicable for cases)
Exceptions	✔ Catch blocks should never be empty, either log, comment or perform some meaningful logic ✔ Cleanup logic should not be duplicated in both try and catch blocks, if needed it should be placed in the finally block ✔ Exceptions should not be thrown and caught from within the same block of code ✔ Exceptions should bubble where appropriate and be handled where logically applicable: catch it only if you can act on it. ✔ Preserve stack trace. Throwing a new exception from a catch block without passing the original exception into the new exception will cause the true stack trace to be lost, and it will make difficult to debug ✔ Threading ✔ Exceptions should never be thrown out of the thread ✔ REST Services ✔ Exceptions in web services should be caught, logged and the appropriate status code should be returned ✔ (500 should not be returned for all exceptions)
Warnings	Clean up all warnings before checkin(Possible to configure in save action)
Formatter	✔ Format the code before checkin(Possible to configure in save action)
Comparison	✔ Write constants first while comparing using equals
Scoping	✔ Most restricted scope should be provided to the variables and methods ✔ Methods are public only they are a public api, i.e. being consumed from outside the resident class ✔ Getters/Setters may remain public even if not consumed from anywhere

	✔ Member and static (excluding constants) variables should be private
Comments	✔ Commented code should be removed before check-in. ✔ Complex code should be commented and maintained throughout changes
Must Haves	✔ Classes should be no more than 400 lines ✔ Methods should be no more than 75 lines
Test Cases	✔ Test cases coverage should be greater than 80
Logical	✔ Ensure unnecessary checks are not written in code Eg. Checking for null a field retrieved from database which has NOT NULL constraint
Database	✔ Use Joins rather than inner query if possible. Joins have better performance than an inner query. ✔ Ensure unnecessary database hits are removed from the application. ✔ Avoid writing native query if possible. The query keywords like “where” etc must be written in uppercase.
General	✔ Follow the OOPS concepts ✔ Use non primitive in model as they are bound to UI and can return a null. ✔ Use latest APIs of particular language to write the code. ✔ Create reusable generic Utility classes in case the functionality can be reused in application ✔ Inherit from an existing class if it makes sense.
Logs	✔ Use logger instead of SYSO or Consoles