

# Software Systems Architecture

## Personal Project

### Summary



In this personal project, you will have to:

- Develop a framework that uses code annotations
- Create an application with a layered architecture that uses this framework
- Define an approach about how to test your architecture
- Evaluate the code and tests using metrics tools

### Framework



**Domain:** The goal is to create a framework for an application to introduce a gamification process in its execution. The framework will add points for the user based on annotations on classes.


**General Vision:** The following are some classes from the framework API:


- Task: an interface to be implemented by the application to add domain-specific logic. It represents something in the application that can worth rewards in the gamification system.
- User: represents a user from the system. It have points associated.
- UserRegistry: this class is used to configure the user for the current thread (see tips about how to do that).
- GamificationFacade: this class is used to execute the tasks and perform the gamification logic. It should be a Singleton.
- @AddPoints and @RemovePoints: annotations that can be added in the classes that implement the Task interface to add or remove points for the current user.
- GamificationListener: an interface that can be registered at GamificationFacade to be invoked every time that an user win or lose points.


**Specification:** A more detailed description from the framework classes.


- Task (interface): Have an execute() method that return Object.
- User (class): Should provide the user name and how many points the user have. Should also provide methods to add and remove points. The student should design the class API.
- UserRegistry (class): It should have the methods setCurrentUser(User) and getCurrentUser(). (Tip: use ThreadLocal for this implementation).
- GamificationFacade (class): This class should be a singleton and have the following methods:
  - void addCallback(GamificationListener) -> set an implementation of GamificationListener to be called when points are added or removed from the User.

	<ul style="list-style-type: none"> <li>○ Object execute(Task) -&gt; Execute the Task adding or removing points according to its annotations and execute the GamificationListener when applicable.</li> <li>• GamificationListener: This interface should have methods that are called when the points are added or removed from a User. The student should design the interface API.</li> <li>• @AddPoints and @RemovePoints: These annotations should have attributes with the number of points to be added or removed.</li> </ul>
--	---


<b>Application</b>  	<p><b>Domain:</b> The application should register the “lessons learned” by a team. It should contain the following functionality: user login, register lesson learned, recommend lesson learned (similar to a “like”), unrecommend lesson learned (similar to a “dislike”), add a comment to a lesson learned, show lessons learned, and show users ranking. When a user enters the system, he can see a list with the lessons learned and an interface to add a new one. When he clicks a lesson learned, he can see it with the recommendations, recommendations and comments. In this screen he can recommend, unrecommend and add a comment. Additionally, the users will receive points for performing the system actions the users should be able to see their ranking.</p> <p><b>Architectural Requirements:</b> The database is located on a server and the application users need to access it remotely. The application can be of any kind: mobile, web, desktop, etc... The application should use the gamification framework created.</p> <p><b>Layer Definition:</b> When you create the application, you should divide it in layers with distinct responsibilities, such as database access and graphical interface. What APIs and libraries each layer have access? How the borders are defined in each case? Between which layers you will put the remote access? What is protocol that you will use for the remote access? Where the gamification framework is going to be added? How does it influence the architecture? All these decisions you should make when you define your layers.</p> <p><b>Technologies:</b> Consider that the team that will develop the application have expertise in the Java language. Not all layers need to be in Java, but you will need to justify your choices. You should use a relational database to store the data.</p> <p><b>Gamification:</b> This application needs use the gamification framework to implement the following point system: (a) adding a lesson learned +20 points; (b) recommending a lesson learned +5 points; (c) unrecommending a lesson learned -3 points; (d) commenting a lesson learned +10 points. The application should implement a GamificationListener to persist the user points. You should try to reduce as much as possible the coupling between the system and the framework.</p>
---	--


<p><b>Automated Tests</b></p> 	<p><b>Test Architecture:</b> You should create tests for each component in the architecture. For each layer you should have an approach for implementing the tests for that kind of class. The tests should be on a separated project.</p> <p><b>Goal:</b> Your goal on this step is to do more than create tests for your application. You are defining the approach that is going to be used for all functionalities for each layer.</p>
---	--

<p><b>Metrics</b></p> 	<p><b>Goal:</b> Perform an analysis of your application software using metric and software visualization. The evaluation should focus on 3 aspects: (a) code quality; (b) dependences between components; (c) test coverage.</p> <p><b>Code quality:</b> Use metrics and software visualization to evaluate the quality of your code and identify potential point with problems.</p> <p><b>Dependences:</b> Evaluate dependences between layers and between the gamification frameworks and the other components;</p> <p><b>Test coverage:</b> Evaluate the test approach verifying if there is any part of the code that is not being covered by your suggested test approach.</p>
---	---

<p><b>Delivery</b></p> 	<p>A .zip file should be created containing all the project information according to the structure described in each item.</p> <p><b>Framework:</b> Create a folder “framework” and put inside: the complete source code developed, a .jar file with the binaries, and a .pdf file with the framework documentation. The documentation should be focused on framework users. It needs to have a step by step “hello world”, a description of the framework features, and an example of the framework usage.</p> <p><b>Application:</b> Create a folder “application” and put inside: the complete source code for the application and a .pdf file with the documentation of the layered architecture. The target audience of this document is not your teacher but a development team that will need to follow that architecture. Give a general view of the entire architecture, and then describe each layer in detail. This version of the application should not include details about the usage of the gamification framework yet.</p> <p><b>Test Architecture:</b> Create a folder “tests” and put inside: the complete source code for the tests and a .pdf file about how tests should be created for each component in the architecture. The report should contain a picture that shows the execution of all tests. The target audience of this document is not your teacher but a development team that will need to created automated tests for that architecture.</p>
--	---

	<p><b>Metrics:</b> Create a folder “metrics” and put inside a .pdf file containing a report based on the goals. The report should contain the information required depending on the alternative chosen. When you refer to a source code, include on the report the part of the source code that you are referring to. When you mention the metrics or a visualization that you used, include this information in the report. If needed, you can take a screenshot from the tool that you used. You should not just present the result from the tools. Your interpretation of the data is very important!</p> <p>After the delivery of the personal project, the students will have to present it. The presentation needs to cover all the topics and should be scheduled with the teacher.</p>
--	--

<p><b>Tips</b></p> 	<p><b>Tip about tips:</b> Many parts of this personal project are similar to exercises from the labs. You can find good tips there!</p> <p><b>Package division:</b> Divide your application in components and packages in a way that can make easy to perform the analysis.</p> <p><b>Use the metrics:</b> If your metrics show that the code quality or the test coverage is not good, remember that you can use it to fix your solution before delivering.</p> <p><b>Evaluating dependences:</b> There are many views that can help you to evaluate the modularity and coupling between the components. One of them is the DSM (Design Structure Matrix) which can be generated by several tools.</p>
--	---

<p><b>Rules</b></p> 	<p><b>You cannot do:</b> Different from the labs, in this personal project you cannot get help from any other student in any part of it.</p> <p><b>You can do:</b> You can take a look on the labs delivered by other students, that do not have the same tasks, but similar ones.</p>
---	--