# High-Quality Code Exam – Buhtig Issue Tracker

You have been assigned to work on an international project with a German company. The task is to implement an issue tracking web application in C# (read **"buhtig"** backwards ☺). Your partner, **Sigismund Grunebaum** has done most of the work, but you still need to refactor the code in order to make it more usable and more easily readable.

Your task is to **refactor the code**, using all best practices in **object-oriented design** and **object-oriented programming**, **SOLID** principles, and **design patterns**. You have to **improve the code quality** so it is easy to read and maintain. You also have to **fix any bugs** your Brazilian friend might have left, and **improve the general performance** (execution speed) of the code. Since the Brazilian company didn't have time to **write** any **unit tests**, they also left all of this to you.

You are given the original code and the design document, specifying the task at hand. The Germans also provided you with two sample test cases to check how the application works. These documents are provided below.

## Overview

An **issue tracking system** consists of **users** and **issues**. Each user can create issue, delete his / her issues, search for issues (by their **tags**) and add **comments**.

There are two types of users – guests and registered users. Initially all users are considered guests (up to the moment when they log into the system).

Users can **register** using a **username** and **password**. For security reasons, all passwords are **hashed** using the **SHA-1** algorithm. After registration, users can **login** using the **username** and **password** they created during the registration. After their session has finished, any logged in user can **logout**. The issue tracking system keeps track of the currently logged in user.

Anyone (guests and users) can **search** for issues by their tags. The search takes one or more tags and returns all issues that match at least one of those tags.

**Logged in users** can **create issues**. An **issues** has a **title**, **description**, **priority** (Low, Medium, High, or Showstopper, in order of increasing importance), a set of **tags**, and a set of **comments**. The issue author is the currently logged in user. When an issue is inserted in the system database, it automatically provides it with a unique **ID**. This is a positive integer, greater than or equal to 1. Even if an issue is deleted at some moment (and its ID becomes free), no other issue may get the same ID.

A user may also **delete his / her issues**. To do this, the user provides the ID of the issue to delete.

Users can **comment** on issues. A **comment** has an **author** (the currently logged in user) and **text**.

At any time, a user may wish to see all issues and all comments created by them.

The issues are printed in a sorted order – in decreasing order of priority first, and by titles in increasing order second.

The comments are printed in the order they have been created.

In case a guest tries to access some functionality available to logged in users only, the system returns an error message and **does not perform** the requested operation.

## System design and functions

The main part of the system is **the engine**. It provides the interface (connection) between the users and the web application. **The engine ignores all whitespace around commands and all empty lines.** The engine catches any errors that might occur in the code and writes their messages back to the user.

The engine accepts **URLs** (called **endpoints**) given by the application users and passes them to a **dispatcher**. The dispatcher dispatches all **actions** to their respective places. An **endpoint** consists of **action name** and **parameters**. The parameters are given as a **URL-encoded query string**.

A sample endpoint is shown below:

ActionName?param1=value1&param2=value2

In case of a **correct action name**, the action dispatcher returns a string. It contains either a success message (if everything went as expected), or an error message (if there was any problem executing the command). In case of **incorrect action name**, the engine throws an **InvalidOperationException** with the message "Invalid action: <action name>", where <action name> is the requested action name.

The execution engine delegates all commands to **an object which contains all information about the issue tracker** (which is an implementation of the **IIssueTracker** interface).

The application's **data layer** keeps track of all issues, users, and comments. It also provides an option to know who the currently logged in user is.

The actions supported by the engine are:

* **RegisterUser?username=<username>&password=<password>&confirmPassword=<confirmPassword>**

Registers a user in the database.

In case of success, the action returns **User <username> registered successfully**

If there is already a logged in user, the action returns **There is already a logged in user**

If the two passwords do not match, the action returns **The provided passwords do not match**

If the username is already taken, the action returns **A user with username <username> already exists**

* **LoginUser?username=<username>&password=<password>**

Logins a user in the application. After login, they become the currently active user.

In case of success, the action returns **User <username> logged in successfully**

If there is already a logged in user, the action returns **There is already a logged in user**

If there is no user with the provided username, the action returns **A user with username <username> does not exist**

If the password is invalid, the action returns **The password is invalid for user <username>**

* **LogoutUser**

Logs out the currently active user.

In case of success, the action returns **User <username> logged out successfully**

If there is no logged in user, the action returns **There is no currently logged in user**

* **CreateIssue?title=<title>&description=<description>&priority=<priority>&tags=<tags>**

Creates a new issue. Assigns the current user as its author. Gives it an ID automatically.

If the issue title is less than 3 symbols long, or if the issue description is less than 5 symbols long, the system throws an **ArgumentException** with an appropriate message. The tags are separated by a single pipe sign ("**|**"). In case there are some repeating tags, the system only registers them once for each issue. There will always be at least one tag per new issue. You do not need to check this explicitly.

In case of success, the action returns **Issue <id> created successfully**

If there is no logged in user, the action returns **There is no currently logged in user**

* **RemoveIssue?id=<id>**

Removes an issue given by the specified ID.

In case of success, the action returns **Issue <id> removed**

If there is no logged in user, the action returns **There is no currently logged in user**

If the issue ID is invalid (i. e., does not exist in the database), the action returns

**There is no issue with ID <id>**

If the issue does not belong to the currently logged in user, the action returns

**The issue with ID <id> does not belong to user <current\_user\_username>**

* **AddComment?id=<id>&text=<text>**

Adds a comment to the issue given by the specified ID. If the text is less than 2 symbols long, the system throws an **ArgumentException** with an appropriate message.

In case of success, the action returns **Comment added successfully to issue <id>**

If there is no logged in user, the action returns **There is no currently logged in user**

If the issue ID is invalid (i. e., does not exist in the database), the action returns

**There is no issue with ID <id>**

* **MyIssues**

Returns the issues created by the currently active user.

In case of success, the action returns the issues sorted by priority (in descending order) first, and by title (in alphabetical order) next. Each issue is printed in a user-friendly way, each on its own line. Refer to the sample outputs to see how exactly an issue should be formatted.

If there are no issues, the action returns **No issues**

If there is no logged in user, the action returns **There is no currently logged in user**

* **MyComments**

Returns the comments created by the currently active user.

In case of success, the action returns the comments sorted by time of adding to the application database. Each comment is printed in a user-friendly way, each on its own line. Refer to the sample outputs to see how exactly a comment should be formatted.

If there are no comments, the action returns **No comments**

If there is no logged in user, the action returns **There is no currently logged in user**

* **Search?tags=<tags>**

Searches for issues containing one or more of the provided tags. If an issue matches several tags, it is included only once in the search results. The tags (if there are two or more) are separated by a single pipe sign ("**|**").

In case of success, the action returns the issues sorted by priority (in descending order) first, and by title (in alphabetical order) next. Each issue is printed in a user-friendly way, each on its own line. Refer to the sample outputs to see how exactly an issue should be formatted.

If there are no tags provided, the action returns **There are no tags provided**

If there are no matching issues, the action returns **There are no issues matching the tags provided**

Model the system and all entities (issues, issue tracker, users, endpoints, etc.) using the best established practices in object-oriented design and object-oriented programming.

The input should be read from the console. It may contain up to 50 000 commands, so the issue system must work as efficiently as possible. The output is written to the console. The input and output formats have been specified in the command descriptions.

## Sample Input 1

|  |
| --- |
| RegisterUser?username=admin&password=pass123&confirmPassword=pass123  LoginUser?username=admin&password=pass123  LogoutUser  LoginUser?username=admin&password=pass123  CreateIssue?title=New%20issue&description=This%20is%20a%20new%20issue&priority=high&tags=new|issue  CreateIssue?title=Another%20issue&description=This%20is%20another%20new%20issue&priority=medium&tags=new|issue|another|issue  AddComment?id=2&text=New%20comment  AddComment?id=2&text=Another%20comment  MyIssues  MyComments  LogoutUser  RegisterUser?username=user&password=user123&confirmPassword=user123  LoginUser?username=user&password=user123  AddComment?id=2&text=Another%20comment%20by%20user  MyComments  LoginUser?username=admin&password=pass123  MyIssues  RemoveIssue?id=1  Search?tags=new  Search?tags=new|issue  End |

## Sample Output 1

|  |
| --- |
| User admin registered successfully  User admin logged in successfully  User admin logged out successfully  User admin logged in successfully  Issue 1 created successfully  Issue 2 created successfully  Comment added successfully to issue 2  Comment added successfully to issue 2  New issue  Priority: \*\*\*  This is a new issue  Tags: issue,new  Another issue  Priority: \*\*  This is another new issue  Tags: another,issue,new  Comments:  New comment  -- admin  Another comment  -- admin  New comment  -- admin  Another comment  -- admin  User admin logged out successfully  User user registered successfully  User user logged in successfully  Comment added successfully to issue 2  Another comment by user  -- user  There is already a logged in user  No issues  The issue with ID 1 does not belong to user user  New issue  Priority: \*\*\*  This is a new issue  Tags: issue,new  Another issue  Priority: \*\*  This is another new issue  Tags: another,issue,new  Comments:  New comment  -- admin  Another comment  -- admin  Another comment by user  -- user  New issue  Priority: \*\*\*  This is a new issue  Tags: issue,new  Another issue  Priority: \*\*  This is another new issue  Tags: another,issue,new  Comments:  New comment  -- admin  Another comment  -- admin  Another comment by user  -- user |

## Sample Input 2

|  |
| --- |
| RegisterUser?username=admin&password=pass123&confirmPassword=dont\_match  RegisterUser?username=admin&password=pass123&confirmPassword=pass123  RegisterUser?username=admin&password=already\_registered&confirmPassword=already\_registered  LoginUser?username=admin&password=pass123  RegisterUser?username=admin&password=pass123&confirmPassword=pass123  LogoutUser  LoginUser?username=invalid&password=pass123  LoginUser?username=admin&password=invalid  LogoutUser  LoginUser?username=admin&password=pass123  CreateIssue?title=Valid%20issue&description=This%20is%20a%20new%20issue&priority=high&tags=new|issue  LoginUser?username=admin&password=pass123  CreateIssue?title=N&description=This%20is%20a%20new%20issue&priority=high&tags=new|issue  CreateIssue?title=New%20issue&description=T&priority=high&tags=new|issue  RemoveIssue?id=30  LogoutUser  RemoveIssue?id=1  AddComment?id=1&text=Invalid%20comment  LoginUser?username=admin&password=pass123  AddComment?id=1&text=I  AddComment?id=30&text=Invalid%20comment  LogoutUser  MyIssues  MyComments  Search?tags=no|issues  End |

## Sample Output 2

|  |
| --- |
| The provided passwords do not match  User admin registered successfully  A user with username admin already exists  User admin logged in successfully  There is already a logged in user  User admin logged out successfully  A user with username invalid does not exist  The password is invalid for user admin  There is no currently logged in user  User admin logged in successfully  Issue 1 created successfully  There is already a logged in user  The title must be at least 3 symbols long  The description must be at least 5 symbols long  There is no issue with ID 30  User admin logged out successfully  There is no currently logged in user  There is no currently logged in user  User admin logged in successfully  The text must be at least 2 symbols long  There is no issue with ID 30  User admin logged out successfully  There is no currently logged in user  There is no currently logged in user  There are no issues matching the tags provided |

## Problem 1. Code Refactoring

**Refactor the source code** to improve its quality following the best practices introduced in the course  
“[High-Quality Code](https://softuni.bg/courses/high-quality-code/)”. You may refactor anything except the **IIssueTracker** interface (though you can document it), as long as it improves the code quality. You may create as many classes, interfaces, enumerations, structures, etc. as you wish.

**30 score**

## Problem 2. StyleCop

Make StyleCop run without any errors on your code (ignore all documentation-related errors).

**4 score**

## Problem 3. Bug Fixing

**Debug the code** and fix any bugs you find.

**6 score**

## Problem 4. Code Documentation

**Document the IIssueTracker** interface declaration and all methods in it using C# XML documentation. Any other documentation is **not** required. Each documentation gives 0.78 score.

**7 score**

## Problem 5. Unit Testing

Design and implement **unit tests for** **the following methods of the IIssueTracker** **interface:**

* **RegisterUser()**
* **CreateIssue()**
* **GetMyIssues()**
* **SearchForIssues()**

Any other code is not required to be tested. The **code coverage** should be **at** **least 90% for the specified methods** (you do not need to cover the class that parses the input commands and prints the output). Be sure to test **all major execution scenarios** + all interesting **border cases** and **special cases**. Use Visual Studio Team Test (VSTT) and VS code coverage.

You may need to call other methods on the implementation of the **IIssueTracker** interface. Ideally, this would be avoided using mocking, but feel free to do it using regular testing only.

**28 score**

## Problem 6. Performance Bottlenecks

Find any **performance bottlenecks** and briefly describe them with the following **comment in the code**:

**// PERFORMANCE: <your description of why you think this is a performance bottleneck>**

**Fix the problems** if possible (and leave the bottlenecks descriptions in addition to the fixes).

**6 score**

## Problem 7. Dependency Injection

Use dependency injection to decouple a class from its dependencies. Introduce **at least one** usage of dependency injection and mark it with the comment:

**// DI: <a short list of what you did to introduce DI>**

**Example: // DI: Added the property MyProperty, removed the class MyClass, removed the interface IMyInterface, added the field myField to the interface IMyInterface2**

**4 score**

## Problem 8. Correct Results in the Judge System

You are given an automated judge system to submit your solution. If your code is correct (all bugs are fixed) and runs fast enough (the performance bottlenecks are fixed), your solution will pass all the tests. The last 2 tests measure performance. The others measure correctness.

**16 score**