# High-Quality Code Exam – Bangalore University Learning System

You have been assigned to work on a big international project with the Bangalore University in India. The task is to implement the Web application serving the university’s learning system. The project leader, [**Ramachander**](http://www.behindthename.com/name/ramachander)[**Mahesha**](http://www.behindthename.com/name/mahesha), has already written most of the system’s functionality. However, when you saw his work, you understood you needed to refactor the code in order to make it easier to read, understand and maintain.

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**. You also have to **fix any bugs** the project leader might have left, and **improve the general performance** (execution speed) of the code. Since the university 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 Indians also have two sample test cases to check how the application works. These documents are provided below.

**Ensure the application follows the design document strictly.**

## Overview

The university system holds information about **users**, **courses** and **lectures**.

A **user** has a **username**, **password**, **role** (Student or Lecturer), and some **courses**. A **course** has a **name**, **lectures** and **students** enrolled in it. A **lecture** has a **name**.

There are three types of users: guests (users which haven’t logged into the system), students and lecturers.

Guests can **register** and after successful registration, they can **log in** the system. They can also view basic information about **all courses**.

Once logged in, a user is a Lecturer or a Student and he / she is able to **log out**. Students can do all things guests can do, and they can additionally **enroll in a course** and **view lectures** (for courses they participate in). Lecturers can do all things available to guests and students, and they can **add new courses** **and lectures**.

## System design

The project follows the **Model-View-Controller** architecture. It has been chosen after a long discussion with the Bangalore University Board of Directors.

The main part of the system is an **engine**. The engine reads URLs from the standard input (console) and decides what action should be performed. After performing the action, the engine writes the result to the standard output. In case some method throws an exception, the engine displays the exception message to the standard output.

A URL follows a strict format. The first part of an URL specifies a **controller name** to be called, and the second part specifies the **action name**. For more information about controllers and actions, read on. A URL may also have a set of **query parameters**: name-value pairs. The series of pairs are separated by an ampersand (**&**), and the parts within each pair are separated by a single equals sign (**=**). The query parameters are **URL-encoded** and can be given **in any order**. You can find an online URL encoder / decoder here: <http://www.url-encode-decode.com/>.

All URLs will be valid and in the format provided. There is no need to check this explicitly.

Examples of valid URLs are given below:

* **/Users/Login?username=new\_user&password=****P@ssw0rd123** → controller name: UsersController, action name: Login, parameters: username = new\_user, password = P@ssw0rd123
* **/Users/Logout** → controller name: UsersController, action name: Logout, parameters: none

Every other URL (not following these examples) is **invalid**.

**Models** are classes containing information about the real-world object the system works with. The models for this project are **User**, **Course** and **Lecture**. Not all models are valid. The validation rules for the models are given below:

* A user’s username must be at least 5 symbols long
* A user’s password must be at least 6 symbols long
* A course name must be at least 5 symbols long
* A lecture name must be at least 3 symbols long

In case validation fails, the system throws an **ArgumentException** with the message: **The [parameter\_name] must be at least [min\_length] symbols long.**

The system stores the passwords in a hashed state, i. e. it doesn’t keep the original (plain-text) password, but its **hash**. A **hash** has two key properties:

* All strings, no matter how long, produce a hash with the same length
* If two strings are the same, they will produce the same hash

The system uses the **SHA-1** hashing algorithm to store password hashes and keeps them as hexadecimal strings.

In order to work with model collections, the project has a **data layer**. The data layer consists of several **repositories**. A repository contains objects of the same type and provides methods for the following:

* **Getting all items** in the repository
* **Getting an item** by its unique ID (**1-based, in order of addition**)
* **Adding** a new item

The repository which stores users can also **look up users by their username**.

A **database** class combines all repositories defined for the application.

**Controllers** contain the main business logic in the application. A controller can contain a database and has at least one action. An **action** is a public method which returns a **view** and can optionally accept parameters. All controllers inherit from a base class.

Controllers keep track of the currently logged in user in the system. They also check if a user is authorized to perform the current action. For example, if a guest tries to add a new course, the system will reject the request. If an authorization check fails, the system throws an **AuthorizationFailedException** with the message **"The current user is not authorized to perform this operation."**.

**Views** contain the **presentation logic** in the system. They contain the results which are given to the user. A view can accept a **model** and it has to display some information, possibly using its model.

All views inherit from a base class. The structure of the views has strict conventions. A view namespace follows the structure of its respective controller.

For example, if there are the following controllers and actions defined in the system:

* **ControllerOne** (ActionA, ActionB)
* **ControllerTwo** (ActionC, ActionD),

then their views must follow the following namespace (and folder) structure:

**|-- Views****| `-- ControllerOne****| | -- ActionA.cs  
| | -- ActionB.cs  
| `-- ControllerTwo  
| | -- ActionC.cs  
| | -- ActionD.cs**

## System functionality

The Bangalore University Learning System responds to the following endpoints (URLs). All messages which the system should return are given in the tables for each endpoint.

* **/Users/Register?username=[string]&password=[string]&confirmPassword=[string]&role=[string]**

Registers a new user in the system. The role will always be **student** or **lecturer**.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] registered successfully. | None |
| The two passwords do not match | The provided passwords do not match. | ArgumentException |
| There is already a logged in user in the system | There is already a logged in user. | ArgumentException |
| There is already a user with the specified username in the system | A user with username [username] already exists. | ArgumentException |

* **/Users/Login?username=[string]&password=[string]**

Logs a registered user into the system.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] logged in successfully. | None |
| There is already a logged in user in the system | There is already a logged in user. | ArgumentException |
| There is no user with the specified username in the system | A user with username [username] does not exist. | ArgumentException |
| The username has been found, but the password does not match the database record | The provided password is wrong. | ArgumentException |

* **/Users/Logout**

Terminates the session of the currently logged user.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] logged out successfully. | None |
| There is no logged in user in the system | There is no currently logged in user. | ArgumentException |

* **/Courses/All**

Displays short information about all courses in the system.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | **In case there are no courses:** No courses.  **In case there are courses, displays all courses sorted alphabetically first, and then by number of students in descending order in the format:**  All courses:  A course (2 students)  Course 1 (20 students)  Course 2 (10 students) | None |

* **/Courses/Details?courseId=[int]**

Displays detailed information about the course with the specified ID.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | **In case there are no lectures:**  [Course name]  No lectures  **In case there are lectures, displays them in order of their addition in the format:**  [Course name]  - First lecture  - Second lecture  - Third lecture | None |
| There is no logged in user in the system | There is no currently logged in user. | ArgumentException |
| The currently logged in user is not a student or a lecturer | The current user is not authorized to perform this operation. | AuthorizationFailed Exception |
| The course does not exist | There is no course with ID [course\_id]. | ArgumentException |
| The course is not in the currently logged in user’s courses | You are not enrolled in this course. | ArgumentException |

* **/Courses/Enroll?courseId=[int]**

Enrolls the currently logged in user into the course with the specified ID.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Student successfully enrolled in course [course\_name]. | None |
| There is no logged in user in the system | There is no currently logged in user. | ArgumentException |
| The currently logged in user is not a student or a lecturer | The current user is not authorized to perform this operation. | AuthorizationFailed Exception |
| The currently logged in user has already enrolled in the course | You are already enrolled in this course. | ArgumentException |
| The course does not exist | There is no course with ID [course\_id]. | ArgumentException |

* **/Courses/Create?name=[string]**

Creates a new course with the specified name and adds it to the system database.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Course [course\_name] created successfully. | None |
| There is no logged in user in the system | There is no currently logged in user. | ArgumentException |
| The currently logged in user is not a lecturer | The current user is not authorized to perform this operation. | AuthorizationFailed Exception |

* **/Courses/AddLecture?courseId=[int]&lectureName=[string]**

Creates a new lecture with the specified name and adds it to the course with the specified ID.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Lecture successfully added to course [course\_name]. | None |
| There is no logged in user in the system | There is no currently logged in user. | ArgumentException |
| The currently logged in user is not a lecturer | The current user is not authorized to perform this operation. | AuthorizationFailed Exception |
| The course does not exist | There is no course with ID [course\_id]. | ArgumentException |

Model the system and all entities 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 above.

## Sample Input 1

|  |
| --- |
| /Users/Register?username=firstLecturer&password=firstPass&confirmPassword=firstPass&role=lecturer  /Users/Login?password=firstPass&username=firstLecturer  /Users/Logout  /Courses/All  /Users/Register?username=firstStudent&password=firstPass&confirmPassword=firstPass&role=student  /Users/Login?password=firstPass&username=firstLecturer  /Courses/All  /Courses/Create?name=Advanced+C%23  /Courses/Create?name=Object-Oriented+Programming  /Courses/Create?name=High-Quality+Code  /Courses/Create?name=Java+Basics  /Courses/AddLecture?courseId=3&lectureName=Naming+Identifiers  /Courses/AddLecture?courseId=3&lectureName=Comments+in+the+Code  /Courses/AddLecture?courseId=3&lectureName=High-Quality+Classes  /Courses/AddLecture?courseId=2&lectureName=Defining+Classes  /Courses/AddLecture?courseId=2&lectureName=OOP+Principles  /Courses/Enroll?courseId=1  /Courses/Enroll?courseId=3  /Courses/All  /Courses/Details?courseId=1  /Courses/Details?courseId=3 |

## Sample Output 1

|  |
| --- |
| User firstLecturer registered successfully.  User firstLecturer logged in successfully.  User firstLecturer logged out successfully.  No courses.  User firstStudent registered successfully.  User firstLecturer logged in successfully.  No courses.  Course Advanced C# created successfully.  Course Object-Oriented Programming created successfully.  Course High-Quality Code created successfully.  Course Java Basics created successfully.  Lecture successfully added to course High-Quality Code.  Lecture successfully added to course High-Quality Code.  Lecture successfully added to course High-Quality Code.  Lecture successfully added to course Object-Oriented Programming.  Lecture successfully added to course Object-Oriented Programming.  Student successfully enrolled in course Advanced C#.  Student successfully enrolled in course High-Quality Code.  All courses:  Advanced C# (1 students)  High-Quality Code (1 students)  Java Basics (0 students)  Object-Oriented Programming (0 students)  Advanced C#  No lectures  High-Quality Code  - Naming Identifiers  - Comments in the Code  - High-Quality Classes |

## Sample Input 2

|  |
| --- |
| /Users/Register?username=a&password=myPassword&confirmPassword=myPassword&role=student  /Users/Register?username=firstUser&password=short&confirmPassword=short&role=student  /Users/Register?username=notMatchingPasswords&password=password123&confirmPassword=Password098&role=student  /Users/Register?username=firstStudent&password=firstPass&confirmPassword=firstPass&role=student  /Users/Login?username=firstStudent&password=firstPass  /Users/Register?username=CantRegisterLoggedIn&password=password123&confirmPassword=password123&role=student  /Users/Register?username=firstStudent&password=firstPass&confirmPassword=firstPass&role=student  /Users/Login?username=firstStudent&password=firstPass  /Users/Logout  /Users/Login?username=nonexistingStudent&password=nonExistingPassword  /Users/Login?username=firstStudent&password=wrongPass  /Users/Logout  /Courses/Details?courseId=1  /Users/Register?username=firstLecturer&password=firstPass&confirmPassword=firstPass&role=lecturer  /Users/Login?password=firstPass&username=firstLecturer  /Courses/Create?name=High-Quality+Code  /Courses/Details?courseId=1  /Users/Logout  /Courses/Details?courseId=1  /Courses/Enroll?courseId=1  /Users/Login?password=firstPass&username=firstLecturer  /Courses/Enroll?courseId=1  /Courses/Enroll?courseId=1  /Courses/Enroll?courseId=10  /Users/Logout  /Courses/Create?name=Object-Oriented+Programming  /Users/Login?password=firstPass&username=firstStudent  /Courses/Create?name=Object-Oriented+Programming  /Users/Logout  /Users/Login?password=firstPass&username=firstLecturer  /Courses/Create?name=a  /Users/Logout  /Courses/AddLecture?courseId=1&lectureName=High-Quality+Methods  /Users/Login?password=firstPass&username=firstStudent  /Courses/AddLecture?courseId=1&lectureName=High-Quality+Methods  /Users/Logout  /Users/Login?password=firstPass&username=firstLecturer  /Courses/AddLecture?courseId=1&lectureName=a  /Courses/AddLecture?courseId=10&lectureName=High-Quality+Methods |

## Sample Output 2

|  |
| --- |
| The username must be at least 5 symbols long.  The password must be at least 6 symbols long.  The provided passwords do not match.  User firstStudent registered successfully.  User firstStudent logged in successfully.  There is already a logged in user.  There is already a logged in user.  There is already a logged in user.  User firstStudent logged out successfully.  A user with username nonexistingStudent does not exist.  The provided password is wrong.  There is no currently logged in user.  There is no currently logged in user.  User firstLecturer registered successfully.  User firstLecturer logged in successfully.  Course High-Quality Code created successfully.  You are not enrolled in this course.  User firstLecturer logged out successfully.  There is no currently logged in user.  There is no currently logged in user.  User firstLecturer logged in successfully.  Student successfully enrolled in course High-Quality Code.  You are already enrolled in this course.  There is no course with ID 10.  User firstLecturer logged out successfully.  There is no currently logged in user.  User firstStudent logged in successfully.  The current user is not authorized to perform this operation.  User firstStudent logged out successfully.  User firstLecturer logged in successfully.  The course name must be at least 5 symbols long.  User firstLecturer logged out successfully.  There is no currently logged in user.  User firstStudent logged in successfully.  The current user is not authorized to perform this operation.  User firstStudent logged out successfully.  User firstLecturer logged in successfully.  The lecture name must be at least 3 symbols long.  There is no course with ID 10. |

## 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, as long as it improves the code quality. You may create as many classes, interfaces, enumerations, structures, etc. as you wish.

**32 score**

## Problem 2. StyleCop

Make StyleCop run without any errors on your code (ignore all documentation-related errors). Use the following StyleCop settings:



**4 score**

## Problem 3. Bug Fixing

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

**5 score**

## Problem 4. Code Documentation

**Document the following interfaces and methods** using C# XML documentation:

* **IRoute** (declaration + members)
* **IRepository** (declaration + members)
* **The controller action which performs user registration**

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:**

* **Get(int id)** on the implementation of **IRepository<T>**
* **Display()** on the implementation of **IView** (note: you may need to inherit a class)
* The method implementing the **/Users/Logout** action
* The method implementing the **/Courses/All** action

Any other code is not required to be tested. The **code coverage** should be **at** **least 80% 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. Ideally, this would be avoided using mocking, but **feel free to do it using regular unit testing only**. However, you may want to create empty classes for some of the tests to simplify your work.

**30 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 bottleneck descriptions in addition to the fixes).

**6 score**

## Problem 7. 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**

## Problem 8. \* Mocking

Test the controller action **/Courses/AddLecture** using mocking. Use the Moq framework.

Note that you may need to make your code testable first.

**10 score (bonus)**