# High-Quality Code Exam – Chepelare Hotel Booking System

**SoftUni Tent Conf 2015** is coming and some of the participants prefer staying in a hotel to freezing to death in a tent. However, they are so many that all local hotels are struggling placing them all. As you are the only person left who’s not busy preparing for the Conf, your task is to help the hotels put an online booking system so they can manage their rooms, bookings and guests more easily.

Luckily, you don’t have to start from scratch. You achieved the “Ninja Googler” badge back in high school and you managed to find an online booking system. The only trouble is, it’s not so well-written and has a few bugs.

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 previous project leader might have left, and **improve the general performance** (execution speed) of the code. Since the project was open source, the team didn't have time to **write** any **unit tests**, so you’ve got to do this as well.

You are given the original code and the design document, specifying the task at hand. You have also found another document with 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 hotel booking system keeps information about **users**, **venues** (hotels), **rooms** and **bookings**.

A **user** has a **username**, **password**, **role** (User or Venue Admin), and some **bookings**. A **venue** has **name**, **address**, **description**, **owner** and some **rooms**. A **room** has **number of places**, **price per day**, some **bookings** and some **dates when it’s available to book**.

The booking system holds information about **users**, **courses** and **lectures**. A **booking** has a **client**, **start** and **end date** and **total price**.

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

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

Once logged in, a person may be a venue admin or a regular user and he / she is able to **log out**. Users can do all things guests can do, and they can additionally **book rooms** (if they are available). Venue admins can do all things available to guests and users, and they can **add new venues** **and rooms**, and add information about when these rooms are available to book.

## System design

The project follows the **Model-View-Controller** architecture.

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 an **error view** containing 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**, **Venue**, **Room** and **Booking**. 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 venue name must be at least 3 symbols long
* A venue address must be at least 3 symbols long
* A room’s places cannot be negative (< 0)
* A room’s price per day cannot be negative (< 0)
* All of a room’s available days must be valid dates
* All start dates must be less than or equal to end dates)

All dates are in the format **dd.MM.yyyy**. All prices are rounded to two numbers after the decimal point.

In case validation fails, the system throws an **ArgumentException** with one of the following messages:

* Minimal length: **The [parameter\_name] must be at least [min\_length] symbols long.**
* Minimal value: **The [parameter\_name] must not be less than [min\_value].**
* Date range: **The date range is invalid.**

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-256** hashing algorithm to store password hashes and keeps them as hexadecimal strings. **The system must NOT keep any passwords as plain text!**

In order to work with model collections, the project has a **data layer**. The data layer consists of several **repositories** and a central **database**. 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
* **Updating** an existing item by its unique ID
* **Deleting** an item by its unique ID

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

A **database** (unit of work) class combines all repositories defined for the application and is responsible for maintaining the integrity of all data transactions.

**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 currently logged in user doesn't have sufficient rights 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.

There is also a **Shared** views folder which holds special views (such as error pages).

The default error page takes an error message as its model and displays it like so:

**Something happened!!1  
[error\_message]**

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  
| `-- Shared  
| | -- Error.cs**

## System functionality

The Chepelare Hotel 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 **user** or **venueAdmin**.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | The user [username] has been registered and may login. | 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 | The user [username] has logged in. | 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/MyProfile**

Displays the profile of the currently logged in user.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | **In case there are no bookings:**  [username] You have not made any bookings yet.  **In case there are some bookings: shows all user bookings (in order of addition)**  [username]  Your bookings:  \* 19.10.2015 - 20.10.2015 ($32.50)  \* 20.10.2015 - 23.10.2015 ($45.00) | None |
| There is no logged in user in the system | There is no currently logged in user. | ArgumentException |

* **/Users/Logout**

Terminates the session of the currently logged user.

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

* **/Venues/All**

Displays short information about all venues in the system (in order of addition).

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | **In case there are no venues:** There are currently no venues to show.  **In case there are venues, displays all courses in order of their addition to the system in the format**  \*[venue\_id] [venue\_name], located at [venue\_address] Free rooms: [number\_of\_rooms] | None |

* **/Venues/Details?id=[int]**

Displays detailed information about the venue with the specified ID.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | **In case there are no rooms:**  [venue\_name]  Located at [venue\_address]  No rooms are currently available.  **In case there are rooms, displays them in order of their addition:**  [venue\_name]  Located at [venue\_address]  Available rooms:  \* 2 places ($20.50 per day)  \* 3 places ($20.00 per day)  \* 1 places ($10.32 per day) | 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 user or an admin | The currently logged in user doesn't have sufficient rights to perform this operation. | AuthorizationFailed Exception |
| The venue does not exist | The venue with ID [id] does not exist. | ArgumentException |

* **/Venues/Rooms?id=[int]**

Displays detailed information about the venue with the specified ID.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | **In case there are no rooms:**  Available rooms for venue [venue\_name]:  No rooms are currently available.  **In case there are rooms, displays them in order of their addition. If there are periods when the room is available to book, displays them sorted by the start date:**  Available rooms for venue [venue\_name]:  \*[room\_id] 2 places, $30.00 per day  This room is not currently available.  \*[room\_id] 3 places, $42.26 per day  Available dates:  - 20.10.2015 - 23.10.2015  - 25.10.2015 - 26.10.2015  \*[room\_id] 1 places, $20.00 per day  This room is not currently available.  \*[room\_id] 2 places, $42.00 per day  Available dates:  - 20.10.2015 - 21.10.2015 | 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 user or an admin | The currently logged in user doesn't have sufficient rights to perform this operation. | AuthorizationFailed Exception |
| The venue does not exist | The venue with ID [id] does not exist. | ArgumentException |

* **/Venues/Add?name=[string]&address=[string]&description=[string]**

Adds a new venue to the system.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | The venue [venue\_name] with ID [venue\_id] has been 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 an admin | The currently logged in user doesn't have sufficient rights to perform this operation. | AuthorizationFailed Exception |

* **/Rooms/Add?venueId=[int]&places=[int]&pricePerDay=[price]**

Adds a new room to the venue with the specified ID.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | The room with ID [room\_id] has been 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 an admin | The currently logged in user doesn't have sufficient rights to perform this operation. | AuthorizationFailed Exception |
| The venue does not exist | The venue with ID [id] does not exist. | ArgumentException |

* **/Rooms/AddPeriod?roomId=[int]&startDate=[date]&endDate=[date]**

Adds a new period to the room with the specified ID.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | The period has been added to room with ID [room\_id]. | 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 an admin | The currently logged in user doesn't have sufficient rights to perform this operation. | AuthorizationFailed Exception |
| The room does not exist | The room with ID [id] does not exist. | ArgumentException |
| The end date is before the start date | The date range is invalid. | ArgumentException |

* **/Rooms/ViewBookings?id=[int]**

Displays all booking information about the room with the specified ID.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | **In case there are no bookings:**  There are no bookings for this room.  **In case there are some bookings, displays them in order of their placement and displays their total price:**  Room bookings:  \* 20.10.2015 - 23.10.2015 ($60.00)  \* 23.10.2015 - 24.10.2015 ($10.00)  Total booking price: $70.00 | 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 an admin | The currently logged in user doesn't have sufficient rights to perform this operation. | AuthorizationFailed Exception |
| The room does not exist | The room with ID [id] does not exist. | ArgumentException |

* **/Rooms/Book?roomId=[int]&startDate=[date]&endDate=[date]&comments=[string]**

Books the room with the specified ID. As a result, creates a new booking entry in the database and updates the room’s available periods.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | **In case there are no bookings:**  There are no bookings for this room.  **In case there are some bookings, displays them in order of their placement and displays their total price:**  Room bookings:  \* 20.10.2015 - 23.10.2015 ($60.00)  \* 23.10.2015 - 24.10.2015 ($10.00)  Total booking price: $70.00 | 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 user or an admin | The currently logged in user doesn't have sufficient rights to perform this operation. | AuthorizationFailed Exception |
| The room does not exist | The room with ID [id] does not exist. | ArgumentException |
| The end date is before the start date | The date range is invalid. | ArgumentException |
| There is no valid period when the room is available | The room is not available to book in the period [start\_date] - [end\_date]. | 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=newUser&password=PassW0rd&confirmPassword=PassW0rd&role=user  /Users/Register?username=anotherUser&password=An0th3rPassW0rd&confirmPassword=An0th3rPassW0rd&role=user  /Users/Register?username=admin&password=AdM!nPassW0rd&confirmPassword=AdM!nPassW0rd&role=venueAdmin  /Users/Login?username=newUser&password=PassW0rd  /Users/MyProfile  /Users/Logout  /Users/MyProfile  /Venues/All  /Users/Login?username=admin&password=AdM!nPassW0rd  /Venues/Add?name=New%20venue&address=Sofia&description=This%20is%20the%20first%20venue%20here.  /Venues/Add?name=New%20venue%202&address=Sofia&description=This%20is%20the%20second%20venue%20here.  /Venues/Add?name=New%20venue%203&address=Sofia&description=This%20is%20the%20third%20venue%20here.  /Venues/All  /Venues/Details?id=1  /Venues/Rooms?id=1  /Rooms/Add?venueId=1&places=2&pricePerDay=30.00  /Rooms/Add?venueId=1&places=2&pricePerDay=40.00  /Rooms/Add?venueId=2&places=2&pricePerDay=35.20  /Venues/All  /Venues/Details?id=1  /Venues/Rooms?id=1  /Rooms/AddPeriod?roomId=1&startDate=20.10.2015&endDate=25.10.2015  /Rooms/AddPeriod?roomId=1&startDate=27.10.2015&endDate=30.10.2015  /Rooms/AddPeriod?roomId=2&startDate=24.10.2015&endDate=26.10.2015  /Rooms/Book?roomId=1&startDate=21.10.2015&endDate=23.10.2015&comments=Nothing%20special  /Venues/Rooms?id=1  /Users/MyProfile  /Rooms/ViewBookings?id=1 |

## Sample Output 1

|  |
| --- |
| The user newUser has been registered and may login.  The user anotherUser has been registered and may login.  The user admin has been registered and may login.  The user newUser has logged in.  newUser  You have not made any bookings yet.  The user newUser has logged out.  Something happened!!1  There is no currently logged in user.  There are currently no venues to show.  The user admin has logged in.  The venue New venue with ID 1 has been created successfully.  The venue New venue 2 with ID 2 has been created successfully.  The venue New venue 3 with ID 3 has been created successfully.  \*[1] New venue, located at Sofia  Free rooms: 0  \*[2] New venue 2, located at Sofia  Free rooms: 0  \*[3] New venue 3, located at Sofia  Free rooms: 0  New venue  located at Sofia  Description: This is the first venue here.  No rooms are currently available.  Available rooms for venue New venue:  No rooms are currently available.  The room with ID 1 has been created successfully.  The room with ID 2 has been created successfully.  The room with ID 3 has been created successfully.  \*[1] New venue, located at Sofia  Free rooms: 2  \*[2] New venue 2, located at Sofia  Free rooms: 1  \*[3] New venue 3, located at Sofia  Free rooms: 0  New venue  located at Sofia  Description: This is the first venue here.  Available rooms:  \* 2 places ($30.00 per day)  \* 2 places ($40.00 per day)  Available rooms for venue New venue:  \*[1] 2 places, $30.00 per day  This room is not currently available.  \*[2] 2 places, $40.00 per day  This room is not currently available.  The period has been added to room with ID 1.  The period has been added to room with ID 1.  The period has been added to room with ID 2.  Room booked from 21.10.2015 to 23.10.2015 for $60.00.  Available rooms for venue New venue:  \*[1] 2 places, $30.00 per day  Available dates:  - 20.10.2015 - 21.10.2015  - 23.10.2015 - 25.10.2015  - 27.10.2015 - 30.10.2015  \*[2] 2 places, $40.00 per day  Available dates:  - 24.10.2015 - 26.10.2015  admin  Your bookings:  \* 21.10.2015 - 23.10.2015 ($60.00)  Room bookings:  \* 21.10.2015 - 23.10.2015 ($60.00)  Total booking price: $60.00 |

## Sample Input 2

|  |
| --- |
| /Users/Register?username=newUser&password=PassW0rd&confirmPassword=PassW0rd&role=user  /Users/Register?username=admin&password=Adm1NPassW0rd&confirmPassword=Adm1NPassW0rd&role=venueAdmin  /Users/Register?username=a&password=PassW0rd&confirmPassword=PassW0rd&role=user  /Users/Register?username=newUser2&password=PassW0rd123&confirmPassword=PassW0rd098&role=user  /Users/Register?username=newUser&password=pass&confirmPassword=pass&role=user  /Users/Logout  /Users/MyProfile  /Venues/All  /Venues/Details?id=1  /Users/Login?username=admin&password=Adm1NPassW0rd  /Venues/Add?name=New%20venue&address=Sofia&description=This%20is%20the%20first%20venue%20here.  /Venues/Details?id=1  /Venues/Rooms?id=1  /Rooms/Add?venueId=1&places=2&pricePerDay=30.00  /Venues/Rooms?id=1  /Users/Logout  /Venues/Details?id=1  /Venues/Details?id=10  /Venues/Rooms?id=1  /Venues/Rooms?id=10  /Venues/All  /Rooms/Add?venueId=1&places=2&pricePerDay=30.00  /Rooms/AddPeriod?roomId=1&startDate=20.10.2015&endDate=25.10.2015  /Rooms/ViewBookings?id=1  /Rooms/Book?roomId=1&startDate=21.10.2015&endDate=23.10.2015&comments=Nothing%20special  /Users/Login?username=admin&password=Adm1NPassW0rd  /Rooms/AddPeriod?roomId=1&startDate=20.10.2015&endDate=25.10.2015  /Users/Logout  /Rooms/ViewBookings?id=1 |

## Sample Output 2

|  |
| --- |
| The user newUser has been registered and may login.  The user admin has been registered and may login.  Something happened!!1  The username must be at least 5 symbols long.  Something happened!!1  The provided passwords do not match.  Something happened!!1  A user with username newUser already exists.  Something happened!!1  There is no currently logged in user.  Something happened!!1  There is no currently logged in user.  There are currently no venues to show.  Something happened!!1  There is no currently logged in user.  The user admin has logged in.  The venue New venue with ID 1 has been created successfully.  New venue  Located at Sofia  Description: This is the first venue here.  No rooms are currently available.  Available rooms for venue New venue:  No rooms are currently available.  The room with ID 1 has been created successfully.  Available rooms for venue New venue:  \*[1] 2 places, $30.00 per day  This room is not currently available.  The user admin has logged out.  Something happened!!1  There is no currently logged in user.  Something happened!!1  There is no currently logged in user.  Available rooms for venue New venue:  \*[1] 2 places, $30.00 per day  This room is not currently available.  Something happened!!1  The venue with ID 10 does not exist.  \*[1] New venue, located at Sofia  Free rooms: 1  Something happened!!1  There is no currently logged in user.  Something happened!!1  There is no currently logged in user.  Something happened!!1  There is no currently logged in user.  Something happened!!1  There is no currently logged in user.  The user admin has logged in.  The period has been added to room with ID 1.  The user admin has logged out.  Something happened!!1  There is no currently logged in user. |

## 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:

* **Controller** (declaration + members)
* **IRepository** (declaration + members)
* **The controller action which adds a new room to a venue**

Any other documentation is **not** required. Each documentation gives 0.5 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>** (test the class, not the interface, i. e., don’t use mocking)
* The **Controller.Authorize()** method
* The method implementing the **/Users/Logout** action
* The method implementing the **/Venues/All** action

Any other code is not required to be tested. The **code coverage** should be **at** **least 80% for the specified methods**. 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 **/Rooms/Add** using mocking. Use the Moq framework.

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

**10 score (bonus)**