# C# Web Basics Exam – 25 October 2020

# Git



Exam problems for the [C# Web Basics course @ SoftUni](https://softuni.bg/courses/csharp-web-basics).

Submit your solutions in the **SoftUni judge** system **(delete all "bin"/"obj" folders).**

**Git** is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

## Technological Requirements

* Use the **SUS**
* Use **Entity Framework Core – 3.1**

The Technological Requirements are **ABSOLUTE**. If you do not follow them, you will **NOT** be scored for other Requirements.

Now that you know the **Technological Requirements**, let us see what the **Functional Requirements** are.

## Database Requirements

The **Database** of **Git** needs to support **3 entities**:

### User

* Has an Id – a **string, Primary Key**
* Has a Username – a string with **min length** **5** and **max length 20** (**required**)
* Has an Email - a string (**required**)
* Has a Password – a string with **min length** **6** and **max length 20** - hashed in the database (**required**)
* Has **Repositories** collection – a **Repository** type
* Has **Commits** collection – a **Commit** type

### Repository

* Has an Id – a **string, Primary Key**
* Has a Name – a string with **min length** **3** and **max length 10** (**required**)
* Has a CreatedOn – a **datetime** (**required**)
* Has a IsPublic – **bool** (**required**)
* Has a OwnerId – a **string**
* Has a Owner – a User object
* Has **Commits** collection – a **Commit** type

### Commit

* Has an **Id** – a **string**, **Primary Key**
* Has a **Description** – a string with **min length** **5** (**required**)
* Has a **CreatedOn** – a **datetime** (**required**)
* Has a **CreatorId** – a **string**
* Has Creator – a User object
* Has **RepositoryId** – a **string**
* Has **Repository**– a Repository object

Implement the entities with the **correct datatypes** and their **relations**.

## Page Requirements

### Index Page (logged-out user) – DONE!!!



### Login Page (logged-out user) – DONE!!!



### Register Page (logged-out user) – DONE!!!



### /Repositories/All (logged-in user) – DONE!!!

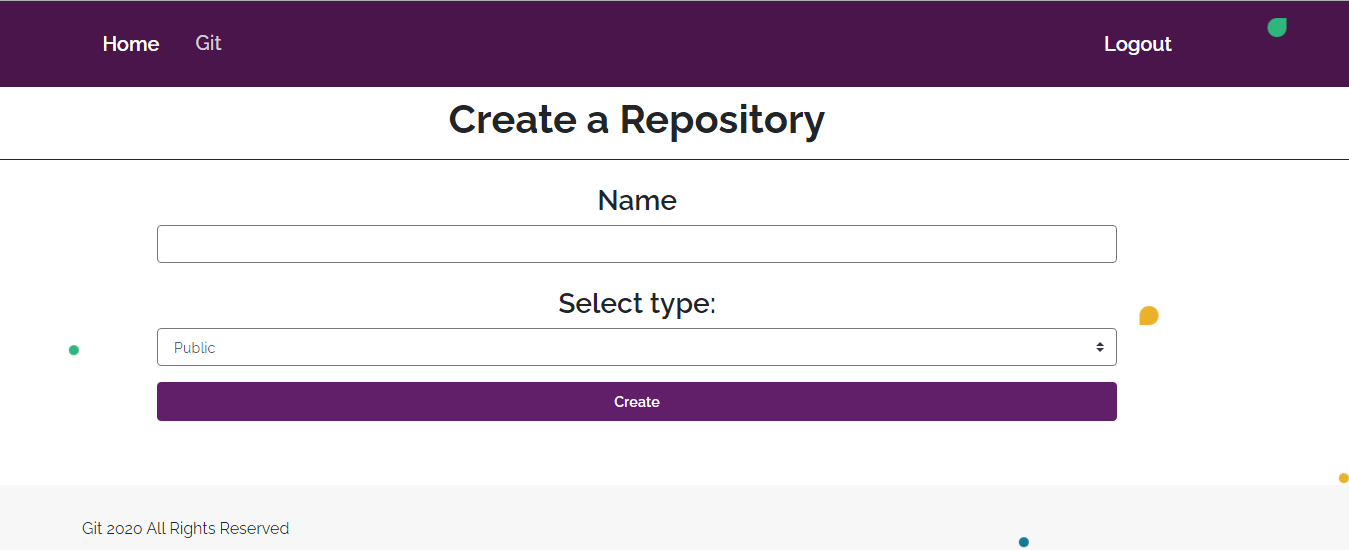


**NOTE**:

If the user is logged in and he tries to go the home page, the application must redirect him to the **/Repositories/All**

**DateTime format does not matter – DONE!!!**

### /Repositories/Create (logged-in user) – DONE!!!



### /Commits/Create?id={id} (logged-in user) – DONE!!!



### /Commits/All (logged-in user) – DONE!!!



DateTime format does not matter

### /Commits/Delete?id ={id} (logged-in user) – DONE!!!

Deletes the given commit. Only the owner can delete his commit. If everything is successful, the user must be **redirect to the all repositories.**

**NOTE**: The templates should look **EXACTLY** as shown above.

**NOTE**: The templates do **NOT** **require** **additional** **CSS** for you to write. Only **bootstrap** and the **given css** are enough.

## Functionality

The functionality of **Git** platform is very simple.

### Users

Guests can see Register, Login, Index and **All Repositories** views.

Users can create repositories, which can be private or public.

User can see all publicrepositorieson the repositories page. From the repository page they can also commit to a repository. They are able to delete their commits.

### Repositories

**Users can add repositories**. All **public** repositories are visualized on the **all repositories page**, each one in its own separate rectangular element.

**Repositories** are visualized on the **all repositories page** as a table with **Name**, **Owner**, **Created On**, **Commits Count** (total commits) and **Commit** action.

**Repositories** are visualized on the **all repositories page** with button – [**Commit**].

* The [**Commit**] button leads to the **create commit** page and creates **commit** for the **particular repository**.

### **Commits**

**Users can make commits on all repositories**. All **commits** are visualized on the **all commits page**, each one in its own separate rectangular element. Only **user's own commits** must be listed.

**Commits** are visualized on the **all commits page** with button – [**Delete**].

* **The [Delete] button deletes the particular commits only if the owner tries to delete it**.

### Redirections – DONE!!!

* Upon successful **Registration** of a **User**, you should be redirected to the **Login** **Page**.- DONE!!
* Upon successful **Login** of a **User**, you should be redirected to the /**Repositories/All**. – DONE!!!
* Upon successful **creation** of **a** **repository**, you should be redirected to the /**Repositories/All**. – DONE!!!
* Upon successful **creating commit to a repository**, should be redirected to the /**Repositories/All**. – DONE!!!
* Upon successful **deletion** of **a commit**, should be redirected to the /**Commits/All. – DONE!!!**
* If any of the validations in the POST forms do not pass, redirect to the same page (reload/refresh it) or return an error. – DONE!!!

## Security – DONE!!!

The Security section mainly describes access requirements. Configurations about which users can access specific functionalities and pages:

* Guest (not logged in) users can access Index page. – DONE!!!
* Guest (not logged in) users can access Login page. – DONE!!!
* Guest (not logged in) users can access Register page. – DONE!!!
* Guest (not logged in) users can access Repositores/All page. – DONE!!!
* Users (logged in) cannot access Guest pages. – DONE!!!
* Users (logged in) can access Repositories/All page and functionality. – DONE!!!
* Users (logged in) can access Repositories/Create page. – DONE!!!
* Users (logged in) can access Commits/Delete page and functionality. – DONE!!!
* Users (logged in) can access Commits/Create page and functionality. – DONE!!!
* Users (logged in) can access Commits/All page and functionality. – DONE!!!
* Users (logged in) can access Logout functionality. – DONE!!!

## Code Quality

Make sure you provide the best architecture possible. Structure your code into different classes, follow the principles of high-quality code (**SOLID**). You will be scored for the Code Quality and Architecture of your project.

## Scoring

### Database Requirements – 10 points.

### Template Requirements – 10 points.

### Functionality – 50 points.

### Security – 10 points.

### Code Quality – 15 points.

### Data Validation – 5 points.