TaskBoard

Problems for exam preparation for the "Software Quality Assurance" course from the official "Applied Programmer" curriculum.

The "Task Board" System

"**Task Board**" is a simple information system for managing **tasks in a task board**. Each task consists of **title** + **description**. Tasks are organized in **boards**, which are displayed as columns (sections): **Open**, **In Progress**, **Done**. Users can **view** the task board with the tasks, **search** for tasks by keyword, **view** task details, **create** new tasks and **edit** existing tasks (and move existing tasks from one board to another).

You are given the RESTful **API** + **Web** client app for the task board system. Your assignment is to write **API tests and UI automated tests** for the system.

You are given the following project assets:

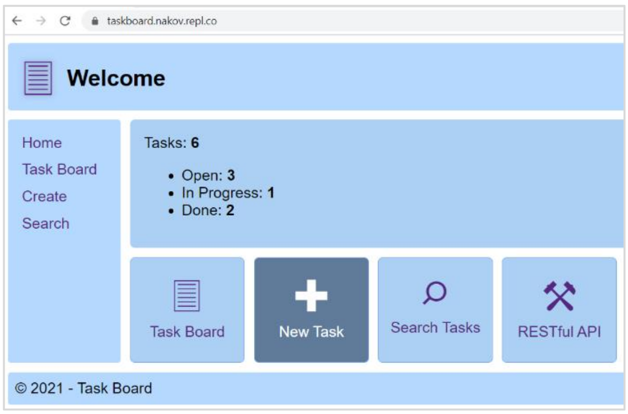
- https://github.com/nakov/TaskBoard – source code of the TaskBoard Web client app and RESTful API

Web App Functionality

The **"**Task Board**" Web app** supports the following operations:

- Home page (view tasks count + menu): **`/`**
- View the boards with tasks: **`/boards`**
- Search tasks form: **`/tasks/search`**
- Search tasks by keyword: **`/tasks/search/:keyword`**
- View task details (by id): **`/tasks/view/:id`**
- Add new task (title + description): **`/tasks/create`**
- Edit task / move to board: **`/tasks/edit/:id`**

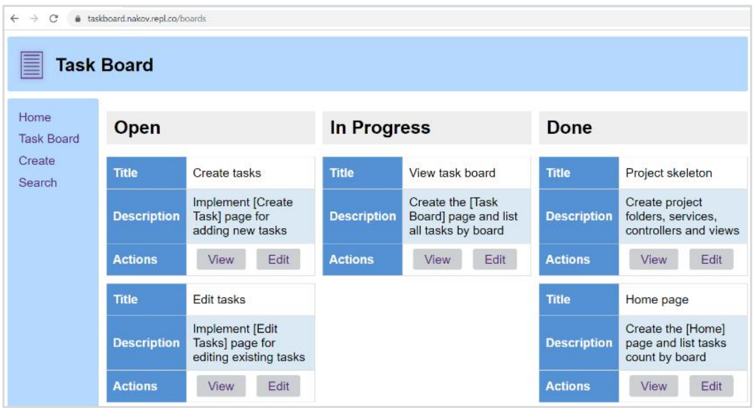
Run the Web app from: https://taskboard-web-app.softuniorg.repl.co.



Tasks: **6**

- Open: **3**
- In Progress: **1**
- Done: **2**

© 2021 - Task Board



Open

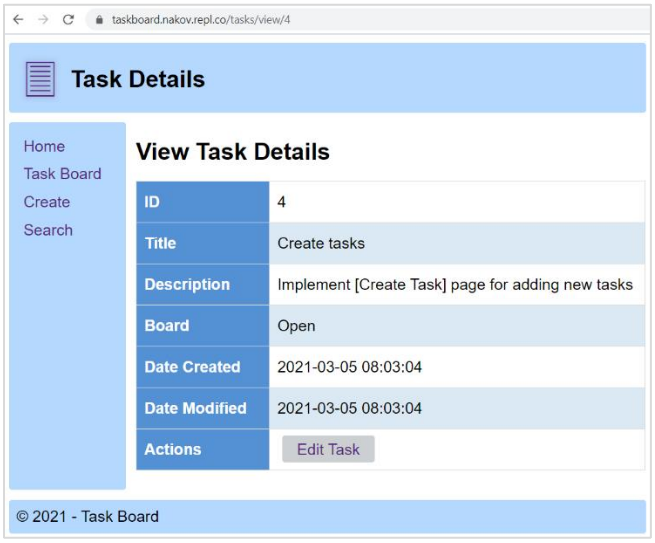
Title	Create tasks
Description	Implement [Create Task] page for adding new tasks
Actions	View Edit
Title	Edit tasks
Description	Implement [Edit Tasks] page for editing existing tasks
Actions	View Edit

In Progress

Title	View task board
Description	Create the [Task Board] page and list all tasks by board
Actions	View Edit

Done

Title	Project skeleton
Description	Create project folders, services, controllers and views
Actions	View Edit
Title	Home page
Description	Create the [Home] page and list tasks count by board
Actions	View Edit


taskboard.nakov.repl.co/tasks/view/4

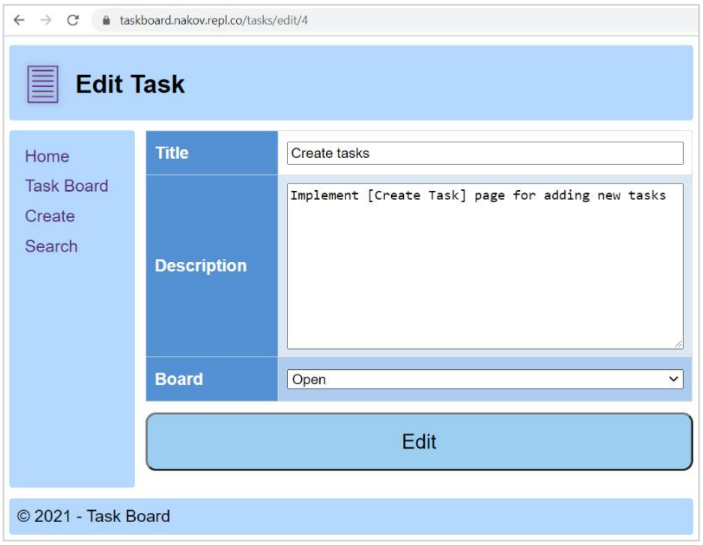
Task Details

Home
Task Board
Create
Search

View Task Details

ID	4
Title	Create tasks
Description	Implement [Create Task] page for adding new tasks
Board	Open
Date Created	2021-03-05 08:03:04
Date Modified	2021-03-05 08:03:04
Actions	Edit Task

© 2021 - Task Board


taskboard.nakov.repl.co/tasks/edit/4

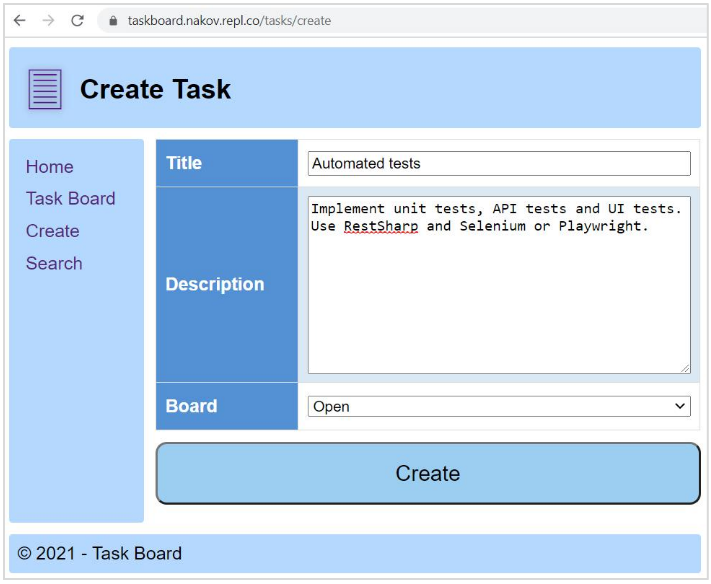
Edit Task

Home
Task Board
Create
Search

Title	Create tasks
Description	Implement [Create Task] page for adding new tasks
Board	Open

Edit

© 2021 - Task Board


taskboard.nakov.repl.co/tasks/create

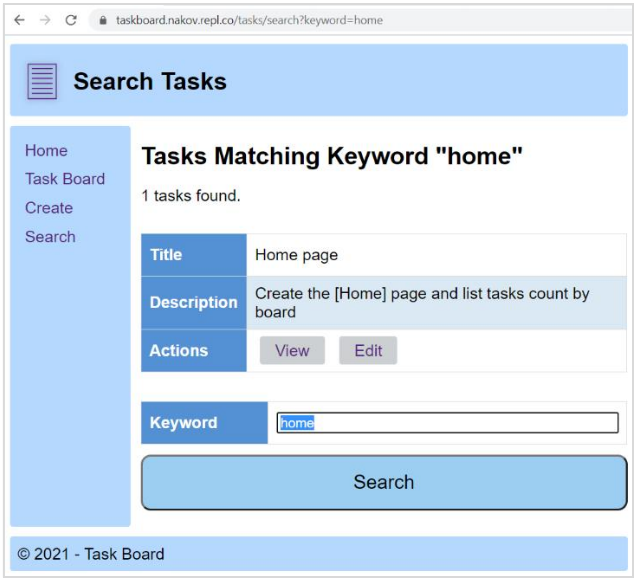
Create Task

Home
Task Board
Create
Search

Title	Automated tests
Description	Implement unit tests, API tests and UI tests. Use RestSharp and Selenium or Playwright.
Board	Open

Create

© 2021 - Task Board


taskboard.nakov.repl.co/tasks/search?keyword=home

Search Tasks

Home
Task Board
Create
Search

Tasks Matching Keyword "home"

1 tasks found.

Title	Home page
Description	Create the [Home] page and list tasks count by board
Actions	View Edit

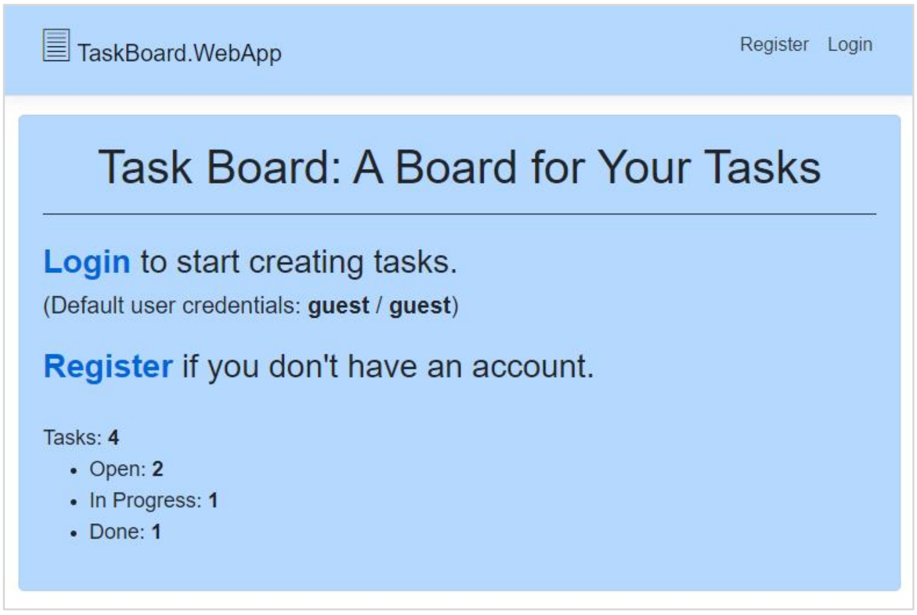
Keyword	home

Search

© 2021 - Task Board

Installing and Running the App

As we have stated the web app can be accessed from: https://taskboard-web-app.softuniorg.repl.co.


TaskBoard.WebApp Register Login

Task Board: A Board for Your Tasks

Login to start creating tasks.

(Default user credentials: **guest** / **guest**)

Register if you don't have an account.

Tasks: **4**
- Open: **2**
- In Progress: **1**
- Done: **1**

Optionally, you can **install** and **run** the app on your **local machine**:

Open the project in Visual Studio, compile and run it. You need to have the following software installed:

- **.NET 5** or later version (https://dotnet.microsoft.com)
- **MS SQL Server LocalDB** (https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/sql-server-express-localdb)
- **Visual Studio 2019** or later version (https://visualstudio.microsoft.com)

API Endpoints

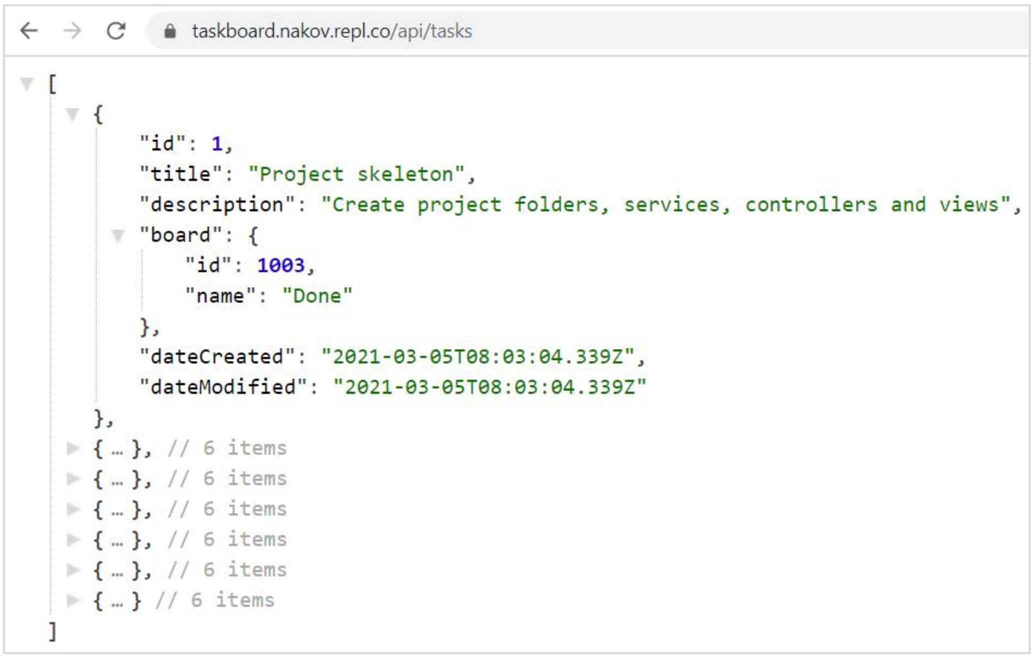
TaskBook exposes a **RESTful API**, available at:

- https://taskboard-web-api.softuniorg.repl.co

The following endpoints are supported:

- **`GET /api/tasks/:id`** – returns a task by given **`id.`**
- **`PUT /api/tasks/:id`** – edits a whole issue by **id**.
- **`POST /api/tasks`** – create a new task (post a JSON object in the request body, e.g. **`{"title":"Add Tests", "description":"API + UI tests", "board":"Open"}`**).
- **`PATCH /api/tasks/:id`** – edit task by **`id`** (send a JSON object in the request body, holding the fields to modify, e.g. **`{"title":"changed title", "board":"Done"}`**).
- **`DELETE /api/tasks/:id`** – delete task by **`id.`**

This is a sample output from an API call to **`/api/tasks`**:


taskboard.nakov.repl.co/api/tasks

```
[
  {
    "id": 1,
    "title": "Project skeleton",
    "description": "Create project folders, services, controllers and views",
    "board": {
      "id": 1003,
      "name": "Done"
    },
    "dateCreated": "2021-03-05T08:03:04.339Z",
    "dateModified": "2021-03-05T08:03:04.339Z"
  },
  {…}, // 6 items
  {…}, // 6 items
  {…}, // 6 items
  {…}, // 6 items
  {…}, // 6 items
  {…} // 6 items
]
```

1. TaskBoard Web App: Unit Tests

Your task is to write **C# unit tests** for the Web App.

- **Instantiate a new task controller** in the "**`StartUp()`**" method and assign a user to the controller with the "**`AssignCurrentUserForController()`**" method.

You should implement the following **automated unit tests**:

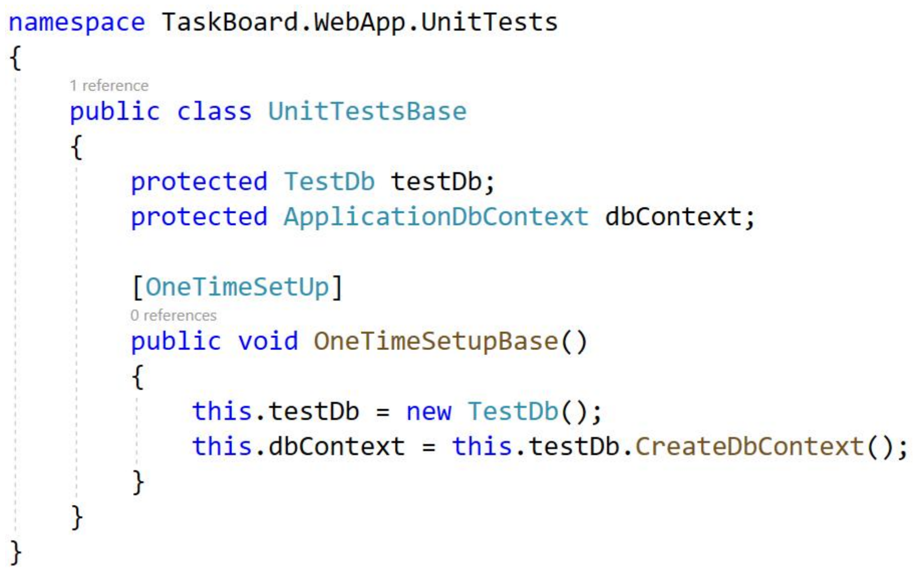
- Test the "**`Create()`**" method by **creating a new task with valid data** and **assert** the database **count increases** and that the **task appears in the database**.
- Test the "**`Delete()`**" method **by id with valid id** and assert the **database count decreases**.
- Test the "**`Delete()`**" method with an **unauthorized** user and **assert** an **unauthorized result** is **returned** and that the **count** has not changed in the **database**.
- Test the "**`Edit()`**" method **with valid and invalid id** and **assert** the task **is edited successfully / has not been edited.**
- Test the "**`Edit()`**" method with an **unauthorized user** and **assert** an **unauthorized result** is returned and that the **count** has **not changed** in the database.

You should use the NUnit **testing framework**.

Hints and Guidelines:

`UnitTestsBase.cs` class:

This class is responsible for **setting up the test database** and the **database context**:

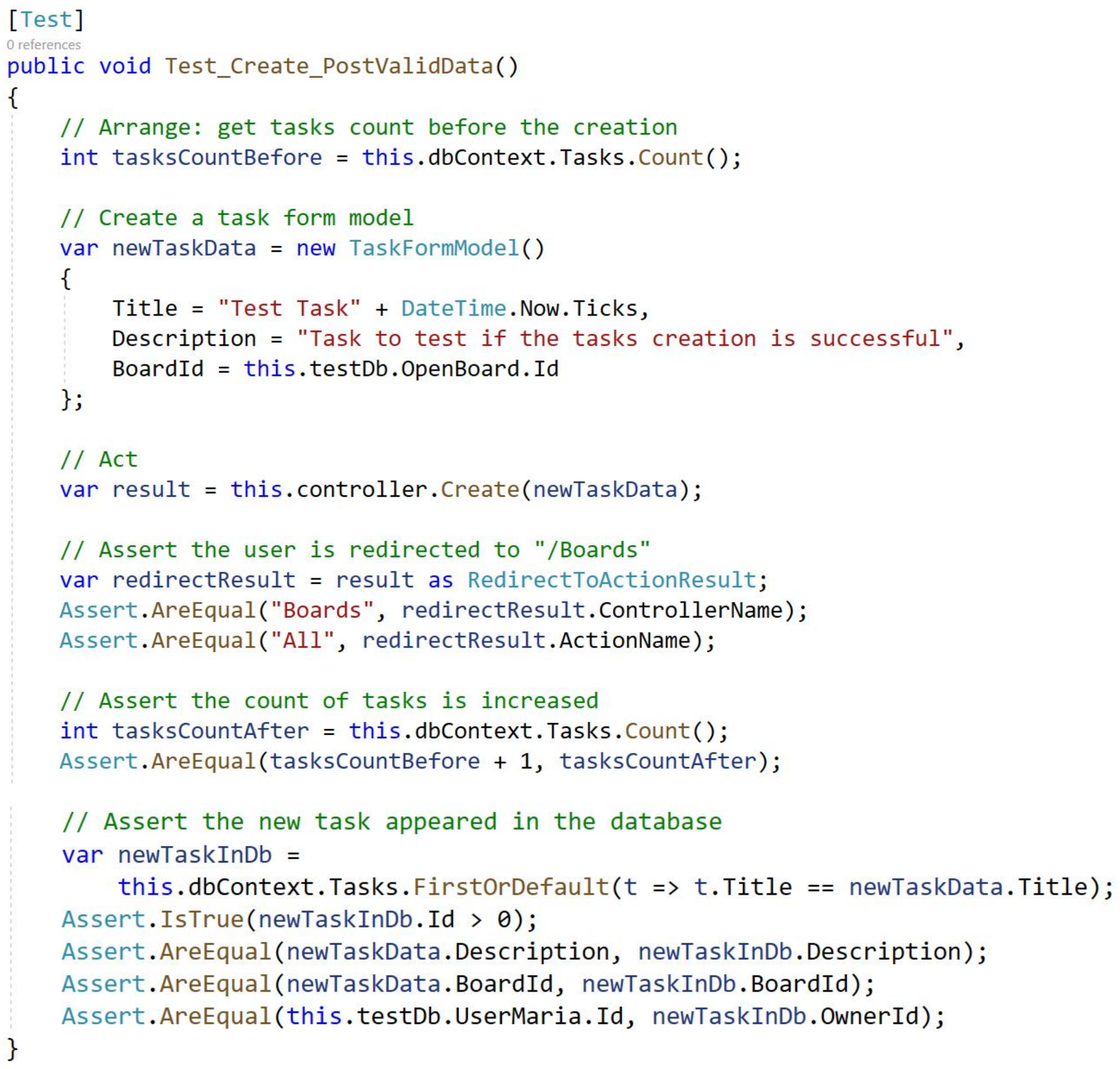


```
namespace TaskBoard.WebApp.UnitTests
{
    public class UnitTestsBase
    {
        protected TestDb testDb;
        protected ApplicationDbContext dbContext;

        [OneTimeSetUp]
        public void OneTimeSetupBase()
        {
            this.testDb = new TestDb();
            this.dbContext = this.testDb.CreateDbContext();
        }
    }
}
```

Test_Create_PostValidData()

- First get the **tasks count**.
- **Create** a task form model.
- Call the "**`Create()`**" method with the new task form model.
- **Assert** the user is redirected to "**`/Boards`**".
- **Assert** the count of tasks is increased.
- **Assert** the new task **appeared** in the **database.**

Here is how the "**`Test_Create_PostValidData()`**" test could look like:

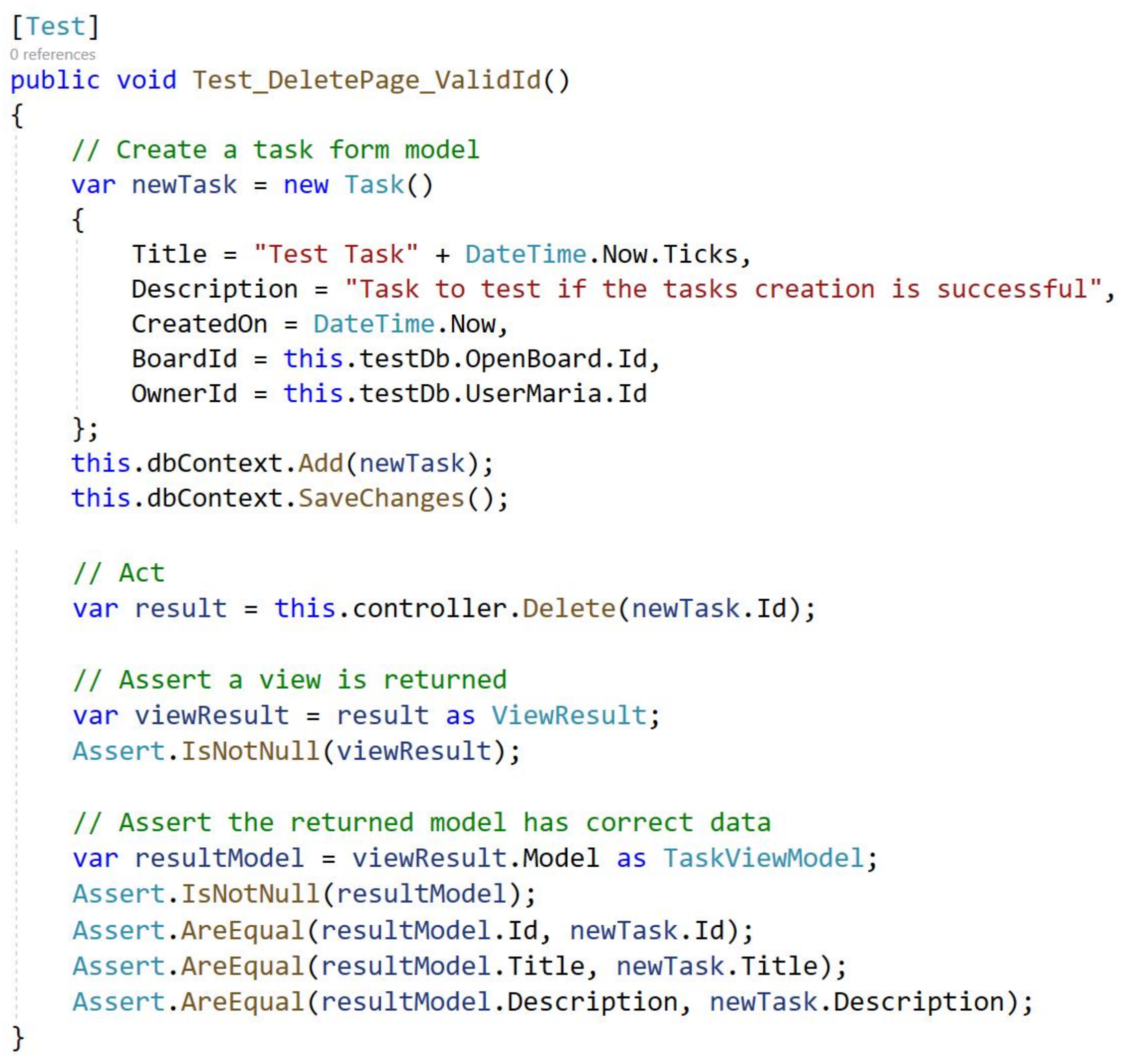


```
[Test]
0 references
public void Test_Create_PostValidData()
{
    // Arrange: get tasks count before the creation
    int tasksCountBefore = this.dbContext.Tasks.Count();

    // Create a task form model
    var newTaskData = new TaskFormModel()
    {
        Title = "Test Task" + DateTime.Now.Ticks,
        Description = "Task to test if the tasks creation is successful",
        BoardId = this.testDb.OpenBoard.Id
    };

    // Act
    var result = this.controller.Create(newTaskData);

    // Assert the user is redirected to "/Boards"
    var redirectResult = result as RedirectToActionResult;
    Assert.AreEqual("Boards", redirectResult.ControllerName);
    Assert.AreEqual("All", redirectResult.ActionName);

    // Assert the count of tasks is increased
    int tasksCountAfter = this.dbContext.Tasks.Count();
    Assert.AreEqual(tasksCountBefore + 1, tasksCountAfter);

    // Assert the new task appeared in the database
    var newTaskInDb =
        this.dbContext.Tasks.FirstOrDefault(t => t.Title == newTaskData.Title);
    Assert.IsTrue(newTaskInDb.Id > 0);
    Assert.AreEqual(newTaskData.Description, newTaskInDb.Description);
    Assert.AreEqual(newTaskData.BoardId, newTaskInDb.BoardId);
    Assert.AreEqual(this.testDb.UserMaria.Id, newTaskInDb.OwnerId);
}
```

Test_DeletePage_ValidId()

- **Create** a task form model.
- Call the "**`Delete()`**" method with the new task id.
- **Assert** a view is returned and that it is not null.
- **Assert** the returned model has correct data (**id**, **title**, **description**).

Here is how the "**`Test_DeletePage_ValidId()`**" test could look like:



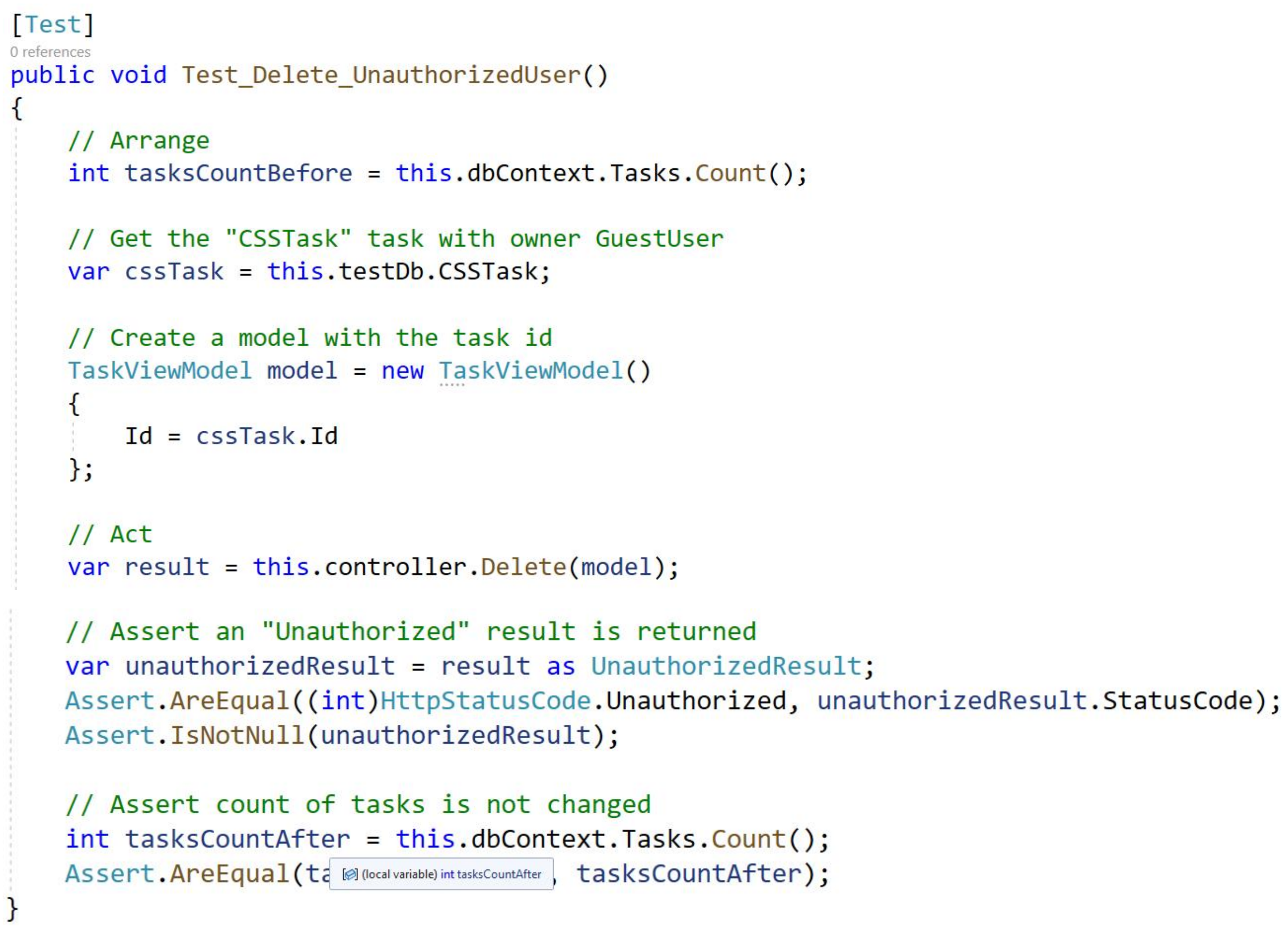
```
[Test]
0 references
public void Test_DeletePage_ValidId()
{
    // Create a task form model
    var newTask = new Task()
    {
        Title = "Test Task" + DateTime.Now.Ticks,
        Description = "Task to test if the tasks creation is successful",
        CreatedOn = DateTime.Now,
        BoardId = this.testDb.OpenBoard.Id,
        OwnerId = this.testDb.UserMaria.Id
    };
    this.dbContext.Add(newTask);
    this.dbContext.SaveChanges();

    // Act
    var result = this.controller.Delete(newTask.Id);

    // Assert a view is returned
    var viewResult = result as ViewResult;
    Assert.IsNotNull(viewResult);

    // Assert the returned model has correct data
    var resultModel = viewResult.Model as TaskViewModel;
    Assert.IsNotNull(resultModel);
    Assert.AreEqual(resultModel.Id, newTask.Id);
    Assert.AreEqual(resultModel.Title, newTask.Title);
    Assert.AreEqual(resultModel.Description, newTask.Description);
}
```

Test_DeletePage_UnauthorizedUser()

For the **unauthorized user** test:

- Get the **tasks count** from the **database**.
- Get the "**`CSSTask`**" from the **database**.
- Create a **model** with the **task id.**
- Call the "**`Delete()`**" method with the model.
- **Assert** an **unauthorized** status code is returned.
- **Assert** the count of tasks has not changed.

Here is how the "**`Test_DeletePage_UnauthorizedUser()`**" test could look like:



```
[Test]
public void Test_Delete_UnauthorizedUser()
{
    // Arrange
    int tasksCountBefore = this.dbContext.Tasks.Count();

    // Get the "CSSTask" task with owner GuestUser
    var cssTask = this.testDb.CSSTask;

    // Create a model with the task id
    TaskViewModel model = new TaskViewModel()
    {
        Id = cssTask.Id
    };

    // Act
    var result = this.controller.Delete(model);

    // Assert an "Unauthorized" result is returned
    var unauthorizedResult = result as UnauthorizedResult;
    Assert.AreEqual((int)HttpStatusCode.Unauthorized, unauthorizedResult.StatusCode);
    Assert.IsNotNull(unauthorizedResult);

    // Assert count of tasks is not changed
    int tasksCountAfter = this.dbContext.Tasks.Count();
    Assert.AreEqual(ta                    , tasksCountAfter);
}
```

Test_Edit_ValidId()

For the **edit task with valid id** test:

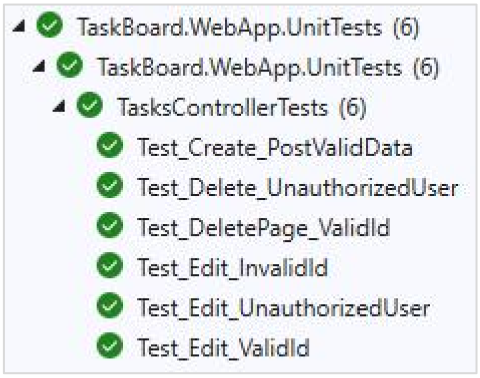
- Get the "**`EditTask`**" from the **database**.
- Call the "**`Edit()`**" method with the **edit task id**.
- **Assert** a not null view is returned.
- **Assert** the returned model has correct data (**title**, **description**, **board id**).

Here is how the "**`Test_Edit_ValidId()`**" test could look like:

```
[Test]
0 references
public void Test_Edit_ValidId()
{
    // Arrange: get the "EditTask" task from the db
    var editTask = this.testDb.EditTask;

    // Act
    var result = this.controller.Edit(editTask.Id);

    // Assert a view is returned
    var viewResult = result as ViewResult;
    Assert.IsNotNull(viewResult);

    // Assert the returned model has correct data
    var resultModel = viewResult.Model as TaskFormModel;
    Assert.IsNotNull(resultModel);
    Assert.AreEqual(resultModel.Title, editTask.Title);
    Assert.AreEqual(resultModel.Description, editTask.Description);
    Assert.AreEqual(resultModel.BoardId, editTask.BoardId);
}
```

Do the rest of the unit tests on your own!

Here are how the tests should look like when run:



```
◢ ✓ TaskBoard.WebApp.UnitTests (6)
  ◢ ✓ TaskBoard.WebApp.UnitTests (6)
    ◢ ✓ TasksControllerTests (6)
        ✓ Test_Create_PostValidData
        ✓ Test_Delete_UnauthorizedUser
        ✓ Test_DeletePage_ValidId
        ✓ Test_Edit_InvalidId
        ✓ Test_Edit_UnauthorizedUser
        ✓ Test_Edit_ValidId
```

2. TaskBoard RESTful API: Automated API Tests

Your task is to write **C# automated tests** for the above provided API endpoints. You should implement the following automated tests:

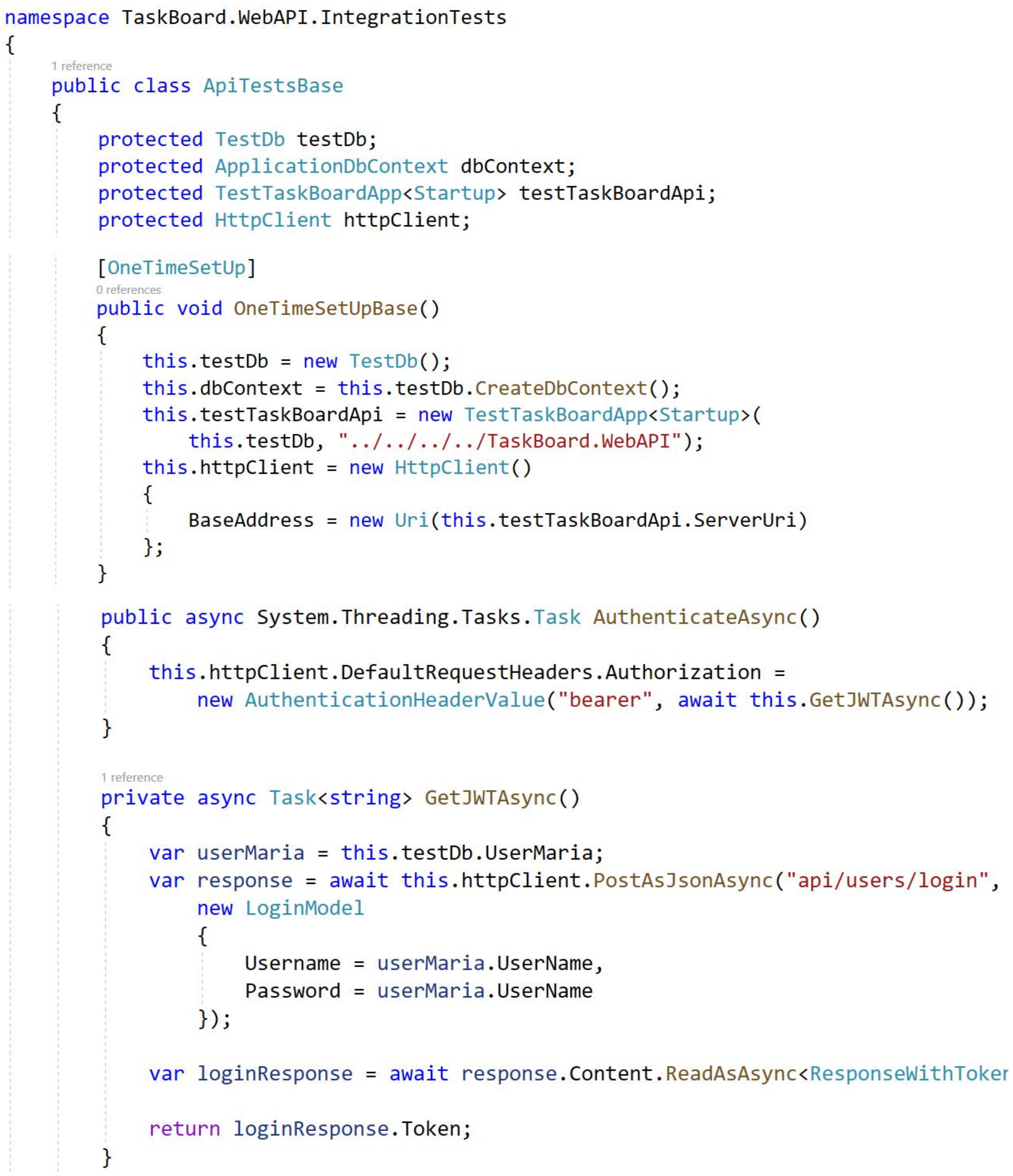
- Create a **new task** (endpoint **POST `/api/tasks`**), holding valid data, and assert the **new task is added** and is properly listed in the task board.
- Create a **new task** (endpoint **DELETE `/api/tasks/:id`**) and then **delete** that task and **assert** that the **deletion is successful**.
- Get the "**`EditTask`**" from the database, edit it, send a **PUT** request (endpoint **PUT `/api/tasks/:id`**), and **assert** the correct **`HTTP Status Code`** is returned. Do the same with an **invalid id**.
- Get the "**`CSSTask`**" from the database and assert **`HTTP Status Code 200 OK`** is returned when **searching** for that **tasks id** (endpoint **GET `/api/tasks/:id`**).
- Get the "**`EditTask`**" from the database, edit **only one part from** it, send a **PATCH** request (endpoint **PATCH `/api/tasks/:id`**), and **assert** the correct **`HTTP Status Code`** is returned.

You should use the NUnit **testing framework** and the **external library RestSharp**.

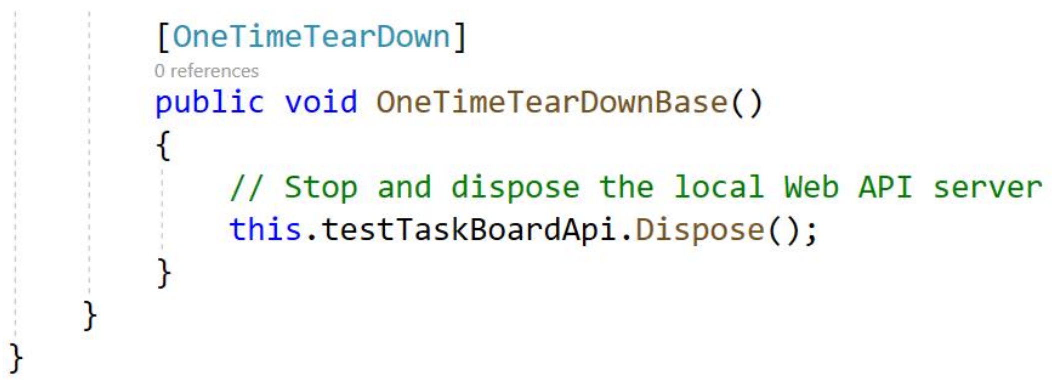
Hints and Guidelines:

`ApiTestsBase.cs` class:

This class is responsible for **setting up the test database** and the **database context** as well as creating a new **TaskBoard** app alongside a **new http client**. The **`HttpClient`** is used for creating an **authentication** with a newly created **`JWT token`**:



```
namespace TaskBoard.WebAPI.IntegrationTests
{
    1 reference
    public class ApiTestsBase
    {
        protected TestDb testDb;
        protected ApplicationDbContext dbContext;
        protected TestTaskBoardApp<Startup> testTaskBoardApi;
        protected HttpClient httpClient;

        [OneTimeSetUp]
        0 references
        public void OneTimeSetUpBase()
        {
            this.testDb = new TestDb();
            this.dbContext = this.testDb.CreateDbContext();
            this.testTaskBoardApi = new TestTaskBoardApp<Startup>(
                this.testDb, "../../../../TaskBoard.WebAPI");
            this.httpClient = new HttpClient()
            {
                BaseAddress = new Uri(this.testTaskBoardApi.ServerUri)
            };
        }

        public async System.Threading.Tasks.Task AuthenticateAsync()
        {
            this.httpClient.DefaultRequestHeaders.Authorization =
                new AuthenticationHeaderValue("bearer", await this.GetJWTAsync());
        }

        1 reference
        private async Task<string> GetJWTAsync()
        {
            var userMaria = this.testDb.UserMaria;
            var response = await this.httpClient.PostAsJsonAsync("api/users/login",
                new LoginModel
                {
                    Username = userMaria.UserName,
                    Password = userMaria.UserName
                });

            var loginResponse = await response.Content.ReadAsAsync<ResponseWithToke

            return loginResponse.Token;
        }
```

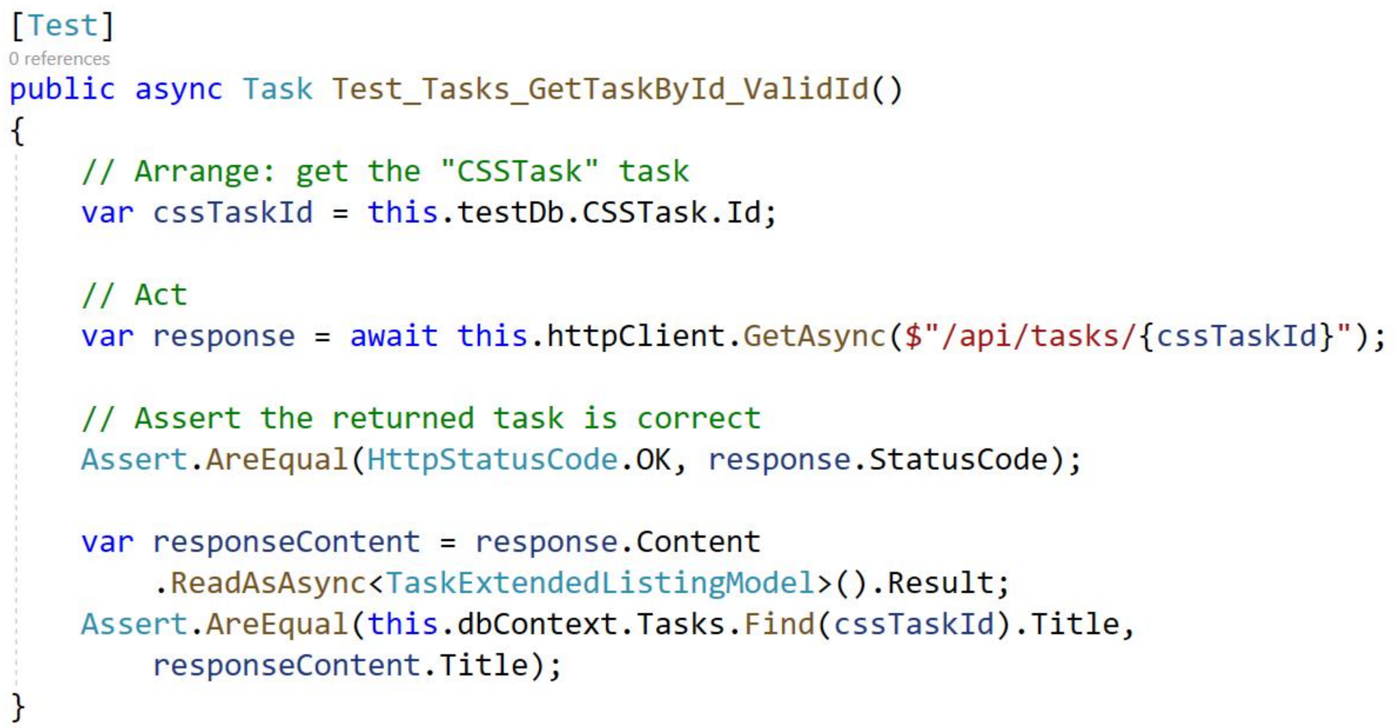


```
        [OneTimeTearDown]
        0 references
        public void OneTimeTearDownBase()
        {
            // Stop and dispose the local Web API server
            this.testTaskBoardApi.Dispose();
        }
    }
}
```

Test_Tasks_GetTaskById_ValidId()

- Get the "**CSSTask**" from the database.
- Call the "**GetAsync()**" method with the endpoint "**/api/tasks/{cssTaskId}**".
- **Assert** the **HttpStatusCode OK** is returned.
- **Assert** the returned task and the task in the database **have the same title**.

Here is how the "**Test_Tasks_GetTaskById_ValidID()**" test could look like:

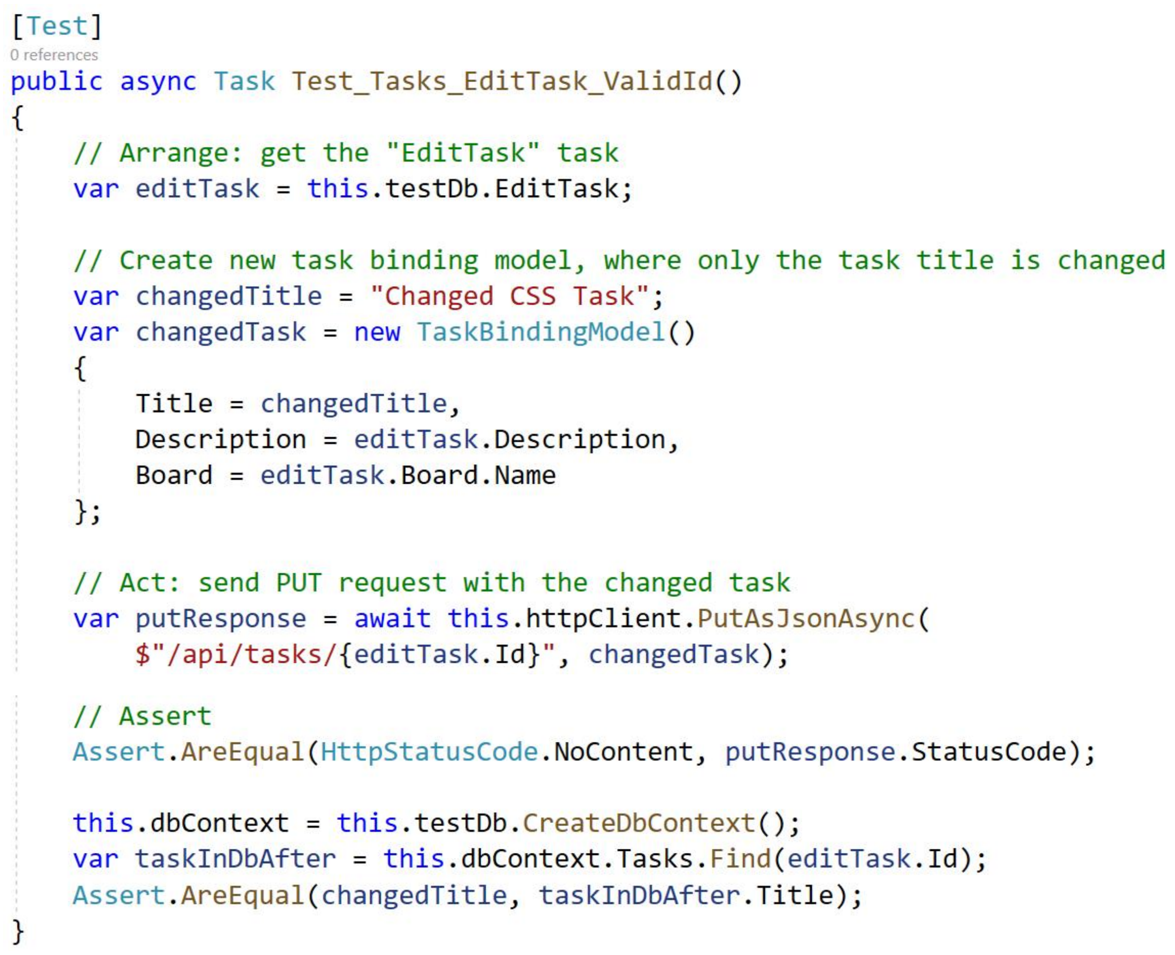


```
[Test]
0 references
public async Task Test_Tasks_GetTaskById_ValidId()
{
    // Arrange: get the "CSSTask" task
    var cssTaskId = this.testDb.CSSTask.Id;

    // Act
    var response = await this.httpClient.GetAsync($"/api/tasks/{cssTaskId}");

    // Assert the returned task is correct
    Assert.AreEqual(HttpStatusCode.OK, response.StatusCode);

    var responseContent = response.Content
        .ReadAsAsync<TaskExtendedListingModel>().Result;
    Assert.AreEqual(this.dbContext.Tasks.Find(cssTaskId).Title,
        responseContent.Title);
}
```

Test_Tasks_EditTask_ValidId()

- Get the "**EditTask**" from the database.
- **Create** new **task binding model**, where only the **task title is changed**.
- Call the "**PutAsJsonAsync()**" method with the endpoint "**/api/tasks/{editTask.Id}**".
- **Assert** the **HttpStatusCode NoContext** is returned.
- **Assert** the task in the database **has a changed title.**

Here is how the "**Test_Tasks_EditTask_ValidId()**" test could look like:



```
[Test]
0 references
public async Task Test_Tasks_EditTask_ValidId()
{
    // Arrange: get the "EditTask" task
    var editTask = this.testDb.EditTask;

    // Create new task binding model, where only the task title is changed
    var changedTitle = "Changed CSS Task";
    var changedTask = new TaskBindingModel()
    {
        Title = changedTitle,
        Description = editTask.Description,
        Board = editTask.Board.Name
    };

    // Act: send PUT request with the changed task
    var putResponse = await this.httpClient.PutAsJsonAsync(
        $"/api/tasks/{editTask.Id}", changedTask);

    // Assert
    Assert.AreEqual(HttpStatusCode.NoContent, putResponse.StatusCode);

    this.dbContext = this.testDb.CreateDbContext();
    var taskInDbAfter = this.dbContext.Tasks.Find(editTask.Id);
    Assert.AreEqual(changedTitle, taskInDbAfter.Title);
}
```

Test_Tasks_DeleteTask_ValidId()

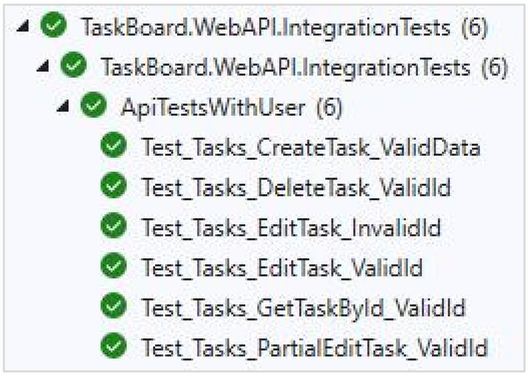
- **Create** a **new task** in the **database** for deleting.
- Call the "**`DeleteAsync()`**" method with the endpoint "**`/api/tasks/{newTask.Id}`**".
- **Assert** the **`HttpStatusCode OK`** is returned.
- **Assert** the task count in the database **has decreased.**
- **Assert** the delete response and the task title are **the same**.

Here is how the "**`Test_Tasks_DeleteTask_ValidId()`**" test could look like:

```
[Test]
0 references
public async Task Test_Tasks_DeleteTask_ValidId()
{
    // Arrange: create a new task in the database for deleting
    Data.Task newTask = new Data.Task()
    {
        Title = "Test task",
        Description = "Test the TaskBoard Web API",
        CreatedOn = DateTime.Now,
        BoardId = this.testDb.OpenBoard.Id,
        OwnerId = this.testDb.UserMaria.Id
    };
    this.dbContext.Add(newTask);
    this.dbContext.SaveChanges();

    var tasksCountBefore = this.dbContext.Tasks.Count();

    // Act: send a DELETE request
    var deleteResponse = await this.httpClient.DeleteAsync(
        $"/api/tasks/{newTask.Id}");

    // Assert the deletion is successfull
    Assert.AreEqual(HttpStatusCode.OK, deleteResponse.StatusCode);

    var deleteResponseContent = deleteResponse.Content
        .ReadAsAsync<TaskListingModel>().Result;
    Assert.AreEqual(newTask.Title, deleteResponseContent.Title);

    var tasksCountAfter = this.dbContext.Tasks.Count();
    Assert.AreEqual(tasksCountBefore - 1, tasksCountAfter);
}
```

Do the rest of the integration tests on your own!

Here are how the tests should look like when run:



```
◢ ✓ TaskBoard.WebAPI.IntegrationTests (6)
  ◢ ✓ TaskBoard.WebAPI.IntegrationTests (6)
    ◢ ✓ ApiTestsWithUser (6)
        ✓ Test_Tasks_CreateTask_ValidData
        ✓ Test_Tasks_DeleteTask_ValidId
        ✓ Test_Tasks_EditTask_InvalidId
        ✓ Test_Tasks_EditTask_ValidId
        ✓ Test_Tasks_GetTaskById_ValidId
        ✓ Test_Tasks_PartialEditTask_ValidId
```

3. TaskBoard Web App: Automated Selenium UI Tests

Write **Selenium-based automated UI tests** for the "**TaskBoard**" app. You should implement the following **automated UI tests**:

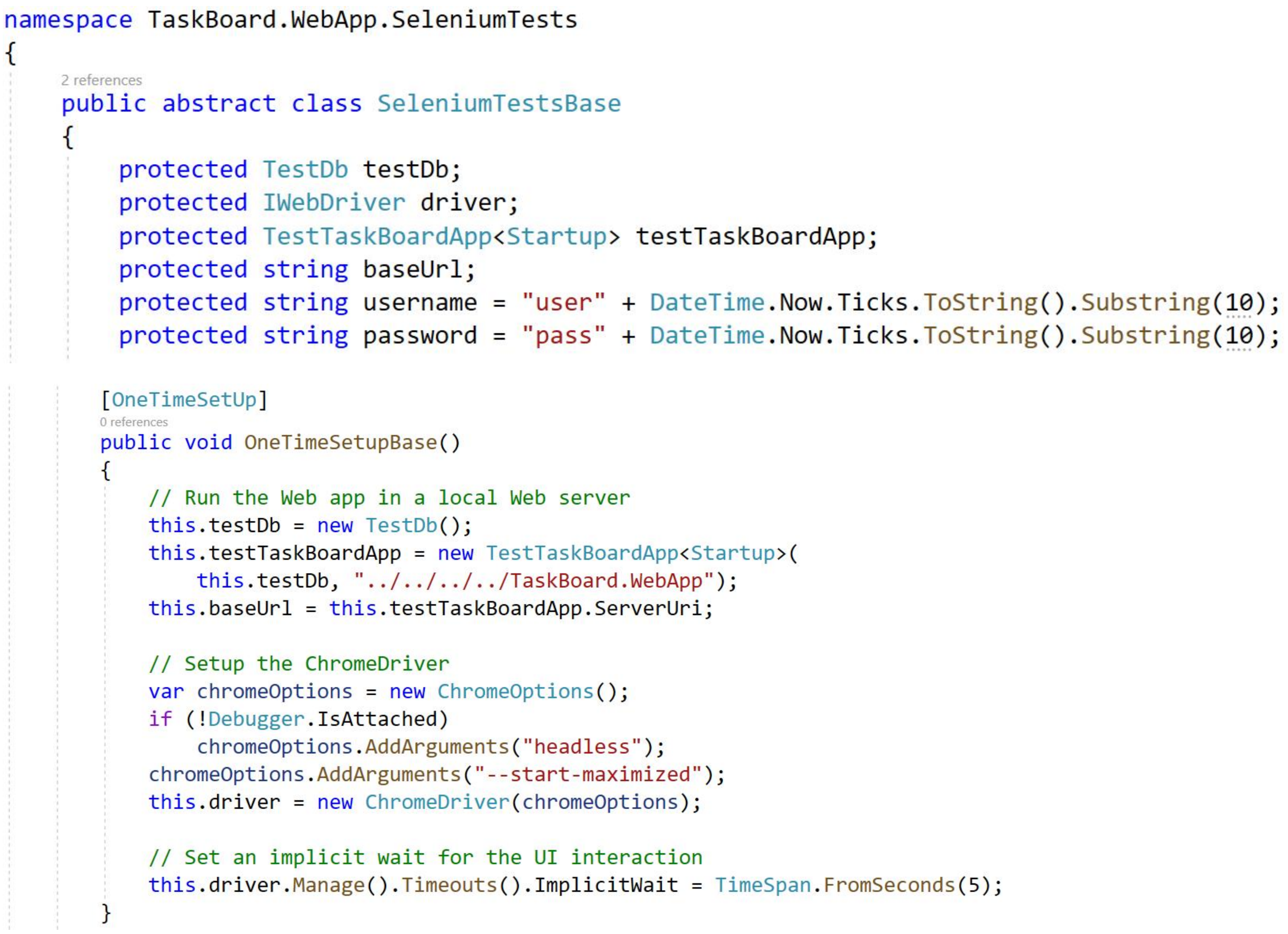
- Navigate to the "`/Tasks/Create`" page, fill in **valid data**, click on the [**Create**] button, and **assert the new task has been created**. Do the same with **invalid data** and **assert** the correct **error messages** show.
- Try to delete a task: Call the "**`CreateTask(out string taskTitle, out string taskDescription)`**" to create a task, **assert** the **new task is created** and that the **user is redirected**. Click on the **delete button**, assert a **redirect to the deletion confirmation page happens**, click on the **delete button** again and finally **assert the task is gone**.
- Try to edit a task: Call the "**`CreateTask(out string taskTitle, out string taskDescription)`**" to create a task, **assert** the **new task is created**. Click on the **edit button**, assert a **redirect to the edit page happens**, change the **tasks title**, click on the **edit button** again and finally **assert the task contains our new title**.
- Try to **click on the login / register buttons** on both the **home page and in the navigation** and **assert the user is redirected.**
- Try to do a **full register / login / logout** sequence. Find the needed buttons, **fill in valid data** and assert the correct actions are happening. (i.e., redirections)

You are free to use a **testing framework** of choice (e. g. NUnit or JUnit), but your primary Web UI automation tool should be **Selenium**. You are free to use **external libraries and tools**.

Hints and Guidelines:

`SeleniumTestsBase.cs` class:

This class is responsible for **setting up the test database** as well as creating a new **TaskBoard** app alongside a **new chrome driver**. The **`HttpClient`** is used for creating an **authentication** with a newly created **`JWT token`**:



```
namespace TaskBoard.WebApp.SeleniumTests
{
    2 references
    public abstract class SeleniumTestsBase
    {
        protected TestDb testDb;
        protected IWebDriver driver;
        protected TestTaskBoardApp<Startup> testTaskBoardApp;
        protected string baseUrl;
        protected string username = "user" + DateTime.Now.Ticks.ToString().Substring(10);
        protected string password = "pass" + DateTime.Now.Ticks.ToString().Substring(10);

        [OneTimeSetUp]
        0 references
        public void OneTimeSetupBase()
        {
            // Run the Web app in a local Web server
            this.testDb = new TestDb();
            this.testTaskBoardApp = new TestTaskBoardApp<Startup>(
                this.testDb, "../../../../TaskBoard.WebApp");
            this.baseUrl = this.testTaskBoardApp.ServerUri;

            // Setup the ChromeDriver
            var chromeOptions = new ChromeOptions();
            if (!Debugger.IsAttached)
                chromeOptions.AddArguments("headless");
            chromeOptions.AddArguments("--start-maximized");
            this.driver = new ChromeDriver(chromeOptions);

            // Set an implicit wait for the UI interaction
            this.driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(5);
        }
        [OneTimeTearDown]
        0 references
        public void OneTimeTearDownBase()
        {
            // Stop and dispose the Selenium driver
            this.driver.Quit();

            // Stop and dispose the local Web server
            this.testTaskBoardApp.Dispose();
        }
    }
}
```

Inside our test class we have **2 helper methods** – the "**RegisterUserForTesting()**" method and the "**`CreateTask()`**" method.

The first one **navigates to the register page**, **finds all the required fields**, **fills them in with valid data and click on the [Register] button**. This will be used for the tests in which we need a registered user. Code it yourself.

The second one **navigates to the create task page**, **finds all the required fields**, **fills them in with valid data and click on the [Create] button**. This will be used for the tests in which we need a new task. Code it yourself.

Test_CreateTask_ValidData()

- Navigate to the "**/Tasks/Create**" page.
- Click on the [**Create**] button.
- **Assert** the user is redirected to "**`/Boards`**".
- **Assert** the new task in on the page and existing.

Here is how the "**`Test_CreateTask_ValidData()`**" test could look like:

```
[Test]
0 references
public void Test_CreateTask_ValidData()
{
    this.driver.Navigate().GoToUrl(this.baseUrl + "/Tasks/Create");
    Assert.That(this.driver.Title.Contains("Create Task"));

    var taskTitle = "Test Task" + DateTime.Now.Ticks;
    var titleField = this.driver.FindElement(By.Id("Title"));
    titleField.Clear();
    titleField.SendKeys(taskTitle);

    var taskDescription = "Task to test if the tasks creation is successful";
    var descriptionField = this.driver.FindElement(By.Id("Description"));
    descriptionField.Clear();
    descriptionField.SendKeys(taskDescription);

    var boardField = this.driver.FindElement(By.Id("BoardId"));
    boardField.Click();

    var inProgressOption = this.driver.FindElement(By.XPath(@"//*[@id='BoardId']/option[2]"));
    inProgressOption.Click();
    // Locate the "Create" button
    var createButton = this.driver
        .FindElement(By.XPath("//input[contains(@value,'Create')]"));

    // Click on the button
    createButton.Click();

    // Assert user is redirected
    Assert.AreEqual(this.baseUrl + "/Boards", this.driver.Url);
    Assert.That(this.driver.Title.Contains("Task Board"));
    Assert.That(this.driver.PageSource.Contains("Task Board"));
```

```
    Assert.That(this.driver.PageSource.Contains(taskTitle));
    Assert.That(this.driver.PageSource.Contains(taskDescription));
    Assert.That(this.driver.PageSource.Contains(this.username));

    var taskTableBodyElement =
        this.driver.FindElement(By.XPath($"//tbody[contains(.,'{taskTitle}')]"));
    var taskTableDataRows = taskTableBodyElement.FindElements(By.TagName("td"));

    Assert.AreEqual(taskTableDataRows[0].Text, taskTitle);
    Assert.AreEqual(taskTableDataRows[1].Text, taskDescription);
    Assert.That(taskTableDataRows[2].Text.Contains("View"));
    Assert.That(taskTableDataRows[2].Text.Contains("Edit"));
    Assert.That(taskTableDataRows[2].Text.Contains("Delete"));
}
```

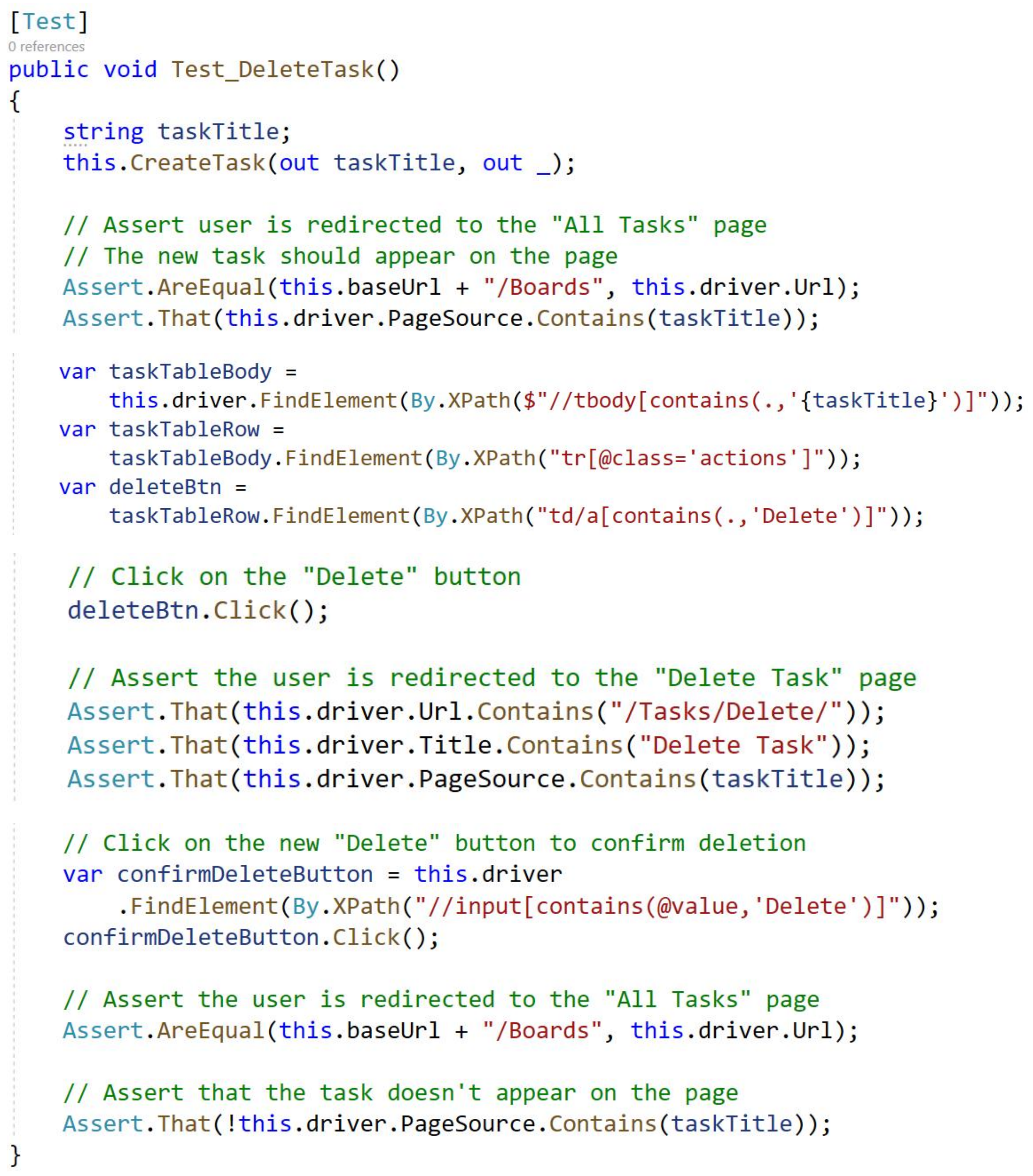
Do the same for the "`Test_CreateTask_InvalidId()`" but this time you need to **input invalid data**, **assert the user stays on the same page and that an error message appears**. Here is how it could look like:

```
    // Assert the user stays on the same page
    Assert.AreEqual(this.baseUrl + "/Tasks/Create", this.driver.Url);

    // Assert that an error message appears on the page
    var errorSpan = this.driver.FindElement(By.Id("Title-error"));
    Assert.AreEqual(errorSpan.Text, "The Title field is required.");
```

Test_DeleteTask()

- Call the "`CreateTask()`" method.
- **Find and click** on the new tasks **[Delete] button.**
- **Assert** the user is **redirected** to the "**Delete Task**" page.
- **Click** on the new **[Delete] button** to **confirm deletion.**
- **Assert** the user is **redirected** to the "**`All Tasks`**" page.
- **Assert** that the task **doesn't appear** on the page.

Here is how the "`Test_DeleteTask()`" test could look like:



```
[Test]
0 references
public void Test_DeleteTask()
{
    string taskTitle;
    this.CreateTask(out taskTitle, out _);

    // Assert user is redirected to the "All Tasks" page
    // The new task should appear on the page
    Assert.AreEqual(this.baseUrl + "/Boards", this.driver.Url);
    Assert.That(this.driver.PageSource.Contains(taskTitle));

    var taskTableBody =
        this.driver.FindElement(By.XPath($"//tbody[contains(.,'{taskTitle}')]"));
    var taskTableRow =
        taskTableBody.FindElement(By.XPath("tr[@class='actions']"));
    var deleteBtn =
        taskTableRow.FindElement(By.XPath("td/a[contains(.,'Delete')]"));

    // Click on the "Delete" button
    deleteBtn.Click();

    // Assert the user is redirected to the "Delete Task" page
    Assert.That(this.driver.Url.Contains("/Tasks/Delete/"));
    Assert.That(this.driver.Title.Contains("Delete Task"));
    Assert.That(this.driver.PageSource.Contains(taskTitle));

    // Click on the new "Delete" button to confirm deletion
    var confirmDeleteButton = this.driver
        .FindElement(By.XPath("//input[contains(@value,'Delete')]"));
    confirmDeleteButton.Click();

    // Assert the user is redirected to the "All Tasks" page
    Assert.AreEqual(this.baseUrl + "/Boards", this.driver.Url);

    // Assert that the task doesn't appear on the page
    Assert.That(!this.driver.PageSource.Contains(taskTitle));
}
```

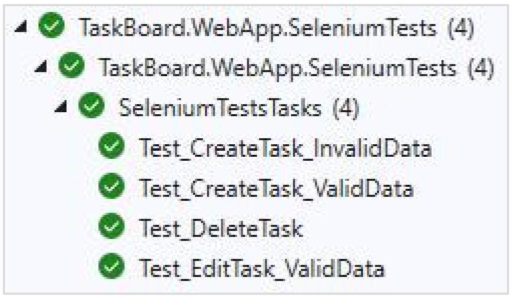
Test_EditTask_ValidData()

- Call the "`CreateTask()`" method.
- **Find and click** on the new tasks **[Edit] button.**
- **Assert** the user is **redirected** to the "**Edit Task**" page.
- **Change the title** of the task.
- **Click** on the new **[Edit] button** to **confirm editing.**
- **Assert** the user is **redirected** to the "**`All Tasks`**" page.
- **Assert** that the task **contains the new title** on the page.

Here is how the "**`Test_EditTask_ValidData()`**" test could look like:

```
[Test]
0 references
public void Test_EditTask_ValidData()
{
    string taskTitle;
    this.CreateTask(out taskTitle, out _);

    var taskTableBody =
        this.driver.FindElement(By.XPath($"//tbody[contains(.,'{taskTitle}')]"));
    var taskTableRow =
        taskTableBody.FindElement(By.XPath("tr[@class='actions']"));
    var editBtn =
        taskTableRow.FindElement(By.XPath("td/a[contains(.,'Edit')]"));

    editBtn.Click();

    // Assert the user is redirected to the "Edit Task" page
    Assert.That(this.driver.Url.Contains("/Tasks/Edit/"));
    Assert.That(this.driver.Title.Contains("Edit Task"));

    // Change the title of the task
    var editTitleField = this.driver.FindElement(By.Id("Title"));
    var changedTitle = "Changed Test Task" + DateTime.Now.Ticks;
    editTitleField.Clear();
    editTitleField.SendKeys(changedTitle);

    var confirmEditButton = this.driver
        .FindElement(By.XPath("//input[contains(@value,'Edit')]"));

    // Click on the new "Edit" button to confirm edition
    confirmEditButton.Click();

    // Assert the user is redirected to the "All Tasks" page
    Assert.AreEqual(this.baseUrl + "/Boards", this.driver.Url);

    // Assert that the page contains the new task title and not the old one
    Assert.That(this.driver.PageSource.Contains(changedTitle));
    Assert.That(!this.driver.PageSource.Contains(taskTitle));
}
```

Here are how the tests should look like when run:



```
◢ ✓ TaskBoard.WebApp.SeleniumTests (4)
  ◢ ✓ TaskBoard.WebApp.SeleniumTests (4)
    ◢ ✓ SeleniumTestsTasks (4)
        ✓ Test_CreateTask_InvalidData
        ✓ Test_CreateTask_ValidData
        ✓ Test_DeleteTask
        ✓ Test_EditTask_ValidData
```