Goal: eu quero criar um site q funcione como uma barra de tarefas em c# que possa adicionar, editar tarefas diarias ou unicas e q possa escolher pela importancia da tarefa o quanto serei notificado

---

Adding Task: Create a new website using C# that functions as a taskbar

---

Adding Task: Implement a feature to add daily or one-time tasks

---

Adding Task: Add an option to prioritize tasks and set notification preferences

---

Adding Task: Test the website to ensure it is working properly

---

Adding Task: Create a new website using C# that functions as a taskbar

---

Writing code...

Executing: "Create a new website using C# that functions as a taskbar": # Taskbar Website in C#

To create a taskbar website in C#, we can use ASP.NET MVC framework. Let's proceed step by step to create this website.

Step 1: Create a new ASP.NET MVC Project

1. Open Visual Studio.
2. Select "Create a new project".
3. Search for "ASP.NET Web Application (.NET Framework)" template and select it.
4. Click "Next".
5. Enter a name for your project, choose a location, and click "Create".
6. In the "Create a new ASP.NET Web Application" dialog, select "MVC" as the project template.
7. Click "Create".

Step 2: Design Taskbar Layout

1. Open the `Views` folder in the Solution Explorer.
2. Right-click on the `Shared` folder and select "Add" -> "View".
3. Enter "Taskbar" as the name and select "Empty (without model)" as the template.
4. Click "Add".

Open the `Taskbar.cshtml` file and add the following code to create the taskbar layout:

```html
<div id="taskbar">
<ul id="taskbar-items">
  <!-- Taskbar items will be dynamically added here -->
</ul>
</div>
```

Step 3: Create Models

1. Right-click on the `Models` folder in the Solution Explorer and select "Add" -> "Class".
2. Enter "Task" as the name and click "Add".

Open the `Task.cs` file and add the following code to define the `Task` model:

```csharp
public class Task
{
public int Id { get; set; }
public string Title { get; set; }
public string Description { get; set; }
public bool IsImportant { get; set; }
  public bool IsCompleted { get; set; }
}
```

Step 4: Create Controllers

1. Right-click on the `Controllers` folder in the Solution Explorer and select "Add" -> "Controller".
2. Enter "TaskController" as the name and click "Add".

Open the `TaskController.cs` file and add the following code:

```csharp
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using YourProjectName.Models;

namespace YourProjectName.Controllers
{
public class TaskController : Controller
    {
    private static List<Task
```

Adding Task: Create a feature to edit tasks and update the taskbar layout to display the task details.

Adding Task: Implement a feature to add daily or one-time tasks

---

Writing code...

Executing: "Implement a feature to add daily or one-time tasks": To implement a feature to add daily or one-time tasks, we can start by creating a class to represent a task. Each task will have properties such as its name, description, date, and importance.

```csharp
public class Task
```

```
{
    public string Name { get; set; }
 public string Description { get; set; }
public DateTime Date { get; set; }
    public int Importance { get; set; }
}
```

Next, we can create a class to manage the tasks. This class will have methods to add daily or one-time tasks, and a list to store the tasks.

```csharp
public class TaskManager
{
private List<Task> tasks;

 public TaskManager()
{
      tasks = new List<Task>();
}

   public void AddDailyTask(string name, string description, int importance)
 {
 Task task = new Task
{
    Name = name,
         Description = description,
        Date = DateTime.Today,
   Importance = importance
     };

       tasks.Add(task);
}

public void AddOneTimeTask(string name, string description, DateTime date, int importance)
{
Task task = new Task
{
   Name = name,
        Description = description,
       Date = date,
            Importance = importance
    };

      tasks.Add(task);
}
}
```

To use the `TaskManager` class, we can create an instance and call the appropriate methods.

```csharp
TaskManager taskManager = new TaskManager();

taskManager.AddDailyTask("Do laundry", "Wash and fold clothes", 3);
taskManager.AddOneTimeTask("Buy groceries", "Visit the supermarket", new DateTime(2022, 1, 15), 4);
```

With this implementation, we can add daily or one-time tasks with their respective properties. We can later extend this functionality to include features like editing tasks or sorting them by importance.

Adding Task: Create a feature to delete tasks from the task manager.

Adding Task: Add an option to prioritize tasks and set notification preferences

---

Writing code...

Executing: "Add an option to prioritize tasks and set notification preferences": To add an option to prioritize tasks and set notification preferences, we can modify the existing code of the website. Here are the steps to achieve that:

1. Modify the data structure:
  - Add a new field called "priority" to the existing task data structure.
- The priority field can be an integer value, where a higher value indicates a higher priority.

2. Modify the user interface:
- Add a dropdown or a set of radio buttons to allow the user to select the priority of the task.
- Add checkboxes or toggle buttons to allow the user to choose the notification preferences.

3. Update the task creation/editing functionality:
- When a new task is created or an existing task is edited, capture the priority value selected by the user and store it in the task data structure.

4. Update the notification functionality:
 - When sending notifications for tasks, check the priority of the task and the notification preferences chosen by the user.
- If the priority of the task is high and the user has chosen to be notified for high priority tasks, send the notification.

Here's an example implementation in C#:

```csharp
// Step 1: Modify the data structure
public class Task
{
public string Name { get; set; }
   public string Description { get; set; }
public int Priority { get; set; } // new field for priority
}

// Step 2: Modify the user interface
```

```
// Add priority dropdown to task creation/editing form
// Add checkboxes for notification preferences

// Step 3: Update the task creation/editing functionality
public void CreateOrUpdateTask(string name, string description, int priority)
{
  // create or update task with the given name, description, and priority
}

// Step 4: Update the notification functionality
public void SendNotification(Task task)
{
  if (task.Priority > 5 && userNotificationPreferences.HighPriority)
   {
   // send notification for high priority task
 }
}
```

Note: This is a simplified example, and the actual implementation may vary based on the specific requirements and architecture of the website. Additionally, this example assumes that there is already functionality in place for creating tasks and sending notifications.

Adding Task: Update the taskbar layout to display the task priority and notification preferences.

Adding Task: Test the website to ensure it is working properly

---

Searching the web for "how to test a website"...

Executing: "Test the website to ensure it is working properly": To test a website and ensure that it is functioning properly, several steps should be taken. These steps include web testing, mobile-friendly testing, website QA test checklist, and web application testing.

Web testing is a software testing practice that is used to test websites or web applications to identify potential bugs before making them live[1](https://www.softwaretestinghelp.com/web-application-testing/). It is an essential step to ensure the smooth functioning of the website.

Mobile-friendly testing is another important aspect of website testing. It involves verifying the compatibility of the website with smartphones and tablets. It also focuses on ensuring that the site navigation is as simple as possible and optimizing the loading time[2](https://geteasyqa.com/qa/test-website/). This ensures that the website is accessible and user-friendly across different devices.

The website QA test checklist provides a comprehensive guide for testing websites. It includes various key aspects such as cross-browser compatibility, responsiveness, functionality testing, and checking for broken links or errors[3](https://www.browserstack.com/guide/how-to-perform-website-qa-testing). Following this checklist helps to identify any potential issues and ensures that the website is working properly.

Web application testing, also known as website testing, is the process of checking a web application or website for potential bugs before it is made live and accessible to users[4](https://www.guru99.com/web-application-testing.html).

This type of testing is crucial to identify and fix any issues that might affect the functionality of the website.

Overall, performing these tests and following the best practices ensures that the website is thoroughly tested and functions properly, providing a seamless user experience.

Adding Task: Update the task manager to add a feature that allows users to set notification preferences for each task based on their priority.

Adding Task: Create a feature to edit tasks and update the taskbar layout to display the task details.

Writing code...

Executing: "Create a feature to edit tasks and update the taskbar layout to display the task details.": # Task Manager Application

Task Class
We start by defining a `Task` class to represent a single task. This class will have properties to store information about the task such as its name, description, importance, and status.

```csharp
public class Task
{
public string Name { get; set; }
public string Description { get; set; }
public int Importance { get; set; } // 1 to 5 scale
public bool IsCompleted { get; set; }

public Task(string name, string description, int importance)
    {
    Name = name;
     Description = description;
Importance = importance;
   IsCompleted = false;
}
}
```

Task Manager Class
Next, we create a `TaskManager` class to handle the management of tasks. This class will have methods to add, edit, and display tasks.

```csharp
public class TaskManager
{
private List<Task> tasks;

public TaskManager()
  {
  tasks = new List<Task>();
```

```
  }

public void AddTask(string name, string description, int importance)
  {
  Task task = new Task(name, description, importance);
   tasks.Add(task);
    }

 public void EditTask(int index, string name, string description, int importance)
{
     if (index >= 0 && index < tasks.Count)
  {
      Task task = tasks[index];
    task.Name = name;
    task.Description = description;
       task.Importance = importance;
     }
  }

public void DisplayTasks()
 {
 foreach (Task task in tasks)
       {
           Console.WriteLine($"Name: {task.Name}");
    Console.WriteLine($"Description: {task.Description}");
           Console.WriteLine($"Importance: {task.Importance}");
      Console.WriteLine($"Status: {(task.IsCompleted ? "Completed" : "Not Completed")}");
   Console.WriteLine("----------");
    }
}
}
```

Main Program
In the main program, we can create an instance of the `TaskManager` class and use its methods to add, edit, and display tasks.

```csharp
class Program
{
   static void Main(string[] args)
{
TaskManager taskManager = new TaskManager();

   taskManager.AddTask("Task 1", "Description 1", 3);
  taskManager.AddTask("Task 2", "Description 2", 2);

  taskManager.DisplayTasks();
```

Adding Task: Create a feature to edit tasks and update the taskbar layout to display the task details.

Adding Task: Create a feature to delete tasks from the task manager.

Max loops of 5 exceeded, shutting down.
