**­Westminster international university in Tashkent.**

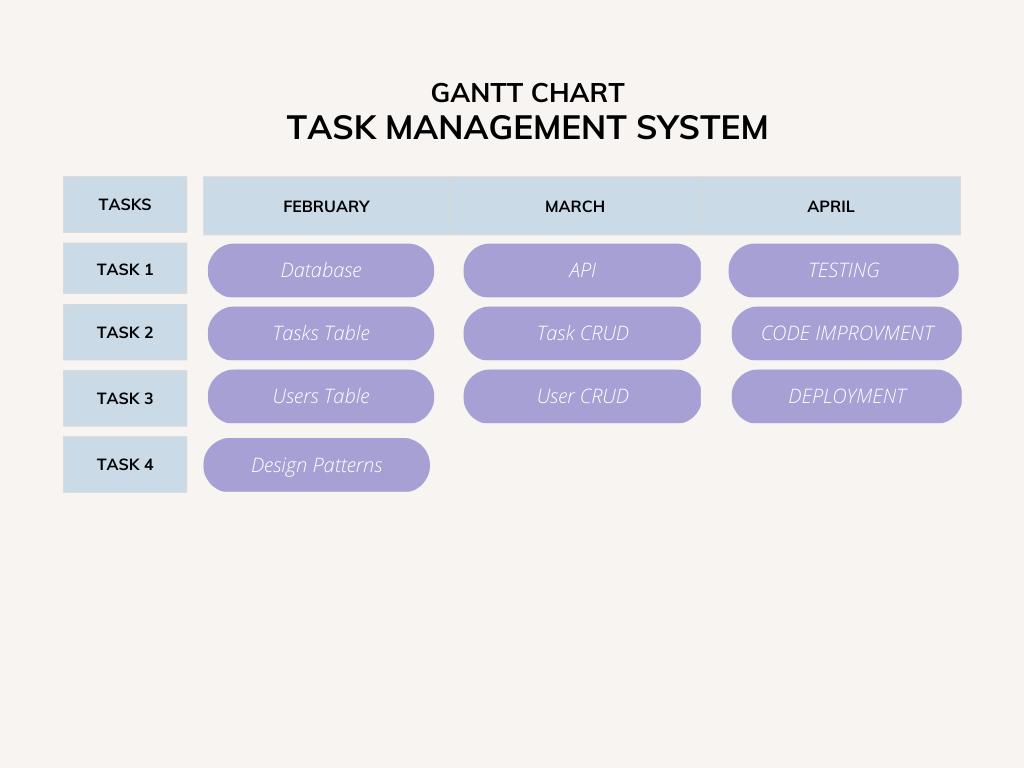
**Development Plan**

For my coursework from Web Application Development module, I am going to create a Task Management System which will mainly be helping people to effectively manage their time and increase productivity. These are my timelines for this project to get done on time.

**MoSCoW method:**

|  |  |  |  |
| --- | --- | --- | --- |
| MUST HAVE | SHOULD HAVE | COULD HAVE | WON’T HAVE |
| ASP.NET CORE.  Database  Entity Framework Core.  CRUD operations. | Assignment to users.  Achieved tasks. | Restore deleted tasks.  Log in system.  Account update. | Two step auth.  Task ownership transfer.  Deadline notifications. |

**Gantt Chart**

****

**SRP, DRY**

**Single Responsibility Principle:**

The SRP states that a class should have only one reason to change. In my task management web application, this means that each component of the application should have a specific responsibility and should not be responsible for multiple tasks. This allows for easier maintenance and testing of the application.

Initially, we might have a single TaskController that handles both creating and updating tasks. However, this violates the SRP because the controller has multiple responsibilities. To apply SRP, we can split the TaskController into two separate controllers: *TaskCreationController* and *TaskUpdateController*. This way, each controller has a single responsibility.

**Before applying SRP**

public class TaskController : Controller

{

public IActionResult Create(TaskRequest request)

{

// code to create a new task

}

public IActionResult Update(TaskRequest request)

{

// code to update a task

}

}

**After applying SRP**

public class TaskCreationController : Controller

{

public IActionResult Create(TaskRequest request)

{

// code to create a new task

}

}

public class TaskUpdateController : Controller

{

public IActionResult Update(TaskRequest request)

{

// code to update a task

}

}

**Don’t Repeat Yourself:**

The DRY principle states that code should not be duplicated within a system. Duplication of code can lead to inconsistencies and can make maintenance more difficult.

Let's consider an example of my task management that has a *Task* model and a *User* model. Initially, we might have duplicate code to get the date of creation in both the Task and User models. To apply DRY, we can define a base model with the date of creation and then inherit from that base model in both the Task and User models.

**Before applying DRY**

public class Task

{

public int Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public DateTime CreatedAt { get; set; }

}

public class User

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public DateTime CreatedAt { get; set; }

}

**After applying DRY**

public class BaseModel

{

public int Id { get; set; }

public DateTime CreatedAt { get; set; }

}

public class Task : BaseModel

{

public string Title { get; set; }

public string Description { get; set; }

}

public class User : BaseModel

{

public string FirstName { get; set; }

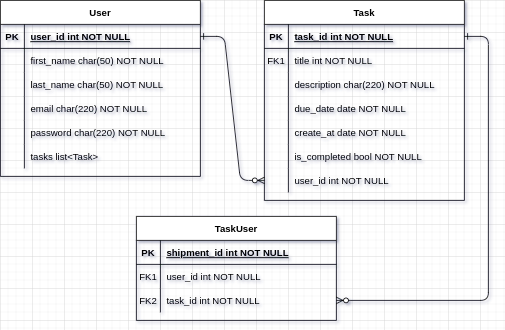
public string LastName { get; set; }

public string Email { get; set; }

}

In this example, I have defined a *BaseModel* class with the common properties that *Task* and *User* share, such as *Id* and *CreatedAt*. Both *Task* and *User*  now inherit from *BaseModel*, which eliminates the duplication of code for those shared properties.

**Diagramm**

****

With TaskServices example

