First Task: Prepares the working environment for different web applications.

**Evidence and Proof Requirements 1,2,3,4,5,6:**

Product Evidence

The student is requested to create an application and present a report on the most suited choice; in terms of cost, availability in the target market, and the application’s needs for the following case study:

**Objective:**

**Develop an ASP.NET Core MVC web application for a Task Management System where a Team Leader can manage multiple projects and tasks with their assigned team members. The dashboard should display a list of all projects with CRUD capabilities. Each project's details should include its tasks and the assigned team members, with functionality to view team member profiles and update task statuses.**

**Requirements:**

1. **Project Setup**

* Create an ASP.NET Core MVC project using Visual Studio.
  + Set up Entity Framework Core with a code-first approach, connecting to a local database.

1. **Entities and Models**
   * Define three main entities: **Task**, **Project**, and **TeamMember**.
     + **Task**:
       - Id (Primary Key)
       - Title (string, max length 100)
       - Description (string, max length 500)
       - Status (Pending, In Progress, Completed)
       - Priority (Low, Medium, High)
       - Deadline (DateTime)
       - ProjectId (Foreign Key to Project)
       - TeamMemberId (Foreign Key to TeamMember)
     + **Project**:
       - Id (Primary Key)
       - Name (string, max length 100)
       - Description (string, max length 500)
       - StartDate (DateTime)
       - EndDate (DateTime)
     + **TeamMember**:
       - Id (Primary Key)
       - Name (string, max length 100)
       - Email (string, max length 100, unique)
       - Role (string, max length 50)
   * Define relationships:
     + A **Project** can have multiple **Tasks** (one-to-many).
     + A **TeamMember** can be assigned to multiple **Tasks** (one-to-many).
2. **Database Context**
   * Create a **TaskManagementContext** class inheriting from **DbContext**.
   * Configure the **DbSet<Task>,** **DbSet<Project>,** and **DbSet<TeamMember>** properties.
   * Use *Fluent API* or *Data Annotations* to configure relationships:
     + One-to-many relationships between **Project** and **Task**, and between **TeamMember** and Task.
     + Set constraints like max lengths and required fields.
3. **Migrations**
   * Use EF Core Migrations to generate and apply the database schema.
4. **Controllers**
   * Create controllers for **Projects, Tasks, and TeamMembers**:
     + Use dependency injection to inject corresponding services.
     + **ProjectsController**:
       - Actions for *Index* (list projects), *Details* (view project details), *Create*, *Edit*, and *Delete*.
       - *Details* action should include:
         * List of tasks in the project.
         * For each task, display the assigned team member.
     + **TasksController**:
       - Actions for *Edit* to update task status.
     + **TeamMembersController:**
       - Actions for *Details* (view team member profile) and *Edit* (update team member details).
5. **Views**
   * Use **Razor views** to display and manage data:
     + **Dashboard** (**Projects Index View**):
       - Display a list of all projects with options to view, create, edit, and delete.
     + **Project Details View:**
       - Display project details and a list of tasks in a table:
         * Columns: Task Title, Status, Priority, Deadline, Assigned Team Member (with a link to their profile).
     + **Task Edit View:**
       - Form to update task status.
     + **Team Member Details View:**
       - Display team member details.
       - Include a list of tasks assigned to the member.
   * Use **ViewModels** to pass data between controllers and views.
   * Use **Bootstrap** for styling.
6. **Validation and Error Handling**
   * Use **data annotations** for validation:
     + Required fields (e.g., Name, Title, Email).
     + Date validations (e.g., Deadline > StartDate).
   * Handle validation errors in the Razor views:
     + Display error messages for invalid inputs.
   * Add global error handling for unexpected issues.