EXQT
Готовые ответы — на EXQT
Доступ к ответам приобретается отдельно.
Найти ответы ↗

Этап 1. C# и ООП-модель

Цель этапа

На этом этапе студент создает основу приложения: понимает предметную область, описывает сущность заявки и реализует ее как класс C#.

Теория

Основы C

C# — объектно-ориентированный язык программирования. В нем программа строится из классов, объектов, методов, свойств и коллекций.

Базовые элементы языка:

Элемент	Для чего нужен	Пример
Переменная	хранит значение	`string title = "Ошибка входа";`
Тип данных	определяет вид значения	`int`, `string`, `bool`, `DateTime`
Условие	выбирает действие	`if (status == "Open")`
Цикл	повторяет действие	`foreach (var item in requests)`
Метод	выполняет операцию	`ChangeStatus()`
Коллекция	хранит набор объектов	`List<Request>`

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗
WhatsApp ↗
MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

ООП

ООП — объектно-ориентированное программирование. Оно помогает описывать предметную область через классы и объекты.

Принцип	Смысл	Пример в проекте
Класс	шаблон объекта	`Request`
Объект	конкретный экземпляр класса	заявка №1
Инкапсуляция	данные и методы собраны вместе	статус меняется через метод
Наследование	класс может расширять другой класс	базовый класс `Entity`
Полиморфизм	один интерфейс, разные реализации	`IRequestRepository`

Модель данных

Модель — это класс, который описывает данные предметной области. Для системы учета заявок главная модель — заявка пользователя.

Заявка должна содержать:

- идентификатор;
- тему;
- описание;
- имя автора;
- дату создания;
- статус;
- приоритет.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Статус заявки

Статус лучше хранить не строкой, а перечислением `enum`. Это уменьшает количество ошибок.

```
public enum RequestStatus
{
    New,
    InProgress,
    Done,
    Cancelled
}
```

Задание

Необходимо создать модель заявки для приложения **«Система учета заявок пользователей»**.

Нужно выполнить:

1. Создать проект C#.
2. Создать папку `Models`.
3. Создать перечисление `RequestStatus`.
4. Создать класс `UserRequest`.
5. Добавить свойства заявки.
6. Добавить метод изменения статуса.
7. Создать несколько тестовых объектов в коде.

Пример кода

Перечисление статусов

```
namespace RequestApp.Models;

public enum RequestStatus
{
    New,
    InProgress,
    Done,
    Cancelled
}
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Класс заявки

```
namespace RequestApp.Models;

public class UserRequest
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Title { get; set; } = string.Empty;
    public string Description { get; set; } = string.Empty;
    public string AuthorName { get; set; } = string.Empty;
    public DateTime CreatedAt { get; set; } = DateTime.Now;
    public RequestStatus Status { get; private set; } = RequestStatus.New;
    public string Priority { get; set; } = "Normal";

    public void ChangeStatus(RequestStatus newStatus)
    {
        Status = newStatus;
    }
}
```

Тестовое создание объекта

```
using RequestApp.Models;

var request = new UserRequest
{
    Title = "Не работает вход",
    Description = "Пользователь не может войти в систему.",
    AuthorName = "Иванов Иван",
    Priority = "High"
};

request.ChangeStatus(RequestStatus.InProgress);

Console.WriteLine($"{request.Title}: {request.Status}");
```

Что добавить в отчет

1. Описание модели заявки.
2. Таблицу свойств класса.
3. Код `RequestStatus`.
4. Код `UserRequest`.
5. Скриншот запуска тестового кода.

Таблица для отчета

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Свойство	Тип	Назначение
`Id`	`Guid`	уникальный идентификатор заявки
`Title`	`string`	тема заявки
`Description`	`string`	подробное описание
`AuthorName`	`string`	автор заявки
`CreatedAt`	`DateTime`	дата создания
`Status`	`RequestStatus`	текущий статус
`Priority`	`string`	приоритет заявки

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Этап 2. Архитектура и паттерны

Цель этапа

На этом этапе студент проектирует структуру приложения: разделяет код на слои и понимает, зачем нужны паттерны `Repository`, `Service Layer`, `MVC/MVVM` и `Singleton`.

Теория

Зачем нужна архитектура

Архитектура помогает не смешивать весь код в одном файле. Если интерфейс, бизнес-логика и работа с файлами находятся вместе, проект быстро становится сложным для изменения.

Приложение удобно делить на слои:

Слой	Назначение	Пример
Presentation	интерфейс пользователя	WinForms, WPF, HTML-страница
Business Logic	правила работы приложения	сервис заявок
Data Access	работа с данными	чтение и запись JSON
Models	сущности предметной области	`UserRequest`

SOLID простыми словами

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Принцип	Смысл
Single Responsibility	класс должен отвечать за одну задачу
Open/Closed	класс можно расширять, не ломая старый код
Liskov Substitution	наследники должны корректно заменять базовый тип
Interface Segregation	интерфейсы не должны быть слишком большими
Dependency Inversion	код зависит от абстракций, а не от конкретных классов

Repository

Repository — паттерн, который скрывает способ хранения данных. Остальной код не должен знать, где лежат заявки: в JSON, базе данных или внешнем API.

В нашем проекте репозиторий будет работать с JSON-файлом.

Service Layer

Service Layer — слой бизнес-логики. Сервис решает, что можно делать с заявками: добавить, изменить, удалить, поменять статус.

Репозиторий только сохраняет и читает данные, а сервис проверяет правила.

MVC и MVVM

Паттерн	Где чаще используется	Идея
MVC	веб-приложения	Model, View, Controller

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Паттерн	Где чаще используется	Идея
MVVM	WPF	Model, View, ViewModel

Главная цель MVC/MVVM — отделить интерфейс от логики.

Singleton

Singleton — паттерн, при котором у класса есть только один экземпляр. Его можно применять осторожно, например для настроек приложения. В ASP.NET Core чаще используется встроенный механизм `Dependency Injection`.

Задание

Необходимо спроектировать архитектуру проекта.

Нужно выполнить:

1. Создать структуру папок.
2. Создать интерфейс репозитория.
3. Создать сервис заявок.
4. Показать, какие классы относятся к каждому слою.
5. Описать, какие паттерны используются в проекте.

Рекомендуемая структура проекта

```
RequestApp/
  Models/
    UserRequest.cs
    RequestStatus.cs
  Repositories/
    IRequestRepository.cs
    JsonRequestRepository.cs
  Services/
    IRequestService.cs
    RequestService.cs
  Controllers/
    RequestsController.cs
  UI/
    ...
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
  Data/
    requests.json
```

Интерфейс репозитория

```
using RequestApp.Models;

namespace RequestApp.Repositories;

public interface IRequestRepository
{
    Task<List<UserRequest>> GetAllAsync();
    Task<UserRequest?> GetByIdAsync(Guid id);
    Task AddAsync(UserRequest request);
    Task UpdateAsync(UserRequest request);
    Task DeleteAsync(Guid id);
}
```

Интерфейс сервиса

```
using RequestApp.Models;

namespace RequestApp.Services;

public interface IRequestService
{
    Task<List<UserRequest>> GetAllAsync();
    Task<UserRequest?> GetByIdAsync(Guid id);
    Task<UserRequest> CreateAsync(UserRequest request);
    Task<bool> UpdateAsync(Guid id, UserRequest request);
    Task<bool> DeleteAsync(Guid id);
    Task<bool> ChangeStatusAsync(Guid id, RequestStatus status);
}
```

Сервис заявок

```
using RequestApp.Models;
using RequestApp.Repositories;

namespace RequestApp.Services;

public class RequestService : IRequestService
{
    private readonly IRequestRepository _repository;

    public RequestService(IRequestRepository repository)
    {
        _repository = repository;
    }
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
    public Task<List<UserRequest>> GetAllAsync()
    {
        return _repository.GetAllAsync();
    }

    public Task<UserRequest?> GetByIdAsync(Guid id)
    {
        return _repository.GetByIdAsync(id);
    }

    public async Task<UserRequest> CreateAsync(UserRequest request)
    {
        if (string.IsNullOrWhiteSpace(request.Title))
            throw new ArgumentException("Тема заявки обязательна.");

        await _repository.AddAsync(request);
        return request;
    }

    public async Task<bool> UpdateAsync(Guid id, UserRequest request)
    {
        var existing = await _repository.GetByIdAsync(id);
        if (existing is null)
            return false;

        request.Id = id;
        await _repository.UpdateAsync(request);
        return true;
    }

    public Task<bool> DeleteAsync(Guid id)
    {
        return DeleteInternalAsync(id);
    }

    public async Task<bool> ChangeStatusAsync(Guid id, RequestStatus status)
    {
        var request = await _repository.GetByIdAsync(id);
        if (request is null)
            return false;

        request.ChangeStatus(status);
        await _repository.UpdateAsync(request);
        return true;
    }

    private async Task<bool> DeleteInternalAsync(Guid id)
    {
        var existing = await _repository.GetByIdAsync(id);
        if (existing is null)
            return false;

        await _repository.DeleteAsync(id);
        return true;
    }
}
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

Что добавить в отчет

1. Схему структуры проекта.
2. Описание слоев приложения.
3. Код интерфейса `IRequestRepository`.
4. Код интерфейса `IRequestService`.
5. Описание паттернов, которые используются в проекте.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Этап 3. JSON-хранилище

Цель этапа

На этом этапе студент реализует хранение заявок в JSON-файле и учится сериализовать и десериализовать данные в C#.

Теория

Что такое JSON

JSON — текстовый формат хранения и передачи данных. Он состоит из объектов, массивов, строк, чисел, логических значений и `null`.

Пример JSON:

```
[
  {
    "id": "0f8fad5b-d9cb-469f-a165-70867728950e",
    "title": "Не работает вход",
    "description": "Пользователь не может войти в систему",
    "authorName": "Иванов Иван",
    "createdAt": "2026-04-29T10:00:00",
    "status": "New",
    "priority": "High"
  }
]
```

Сериализация и десериализация

Процесс	Что делает
Сериализация	превращает объект C# в JSON
Десериализация	превращает JSON в объект C#

В C# можно использовать:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- `System.Text.Json`;
- `Newtonsoft.Json`.

Для практики рекомендуется `System.Text.Json`, потому что он встроен в .NET.

async/await

Работа с файлами и API может выполняться асинхронно. Это позволяет приложению не зависать во время чтения или записи данных.

Ключевые слова:

- `async` — метод выполняется асинхронно;
- `await` — ожидание результата операции;
- `Task` — объект асинхронной операции.

Задание

Необходимо создать репозиторий, который хранит заявки в JSON-файле.

Нужно выполнить:

1. Создать папку `Data`.
2. Создать файл `requests.json`.
3. Создать класс `JsonRequestRepository`.
4. Реализовать чтение всех заявок.
5. Реализовать добавление заявки.
6. Реализовать редактирование заявки.
7. Реализовать удаление заявки.
8. Проверить, что данные сохраняются после перезапуска приложения.

Пример JSON-файла

```
[]
```

Файл можно начать с пустого массива. После добавления заявок приложение будет записывать туда объекты.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Репозиторий для JSON

```
using System.Text.Json;
using System.Text.Json.Serialization;
using RequestApp.Models;

namespace RequestApp.Repositories;

public class JsonRequestRepository : IRequestRepository
{
    private readonly string _filePath;
    private readonly JsonSerializerOptions _options = new()
    {
        WriteIndented = true,
        Converters = { new JsonStringEnumConverter() }
    };

    public JsonRequestRepository(string filePath)
    {
        _filePath = filePath;
    }

    public async Task<List<UserRequest>> GetAllAsync()
    {
        if (!File.Exists(_filePath))
            return new List<UserRequest>();

        var json = await File.ReadAllTextAsync(_filePath);

        if (string.IsNullOrWhiteSpace(json))
            return new List<UserRequest>();

        return JsonSerializer.Deserialize<List<UserRequest>>(json, _options)
               ?? new List<UserRequest>();
    }

    public async Task<UserRequest?> GetByIdAsync(Guid id)
    {
        var requests = await GetAllAsync();
        return requests.FirstOrDefault(x => x.Id == id);
    }

    public async Task AddAsync(UserRequest request)
    {
        var requests = await GetAllAsync();
        requests.Add(request);
        await SaveAllAsync(requests);
    }

    public async Task UpdateAsync(UserRequest request)
    {
        var requests = await GetAllAsync();
        var index = requests.FindIndex(x => x.Id == request.Id);

        if (index >= 0)
        {
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
            requests[index] = request;
            await SaveAllAsync(requests);
        }
    }

    public async Task DeleteAsync(Guid id)
    {
        var requests = await GetAllAsync();
        requests.RemoveAll(x => x.Id == id);
        await SaveAllAsync(requests);
    }

    private async Task SaveAllAsync(List<UserRequest> requests)
    {
        var directory = Path.GetDirectoryName(_filePath);

        if (!string.IsNullOrEmpty(directory))
            Directory.CreateDirectory(directory);

        var json = JsonSerializer.Serialize(requests, _options);
        await File.WriteAllTextAsync(_filePath, json);
    }
}
```

Обработка исключений

При работе с файлами могут быть ошибки: файл занят, нет прав доступа, неверный JSON. Поэтому в реальном приложении важно использовать `try-catch`.

Пример:

```
try
{
    var requests = await repository.GetAllAsync();
}
catch (IOException ex)
{
    Console.WriteLine($"Ошибка чтения файла: {ex.Message}");
}
catch (JsonException ex)
{
    Console.WriteLine($"Ошибка формата JSON: {ex.Message}");
}
```

Что добавить в отчет

1. Описание структуры JSON.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

2. Код `JsonRequestRepository`.
3. Пример файла `requests.json`.
4. Скриншот JSON-файла после добавления заявок.
5. Краткое описание сериализации и десериализации.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Этап 4. REST API

Цель этапа

На этом этапе студент создает REST API, через которое интерфейс или внешний клиент сможет получать, добавлять, изменять и удалять заявки.

Теория

Что такое REST API

REST API — это способ обмена данными между клиентом и сервером через HTTP-запросы.

Клиент отправляет запрос, сервер выполняет действие и возвращает ответ.

Пример:

```
GET /api/requests
```

Означает: получить список заявок.

HTTP-методы

Метод	Назначение	Пример
`GET`	получить данные	получить список заявок
`POST`	создать данные	добавить новую заявку
`PUT`	полностью обновить данные	изменить заявку
`DELETE`	удалить данные	удалить заявку

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Статус-коды

Код	Значение
`200 OK`	запрос выполнен успешно
`201 Created`	объект создан
`400 Bad Request`	ошибка в данных запроса
`404 Not Found`	объект не найден
`500 Internal Server Error`	ошибка сервера

JSON в REST API

REST API часто принимает и возвращает данные в JSON.

Пример тела запроса `POST`:

```
{
  "title": "Ошибка входа",
  "description": "Пользователь не может войти в систему",
  "authorName": "Иванов Иван",
  "priority": "High"
}
```

Dependency Injection

Dependency Injection — механизм, при котором зависимости передаются в класс извне.

В ASP.NET Core это используется так:

```
builder.Services.AddSingleton<IRequestRepository>(repository);
builder.Services.AddScoped<IRequestService, RequestService>();
```

Контроллер получает сервис через конструктор и не создает его вручную.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Задание

Необходимо создать REST API для работы с заявками.

Нужно реализовать:

1. `GET /api/requests` — получить все заявки.
2. `GET /api/requests/{id}` — получить одну заявку.
3. `POST /api/requests` — добавить заявку.
4. `PUT /api/requests/{id}` — обновить заявку.
5. `PUT /api/requests/{id}/status` — изменить статус заявки.
6. `DELETE /api/requests/{id}` — удалить заявку.

Создание Web API проекта

```
dotnet new webapi -n RequestApi
cd RequestApi
```

Настройка `Program.cs`

```
using RequestApp.Repositories;
using RequestApp.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var dataPath = Path.Combine(AppContext.BaseDirectory, "Data",
"requests.json");

builder.Services.AddSingleton<IRequestRepository>(
    new JsonRequestRepository(dataPath));
builder.Services.AddScoped<IRequestService, RequestService>();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
app.MapControllers();
app.Run();
```

Контроллер заявок

```
using Microsoft.AspNetCore.Mvc;
using RequestApp.Models;
using RequestApp.Services;

namespace RequestApp.Controllers;

[ApiController]
[Route("api/requests")]
public class RequestsController : ControllerBase
{
    private readonly IRequestService _service;

    public RequestsController(IRequestService service)
    {
        _service = service;
    }

    [HttpGet]
    public async Task<ActionResult<List<UserRequest>>> GetAll()
    {
        return Ok(await _service.GetAllAsync());
    }

    [HttpGet("{id:guid}")]
    public async Task<ActionResult<UserRequest>> GetById(Guid id)
    {
        var request = await _service.GetByIdAsync(id);

        if (request is null)
            return NotFound();

        return Ok(request);
    }

    [HttpPost]
    public async Task<ActionResult<UserRequest>> Create(UserRequest request)
    {
        var created = await _service.CreateAsync(request);
        return CreatedAtAction(nameof(GetById), new { id = created.Id },
created);
    }

    [HttpPut("{id:guid}")]
    public async Task<IActionResult> Update(Guid id, UserRequest request)
    {
        var updated = await _service.UpdateAsync(id, request);

        if (!updated)
            return NotFound();
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
        return NoContent();
    }

    [HttpPut("{id:guid}/status")]
    public async Task<IActionResult> ChangeStatus(Guid id, RequestStatus
status)
    {
        var changed = await _service.ChangeStatusAsync(id, status);

        if (!changed)
            return NotFound();

        return NoContent();
    }

    [HttpDelete("{id:guid}")]
    public async Task<IActionResult> Delete(Guid id)
    {
        var deleted = await _service.DeleteAsync(id);

        if (!deleted)
            return NotFound();

        return NoContent();
    }
}
```

Проверка API

Проверить API можно через:

- Swagger;
- Postman;
- Insomnia;
- `curl`;
- браузер для `GET`-запросов.

Пример `curl`:

```
curl http://localhost:5000/api/requests
```

Пример `POST`:

```
curl -X POST http://localhost:5000/api/requests \
  -H "Content-Type: application/json" \
  -d '{"title":"Ошибка входа","description":"Не работает
пароль","authorName":"Иванов Иван","priority":"High"}'
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Что добавить в отчет

1. Описание REST API.
2. Таблицу маршрутов API.
3. Код `RequestsController`.
4. Скриншоты Swagger или Postman.
5. Пример JSON-запроса и JSON-ответа.

Таблица маршрутов

Метод	URL	Назначение
`GET`	`/api/requests`	получить все заявки
`GET`	`/api/requests/{id}`	получить заявку по ID
`POST`	`/api/requests`	создать заявку
`PUT`	`/api/requests/{id}`	обновить заявку
`PUT`	`/api/requests/{id}/status`	изменить статус
`DELETE`	`/api/requests/{id}`	удалить заявку

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Этап 5. GUI или веб-интерфейс

Цель этапа

На этом этапе студент создает интерфейс, через который пользователь сможет работать с заявками: просматривать, добавлять, редактировать, удалять и менять статус.

Теория

GUI и веб-интерфейс

Интерфейс — это слой `Presentation`. Он не должен напрямую работать с JSON-файлом. Правильная схема:

```
Интерфейс → REST API → сервис → репозиторий → JSON-файл
```

Такой подход позволяет заменить интерфейс, не переписывая бизнес-логику и хранение данных.

Вариант 1: Windows Forms

Windows Forms подходит для простого desktop-приложения.

Нужно знать:

- формы;
- кнопки;
- поля ввода;
- таблицы `DataGridView`;
- обработчики событий;
- `HttpClient` для обращения к API.

Вариант 2: WPF

WPF подходит для более структурированного desktop-приложения.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Важно понимать:

- XAML;
- привязку данных;
- `ObservableCollection`;
- MVVM;
- команды;
- разделение `View` и `ViewModel`.

Вариант 3: веб-интерфейс

Веб-интерфейс можно сделать через:

- Razor Pages;
- MVC;
- Blazor;
- простую HTML-страницу с JavaScript `fetch`.

Для учебной практики самый простой вариант — HTML + `fetch`.

Задание

Необходимо создать один интерфейс на выбор:

1. Windows Forms.
2. WPF.
3. Веб-страница.

Интерфейс должен позволять:

- вывести список заявок;
- добавить заявку;
- изменить заявку;
- удалить заявку;
- изменить статус заявки;
- получить данные через REST API;
- отправить данные в REST API.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Пример веб-интерфейса

HTML

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <title>Система заявок</title>
</head>
<body>
  <h1>Система учета заявок</h1>

  <input id="title" placeholder="Тема заявки">
  <input id="author" placeholder="Автор">
  <textarea id="description" placeholder="Описание"></textarea>
  <button onclick="createRequest()">Добавить</button>

  <table border="1">
    <thead>
      <tr>
        <th>Тема</th>
        <th>Автор</th>
        <th>Статус</th>
        <th>Действия</th>
      </tr>
    </thead>
    <tbody id="requests"></tbody>
  </table>

  <script src="app.js"></script>
</body>
</html>
```

JavaScript для REST API

```
const apiUrl = "https://localhost:5001/api/requests";

async function loadRequests() {
  const response = await fetch(apiUrl);
  const requests = await response.json();

  const tbody = document.getElementById("requests");
  tbody.innerHTML = "";

  requests.forEach(item => {
    const row = document.createElement("tr");

    row.innerHTML = `
      <td>${item.title}</td>
      <td>${item.authorName}</td>
      <td>${item.status}</td>
      <td>
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
        <button onclick="deleteRequest('${item.id}')">Удалить</button>
      </td>
    `;

    tbody.appendChild(row);
  });
}

async function createRequest() {
  const request = {
    title: document.getElementById("title").value,
    authorName: document.getElementById("author").value,
    description: document.getElementById("description").value,
    priority: "Normal"
  };

  await fetch(apiUrl, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(request)
  });

  await loadRequests();
}

async function deleteRequest(id) {
  await fetch(`${apiUrl}/${id}`, {
    method: "DELETE"
  });

  await loadRequests();
}

loadRequests();
```

Пример Windows Forms через `HttpClient`

```
using System.Net.Http.Json;
using RequestApp.Models;

public partial class MainForm : Form
{
    private readonly HttpClient _httpClient = new()
    {
        BaseAddress = new Uri("https://localhost:5001/")
    };

    public MainForm()
    {
        InitializeComponent();
    }

    private async void MainForm_Load(object sender, EventArgs e)
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
    {
        await LoadRequestsAsync();
    }

    private async Task LoadRequestsAsync()
    {
        var requests = await _httpClient.GetFromJsonAsync<List<UserRequest>>(
            "api/requests");

        dataGridView1.DataSource = requests;
    }

    private async void addButton_Click(object sender, EventArgs e)
    {
        var request = new UserRequest
        {
            Title = titleTextBox.Text,
            Description = descriptionTextBox.Text,
            AuthorName = authorTextBox.Text,
            Priority = "Normal"
        };

        await _httpClient.PostAsJsonAsync("api/requests", request);
        await LoadRequestsAsync();
    }
}
```

Что добавить в отчет

1. Какой интерфейс выбран: WinForms, WPF или веб.
2. Скриншот главного окна или веб-страницы.
3. Описание элементов интерфейса.
4. Код обращения к REST API.
5. Скриншот добавления и отображения заявки.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Этап 6. Итоговая сборка проекта

Цель этапа

На этом этапе студент собирает все части проекта в одно рабочее приложение и проверяет полный сценарий работы системы учета заявок.

Теория

Полный поток данных

В готовом приложении данные проходят через несколько слоев:

```
Пользователь
  → GUI / веб-интерфейс
  → REST API
  → Service Layer
  → Repository
  → JSON-файл
```

Каждый слой выполняет свою задачу:

Слой	Что делает
Интерфейс	показывает данные и принимает ввод
API	принимает HTTP-запросы
Сервис	проверяет правила и вызывает репозиторий
Репозиторий	читает и записывает JSON
JSON-файл	хранит данные

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Что считается готовым проектом

Проект считается готовым, если пользователь может:

- открыть интерфейс;
- увидеть список заявок;
- добавить новую заявку;
- изменить существующую заявку;
- удалить заявку;
- изменить статус;
- закрыть приложение и после запуска снова увидеть сохраненные данные.

Задание

Необходимо собрать и проверить готовое приложение.

Нужно выполнить:

1. Проверить структуру проекта.
2. Запустить REST API.
3. Проверить API через Swagger или Postman.
4. Запустить GUI или веб-интерфейс.
5. Добавить несколько заявок.
6. Изменить статус одной заявки.
7. Удалить одну заявку.
8. Проверить файл `requests.json`.
9. Подготовить краткое описание архитектуры.

Рекомендуемая итоговая структура

```
RequestApp/
  RequestApi/
    Controllers/
    Models/
    Repositories/
    Services/
    Data/
      requests.json
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
  Program.cs
RequestUi/
  ...
README.md
```

Чек-лист проверки

Что проверить	Результат
Проект запускается без ошибок	[заполнить]
API возвращает список заявок	[заполнить]
Новая заявка добавляется	[заполнить]
Заявка редактируется	[заполнить]
Статус заявки меняется	[заполнить]
Заявка удаляется	[заполнить]
Данные сохраняются в JSON	[заполнить]
Интерфейс получает данные через API	[заполнить]

Пример `README.md` для проекта

```
# Система учета заявок пользователей

Приложение на C# для работы с заявками пользователей.

## Возможности

- добавление заявок;
- просмотр списка заявок;
- редактирование заявок;
- удаление заявок;
- изменение статуса;
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
- хранение данных в JSON;
- работа через REST API.

## Стек

- C#;
- ASP.NET Core Web API;
- System.Text.Json;
- Windows Forms / WPF / Web UI.

## Запуск

1. Запустить API:

   dotnet run

2. Открыть Swagger:

   https://localhost:5001/swagger

3. Запустить интерфейс.
```

Что добавить в отчет

1. Итоговую структуру проекта.
2. Описание слоев приложения.
3. Таблицу использованных паттернов.
4. Скриншоты API.
5. Скриншоты интерфейса.
6. Пример содержимого `requests.json`.
7. Краткий вывод о выполненной работе.

Таблица паттернов для отчета

Паттерн	Где используется	Зачем нужен
Repository	`JsonRequestRepository`	скрывает способ хранения данных

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Паттерн	Где используется	Зачем нужен
Service Layer	`RequestService`	содержит бизнес-логику
MVC / MVVM	интерфейс или веб-часть	отделяет UI от логики
Singleton / DI	регистрация зависимостей	управляет созданием объектов

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru