**Лабораторная работа №3: "Создание RESTful API и его использование в клиентском приложении"**

**Цель:**

Научиться проектировать и реализовывать RESTful API с использованием ASP.NET Core, а также создавать клиентское приложение (например, консольное или WPF) для взаимодействия с этим API.

**Теоретическая часть:**

**1. Что такое RESTful API?**

REST (Representational State Transfer) — это архитектурный стиль для проектирования сетевых приложений, основанный на взаимодействии клиентов и серверов через HTTP-протокол. RESTful API — это интерфейс, который позволяет клиентским приложениям взаимодействовать с сервером, используя стандартные методы HTTP: GET, POST, PUT/PATCH, DELETE.

**Основные принципы REST:**

* Клиент-серверная архитектура : Клиент и сервер являются независимыми компонентами.
* Без состояния (stateless) : Каждый запрос должен содержать всю необходимую информацию для его обработки.
* Единая интерфейсная система : Использование URI для идентификации ресурсов и HTTP-методов для операций над ними.
* Слоистая система : Архитектура может включать промежуточные слои, такие как кэши или балансиры нагрузки.

**2. CRUD-операции в RESTful API**

CRUD (Create, Read, Update, Delete) — это базовые операции для управления данными. В RESTful API они соответствуют следующим HTTP-методам:

* GET : Получение данных (Read).
* POST : Создание новых данных (Create).
* PUT/PATCH : Обновление существующих данных (Update).
  + PUT обычно используется для полного обновления ресурса.
  + PATCH используется для частичного обновления.
* DELETE : Удаление данных.

**Пример URL для доступа к ресурсам:**

* *GET /api/books* — Получить список всех книг.
* *GET /api/books/{id}* — Получить книгу по ID.
* *POST /api/books* — Создать новую книгу.
* *PUT /api/books/{id}* — Обновить книгу по ID.
* *DELETE /api/books/{id}* — Удалить книгу по ID.

**3. ASP.NET Core Web API**

ASP.NET Core — это кроссплатформенный фреймворк для разработки веб-приложений и API. Он предоставляет мощные инструменты для создания RESTful API, включая автоматическую маршрутизацию, валидацию данных и поддержку различных форматов ответов (JSON, XML).

**Основные компоненты ASP.NET Core Web API:**

* Контроллеры : Отвечают за обработку входящих запросов и возврат ответов.
* Маршрутизация : Определяет, какой контроллер и действие должны быть вызваны для конкретного URL.
* Фильтры : Позволяют выполнять дополнительную логику до или после выполнения действия контроллера.
* Middleware : Обрабатывает запросы на уровне приложения (например, аутентификация, логирование).

**Пример простого контроллера:**

|  |
| --- |
| [Route("api/[controller]")]  [ApiController]  public class BooksController : ControllerBase  {  private readonly ApplicationDbContext \_context;  public BooksController(ApplicationDbContext context)  {  \_context = context;  }  // GET: api/Books  [HttpGet]  public async Task<ActionResult<IEnumerable<Book>>> GetBooks()  {  return await \_context.Books.ToListAsync();  }  // POST: api/Books  [HttpPost]  public async Task<ActionResult<Book>> PostBook(Book book)  {  \_context.Books.Add(book);  await \_context.SaveChangesAsync();  return CreatedAtAction(nameof(GetBook), new { id = book.Id }, book);  }  } |

**4. Entity Framework Core**

Entity Framework Core (EF Core) — это ORM (Object-Relational Mapping) для работы с базами данных в .NET. Он позволяет использовать объектно-ориентированный подход для взаимодействия с базой данных, абстрагируясь от SQL-запросов.

Основные функции EF Core:

* **Миграции** : Автоматическое создание и обновление структуры базы данных.
* **LINQ** : Язык запросов для работы с данными.
* **Поддержка разных СУБД** : SQLite, SQL Server, PostgreSQL, MySQL и др.

**Пример использования EF Core:**

|  |
| --- |
| public class ApplicationDbContext : DbContext  {  public DbSet<Book> Books { get; set; }  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  {  optionsBuilder.UseSqlite("Data Source=library.db");  }  } |

**5. HttpClient для взаимодействия с API**

**HttpClient** — это класс из библиотеки System.Net.Http, который позволяет отправлять HTTP-запросы и получать ответы. Он используется для взаимодействия между клиентским приложением и API.

**Пример использования HttpClient:**

|  |
| --- |
| using var client = new HttpClient();  var response = await client.GetAsync("https://localhost:5001/api/books");  if (response.IsSuccessStatusCode)  {  var books = await response.Content.ReadFromJsonAsync<List<Book>>();  Console.WriteLine($"Received {books.Count} books.");  } |

**6. Авторизация и аутентификация с помощью JWT**

JWT (JSON Web Token) — это стандарт для передачи информации между сторонами в виде JSON-объекта. Он часто используется для аутентификации пользователей в RESTful API.

Основные этапы реализации JWT:

1. Генерация токена при успешной аутентификации пользователя.
2. Добавление токена в заголовок **Authorization** каждого последующего запроса.
3. Проверка токена на сервере для обеспечения безопасности.

Пример генерации JWT:

|  |
| --- |
| var token = new JwtSecurityToken(  issuer: "issuer",  audience: "audience",  claims: new[] { new Claim("name", "user") },  expires: DateTime.UtcNow.AddMinutes(30),  signingCredentials: new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256)  );  return new JwtSecurityTokenHandler().WriteToken(token); |

**7. Логирование с использованием Serilog**

Serilog — это популярная библиотека для логирования в .NET. Она позволяет записывать события в различные источники (файлы, консоль, базы данных).

Пример настройки Serilog:

|  |
| --- |
| Log.Logger = new LoggerConfiguration()  .WriteTo.Console()  .WriteTo.File("logs/log-.txt", rollingInterval: RollingInterval.Day)  .CreateLogger();  // Использование  Log.Information("API request received."); |

**Задачи:**

1. **Разработка RESTful API** :
   * Создать проект ASP.NET Core Web API.
   * Реализовать несколько контроллеров для управления ресурсами (например, пользователи, товары, заказы).
   * Реализовать CRUD-операции для каждого ресурса:
     + GET (получение списка или одного элемента)
     + POST (создание нового элемента)
     + PUT/PATCH (обновление существующего элемента)
     + DELETE (удаление элемента)
   * Добавить базовую валидацию данных.
   * Использовать Entity Framework Core для работы с базой данных (можно использовать SQLite или SQL Server Express).
2. **Создание клиентского приложения** :
   * Разработать клиентское приложение (консольное или WPF) для взаимодействия с API.
   * Реализовать основные функции:
     + Просмотр списка ресурсов.
     + Добавление новых записей.
     + Редактирование существующих записей.
     + Удаление записей.
   * Использовать HTTP-клиент (**HttpClient**) для выполнения запросов к API.
3. **Дополнительные задачи**:
   * Добавить авторизацию и аутентификацию с помощью JWT (JSON Web Tokens).
   * Добавить возможность загрузки/скачивания файлов через API.
   * Настроить логирование операций с использованием библиотеки (например, Serilog).

**Варианты заданий:**

**1. Библиотечная система**

**Описание:** Разработка системы управления библиотекой, где можно управлять книгами, читателями и их взаимодействием.

* **Ресурсы API:**
  + Книги (Books): CRUD-операции для добавления/редактирования/удаления книг.
  + Читатели (Readers): CRUD-операции для управления информацией о читателях.
  + Выдачи книг (BookLoans): Записи о том, какие книги выданы каким читателям.
* **Клиентское приложение:**
  + Просмотр списка книг, фильтрация по автору или жанру.
  + Добавление новых книг.
  + Выдача и возврат книг читателям.
  + Отображение истории взаимодействия читателя с книгами.

**2. Система управления задачами (To-Do List)**

**Описание:** Создание простого приложения для управления задачами, которое позволяет пользователям создавать списки задач и отслеживать их выполнение.

* **Ресурсы API:**
  + Задачи (Tasks): CRUD-операции для управления задачами.
  + Пользователи (Users): CRUD-операции для регистрации и входа пользователей.
  + Категории задач (Categories): Группировка задач по категориям.
* **Клиентское приложение:**
  + Просмотр списка задач с возможностью фильтрации по статусу (выполнено/не выполнено).
  + Добавление новых задач с указанием категории и дедлайна.
  + Редактирование статуса задачи (например, отметка как "выполнено").
  + Удаление завершенных задач.

**3. Онлайн-журнал учета расходов**

**Описание:** Приложение для учета личных финансов, где пользователи могут отслеживать свои доходы и расходы.

* **Ресурсы API:**
  + Транзакции (Transactions): CRUD-операции для добавления/редактирования/удаления транзакций.
  + Категории транзакций (Categories): Группировка расходов и доходов.
  + Пользователи (Users): Учетные записи для каждого пользователя.
* **Клиентское приложение:**
  + Просмотр истории транзакций за определенный период.
  + Добавление новых транзакций с указанием суммы, категории и даты.
  + Генерация отчетов по категориям расходов.
  + Возможность экспорта данных в файл CSV.

**4. Система управления блогом**

**Описание:** Разработка платформы для создания и управления блогами, где пользователи могут публиковать статьи, комментировать их и оценивать.

* **Ресурсы API:**
  + Публикации (Posts): CRUD-операции для управления статьями.
  + Комментарии (Comments): Комментарии к статьям.
  + Пользователи (Users): Регистрация и аутентификация пользователей.
  + Теги (Tags): Категоризация статей.
* **Клиентское приложение:**
  + Просмотр списка статей с возможностью фильтрации по тегам.
  + Добавление новых статей с заголовком, текстом и тегами.
  + Добавление комментариев к статьям.
  + Оценка статей (например, лайки/дизлайки).

**5. Система управления событиями**

**Описание:** Приложение для организации мероприятий, где можно создавать события, регистрировать участников и отслеживать их участие.

* **Ресурсы API:**
  + События (Events): CRUD-операции для управления событиями.
  + Участники (Participants): Информация о зарегистрированных участниках.
  + Категории событий (Categories): Группировка событий по типу (конференции, концерты и т.д.).
* **Клиентское приложение:**
  + Просмотр списка событий с детальной информацией.
  + Регистрация на события с указанием количества мест.
  + Отслеживание списка зарегистрированных участников.
  + Отправка уведомлений участникам о начале события.

**Общие элементы для всех вариантов:**

* **Аутентификация и авторизация:** Можно добавить JWT для защиты доступа к ресурсам.
* **Логирование:** Использовать Serilog или другую библиотеку для отслеживания операций.
* **Загрузка файлов:** Например, для загрузки изображений книг, аватарок пользователей или скриншотов транзакций.