# Отчет по Учебному Плану: Неделя 7 — Проектирование и Реализация REST API (Calendar Service)

## Введение и Методологическое Обоснование

### Смена Парадигмы: От Игрового Цикла к Распределенным Контрактам

Переход от разработки на Unity к Enterprise-инженерии на платформе.NET Core требует не просто изучения нового синтаксиса, но фундаментальной перестройки ментальной модели разработчика. В среде Unity (GameDev) выполнение кода подчинено детерминированному игровому циклу (Game Loop), где состояние (State) является локальным, мутабельным и сохраняется в оперативной памяти в течение сессии.1 Взаимодействие между компонентами часто происходит синхронно в рамках одного кадра, а сетевое взаимодействие оптимизировано под минимизацию задержек (Latency) через UDP-протоколы, где потеря пакетов допустима ради скорости.

В противоположность этому, backend-разработка оперирует в среде, которая по своей природе является stateless (без сохранения состояния), асинхронной и распределенной. Здесь «кадр» заменяется на «запрос» (Request), который должен быть обработан изолированно, атомарно и надежно. Если в игре ошибка физики может привести к визуальному глитчу, то ошибка в backend-транзакции финансового сервиса или календаря может привести к нарушению целостности данных бизнеса.

Данный отчет представляет собой исчерпывающее руководство для **7-й недели обучения**, разработанное специально для трансформации мышления Unity-разработчика в мышление системного архитектора. Программа недели сфокусирована на проектировании и реализации **REST API** для сервиса календаря (**Calendar Service**). Это не просто упражнение в написании контроллеров; это глубокое погружение в философию HTTP, управление контрактами данных, валидацию и работу с распределенным временем — концепциями, критически важными для перехода на уровень Middle+/Senior Backend Developer.

### Цели и Задачи Недели

Основная цель недели — создать полностью функциональный, соответствующий промышленным стандартам (Production-Grade) веб-сервис на базе ASP.NET Core (.NET 9), реализующий логику планирования событий.

Ключевые образовательные векторы:

1. **Архитектурное проектирование API:** Понимание REST не как набора методов, а как архитектурного стиля, основанного на ресурсах (Resource-Oriented Architecture). Изучение модели зрелости Ричардсона.
2. **Управление данными и DTO:** Строгое разделение доменных моделей и контрактов данных (Data Transfer Objects) для предотвращения уязвимостей Mass Assignment и обеспечения независимости версионирования.
3. **Надежность и Валидация:** Внедрение паттернов защитного программирования через FluentValidation и стандартизация обработки ошибок согласно RFC 7807 (Problem Details).
4. **Асинхронность и Поточность:** Практическое применение знаний о Task и ThreadPool, полученных на 3-й неделе, в контексте обработки HTTP-запросов, где блокировка потока означает потерю пропускной способности (Throughput).

Ниже представлена детальная структура учебного плана, адаптированная для интеграции в Google Docs, с ежедневной разбивкой теории, практики и протоколов взаимодействия с AI-ментором.

## Структура Документа Google Docs: Неделя 7

### День 1: Философия REST и Проектирование Контрактов

Общее время: 4 часа

Фокус: Архитектура, HTTP-семантика, Swagger/OpenAPI

#### 1.1. Теоретический Модуль: От RPC к RESTful Ресурсам

[Создание документа по REST API день 1](https://docs.google.com/document/d/1EDTSEegE4tD0D9jFA7gjcmCBYBkeuxNc8EJLWO3yADo/edit?usp=drive_link)

В игровой индустрии сетевой код часто строится на парадигме удаленного вызова процедур (Remote Procedure Call - RPC). Разработчик мыслит категориями действий: SpawnPlayer, MoveUnit, FireWeapon. Это императивный подход. REST (Representational State Transfer) требует декларативного подхода, где мы оперируем существительными — **Ресурсами**.

Для Unity-разработчика это можно сравнить с переходом от написания скриптов поведения к проектированию структуры данных в ScriptableObject, где данные первичны, а операции над ними стандартизированы.

Модель Зрелости Ричардсона (Richardson Maturity Model):

Понимание этой модели критично для оценки качества API.2

| **Уровень** | **Описание** | **Аналогия с Unity/GameDev** | **Реализация в Enterprise** |
| --- | --- | --- | --- |
| **Уровень 0: The Swamp of POX** | Использование HTTP только как туннеля для передачи данных (обычно XML или JSON) через единственный endpoint (часто POST). Игнорирование семантики протокола. | Отправка сетевых пакетов с собственными OpCode (например, OpCode.Move, OpCode.Attack) через единый сокет. | SOAP-сервисы, старые XML-RPC системы. Тяжело кэшировать и масштабировать. |
| **Уровень 1: Ресурсы** | Появление множества URI. Каждый объект имеет свой адрес (например, /users/1, /users/2). Но глаголы HTTP все еще используются некорректно (все через POST). | Присвоение каждому сетевому объекту уникального NetworkIdentity или GUID, что позволяет обращаться к ним индивидуально. | API, где есть структура URL, но действия выполняются через параметры URL (/deleteUser?id=1). |
| **Уровень 2: HTTP Глаголы** | Использование стандартных методов (GET, POST, PUT, DELETE, PATCH) по их назначению. Корректное использование статус-кодов (200, 404, 201, 500). **Цель этой недели.** | Использование встроенных событий движка (OnTriggerEnter, Start, Destroy) вместо написания своей системы управления жизненным циклом. | Стандарт для 95% современных Web API. Позволяет использовать инфраструктуру интернета (CDN, кэширующие прокси) эффективно. |
| **Уровень 3: HATEOAS** | Hypermedia As The Engine Of Application State. Ответ API содержит ссылки на возможные следующие действия (переходы состояний). | Контекстное меню в игре: кнопка "Сесть в машину" появляется только когда игрок рядом с машиной. Сервер управляет доступными действиями. | Встречается редко из-за сложности реализации клиента, но считается идеалом REST. |

Анализ передового опыта (Best Practices):

Согласно современным стандартам 5, проектирование API должно начинаться не с кода, а с контракта.

* **Именование:** Используйте множественное число для коллекций (/events, а не /event).
* **Вложенность:** Логическая иерархия (/users/123/events — события конкретного пользователя).
* **Версионирование:** Обязательное наличие версии в URL или заголовке (/api/v1/...) для обеспечения обратной совместимости.8

#### 1.2. Практическое Задание: "Архитектор Контрактов"

Контекст: Вы разрабатываете "Calendar Service". Прежде чем написать первую строчку кода на C#, вы должны спроектировать его интерфейс.

Инструмент: Swagger Editor (или просто текстовое описание в формате OpenAPI YAML/JSON).

Задача:

Описать спецификацию API для ресурса Event (Событие календаря).

Поля ресурса CalendarEvent:

* Id (Guid) — Уникальный идентификатор.
* Title (string) — Заголовок.
* Description (string) — Описание.
* StartTime (DateTimeOffset) — Время начала (обязательно с часовым поясом, см. День 5).
* Duration (TimeSpan) — Длительность.
* Location (string) — Место проведения.

**Требуемые Эндпоинты:**

1. **Создание:** POST /api/v1/events. Должен возвращать код 201 Created и заголовок Location с ссылкой на созданный ресурс.
2. **Чтение (List):** GET /api/v1/events. Поддержка фильтрации по диапазону дат (from, to).
3. **Чтение (Single):** GET /api/v1/events/{id}. Возвращает 404 Not Found, если событие не найдено.
4. **Полное обновление:** PUT /api/v1/events/{id}. Заменяет ресурс целиком. Идемпотентная операция.
5. **Частичное обновление:** PATCH /api/v1/events/{id}. Обновляет только переданные поля. (Более сложная реализация, обсудим JSON Patch).
6. **Удаление:** DELETE /api/v1/events/{id}. Возвращает 204 No Content.

#### 1.3. Промпт для AI-Ментора (День 1)

Роль: Senior Backend Architect (C#/.NET Expert).

Контекст: Я Unity-разработчик, изучаю проектирование REST API. Я составил спецификацию эндпоинтов для сервиса календаря.

Задача: Проведи ревью (Code Review) моего дизайна API.

Критерии проверки:

1. Соответствует ли дизайн 2-му уровню модели Ричардсона?
2. Правильно ли выбраны HTTP глаголы для действий?
3. Сравни мой подход к передаче времени (DateTime) с лучшими практиками (ISO 8601, UTC). Почему в распределенной системе нельзя просто отправить "строку времени"?
4. Объясни, почему для операции удаления мы возвращаем 204, а не 200 с текстом "Deleted".  
   Вводные данные: [Вставь описание своих эндпоинтов здесь]

### День 2: Контроллеры, Маршрутизация и Жизненный Цикл Запроса

Общее время: 4 часа

Фокус: ASP.NET Core MVC, Dependency Injection, Async/Await

#### 2.1. Теоретический Модуль: Анатомия Контроллера

[неделя 7 день 2](https://docs.google.com/document/d/1O3flV1wYklqMCjg53BFBpP71B_wvwTCyXKw2fTwHJss/edit?usp=drive_link)

В Unity основным "кирпичиком" логики является MonoBehaviour. В ASP.NET Core Web API (классическом подходе) это ControllerBase.

Главное отличие: MonoBehaviour живет в сцене долго, хранит состояние (поля класса), и его методы (Update) вызываются каждый кадр. Controller создается на каждый HTTP-запрос заново и уничтожается сразу после отправки ответа. Он stateless (не хранит состояние запроса в полях класса).

Controllers vs Minimal API:

В.NET 9 активно продвигается подход Minimal API — легковесное определение эндпоинтов прямо в Program.cs.9

* *Преимущества Minimal API:* Меньше бойлерплейта, выше производительность (меньше накладных расходов на создание классов и фильтры), идеально для микросервисов.
* *Преимущества Controllers:* Структурированность, привычная ООП-модель, встроенная поддержка сложных фильтров и валидации.
* *Решение для курса:* Мы начинаем с **Controllers**.10 Для Unity-разработчика, привыкшего к классам и компонентам, структура контроллера будет понятнее для организации кода. Minimal API мы рассмотрим как альтернативу на 10-й неделе.

Атрибутная Маршрутизация (Attribute Routing):

Вместо таблицы маршрутизации, мы декорируем методы атрибутами, похожими на или в Unity, но управляющими доступом из сети.

C#

[ApiController]  
")] // Автоматически подставит имя контроллера  
public class EventsController : ControllerBase  
{  
 private readonly ICalendarService \_service;  
  
 // Constructor Injection — паттерн, изученный на 6-й неделе  
 public EventsController(ICalendarService service)  
 {  
 \_service = service;  
 }  
  
 [HttpGet("{id:guid}")] // Ограничение маршрута типом GUID  
 public async Task<IActionResult> GetById(Guid id)  
 {  
 var result = await \_service.GetEventAsync(id);  
 if (result == null) return NotFound(); // Helper method returns 404  
 return Ok(result); // Helper method returns 200 + JSON body  
 }  
}

Асинхронность (Async/Await) в Web API:

В Unity async/await (или Coroutines) используются, чтобы не фризить UI/Render Thread. В Web API это используется, чтобы не блокировать ThreadPool. Если поток заблокирован ожиданием БД, он не может обрабатывать другие входящие запросы. Это критично для масштабируемости (Scalability).12

#### 2.2. Практическое Задание: "Каркас Приложения"

**Задача:** Реализовать базовый EventsController с заглушкой сервиса.

1. **Проект:** Создать новый проект ASP.NET Core Web API (.NET 9).
2. **Сервис:** Реализовать InMemoryCalendarService (используя ConcurrentDictionary для потокобезопасного хранения данных в памяти).
   * *Важно:* Поскольку контроллер создается каждый раз заново, сервис должен быть зарегистрирован как Singleton в DI-контейнере (builder.Services.AddSingleton<ICalendarService,...>), чтобы данные "жили" между запросами.1
3. **Контроллер:** Реализовать методы Create, GetById, GetAll.
4. **Тест:** Использовать встроенный Swagger UI для отправки запросов. Убедиться, что созданное событие можно получить по ID.

#### 2.3. Промпт для AI-Ментора (День 2)

Роль: Senior.NET Developer.

Контекст: Я пишу свой первый контроллер на ASP.NET Core. Использую InMemory хранилище.

Задача:

1. Объясни разницу между Singleton, Scoped и Transient временем жизни сервиса в контексте моего InMemoryCalendarService. Почему, если я сделаю его Transient, данные будут пропадать после каждого запроса?
2. Проанализируй мой код контроллера. Правильно ли я использую ActionResult<T>?
3. Сравни механизм [HttpGet] атрибута с системой событий в Unity. Как фреймворк узнает, какой метод вызвать?

### День 3: DTO, Маппинг и Защита Данных

Общее время: 4 часа

Фокус: Безопасность, Разделение ответственности, AutoMapper

#### 3.1. Теоретический Модуль: DTO vs Domain Entities

[день 3: DTO и маппингу](https://docs.google.com/document/d/1xlEgfzZtiIGduo1ZFbv-U7ctv8cxI6zwRA2KmP808jU/edit?usp=drive_link)

В Unity часто используется один класс для логики, сериализации и сохранения (например, класс PlayerStats, помеченный ``). В Backend это **антипаттерн**.

Проблема Mass Assignment (Массовое присваивание):

Если вы используете сущность БД (Entity) напрямую в качестве входного параметра API метода, злоумышленник может передать в JSON поле IsAdmin: true. Если это поле есть в сущности, оно будет перезаписано.

Решение: DTO (Data Transfer Objects). Это простые классы (или record types), содержащие только те данные, которые клиент имеет право отправить или получить.

Records в C# 9+:

Идеальны для DTO благодаря иммутабельности и лаконичности синтаксиса.

C#

public record CreateEventRequest(  
 string Title,  
 string Description,  
 DateTimeOffset StartTime,  
 TimeSpan Duration  
);

Маппинг (Mapping):

Преобразование DTO -> Entity и Entity -> DTO. Может быть ручным (для явного контроля) или автоматическим (AutoMapper, Mapster). Для обучения начнем с ручного, затем внедрим AutoMapper.

#### 3.2. Практическое Задание: "Разделяй и Властвуй"

**Задача:** Рефакторинг контроллера для использования DTO.

1. **Создать DTOs:**
   * CreateEventRequest (входящий).
   * UpdateEventRequest (входящий).
   * EventResponse (исходящий). Исключить служебные поля, если они есть в сущности, добавить вычисляемые поля (например, EndTime = StartTime + Duration).
2. **Logic:**
   * В методе Create принимать CreateEventRequest.
   * Создавать сущность CalendarEvent, копируя поля.
   * Передавать сущность в сервис.
   * Преобразовывать возвращенную сущность в EventResponse перед возвратом Ok().
3. **AutoMapper:** Подключить библиотеку AutoMapper. Создать MappingProfile, настроить правила преобразования. Заменить ручной код маппинга на \_mapper.Map<Target>(source).

#### 3.3. Промпт для AI-Ментора (День 3)

Роль: Security Specialist &.NET Architect.

Контекст: Я разделил модели на Entity и DTO.

Задача:

1. Объясни на примере "взлома", что могло бы произойти, если бы я принимал сущность CalendarEvent (содержащую поле OwnerId) напрямую в контроллере.
2. Проверь мои record определения. Есть ли смысл делать их sealed?
3. Сравни производительность ручного маппинга и AutoMapper (используя рефлексию). Насколько это критично для высоконагруженного API?

### День 4: Валидация и Обработка Ошибок (Problem Details)

Общее время: 4 часа

Фокус: Надежность, Middleware, RFC 7807

#### 4.1. Теоретический Модуль: "Никогда не верь клиенту"

[день 4: Валидация и обработка ошибок REST API](https://docs.google.com/document/d/1vznX9h0jz1CFZkhUrhyK915QwOwf0JIb4ABFR_cwPCs/edit?usp=drive_link)

В Unity валидация часто происходит на уровне UI (Input Field restrict) или внутри сеттеров свойств. В API валидация должна быть жесткой и происходить до выполнения бизнес-логики.

FluentValidation:

Стандарт де-факто в экосистеме.NET.13 Позволяет вынести правила валидации из классов моделей в отдельные классы-валидаторы. Это соответствует принципу Single Responsibility (SOLID).

* *Преимущество:* Возможность сложных правил ("поле A обязательно, только если поле Б заполнено"), асинхронная валидация (проверка уникальности email в БД).

Глобальная обработка ошибок:

Вместо try-catch в каждом методе, используется Middleware. Любое неперехваченное исключение поднимается вверх по пайплайну. Middleware ловит его и формирует красивый JSON-ответ.

RFC 7807 (Problem Details):

Стандартизированный формат ошибки API.16

Вместо простого текста "Error", возвращается JSON объект:

JSON

{  
 "type": "https://calendar.com/errors/validation",  
 "title": "Validation Failed",  
 "status": 400,  
 "errors": { "Title": ["Must not be empty"] }  
}

#### 4.2. Практическое Задание: "Защитный Периметр"

1. **FluentValidation:**
   * Установить FluentValidation.AspNetCore.
   * Создать CreateEventValidator:
     + Title: не пустой, макс 100 символов.
     + StartTime: Не может быть в прошлом (нюанс с тестированием!).
     + Duration: Должна быть больше 0.
   * Подключить валидацию к пайплайну (через DI или SharpGrip.FluentValidation.AutoValidation паттерн).
2. **Exception Middleware:**
   * Реализовать кастомный Middleware или использовать IExceptionHandler (новшество.NET 8/9).
   * Ловить KeyNotFoundException -> Возвращать 404 Problem Details.
   * Ловить ValidationException -> Возвращать 400 Problem Details.
   * Ловить Exception (все остальное) -> Возвращать 500 (и логировать!).

#### 4.3. Промпт для AI-Ментора (День 4)

Роль: Senior Backend Developer.

Контекст: Я внедряю валидацию и обработку ошибок.

Задача:

1. Посмотри на мое правило RuleFor(x => x.StartTime).GreaterThan(DateTime.Now). Почему это правило сделает мои юнит-тесты нестабильными (flaky)? Как правильно валидировать время, чтобы тесты были детерминированными?
2. Объясни концепцию Middleware Pipeline в ASP.NET Core, используя аналогию с конвейером обработки физики или рендеринга в Unity.
3. Проверь JSON-структуру ответа об ошибке на соответствие RFC 7807.

### День 5: Продвинутые Темы — Время, Идемпотентность, Пагинация

Общее время: 4 часа

Фокус: Распределенные системы, Сложность реального мира

#### 5.1. Теоретический Модуль: Ловушки Времени и Сети

[день 5: неделя 7](https://docs.google.com/document/d/1chlQ8wX5lyh7y7kBqefgqWJk8QdJvMF6Te16vu3bgIE/edit?usp=drive_link)

Управление Временем (Timezones):

В глобальном сервисе "Завтра в 10 утра" — понятие относительное.

* *Правило 1:* Всегда хранить время в UTC или использовать DateTimeOffset, чтобы сохранить смещение.19
* *Правило 2:* API возвращает время в формате ISO 8601 (2023-10-05T14:30:00Z). Клиент сам переводит в локальное время пользователя.

Идемпотентность (Idempotency):

В мобильных сетях (клиенты вашего API) запросы могут дублироваться (retries). Если клиент дважды отправит POST /events (создать событие) из-за лага сети, сервер не должен создавать два события.21

* *Решение:* Idempotency-Key заголовок. Сервер запоминает ключ и результат обработки. При повторном ключе возвращает сохраненный результат без повторного выполнения логики.

Пагинация (Pagination):

Метод GetAll опасен. Если событий миллион, сервер упадет c OutOfMemoryException (аналог переполнения кучи в Unity при инстанцировании тысяч префабов).

* *Offset-based:* Skip(10).Take(10). Просто, но медленно на больших данных.
* *Cursor-based:* "Дай мне 10 записей *после* вот этого ID". Быстро, надежно.23

#### 5.2. Практическое Задание: "Production Ready"

1. **Timezone Guard:** Убедиться, что DTO принимают DateTimeOffset, и при маппинге в Entity время приводится к UTC.
2. **Pagination:** Изменить метод GetAll на GetList(int page, int pageSize). Реализовать валидацию pageSize (не более 100). Вернуть метаданные пагинации (TotalCount, HasNextPage) в заголовках ответа или в теле-обертке.
3. **Idempotency (Bonus):** Реализовать простейший IdempotencyFilterAttribute, который проверяет наличие заголовка и сохраняет ключи в MemoryCache (имитация Distributed Cache).

#### 5.3. Итоговое Задание Недели (Capstone Task)

Тема: Реализация "Умного" Эндпоинта Повторяющихся Событий.

Задача:

Добавить в CreateEventRequest поле RecurrenceRule (строка формата iCal, например FREQ=WEEKLY;INTERVAL=1).

Реализовать эндпоинт GET /api/v1/events/occurrences?from=...&to=....

* Логика: Этот метод не просто читает базу. Он берет повторяющиеся события и "разворачивает" (calculate/expand) их в памяти, возвращая список конкретных инстансов событий на заданный период. Это алгоритмическая задача, близкая к логике игровых механик.

#### 5.4. Промпт для AI-Ментора (День 5)

Роль: System Architect.

Контекст: Я дорабатываю API для работы в реальных условиях (время, сеть).

Задача:

1. Объясни "The 5 Laws of API Dates". Почему использование DateTime без указания Kind (Utc/Local) является бомбой замедленного действия?
2. Оцени мою стратегию пагинации. В чем минусы Skip/Take подхода при работе с SQL базой данных, в которой 10 миллионов записей?
3. Как правильно тестировать идемпотентность? Какие сценарии (Race Conditions) нужно учесть?

## Дополнительные Материалы и Ресурсы

### Список Литературы для Глубокого Погружения

1. **"REST API Design Rulebook" (Mark Masse)** — справочник по правильному использованию HTTP.
2. **"Concurrency in C#" (Stephen Cleary)** — обязательно главы про async/await в ASP.NET (отличие от UI/Unity контекста).
3. **Документация Microsoft:** "Controller-based APIs in ASP.NET Core".

### Чек-лист Самопроверки (Definition of Done)

* [ ] Проект компилируется без предупреждений (Warnings).
* [ ] Swagger UI открывается и позволяет протестировать все методы.
* [ ] Валидация работает: попытка создать событие в прошлом возвращает 400 и JSON с деталями.
* [ ] При возникновении исключения сервер не "падает", а возвращает 500 и JSON (Problem Details).
* [ ] Структура проекта разделена: Контроллеры не знают о базе данных, DTO отделены от Сущностей.
* [ ] Код асинхронный (async/await) на всем пути от Контроллера до (имитации) БД.

## Аналитическая Записка: Инсайты и Обоснование Подхода

### Педагогическая Стратегия: От Императива к Декларативности

Анализ материалов 1 показывает, что главной сложностью для Unity-разработчика является не синтаксис C# (который обычно на высоком уровне), а смена парадигмы управления состоянием. В Unity разработчик привык управлять состоянием каждый кадр. В REST API разработчик описывает состояние и делегирует управление фреймворку.

Поэтому программа 7-й недели построена на жестком разделении ответственности (Separation of Concerns). Мы намеренно запрещаем использовать Minimal API на этом этапе 10, так как структура контроллеров, фильтров и middleware создает ментальные "полочки", аналогичные компонентам и системам в Unity, что облегчает усвоение материала.

### Важность Стандартизации Ошибок

Внедрение RFC 7807 16 на раннем этапе критично. В геймдеве ошибки часто логируются в консоль, а игрок видит просто "Connection Lost". В Enterprise детальная, машиночитаемая информация об ошибке — это часть контракта. Это учит разработчика относиться к ошибкам не как к исключительным ситуациям, а как к еще одному варианту потока данных (Control Flow).

### Работа со Временем как Тест на Зрелость

Включение темы Timezones и ISO 8601 19 является фильтром, отделяющим Junior от Middle разработчиков. Unity-разработчики привыкли к Time.deltaTime и локальному времени устройства. Столкновение с реальностью распределенных часовых поясов заставляет задуматься о природе данных и их представлении, что является ключевым навыком backend-инженера.

#### Источники

1. Разработка бэкенда: план обучения C#
2. Richardson Maturity Model - REST API Tutorial, дата последнего обращения: декабря 5, 2025, <https://restfulapi.net/richardson-maturity-model/>
3. Richardson Maturity Model - Martin Fowler, дата последнего обращения: декабря 5, 2025, <https://martinfowler.com/articles/richardsonMaturityModel.html>
4. REST API Maturity Levels - DEV Community, дата последнего обращения: декабря 5, 2025, <https://dev.to/rafaeljcamara/rest-api-maturity-levels-41ec>
5. 14 Best Practices for Designing RESTful APIs | .NET Core Web API - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@jeslurrahman/14-best-practices-for-designing-restful-apis-net-core-web-api-1f34d6b8303e>
6. Best practices for RESTful web API design - Azure - Microsoft Learn, дата последнего обращения: декабря 5, 2025, <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>
7. Building RESTful APIs with .NET 9: A Comprehensive Guide for Modern Developers, дата последнего обращения: декабря 5, 2025, <https://dev.to/leandroveiga/building-restful-apis-with-net-9-a-comprehensive-guide-for-modern-developers-47hc>
8. API Versioning Strategies: Best Practices Guide - Daily.dev, дата последнего обращения: декабря 5, 2025, <https://daily.dev/blog/api-versioning-strategies-best-practices-guide>
9. APIs overview | Microsoft Learn, дата последнего обращения: декабря 5, 2025, <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/apis?view=aspnetcore-10.0>
10. Why aren't you using Minimal APIs? - By dotnet team members - Reddit, дата последнего обращения: декабря 5, 2025, <https://www.reddit.com/r/dotnet/comments/1m0s1y4/why_arent_you_using_minimal_apis_by_dotnet_team/>
11. Comparing Minimal and Controller-Based APIs in ASP.NET | by Peter Semrau-Boughton, дата последнего обращения: декабря 5, 2025, <https://medium.com/@petersembou/comparing-minimal-and-controller-based-apis-in-asp-net-6f0fc1dec078>
12. Chapter 5 — Part 1— Best Practices in ASP.NET Core Web API, дата последнего обращения: декабря 5, 2025, <https://medium.com/@UlbertAO/chapter-5-best-practices-in-asp-net-core-web-api-18d7d431d295>
13. ASP.NET Core — FluentValidation documentation, дата последнего обращения: декабря 5, 2025, <https://fluentvalidation.net/aspnet>
14. Optimal Strategies for Implementing FluentValidation in .NET - EVOKEHUB, дата последнего обращения: декабря 5, 2025, <https://evokehub.com/optimal-strategies-for-implementing-fluentvalidation-in-net/>
15. Fluent Validation: How To Register All Validators in .NET Core Project | by Mustafa ŞAHİN, дата последнего обращения: декабря 5, 2025, <https://medium.com/@58mustafasahin/fluent-validation-how-to-register-all-validators-in-net-core-project-b384d00f0c3b>
16. ProblemDetails in ASP.NET Core – Standardizing API Error Responses - codewithmukesh, дата последнего обращения: декабря 5, 2025, <https://codewithmukesh.com/blog/problem-details-in-aspnet-core/>
17. RFC 7807: Problem Details for HTTP APIs, дата последнего обращения: декабря 5, 2025, <https://www.rfc-editor.org/rfc/rfc7807.html>
18. Problem Details for ASP.NET Core APIs - Milan Jovanović, дата последнего обращения: декабря 5, 2025, <https://www.milanjovanovic.tech/blog/problem-details-for-aspnetcore-apis>
19. Best Practices Using Time Zone API Web Services - Google for Developers, дата последнего обращения: декабря 5, 2025, <https://developers.google.com/maps/documentation/timezone/web-service-best-practices>
20. The 5 laws of API dates and times, дата последнего обращения: декабря 5, 2025, <https://apiux.com/2013/03/20/5-laws-api-dates-and-times/>
21. Implementing Idempotency Keys in REST APIs | Zuplo Learning Center, дата последнего обращения: декабря 5, 2025, <https://zuplo.com/learning-center/implementing-idempotency-keys-in-rest-apis-a-complete-guide>
22. Idempotency - What is an Idempotent REST API? - REST API Tutorial, дата последнего обращения: декабря 5, 2025, <https://restfulapi.net/idempotent-rest-apis/>
23. A guide to REST API pagination - Merge.dev, дата последнего обращения: декабря 5, 2025, <https://www.merge.dev/blog/rest-api-pagination>
24. Pagination Best Practices in REST API Design - Speakeasy, дата последнего обращения: декабря 5, 2025, <https://www.speakeasy.com/api-design/pagination>