# Архитектура Распределенных Систем и Надежность Бэкенда

## Глубокий Анализ Теоретических Основ Пятого Дня (Неделя 7)

### Аннотация

Данный документ представляет собой исчерпывающее техническое руководство и теоретический анализ, разработанный для седьмой недели учебного плана "REST API для бэкенда". Основная цель отчета — обеспечить фундаментальную трансформацию ментальной модели разработчика, переходящего из среды Unity (GameDev) в сферу корпоративной разработки на платформе.NET Core. Если предыдущие дни фокусировались на синтаксисе и базовых паттернах, то пятый день посвящен системному проектированию: управлению временем в распределенной среде, обеспечению идемпотентности операций и эффективной навигации по большим массивам данных. Документ подробно рассматривает стандарты ISO 8601, RFC 5545 и RFC 7807, а также современные механизмы.NET 9, такие как TimeProvider и стратегии Keyset Pagination.

## 1. Введение: Смена Парадигмы от Игрового Цикла к Распределенному Состоянию

Переход от разработки клиентских симуляций (Unity) к серверным системам требует отказа от концепции детерминированного времени и локального состояния. В игровом движке время линейно, контролируемо и локально: Time.deltaTime — это разница между кадрами на конкретном процессоре, а состояние мира живет в оперативной памяти и обновляется 60 раз в секунду.1

В бэкенд-разработке мы сталкиваемся с принципиально иной реальностью:

1. **Недетерминизм:** Запросы обрабатываются асинхронно, порядок их поступления не гарантирует порядок выполнения.
2. **Распределенность:** "Сейчас" — это относительное понятие. Сервер может находиться в UTC+0, база данных в UTC-5, а клиент в UTC+3.
3. **Эфемерность:** Состояние не хранится в памяти процесса (Stateless). Каждый HTTP-запрос должен содержать всю необходимую информацию для его обработки или ссылаться на персистентное хранилище.

Анализ учебного плана показывает, что пятый день является критическим рубежом, отделяющим уровень Junior от Middle. Понимание того, как время течет в распределенной системе, и как защитить данные от сетевых сбоев (идемпотентность), является квалификационным требованием для проектирования надежных API.1

## 2. Темпоральная Архитектура: Управление Временем

### 2.1 Фундаментальная Проблема Времени в Распределенных Системах

В Unity разработчик привык доверять системному времени. Вызов DateTime.Now возвращает время на устройстве игрока, что достаточно для локальной логики (например, кулдаун способности). Однако в распределенной системе использование локального времени сервера является одной из самых опасных архитектурных ошибок, ведущих к так называемому "темпоральному дрейфу" (temporal drift).3

Представьте сценарий:

* Сервер А (Лондон) принимает запрос на создание события.
* Сервер Б (Нью-Йорк) обрабатывает запрос на чтение этого события.
* Если оба сервера используют локальное время без привязки к смещению, событие может "исчезнуть" или появиться в будущем из-за разницы часовых поясов.4

### 2.2 Пять Законов API для Работы с Датами

Для решения проблем синхронизации индустрия выработала свод правил, известный как "5 Laws of API Dates". Эти законы являются императивом для проектирования Calendar Service.

#### Закон 1: Используйте ISO-8601

В отличие от культурно-зависимых форматов (например, 01/02/2023, который в США означает 2 января, а в Европе — 1 февраля), формат ISO-8601 (YYYY-MM-DDTHH:mm:ss.sssZ) является однозначным и лексикографически сортируемым.5

* **Применение:** API календаря никогда не должен принимать или возвращать даты в нестандартных форматах. JSON-сериализатор в.NET Core по умолчанию настроен на этот стандарт.

#### Закон 2: Принимайте Любой Часовой Пояс

API должен быть либерален к входным данным. Клиент может отправить время в своем локальном поясе (например, с суффиксом -05:00). Сервер обязан корректно распарсить это значение, учитывая смещение, а не игнорировать его.5

#### Законы 3 и 4: Храните и Возвращайте в UTC

Это "золотое правило" бэкенда. В базе данных время должно храниться в UTC (Coordinated Universal Time). Это "нормализация" времени.

* **Обоснование:** UTC не имеет переходов на летнее/зимнее время (DST). Это непрерывная шкала времени. Если хранить локальное время, то при переходе на зимнее время один час повторяется дважды, что делает невозможным уникальную идентификацию момента времени.6
* **Исключение:** Для календарных событий часто требуется хранить не только момент времени UTC, но и *исходный* часовой пояс пользователя, чтобы корректно обрабатывать пересчеты при изменении правил DST в конкретной стране (политические изменения часовых поясов).8

#### Закон 5: Не Используйте Время, Если Оно Не Нужно

Для событий типа "День рождения" использование DateTime ошибочно. Рождение происходит в дату, а не в миллисекунду. Использование DateTime с установленным временем 00:00:00 может привести к смещению даты на предыдущий день при конвертации часовых поясов.

* **Решение:** Использовать тип DateOnly, введенный в.NET 6, который семантически точно описывает "календарную дату" без привязки к суточному времени.5

### 2.3 Выбор Типов Данных в.NET: DateTime vs DateTimeOffset

Для Unity-разработчика привычен тип DateTime. Однако в контексте.NET Server-Side этот тип является источником множества багов из-за свойства Kind.

| **Характеристика** | **DateTime** | **DateTimeOffset** |
| --- | --- | --- |
| **Структура** | Тики + Kind (Local/Utc/Unspecified) | Тики + Смещение (Offset) |
| **Однозначность** | Нет. 10:00 может быть UTC или Local. | Да. 10:00+02:00 — это абсолютный момент. |
| **Арифметика** | Зависит от настроек сервера. | Всегда корректна (приводится к UTC). |
| **SQL Mapping** | datetime / datetime2 | datetimeoffset |
| **Рекомендация** | Только для Legacy или UI логики. | **Стандарт для API и БД**. |

**Анализ:** DateTimeOffset хранит момент времени относительно UTC и само смещение. Это позволяет восстановить локальное время пользователя, что критично для календаря ("Встреча в 10 утра по Москве" должна остаться в 10 утра по Москве, даже если правила перехода времени изменятся).8

Использование DateTime с Kind = Unspecified в API считается антипаттерном, так как JSON-сериализатор будет интерпретировать его как локальное время сервера, что в облаке (Azure/AWS) обычно означает UTC, но на локальной машине разработчика — локальное время, создавая эффект "на моем компьютере работает".11

### 2.4 Тестирование Времени: TimeProvider в.NET 8/9

Одной из сложнейших задач в Unity было тестирование таймеров и кулдаунов — часто приходилось ждать реальное время или писать сложные обертки. В.NET 8 был представлен абстрактный класс TimeProvider, который революционизировал тестирование темпоральной логики.12

#### Проблема DateTime.UtcNow в Тестах

Использование статического свойства DateTime.UtcNow делает код жестко связанным с системными часами.

1. **Недетерминированность:** Тест, запущенный в 23:59:59, может упасть, если выполнение перейдет на следующие сутки.
2. **Медлительность:** Тестирование таймаута в 5 минут требует реального ожидания или хаков с Thread.Sleep.

#### Решение: Внедрение Зависимости TimeProvider

Вместо вызова статики, мы внедряем TimeProvider в конструктор сервиса.

C#

public class MeetingService(TimeProvider timeProvider)  
{  
 public bool IsMeetingStarting(Meeting meeting)  
 {  
 // Получаем время через абстракцию, а не напрямую от системы  
 var now = timeProvider.GetUtcNow();  
 return meeting.StartTime <= now.AddMinutes(15);  
 }  
}

В тестах мы используем Microsoft.Extensions.Time.Testing.FakeTimeProvider:

C#

[Fact]  
public void Should\_Alert\_When\_Meeting\_Is\_Soon()  
{  
 // Arrange  
 var fakeTime = new FakeTimeProvider();  
 fakeTime.SetUtcNow(new DateTimeOffset(2024, 1, 1, 10, 0, 0, TimeSpan.Zero));  
   
 var service = new MeetingService(fakeTime);  
 var meeting = new Meeting { StartTime = new DateTimeOffset(2024, 1, 1, 10, 10, 0, TimeSpan.Zero) };  
  
 // Act & Assert  
 Assert.True(service.IsMeetingStarting(meeting));  
   
 // Мгновенное "путешествие во времени"  
 fakeTime.Advance(TimeSpan.FromMinutes(20));  
 // Проверка логики после завершения встречи...  
}

Этот подход позволяет проверять сложные сценарии (високосные годы, переход через полночь, переход DST) детерминированно и мгновенно.12

## 3. Идемпотентность: Гарантия Надежности в Ненадежной Сети

### 3.1 Природа Сетевых Сбоев

В отличие от вызова метода внутри одного процесса Unity (Update()), HTTP-запрос проходит через множество узлов. Сетевой сбой может произойти в момент получения ответа.

Сценарий:

1. Мобильный клиент отправляет POST /events (создать встречу).
2. Сервер получает запрос, сохраняет встречу в БД, списывает средства (если это платный сервис).
3. Сервер отправляет ответ 201 Created.
4. Связь обрывается (клиент зашел в лифт). Клиент получает TimeoutException.
5. Клиент, думая, что запрос не прошел, автоматически повторяет его (Retry policy).

Без защиты сервер создаст *вторую* встречу и спишет средства *дважды*. Это недопустимо в Enterprise-системах.16

### 3.2 Математическое и Техническое Определение

Операция считается **идемпотентной**, если многократное её выполнение приводит к тому же состоянию системы, что и однократное: f(f(x)) = f(x).18

* GET, PUT, DELETE — идемпотентны по стандарту HTTP. Удаление удаленного объекта возвращает 404 или 204, но состояние системы не меняется (объект удален).
* POST — **не идемпотентен**. Он создает новый ресурс при каждом вызове.19

### 3.3 Паттерн Idempotency Key

Для решения проблемы используется паттерн ключа идемпотентности, описанный в черновике стандарта IETF и популяризированный Stripe.6

**Алгоритм Реализации:**

1. **Клиент:** Генерирует уникальный ключ (обычно UUID v4) для *действия* создания. Этот ключ передается в заголовке Idempotency-Key.21 Важно: ключ генерируется *до* первой попытки отправки и переиспользуется при ретраях.
2. **Middleware (Сервер):** Перехватывает запрос до попадания в контроллер.
3. **Проверка Кеша:** Сервер проверяет распределенный кеш (Redis) на наличие ключа.
   * Если ключ найден и операция завершена -> возвращается сохраненный результат (тот же JSON, тот же Status Code).
   * Если ключ найден, но операция "В процессе" -> возвращается 409 Conflict или происходит ожидание (Distributed Lock).
   * Если ключ не найден -> управление передается контроллеру.
4. **Сохранение:** После выполнения контроллера, результат (код ответа и тело) сохраняется в кеш с привязкой к ключу.22

### 3.4 Практическая Реализация в ASP.NET Core

Для учебного проекта мы используем IMemoryCache как имитацию распределенного кеша. Однако в продакшене обязательно использование IDistributedCache (Redis), так как при масштабировании (несколько экземпляров API) локальная память у каждого инстанса своя.24

**Атрибут Идемпотентности (Код):**

C#

public class IdempotentAttribute : Attribute, IAsyncActionFilter  
{  
 private const string HeaderName = "Idempotency-Key";  
   
 public async Task OnActionExecutionAsync(ActionExecutingContext context, ActionExecutionDelegate next)  
 {  
 if (!context.HttpContext.Request.Headers.TryGetValue(HeaderName, out var keyValues))  
 {  
 context.Result = new BadRequestObjectResult("Idempotency-Key header is missing");  
 return;  
 }  
   
 var key = keyValues.ToString();  
 var cache = context.HttpContext.RequestServices.GetRequiredService<IMemoryCache>();  
   
 // 1. Проверяем наличие в кеше  
 if (cache.TryGetValue(key, out IdempotencyRecord? cachedRecord))  
 {  
 // Возвращаем сохраненный результат без выполнения логики  
 context.Result = new ObjectResult(cachedRecord.Body) { StatusCode = cachedRecord.StatusCode };  
 return;  
 }  
  
 // 2. Выполняем действие (контроллер)  
 var executedContext = await next();  
  
 // 3. Сохраняем результат  
 if (executedContext.Result is ObjectResult result)  
 {  
 var record = new IdempotencyRecord(result.StatusCode?? 200, result.Value);  
 cache.Set(key, record, TimeSpan.FromHours(24)); // TTL 24 часа  
 }  
 }  
}

*Примечание:* Реализация выше упрощена. В реальной системе необходимо атомарно блокировать ключ на время выполнения, чтобы избежать "гонки", когда два запроса приходят одновременно.16

**TTL (Time-To-Live):** Ключи не хранятся вечно. Стандартная практика — хранить ключ 24-48 часов. Этого достаточно для покрытия любых разумных ретраев со стороны клиента.26

## 4. Навигация по Данным: Стратегии Пагинации

### 4.1 Ограничения Offset Pagination (Skip/Take)

Классический подход Skip(10).Take(10) (в SQL OFFSET 10 LIMIT 10) понятен интуитивно, но имеет два критических недостатка в высоконагруженных системах.7

1. **Проблема Производительности ($O(N)$):** База данных не может просто "перепрыгнуть" 1 000 000 записей. Она должна прочитать их с диска, десериализовать, отсчитать и выбросить. Чем больше страница (глубина пагинации), тем медленнее запрос. Для 10 миллионов событий запрос последней страницы может "положить" базу.29
2. **Проблема Дрейфа Данных (Data Drift):** Если пользователь смотрит 1-ю страницу (события 1-10), и в этот момент добавляется новое событие в начало списка, то событие, бывшее 10-м, станет 11-м. Когда пользователь перейдет на 2-ю страницу (события 11-20), он увидит это событие снова. Это дублирование данных на клиенте.28

### 4.2 Cursor-Based (Keyset) Pagination

Для бесконечных лент (Infinite Scroll) и больших списков стандартом является пагинация по курсору (Keyset Pagination).29

Принцип работы:

Вместо "дай мне записи, пропустив 100 штук", клиент говорит: "Дай мне 10 записей, которые идут после записи с ID=X и Датой=Y".

**SQL эквивалент:**

SQL

SELECT \* FROM Events  
WHERE (StartTime, Id) > (@LastStartTime, @LastId)  
ORDER BY StartTime, Id  
LIMIT 10

Использование композитного сравнения (StartTime, Id) необходимо, так как время старта может быть неуникальным. Id используется как "tie-breaker" (разрешитель коллизий).32

**Преимущества:**

* **Производительность $O(1)$:** Если поле индексировано (B-Tree), БД мгновенно находит нужную запись и берет следующие 10. Скорость не зависит от того, запрашиваем мы 1-ю или 1-миллионную страницу.34
* **Стабильность:** Вставка новых записей не сдвигает "окно" просмотра для уже загруженных данных.

### 4.3 Реализация в Entity Framework Core

Реализация Keyset пагинации требует аккуратного построения LINQ запроса, так как EF Core должен транслировать его в эффективный SQL.

C#

public async Task<List<CalendarEvent>> GetEventsAfterAsync(DateTimeOffset? lastTime, Guid? lastId, int pageSize)  
{  
 var query = \_context.Events.AsNoTracking();  
  
 if (lastTime.HasValue && lastId.HasValue)  
 {  
 // Эмуляция сравнения кортежей в SQL: (Time, Id) > (LastTime, LastId)  
 // Логика: Либо время строго больше, ЛИБО время такое же, но ID больше  
 query = query.Where(e => e.StartTime > lastTime |  
  
|   
 (e.StartTime == lastTime && e.Id.CompareTo(lastId.Value) > 0));  
 }  
  
 return await query  
 .OrderBy(e => e.StartTime)  
 .ThenBy(e => e.Id) // Обязательная детерминированная сортировка  
 .Take(pageSize)  
 .ToListAsync();  
}

Этот код гарантирует использование индекса и стабильную выдачу результатов.36 Клиенту возвращается не номер страницы, а nextCursor (зашифрованная строка, содержащая StartTime и Id последнего элемента), которую он передаст в следующем запросе.38

## 5. Продвинутое Моделирование: Повторяющиеся События (Recurring Events)

### 5.1 Проблема "Бесконечного" Календаря

Одной из сложнейших задач календарного сервиса является реализация повторяющихся событий (например, "Планерка каждый понедельник").

* **Наивный подход:** Создать записи в БД на следующие 50 лет.
  + **Недостатки:** Огромный расход места, невозможность изменить правило (перенести планерку на вторник потребует обновления тысяч записей), ограничение по времени.39
* **Правильный подход:** Хранить **Правило**, а не инстансы.

### 5.2 Стандарт RFC 5545 и RRULE

Индустрия использует стандарт iCalendar (RFC 5545), описывающий правила повторения строкой RRULE.

Пример: FREQ=WEEKLY;INTERVAL=1;BYDAY=MO (Еженедельно по понедельникам).40

В базе данных хранится одна запись "Master Event" с полем RecurrenceRule. Конкретные встречи (Instances) не существуют в БД как отдельные строки, они "виртуальны".

### 5.3 Алгоритм Экспансии (Expansion)

Для реализации эндпоинта GET /events, который показывает пользователю список дел на неделю, сервер должен динамически развернуть правила в список событий.

**Capstone Task (Итоговое задание):** Реализация "умного" эндпоинта.

1. **Вход:** Диапазон дат (например, 1-7 января).
2. **Шаг 1:** Выборка из БД обычных (одиночных) событий, попадающих в диапазон.
3. **Шаг 2:** Выборка "Мастер-событий", чьи правила *могут* пересекаться с диапазоном.
4. **Шаг 3:** Использование библиотеки Ical.Net для калькуляции. Мы "проецируем" правило RRULE на запрашиваемое окно времени.42  
   C#  
   var recurrence = new RecurrencePattern("FREQ=WEEKLY;BYDAY=MO");  
   var generator = new RecurrencePatternEvaluator(recurrence);  
   var instances = generator.Evaluate(masterEvent.StartTime, windowStart, windowEnd);
5. **Шаг 4 (Сложность):** Обработка Исключений (Exceptions). Если пользователь перенес *одну* встречу из серии, в БД создается запись-исключение с RecurrenceId, указывающим на дату оригинальной встречи. Алгоритм экспансии должен заменить виртуальный инстанс на это реальное исключение.43

### 5.4 Производительность и Лимиты

Экспансия — дорогая операция CPU. Google Calendar API, например, вводит ограничения (лимит на количество инстансов или обязательные timeMin/timeMax), чтобы предотвратить DoS-атаки через сверхсложные правила повторения.45

В вашем сервисе необходимо валидировать RRULE и ограничивать окно экспансии (например, не более 2-х месяцев за запрос).

## 6. Заключение

Пятый день седьмой недели знаменует переход к разработке систем корпоративного уровня.

1. **Время** перестало быть простым числом и стало сложной структурой DateTimeOffset с обязательной нормализацией в UTC.
2. **Надежность** обеспечивается не надеждой на стабильный Wi-Fi, а математически обоснованным механизмом Идемпотентности.
3. **Доступ к данным** оптимизирован через Keyset Pagination, обеспечивающую стабильную работу при любых объемах данных.
4. **Бизнес-логика** перешла от простых CRUD операций к сложным алгоритмам динамической генерации данных на основе стандартов RFC.

Эти паттерны являются фундаментом для построения масштабируемых, надежных и корректных бэкенд-систем, способных обслуживать миллионы пользователей в разных часовых поясах.

### Сравнительная Таблица Ключевых Решений

| **Компонент** | **Подход новичка (GameDev)** | **Промышленный стандарт (Enterprise)** | **Преимущество** |
| --- | --- | --- | --- |
| **Время** | DateTime.Now (Local) | DateTimeOffset + UTC | Защита от часовых поясов и DST |
| **Повторы (Сеть)** | Клиентский Retry | Idempotency Key + Кеш | Гарантия "Exactly-Once" (списания) |
| **Пагинация** | Skip/Take (Offset) | Keyset (Where ID > X) | $O(1)$ скорость, нет дублей при скролле |
| **Календарь** | Создание строк в БД | RRULE + Экспансия в памяти | Бесконечные повторы, экономия БД |
| **Тесты** | Thread.Sleep | TimeProvider (Mock) | Мгновенные, стабильные тесты |

#### Источники

1. План обучения REST API для бэкенда неделя 7
2. Functional Unity Architecture: A Developer's Guide | by Bruno Mikoski | Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@bada/functional-unity-architecture-a-developers-guide-359e5111c5c2>
3. When have you experienced time drift distributed systems related projects at work - Reddit, дата последнего обращения: декабря 5, 2025, <https://www.reddit.com/r/ExperiencedDevs/comments/1lywuz7/when_have_you_experienced_time_drift_distributed/>
4. DateTime.Now and DateTime.UtcNow - Code Maze, дата последнего обращения: декабря 5, 2025, <https://code-maze.com/csharp-datetime-now-vs-datetime-utcnow/>
5. 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/>
6. Recommended date format for REST GET API [closed] - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/9581692/recommended-date-format-for-rest-get-api>
7. Date/time localization - general - OpenBoxes, дата последнего обращения: декабря 5, 2025, <https://community.openboxes.com/t/date-time-localization/721>
8. c# - DateTime vs DateTimeOffset - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/4331189/datetime-vs-datetimeoffset>
9. Choose between DateTime, DateOnly, DateTimeOffset, TimeSpan, TimeOnly, and TimeZoneInfo - Microsoft Learn, дата последнего обращения: декабря 5, 2025, <https://learn.microsoft.com/en-us/dotnet/standard/datetime/choosing-between-datetime>
10. DateTimeOffset vs DateTime in C# - Code Maze, дата последнего обращения: декабря 5, 2025, <https://code-maze.com/csharp-datetimeoffset-vs-datetime/>
11. DateTime.Now vs. DateTime.UtcNow - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/62151/datetime-now-vs-datetime-utcnow>
12. How to use (and test) TimeProvider timers in .NET - Grant Winney, дата последнего обращения: декабря 5, 2025, <https://grantwinney.com/how-to-use-timeprovider-and-faketimeprovider-to-test-timers/>
13. What is the TimeProvider class - .NET - Microsoft Learn, дата последнего обращения: декабря 5, 2025, <https://learn.microsoft.com/en-us/dotnet/standard/datetime/timeprovider-overview>
14. Testing Time-Dependent Code With TimeProvider in .NET, дата последнего обращения: декабря 5, 2025, <https://code-maze.com/csharp-testing-time-dependent-code-with-timeprovider/>
15. Simplifying Time-Based Testing with `TimeProvider` in .NET 8 - DEV Community, дата последнего обращения: декабря 5, 2025, <https://dev.to/moh_moh701/using-timeprovider-for-easier-unit-testing-in-net-8-5fl3>
16. Designing robust and predictable APIs with idempotency - Stripe, дата последнего обращения: декабря 5, 2025, <https://stripe.com/blog/idempotency>
17. Idempotent requests | Stripe API Reference, дата последнего обращения: декабря 5, 2025, <https://docs.stripe.com/api/idempotent_requests>
18. Idempotency and concurrency in a REST web service, дата последнего обращения: декабря 5, 2025, <https://softwareengineering.stackexchange.com/questions/323543/idempotency-and-concurrency-in-a-rest-web-service>
19. Implementing Idempotent REST APIs in ASP.NET Core, дата последнего обращения: декабря 5, 2025, <https://www.milanjovanovic.tech/blog/implementing-idempotent-rest-apis-in-aspnetcore>
20. The Idempotency-Key HTTP Header Field - IETF, дата последнего обращения: декабря 5, 2025, <https://www.ietf.org/archive/id/draft-ietf-httpapi-idempotency-key-header-01.html>
21. Idempotency-Key header - HTTP - MDN Web Docs, дата последнего обращения: декабря 5, 2025, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Idempotency-Key>
22. 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>
23. Idempotency in REST APIs in ASP.NET Core | by Daniel Alabuja | Oct, 2025 - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@Alabuja/idempotency-in-rest-apis-in-asp-net-core-4a3f30cb8e94>
24. Cache in-memory in ASP.NET Core - Microsoft Learn, дата последнего обращения: декабря 5, 2025, <https://learn.microsoft.com/en-us/aspnet/core/performance/caching/memory?view=aspnetcore-10.0>
25. How to implement Idempotent key into an asp.net web api? - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/53039359/how-to-implement-idempotent-key-into-an-asp-net-web-api>
26. How to Design Idempotent APIs Safely: What to Cache and What to Ignore - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@mathildaduku/how-to-design-idempotent-apis-safely-what-to-cache-and-what-to-ignore-feb93a16fc00>
27. Idempotency Key Persistence, from now until forever? : r/softwarearchitecture - Reddit, дата последнего обращения: декабря 5, 2025, <https://www.reddit.com/r/softwarearchitecture/comments/1l8aakv/idempotency_key_persistence_from_now_until_forever/>
28. A Developer's Guide to API Pagination: Offset vs. Cursor-Based - Embedded Blog, дата последнего обращения: декабря 5, 2025, <https://embedded.gusto.com/blog/api-pagination/>
29. Limit/Offset Pagination vs. Cursor Pagination in MySQL - TiDB, дата последнего обращения: декабря 5, 2025, <https://www.pingcap.com/article/limit-offset-pagination-vs-cursor-pagination-in-mysql/>
30. Offset vs Where performance for pagination with index - Database Administrators Stack Exchange, дата последнего обращения: декабря 5, 2025, <https://dba.stackexchange.com/questions/261714/offset-vs-where-performance-for-pagination-with-index>
31. Keyset pagination: how it works, examples, and pros and cons - Merge.dev, дата последнего обращения: декабря 5, 2025, <https://www.merge.dev/blog/keyset-pagination>
32. Cursor-based vs. Offset Pagination for an Infinite Scroll Book Library – Which is Better?, дата последнего обращения: декабря 5, 2025, <https://www.reddit.com/r/dotnet/comments/1jxlu89/cursorbased_vs_offset_pagination_for_an_infinite/>
33. Pagination - EF Core - Microsoft Learn, дата последнего обращения: декабря 5, 2025, <https://learn.microsoft.com/en-us/ef/core/querying/pagination>
34. MongoDB Pagination: Offset-Based vs Keyset vs Pre-generated Result Pages - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/mongodb/mongodb-pagination-offset-based-vs-keyset-vs-pre-generated-result-pages-4177e05d88ec>
35. Keyset Cursors for Postgres Pagination: Fast, Accurate, Scalable - Stacksync, дата последнего обращения: декабря 5, 2025, <https://www.stacksync.com/blog/keyset-cursors-postgres-pagination-fast-accurate-scalable>
36. Keyset Pagination in Entity Framework Core for Efficient Data Retrieval - C# Corner, дата последнего обращения: декабря 5, 2025, <https://www.c-sharpcorner.com/article/keyset-pagination-in-entity-framework-core-for-efficient-data-retrieval/>
37. Keyset (Cursor) Pagination in .NET using EF Core - NikolaTech, дата последнего обращения: декабря 5, 2025, <https://www.nikolatech.net/blogs/keyset-cursor-pagination-with-efcore-in-aspnetcore>
38. 10 API pagination best practices in 2025 - Merge.dev, дата последнего обращения: декабря 5, 2025, <https://www.merge.dev/blog/api-pagination-best-practices>
39. Recurring Calendar Events — Database Design. | by loribean - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@aureliadotlim/recurring-calendar-events-database-design-dc872fb4f2b5>
40. RFC 5545 - Internet Calendaring and Scheduling Core Object Specification (iCalendar), дата последнего обращения: декабря 5, 2025, <https://datatracker.ietf.org/doc/html/rfc5545>
41. 4. iCalendar Object Examples, дата последнего обращения: декабря 5, 2025, <https://icalendar.org/iCalendar-RFC-5545/4-icalendar-object-examples.html>
42. Working with recurring elements · ical-org/ical.net Wiki - GitHub, дата последнего обращения: декабря 5, 2025, <https://github.com/ical-org/ical.net/wiki/Working-with-recurring-elements>
43. Calendar API v3 Best Practices: Recurring Events - Google Workspace Developers blog, дата последнего обращения: декабря 5, 2025, <https://gsuite-developers.googleblog.com/2011/12/calendar-v3-best-practices-recurring.html>
44. Data storage strategies - a balancing act – SQLServerCentral Forums, дата последнего обращения: декабря 5, 2025, <https://www.sqlservercentral.com/forums/topic/data-storage-strategies-a-balancing-act>
45. Google Calendar API - recurring event with max time limit - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/30599745/google-calendar-api-recurring-event-with-max-time-limit>
46. Create a recurring event - Computer - Google Calendar Help, дата последнего обращения: декабря 5, 2025, <https://support.google.com/calendar/answer/37115?hl=en&co=GENIE.Platform%3DDesktop>