# Архитектурная спецификация: Теория временного моделирования и сложной доменной логики в распределенных системах Enterprise-уровня (День 2)

## Аннотация

Данный документ представляет собой исчерпывающее теоретическое руководство, разработанное в рамках программы архитектурной трансформации специалистов из сферы GameDev (Unity/C#) в среду Enterprise Backend Development (.NET/PostgreSQL). Документ фокусируется на фундаментальных проблемах проектирования систем управления ресурсами, связанных с моделированием времени, обработкой рекуррентных событий (повторяющихся расписаний) и обеспечением целостности данных в условиях сложной бизнес-логики.

В отличие от детерминированных сред реального времени, характерных для игровых движков, корпоративные распределенные системы функционируют в условиях геополитической и темпоральной неопределенности. В документе детально рассматриваются ограничения стандартных подходов к хранению времени (UTC-центризм), математический аппарат стандарта RFC 5545 (iCalendar) для описания рекурсии, а также применение паттернов Domain-Driven Design (DDD) и специфических возможностей PostgreSQL для решения задач планирования.

## 1. Эпистемологический сдвиг: От игрового цикла к календарному времени

### 1.1. Фундаментальные различия моделей времени

Переход от разработки на Unity к проектированию Enterprise-бэкенда требует отказа от привычных ментальных моделей времени. В игровом движке время (Time) является непрерывной, монотонно возрастающей величиной, отсчитываемой от запуска приложения или уровня.

* **Unity Time:** $T\_{current} = T\_{start} + \sum \Delta t\_{frame}$. Время здесь — это физическая величина, используемая для интеграции уравнений движения. Оно линейно, предсказуемо и, как правило, не зависит от внешнего мира.1
* **Enterprise Time:** Календарное время — это социальный и политический конструкт. Оно дискретно, нелинейно (из-за високосных секунд и переходов на летнее время) и контекстно-зависимо.

Для разработчика, привыкшего оперировать Time.deltaTime, столкновение с концепцией "будущего времени" в бизнес-приложениях часто становится источником критических ошибок. Главная ошибка заключается в предположении, что физическое время (Instant) и календарное время (Wall Clock Time) изоморфны. В реальности между ними существует сложная функция преобразования, зависящая от географической локации и политических решений правительств, которые могут изменяться произвольно.3

### 1.2. Критика UTC-центризма для будущего планирования

В индустрии существует распространенная догма: "Всегда храни время в UTC, конвертируй в локальное на клиенте". Этот подход, безупречно работающий для **исторических данных** (логов, транзакций, аудита), является архитектурно ошибочным для **планирования будущих событий**.5

#### 1.2.1. Проблема стабильности смещения (Offset Stability)

UTC (Coordinated Universal Time) — это стандарт атомного времени. Локальное время определяется как $T\_{local} = T\_{UTC} + Offset$. Проблема заключается в том, что $Offset$ не является константой. Он является функцией от времени и локации: $Offset(t, Location)$.

Рассмотрим сценарий:

1. Пользователь в Нью-Йорке планирует встречу на "10:00 утра 1 января 2026 года".
2. Система, следуя наивной UTC-стратегии, вычисляет смещение для Нью-Йорка (EST, UTC-5) и сохраняет в базу данных метку 2026-01-01T15:00:00Z.
3. Через месяц Конгресс США принимает закон об отмене перевода часов и постоянном переходе на летнее время (EDT, UTC-4), что уже обсуждалось в законодательных инициативах.3
4. Наступает 1 января 2026 года. Смещение для Нью-Йорка теперь UTC-4.
5. Система берет сохраненное 15:00:00Z и конвертирует его обратно для пользователя: $15:00 - 4 = 11:00$.
6. **Результат:** Встреча сместилась на час вперед. Пользователь пропустил свое время.

#### 1.2.2. Концепция сохранения намерения (Intent Preservation)

В системах бронирования первичным является **намерение пользователя** (User Intent). Если пользователь сказал "10 утра", он имеет в виду положение стрелок на часах в конкретной комнате, вне зависимости от того, как это время соотносится с UTC.

Архитектурное решение:

Для будущих событий необходимо хранить кортеж:

1. **Локальное время (Local Date Time):** 2026-01-01T10:00:00 (без смещения).
2. **Идентификатор часового пояса (IANA Time Zone ID):** America/New\_York.6

Вычисление UTC-метки (Instant) должно производиться **Just-In-Time (JIT)** — в момент проверки доступности или отправки уведомления, используя актуальную на тот момент версию базы данных часовых поясов (tzdb).8 Это гарантирует, что любые изменения в законодательстве будут автоматически учтены системой без миграции данных.

| **Характеристика** | **Исторические события (Логи)** | **Будущие события (Расписание)** |
| --- | --- | --- |
| **Природа** | Факт, который уже свершился. | Намерение, которое должно свершиться. |
| **Инвариант** | Абсолютный момент времени (Instant). | Локальное время (Wall Time) + Локация. |
| **Влияние изменений TZ** | Нет (прошлое неизменно). | Да (правила могут измениться до наступления события). |
| **Рекомендуемое хранение** | DateTimeOffset (UTC) или long (Unix Timestamp). | LocalDateTime + TimeZoneId (Text). |
| **PostgreSQL тип** | timestamptz | timestamp + text |

### 1.3. Инструментарий.NET 8: TimeProvider и NodaTime

Стандартная структура System.DateTime в.NET исторически является источником ошибок из-за свойства Kind (Unspecified, Utc, Local), которое часто теряется при сериализации или арифметических операциях. Кроме того, DateTime не хранит информацию о часовом поясе, только смещение (в DateTimeOffset), чего недостаточно для будущих вычислений (смещение -4 может означать как America/New\_York, так и America/Santiago, у которых разные правила перехода на летнее время).9

#### 1.3.1. NodaTime: Типобезопасность времени

Библиотека NodaTime, разработанная Джоном Скитом (Jon Skeet), предлагает строгую систему типов, исключающую возможность семантических ошибок 8:

* **Instant:** Глобальное время (аналог UTC).
* **LocalDateTime:** Дата и время без привязки к смещению (то, что видит пользователь на настенных часах).
* **ZonedDateTime:** Полная комбинация (Instant + LocalDateTime + DateTimeZone).
* **Period:** Календарная длительность (например, "1 месяц"). Важно отличать от TimeSpan (физическая длительность), так как добавление "1 месяца" к дате может дать разное количество часов (28-31 день).

В рамках "Дня 2" мы рекомендуем использовать NodaTime для внутренней логики расчета расписаний, так как она принуждает разработчика явно обрабатывать такие граничные случаи, как "пропавший час" при переходе на летнее время или "дублирующийся час" при переходе на зимнее.

#### 1.3.2. Абстракция TimeProvider в.NET 8

Начиная с.NET 8, Microsoft представила абстрактный класс TimeProvider, решающий проблему тестируемости времени.12 В Unity для управления временем в тестах часто приходилось писать самодельные обертки над Time.time. В Enterprise-разработке внедрение TimeProvider позволяет:

* Зафиксировать "текущее" время в Unit-тестах.
* Эмулировать прохождение времени (например, смену DST) без ожидания реального времени.
* Управлять таймерами (ITimer) детерминированным образом.

Использование TimeProvider является обязательным стандартом для обеспечения тестируемости бизнес-логики бронирования.13

## 2. Теория Рекуррентности: Стандарт RFC 5545 и Алгоритмы Развертывания

Второй ключевой вызов модуля — переход от хранения сущностей к хранению правил. Если в простой системе одна встреча — это одна запись в БД, то в Enterprise-системе запись "Еженедельная планерка по понедельникам" — это бесконечный ряд событий, описываемый одной строкой.14

### 2.1. Грамматика RRULE (RFC 5545)

Индустриальным стандартом для описания повторяющихся событий является спецификация **iCalendar (RFC 5545)**. Понимание грамматики RRULE (Recurrence Rule) критически важно для проектирования базы данных.14

Типичная строка RRULE выглядит так:

FREQ=WEEKLY;INTERVAL=2;BYDAY=MO,WE;UNTIL=20251231T235959Z

#### Декомпозиция компонентов:

1. **FREQ (Frequency):** Базовая единица повторения. Допустимые значения: SECONDLY, MINUTELY, HOURLY, DAILY, WEEKLY, MONTHLY, YEARLY. В контексте бронирования переговорных наиболее актуальны DAILY, WEEKLY, MONTHLY.
2. **INTERVAL:** Множитель частоты. FREQ=WEEKLY;INTERVAL=2 означает "каждые две недели". По умолчанию равен 1.
3. **BY-Rules (Фильтры и Расширители):** Самая мощная часть спецификации.
   * BYDAY: Указывает дни недели. MO,TH (понедельник, четверг). Может содержать позиционный модификатор: 1FR (первая пятница месяца), -1MO (последний понедельник месяца).
   * BYMONTHDAY: Числа месяца (1, 15, -1 для последнего дня).
   * BYSETPOS: Выбирает n-й элемент из набора, сгенерированного предыдущими правилами.
4. **WKST (Week Start):** День начала недели. Критичен для расчета INTERVAL > 1 при еженедельных повторениях. В США это воскресенье (SU), в Европе — понедельник (MO).14
5. **Ограничители:**
   * COUNT: Количество повторений (конечный ряд).
   * UNTIL: Дата окончания (конечный ряд).
   * Если оба отсутствуют, ряд считается **бесконечным**.

**Важно:** Стандарт RFC 5545 описывает *генеративный алгоритм*. Правило само по себе не содержит дат, оно содержит *инструкцию* по их вычислению относительно начальной даты (DTSTART).16

### 2.2. Стратегии хранения и паттерн Master-Occurrence

Существует фундаментальная дилемма при проектировании схемы БД для рекуррентных событий: **Материализация** против **Вычисления на лету**.15

#### 2.2.1. Стратегия полной материализации

При создании повторяющейся встречи система генерирует N записей в таблице Bookings на 5 лет вперед.

* **Плюсы:**
  + Тривиальные SQL-запросы (SELECT \* FROM Bookings WHERE Date BETWEEN X AND Y).
  + Простая проверка конфликтов (Unique Constraint или Exclusion Constraint).
* **Минусы:**
  + **Взрыв данных:** Одна ежедневная встреча на 10 лет генерирует 3650 записей.
  + **Проблема бесконечности:** Невозможно поддержать бесконечные встречи.
  + **Сложность обновления:** Изменение правила ("перенести с 10:00 на 11:00") требует массового обновления (UPDATE) тысяч строк, что блокирует таблицу и создает нагрузку на WAL (Write Ahead Log) в PostgreSQL.15
  + **Дрифт часовых поясов:** При изменении правил DST нужно пересчитывать миллионы записей.

#### 2.2.2. Гибридная стратегия: Master-Exception (Рекомендуемая)

Мы храним только **определение правила** (Master) и **исключения** (Exceptions). "Виртуальные" встречи вычисляются в оперативной памяти приложения при запросе.18

**Структура данных:**

1. **Таблица RecurringBookings (Master):**
   * Id, RRule (Text), StartDate, StartTime, Duration, TimeZoneId.
2. **Таблица Bookings (Single Instances & Exceptions):**
   * Обычные одиночные встречи.
   * Исключения из рекурсии (перенесенные встречи).
   * Связь: RecurringBookingId (FK, nullable).

Алгоритм чтения расписания (The Expansion Pipeline):

Когда клиент запрашивает расписание на неделю $W$:

1. **Выборка:** Загрузить из БД:
   * Одиночные встречи, попадающие в $W$.
   * Рекуррентные встречи, которые *начались до конца $W$* и *не закончились до начала $W$* (проверка пересечения диапазонов).
   * Исключения для этих рекуррентных встреч.
2. **Экспансия (In-Memory):**
   * Для каждой рекуррентной встречи использовать парсер (например, Ical.Net) для генерации кандидатов-встреч внутри $W$.
   * Начальное состояние генератора устанавливается на DTSTART.20
3. **Применение исключений (Map/Reduce):**
   * Для каждого сгенерированного кандидата проверить наличие записи в таблице исключений (по ключу OriginalStartDate).
   * Если есть исключение типа "Cancelled" (EXDATE) — удалить кандидата.21
   * Если есть исключение типа "Modified" — заменить кандидата данными из исключения.
4. **Слияние:** Объединить список одиночных встреч и развернутых рекуррентных.

Этот подход обеспечивает $O(1)$ на запись (сохранение правила) и $O(N)$ на чтение (где N — количество правил, а не встреч), что оптимально для систем с преобладанием чтения (Read-Heavy), коими являются календари.

### 2.3. Обработка исключений (Exceptions) в iCalendar

Обработка пользовательского сценария "Я хочу перенести *только эту* встречу из серии на другой день" является наиболее сложной частью логики. В терминах RFC 5545 это реализуется через механизм **Recurrence-ID**.22

* **EXDATE (Exception Date):** Список дат, которые нужно *удалить* из генерации. В базе данных это реализуется либо как отдельная таблица BookingExceptions со статусом Cancelled, либо как массив в документе (если используется JSONB/NoSQL).23
* **Реализация в БД:** Для поддержки целостности данных рекомендуется использовать **реляционную таблицу исключений**, связанную внешним ключом с мастер-записью. Это позволяет использовать каскадное удаление (Cascade Delete) — если пользователь удаляет всю серию, исключения удаляются автоматически.24

## 3. Инженерия данных: PostgreSQL и темпоральная целостность

Выбор PostgreSQL обусловлен его продвинутой поддержкой геометрических и темпоральных типов данных, которые позволяют переложить часть сложной логики проверки пересечений с уровня приложения на уровень базы данных, гарантируя строгую консистентность (Strong Consistency).

### 3.1. Типы диапазонов (Range Types)

Вместо хранения start\_time и end\_time как двух отдельных колонок, мы используем тип tstzrange (timestamp with time zone range).

tstzrange хранит диапазон как единый математический объект, поддерживающий алгебру интервалов.25

Формат: `

SQL DDL для таблицы бронирований:

SQL

CREATE EXTENSION IF NOT EXISTS btree\_gist;  
  
CREATE TABLE bookings (  
 id UUID PRIMARY KEY,  
 room\_id UUID NOT NULL,  
 period tstzrange NOT NULL,  
 -- Другие поля...  
 EXCLUDE USING GIST (  
 room\_id WITH =, -- Запретить, если room\_id совпадают...  
 period WITH && --...И периоды пересекаются (&& - оператор пересечения)  
 )  
);

Этот механизм работает **атомарно** на уровне движка БД. Любая попытка вставки пересекающегося интервала вызовет мгновенное исключение 23P01 exclusion\_violation, которое легко перехватывается в коде C# и конвертируется в ошибку 409 Conflict.26

Важное теоретическое ограничение:

Exclusion Constraints работают только для материализованных строк. Они идеально защищают от накладок одиночных встреч и исключений. Однако они не могут защитить от пересечений с "виртуальными" рекуррентными встречами, которые хранятся как RRULE.

Следовательно, архитектура должна предусматривать гибридный подход:

1. **Optimistic Lock** (на уровне приложения) для проверки доступности по RRULE (с использованием Redis Distributed Lock — тема Дня 3).
2. **Hard Constraint** (на уровне БД) для всех физически существующих записей.30

## 4. Domain-Driven Design (DDD): Богатая модель предметной области

В противовес "анемичной модели" (Anemic Domain Model), где объекты являются лишь контейнерами данных (DTO), DDD требует инкапсуляции бизнес-правил внутри сущностей и объектов-значений.31

### 4.1. Value Objects: Инкапсуляция инвариантов

В системе бронирования существуют концепции, которые определяются своим значением, а не идентичностью.

#### 4.1.1. TimeSlot (Временной Слот)

Вместо передачи пары DateTime start, DateTime end, мы создаем Value Object TimeSlot.

Инварианты (правила валидации), проверяемые в конструкторе:

* Start < End.
* Duration > MinDuration (например, 15 минут).
* Start и End должны быть выровнены по сетке (например, кратно 15 минутам).

**Поведение:**

* Overlaps(TimeSlot other): Возвращает true, если слоты пересекаются.
* Contains(DateTime instant): Содержит ли слот данный момент.
* Gap(TimeSlot other): Возвращает слот-промежуток между двумя слотами.

#### 4.1.2. RecurrencePattern (Паттерн Рекурсии)

Value Object, оборачивающий строку RRULE.

Ответственность:

* Парсинг строки RRULE.
* Валидация (например, запрет SECONDLY частоты для встреч).
* Метод GetOccurrences(TimeSlot window), возвращающий IEnumerable<TimeSlot>.

Использование C# record (начиная с C# 9) идеально подходит для реализации Value Objects благодаря встроенной иммутабельности и структурному равенству.32

### 4.2. Агрегаты и Границы Транзакций

RecurringBooking выступает в роли Корня Агрегата (Aggregate Root).

Это означает, что никто (ни API, ни другой сервис) не имеет права создавать, удалять или изменять Исключение (BookingException) напрямую. Все операции должны проходить через корень.

Пример метода в агрегате:

C#

public void RescheduleOccurrence(DateTime originalStart, TimeSlot newSlot)  
{  
 // 1. Проверка: Действительно ли существует встреча в originalStart согласно RRULE?  
 if (!this.Pattern.OccursOn(originalStart))   
 throw new DomainException("Встреча не найдена в расписании");  
  
 // 2. Создание исключения  
 var exception = new BookingException(originalStart, newSlot);  
   
 // 3. Добавление во внутреннюю коллекцию (EF Core отследит это)  
 this.\_exceptions.Add(exception);  
   
 // 4. Регистрация доменного события  
 this.AddDomainEvent(new OccurrenceRescheduledEvent(this.Id, originalStart, newSlot));  
}

Этот подход гарантирует, что агрегат всегда находится в валидном состоянии.34

### 4.3. Доменные События и Outbox Pattern

Изменения в расписании (особенно рекуррентном) имеют побочные эффекты: отправка писем участникам, обновление поискового индекса, уведомление систем "Умного офиса".

В DDD эти эффекты моделируются через Domain Events.

Критически важный момент для "Дня 2": Атомарность.

Нельзя отправить письмо до того, как транзакция БД успешно зафиксирована. Нельзя отправить письмо после транзакции, если процесс упадет сразу после SaveChanges().

Решение: Transactional Outbox Pattern.

1. В рамках одной транзакции БД сохраняется состояние Агрегата (Bookings) И событие сериализуется в таблицу OutboxMessages.
2. Фоновый процесс (Dispatcher) читает OutboxMessages и отправляет их в брокер сообщений (RabbitMQ) или выполняет действия (Email).35

## 5. Entity Framework Core: Реализация Mapping

Приземление теории DDD на практику EF Core требует использования продвинутых техник маппинга, так как реляционная модель не всегда совпадает с объектной.

### 5.1. Complex Types (ранее Owned Types)

В.NET 8 введены **Complex Types**, которые идеально подходят для маппинга Value Objects. В отличие от Owned Types, они не требуют наличия скрытого ключа ID и концептуально ближе к "части" родительской таблицы.36

Конфигурация DbContext:

C#

builder.Entity<Booking>()  
 .ComplexProperty(b => b.TimeSlot, ts =>   
 {  
 ts.Property(p => p.Start).HasColumnName("start\_time");  
 ts.Property(p => p.End).HasColumnName("end\_time");  
 });

Это позволяет работать в коде с объектом booking.TimeSlot, в то время как в БД это будут плоские колонки той же таблицы.

### 5.2. Конвертация типов (Value Conversion)

Для хранения TimeZoneId (который в коде может быть представлен типом NodaTime.DateTimeZone или System.TimeZoneInfo) необходимо использовать Value Converters. EF Core позволяет прозрачно конвертировать сложный тип в строку при записи и обратно при чтении.37

Аналогично, RRULE хранится как строка, но в доменной модели представлен объектом RecurrencePattern.

C#

builder.Entity<RecurringBooking>()  
 .Property(e => e.Pattern)  
 .HasConversion(  
 v => v.ToString(), // В базу: RRule string  
 v => RecurrencePattern.Parse(v) // Из базы: Object  
 );

### 5.3. Перехватчики (Interceptors)

Для реализации аудита (кто и когда изменил встречу) и автоматической проверки инвариантов перед сохранением можно использовать SaveChangesInterceptor. Это мощный механизм EF Core, позволяющий вклиниться в процесс SaveChanges и выполнить валидацию или обогащение данных.35

## 6. Алгоритм Сервиса Доступности (Availability Service)

Финальным теоретическим блоком является алгоритм проверки доступности, который должен объединить все вышеописанные концепции.

**Входные данные:**

* ResourceId (Комната)
* Range (Период просмотра: Start, End)

**Шаги алгоритма:**

1. **Fetch (Извлечение):**
   * Загрузить "жесткие" брони (Single Bookings) внутри Range.
   * Загрузить "мягкие" брони (Recurring Bookings), чьи мастер-серии пересекаются с Range (даже если конкретные встречи не попадают). Оптимизация: использовать Tstzrange пересечение для фильтрации мастер-записей.39
   * Загрузить исключения для найденных рекурсий.
2. **Expansion (Развертывание):**
   * Инициализировать коллекцию occupiedSlots.
   * Добавить "жесткие" брони в occupiedSlots.
   * Для каждой рекуррентной серии:
     + Вызвать pattern.GetOccurrences(Range).
     + Для каждого вхождения проверить таблицу исключений.
     + Если исключение — отмена: пропустить.
     + Если исключение — перенос: пропустить (перенесенная встреча будет загружена как "жесткая" или как отдельная запись в шаге 1, в зависимости от реализации).
     + Остальные добавить в occupiedSlots.
3. **Subtraction (Вычитание):**
   * Создать полный временной слот Range.
   * "Вычесть" из него все occupiedSlots.
   * Результат: список availableSlots.
4. **Optimization (Кэширование):**
   * Поскольку операция Expansion вычислительно дорога (CPU-bound), результат для часто запрашиваемых недель кэшируется (Redis). Ключ кэша должен инвалидироваться при любом изменении (Create/Update/Delete) любой встречи в этой комнате (паттерн Cache-Aside, детально рассматриваемый в День 3).40

## Заключение

Теоретический базис второго дня обучения закладывает фундамент для построения систем Enterprise-уровня. Мы отказываемся от упрощенных моделей времени Unity в пользу геополитически корректного моделирования с использованием NodaTime и сохранения "намерения пользователя". Мы внедряем стандарт RFC 5545 для управления бесконечными сериями событий, используя гибридную модель хранения (Master-Exception), балансирующую между эффективностью записи и сложностью чтения. Мы применяем строгие паттерны DDD для защиты инвариантов и используем специфические возможности PostgreSQL (Exclusion Constraints) для гарантии целостности данных на самом низком уровне.

Реализация этих концепций требует высокой дисциплины кодирования и глубокого понимания механики работы используемых инструментов (.NET, EF Core, Postgres), что и является целью данного этапа трансформации.

#### Источники

1. Неделя 14
2. PICK GameDev VS Enterprise? - YouTube, дата последнего обращения: декабря 6, 2025, <https://www.youtube.com/watch?v=8CZ8khKdFKM>
3. Things Learned: Do Not Use UTC to Save Future Timestamps - Vishnu's Pages, дата последнего обращения: декабря 6, 2025, <https://iamvishnu.com/posts/do-not-use-utc-to-save-future-timestamps>
4. Storing times for human events - Simon Willison's Weblog, дата последнего обращения: декабря 6, 2025, <https://simonwillison.net/2024/Nov/27/storing-times-for-human-events/>
5. Handling Date and Time Formats Internationally: Best Practices for Global Software Development - Page One Formula, дата последнего обращения: декабря 6, 2025, <https://pageoneformula.com/handling-date-and-time-formats-internationally/>
6. dealing with UTC dates and the future - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/74753559/dealing-with-utc-dates-and-the-future>
7. What is an appropriate data type to store a timezone? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/13837258/what-is-an-appropriate-data-type-to-store-a-timezone>
8. How to do Recurring weekly events with timezones? - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/dotnet/comments/zycin0/how_to_do_recurring_weekly_events_with_timezones/>
9. Working with Date and Time in .NET (Updated for .NET 8) | by Softinbit | Dev Genius, дата последнего обращения: декабря 6, 2025, <https://blog.devgenius.io/working-with-date-and-time-in-net-updated-for-net-8-c66890e236db>
10. Not sure if this is a proper use case for using DateTime2 instead of DateTime? : r/dotnet, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/dotnet/comments/1afm312/not_sure_if_this_is_a_proper_use_case_for_using/>
11. DateTime vs DateTimeOffset · dotnet runtime · Discussion #108488 - GitHub, дата последнего обращения: декабря 6, 2025, <https://github.com/dotnet/runtime/discussions/108488>
12. What is the TimeProvider class - .NET - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/standard/datetime/timeprovider-overview>
13. TimeProvider — What's new in .NET 8? | by Stephen Atkinson | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@stephen.p.atkinson/timeprovider-whats-new-in-net-8-c3891875551c>
14. RFC 5545 - Internet Calendaring and Scheduling Core Object Specification (iCalendar), дата последнего обращения: декабря 6, 2025, <https://datatracker.ietf.org/doc/html/rfc5545>
15. Recurring Calendar Events — Database Design. | by loribean - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@aureliadotlim/recurring-calendar-events-database-design-dc872fb4f2b5>
16. 3.8.5.3. Recurrence Rule - iCalendar.org, дата последнего обращения: декабря 6, 2025, <https://icalendar.org/iCalendar-RFC-5545/3-8-5-3-recurrence-rule.html>
17. Storing vs Calculating Recurring Events - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/25710764/storing-vs-calculating-recurring-events>
18. Recurring Events Database Model - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/50204299/recurring-events-database-model>
19. Is there a best practice for recurring calendar events - Database Administrators Stack Exchange, дата последнего обращения: декабря 6, 2025, <https://dba.stackexchange.com/questions/91367/is-there-a-best-practice-for-recurring-calendar-events>
20. Working with recurring elements · ical-org/ical.net Wiki - GitHub, дата последнего обращения: декабря 6, 2025, <https://github.com/ical-org/ical.net/wiki/Working-with-recurring-elements>
21. RFC Errata Report » RFC Editor, дата последнего обращения: декабря 6, 2025, <https://www.rfc-editor.org/errata/rfc5545>
22. How do you specify an exception in an iCal feed? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/22669693/how-do-you-specify-an-exception-in-an-ical-feed>
23. iCalendar Recurrences - Introduction - CalConnect, дата последнего обращения: декабря 6, 2025, <https://devguide.calconnect.org/iCalendar-Topics/Recurrences/>
24. Exception dates in recurring events - IBM, дата последнего обращения: декабря 6, 2025, <https://www.ibm.com/docs/en/tap/5.0.0?topic=events-exception-dates-in-recurring>
25. Recurring Events and PostgreSQL - Thoughtbot, дата последнего обращения: декабря 6, 2025, <https://thoughtbot.com/blog/recurring-events-and-postgresql>
26. Documentation: 18: 8.17. Range Types - PostgreSQL, дата последнего обращения: декабря 6, 2025, <https://www.postgresql.org/docs/current/rangetypes.html>
27. Exclusion constraints in PostgreSQL and a tricky problem, дата последнего обращения: декабря 6, 2025, <https://www.cybertec-postgresql.com/en/exclusion-constraints-in-postgresql-and-a-tricky-problem/>
28. How Do I Manage Daylight Savings When Storing Datetimes in UTC? - Wappler Community, дата последнего обращения: декабря 6, 2025, <https://community.wappler.io/t/how-do-i-manage-daylight-savings-when-storing-datetimes-in-utc/53415>
29. Deep Dive into the Value Object Pattern in C#: Basics to Advanced | by Laks Tutor | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@lakstutor/deep-dive-into-the-value-object-pattern-in-c-basics-to-advanced-b058b49d8565>
30. DDD Value Objects as C# Records: The Missing Manual - DEV Community, дата последнего обращения: декабря 6, 2025, <https://dev.to/x789/ddd-value-objects-as-c-records-the-missing-manual-2fja>
31. Value Objects in .NET (DDD Fundamentals) - Milan Jovanović, дата последнего обращения: декабря 6, 2025, <https://www.milanjovanovic.tech/blog/value-objects-in-dotnet-ddd-fundamentals>
32. Creating Domain-Driven Design entity classes with Entity Framework Core, дата последнего обращения: декабря 6, 2025, <https://www.thereformedprogrammer.net/creating-domain-driven-design-entity-classes-with-entity-framework-core/>
33. Reliable Messaging in .NET: Domain Events and the Outbox Pattern with EF Core Interceptors - DEV Community, дата последнего обращения: декабря 6, 2025, <https://dev.to/stevsharp/reliable-messaging-in-net-domain-events-and-the-outbox-pattern-with-ef-core-interceptors-pjp>
34. Using EFCore to represent and store Value Objects without using Owned Types, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/75146382/using-efcore-to-represent-and-store-value-objects-without-using-owned-types>
35. Best Practices for Using Entity Framework Core in ASP.NET Core Applications with .NET 8, дата последнего обращения: декабря 6, 2025, <https://medium.com/@solomongetachew112/best-practices-for-using-entity-framework-core-in-asp-net-core-applications-with-net-8-9e4d796c02ac>
36. .NET events - EF Core | Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/ef/core/logging-events-diagnostics/events>
37. How to query recurring appointments stored as iCal RRULE in a database table?, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/39754097/how-to-query-recurring-appointments-stored-as-ical-rrule-in-a-database-table>
38. Handling Date/Time in Your Backend Applications - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@thebackendgrip/handling-date-time-in-your-backend-applications-d8c24c9df5e0>