# Архитектурная Трансформация: Интенсивный Протокол Перехода от Unity к Enterprise.NET Backend (Недели 19-20)

## 1. Стратегический анализ и контекст обучения

### 1.1. Смена парадигмы: От Игрового Цикла к Распределенным Транзакциям

Переход квалифицированного разработчика из среды Unity в экосистему Enterprise.NET Backend — это процесс, требующий фундаментальной деконструкции ментальных моделей. В мире GameDev инженер привык оперировать детерминированным игровым циклом (Update, FixedUpdate), где управление состоянием происходит в памяти, а время измеряется кадрами (frames). Вся вселенная игры, как правило, живет в адресном пространстве одного процесса, где доступ к данным мгновенен, а конкурентность ограничена основным потоком (Main Thread) и сопрограммами (Coroutines).1

В противовес этому, разработка высоконагруженного бэкенда требует мышления категориями асинхронных конвейеров (request-response pipelines), отсутствия состояния (statelessness) и распределенной конкурентности. Здесь "соседний объект" может находиться на другом сервере, доступ к нему требует сетевого вызова, который может завершиться тайм-аутом, а тысячи пользователей могут одновременно пытаться изменить одну и ту же запись в базе данных.

Данный отчет представляет собой исчерпывающее руководство, разработанное для финальной фазы обучения (Недели 19-20), сжатой в интенсивный 20-часовой спринт (Capstore Project). Цель — не просто написать код, а спроектировать и реализовать **Систему Бронирования Переговорных Комнат (Meeting Room Booking System)**. Эта задача выбрана не случайно: она является каноническим примером, заставляющим разработчика столкнуться с тремя главными демонами распределенных систем: Временем (планирование и часовые пояса), Состоянием (конкурентный доступ и блокировки) и Наблюдаемостью (метрики и мониторинг).1

### 1.2. Архитектурные Цели и Ограничения

Проектируемая система должна соответствовать строгим критериям качества, характерным для Enterprise-уровня. Мы отказываемся от упрощений "учебных проектов" в пользу промышленных стандартов.

| **Характеристика** | **Требование Unity (Mental Model)** | **Требование Enterprise Backend (Target Model)** | **Архитектурное Решение** |
| --- | --- | --- | --- |
| **Жизненный цикл** | Синхронный, Frame-based. | Асинхронный, I/O-bound. | Использование async/await для всех операций ввода-вывода, освобождение потоков ThreadPool.1 |
| **Безопасность** | Локальная или платформенная (SteamID). | Федеративная (OIDC/OAuth 2.0). | Внедрение **OpenIddict** для реализации Authorization Code Flow с PKCE.2 |
| **Целостность данных** | Однопоточная логика (Race conditions редки). | Высокая конкурентность (Race conditions неизбежны). | Распределенные блокировки через **Redis (Redlock)** для предотвращения double-booking.4 |
| **Масштабируемость** | Вертикальная (оптимизация CPU/GPU). | Горизонтальная (Stateless instances). | Вынос состояния сессии и блокировок в распределенный кэш (Redis).5 |
| **Наблюдаемость** | Visual Studio Debugger, Console.Log. | Telemetry, Metrics, Tracing. | Инструментация через **OpenTelemetry**, сбор метрик в **Prometheus**, визуализация в **Grafana**.6 |

## 2. Модуль 1: Инфраструктура Идентификации и Безопасности (День 1)

Временной бюджет: 4 часа

Фокус: OAuth 2.0, OpenID Connect, JWT, OpenIddict.

### 2.1. Теоретический Базис: Современная Архитектура Идентичности

Для разработчика Unity аутентификация часто является "черным ящиком" — вызовом метода SDK, который возвращает UserId. В Enterprise-разработке мы должны понимать механику этого процесса досконально. Мы переходим от монолитной аутентификации (где приложение само хранит и проверяет пароли) к федеративной, делегируя ответственность специализированному Сервису Идентификации (Identity Provider - IdP).

#### 2.1.1. Протоколы OAuth 2.0 и OpenID Connect (OIDC)

Важно различать эти два стандарта, которые часто путают.

* **OAuth 2.0** — это протокол *авторизации*. Он отвечает на вопрос "Что этот пользователь имеет право делать?". Это механизм делегирования доступа, работающий как "парковочный талон" (valet key): он дает ограниченный доступ к ресурсу (машине/API) без передачи ключей владельца (пароля).8
* **OpenID Connect** — это слой *аутентификации* поверх OAuth 2.0. Он отвечает на вопрос "Кто этот пользователь?". Он добавляет стандартизированный токен (ID Token), который содержит информацию о пользователе.2

В нашем проекте, где фронтендом может выступать SPA (Single Page Application) на React или даже клиент Unity, критически важно использовать **Authorization Code Flow с PKCE (Proof Key for Code Exchange)**. Классический Implicit Flow, часто упоминаемый в старых туториалах, признан небезопасным и устаревшим (deprecated). PKCE гарантирует, что даже если код авторизации будет перехвачен злоумышленником, он не сможет обменять его на токен доступа без секретного верификатора, который был сгенерирован клиентом и никогда не передавался по сети.9

#### 2.1.2. Выбор Провайдера: OpenIddict против IdentityServer

Исторически, стандартом де-факто в.NET был IdentityServer4. Однако его преемник, Duende IdentityServer, перешел на коммерческую лицензию, требующую оплаты для компаний с доходом выше 1 млн долларов.10 Хотя для учебных целей он может быть доступен, использование OpenIddict является более дальновидным выбором для open-source и портфолио-проектов.

OpenIddict — это гибкий фреймворк, который позволяет реализовать свой собственный OIDC-сервер на базе ASP.NET Core, полностью контролируя хранилище данных и UI. Он обеспечивает низкоуровневый контроль, что идеально подходит для образовательных целей, заставляя разработчика понимать каждый шаг протокола, а не полагаться на "магию" коробочных решений.3

### 2.2. План Практической Работы

Мы разделим систему на два микросервиса (или логических модуля в монолите):

1. **Auth Service:** Отвечает за хранение пользователей (ASP.NET Core Identity), выдачу токенов (OpenIddict) и управление сессиями.
2. **Booking Service:** Ресурсный сервер (Resource Server), который доверяет токенам, подписанным Auth Service, и выполняет бизнес-логику бронирования.

**Ключевая задача:** Настроить выпуск JWT (JSON Web Token), содержащих необходимые Claims (утверждения): sub (ID пользователя), email, role (Admin/User). Booking Service будет валидировать подпись токена, используя публичные ключи, загруженные через Discovery Endpoint (.well-known/openid-configuration).8

### 2.3. Расписание Дня 1

| **Время** | **Этап** | **Деятельность** | **Детали и Ресурсы** |
| --- | --- | --- | --- |
| **00:00 - 01:00** | **Теория** | Глубокое погружение в OAuth 2.0. | Изучение диаграмм sequence flow для Authorization Code + PKCE. Разбор структуры JWT (Header, Payload, Signature). Анализ 8 и.9 |
| **01:00 - 01:30** | **AI-Промптинг** | Архитектурная валидация с Gemini. | Использование промптов для генерации конфигурации OpenIddict (см. ниже). |
| **01:30 - 03:00** | **Практика** | Реализация Auth Service. | Настройка DbContext с OpenIddict, регистрация клиентов (ClientId, RedirectUri), включение потоков. Использование SQLite для локальной разработки. |
| **03:00 - 04:00** | **Интеграция** | Защита API. | Настройка JwtBearer аутентификации в Booking Service. Проверка доступа через Swagger/Postman с токеном и без. |

### 2.4. Протоколы Взаимодействия с AI (Промпты)

Для эффективного обучения мы используем AI как "Сеньор-Ментора". Не просите его "написать код", просите "объяснить решение" и "провести ревью".

**Промпт 1 (Конфигурация):**

"Ты — архитектор безопасности.NET. Я создаю сервер аутентификации на базе OpenIddict для системы бронирования. Мой клиент — это SPA на React (или Unity).

1. Объясни, почему мне необходимо использовать Authorization Code Flow с PKCE вместо Implicit Flow. В чем конкретно заключается уязвимость перехвата токена в браузере?
2. Сгенерируй пример конфигурации AddOpenIddict().AddServer(...) в Program.cs для ASP.NET Core 8, который включает этот поток, отключает шифрование токенов (для отладки JWT.io) и использует сертификаты разработчика.
3. Как мне настроить время жизни Access Token и Refresh Token для баланса между безопасностью и удобством?"

**Промпт 2 (Валидация):**

"Я настроил Resource Server (API Бронирования). Вот мой код настройки AddAuthentication().AddJwtBearer(...).

Проверь его на безопасность. Правильно ли я указал Authority? Что произойдет, если Auth Service будет недоступен при старте API? Как настроить валидацию Audience, чтобы мой токен не мог быть использован в других API?"

## 3. Модуль 2: Моделирование Времени и Сложная Доменная Логика (День 2)

Временной бюджет: 4 часа

Фокус: Time Zones, Recurring Events, EF Core, Domain Driven Design.

### 3.1. Теоретический Базис: Многомерность Времени

В Unity время линейно (Time.deltaTime). В Enterprise-системах время — это геополитическая конструкция, полная ловушек. Создание системы календаря кажется тривиальным, пока вы не столкнетесь с часовыми поясами и повторяющимися событиями.

#### 3.1.1. Дилемма UTC и Локального Времени

Распространенное заблуждение: "Всегда храни время в UTC". Это правило работает для *прошлых* событий (логов), но опасно для *будущих* событий, привязанных к локальному контексту.12

* **Сценарий:** Пользователь бронирует переговорку в Нью-Йорке на "Каждый понедельник в 9:00 утра" на год вперед.
* **Проблема:** Если вы сохраните это как 14:00 UTC, то после перехода США на летнее/зимнее время (DST), встреча сместится на 13:00 или 15:00 UTC, что останется 9:00 утра в Нью-Йорке, но если правила перехода изменятся (как было в 2007 году), конвертация UTC -> Local станет неверной.
* **Решение:** Для повторяющихся событий необходимо хранить **Локальное Время Стены (Wall Clock Time)** и **Идентификатор Часового Пояса IANA** (например, America/New\_York). Конвертация в UTC должна происходить "на лету" (Just-In-Time) при формировании ответа клиенту или при проверке конфликтов на конкретную дату.14

#### 3.1.2. Паттерны Хранения Повторяющихся Событий

Мы не можем создавать миллион записей в БД для события "Ежедневно без конца". Это неэффективно и делает редактирование серии ("Перенести все будущие встречи на час позже") невозможным.16

Оптимальная Схема (Pattern: Master-Occurrence):

1. **Таблица Events (Master):** Хранит определение серии. Поля: Id, RRule (стандарт RFC 5545, например FREQ=WEEKLY;BYDAY=MO), StartTime, Duration, TimeZone.
2. **Таблица Exceptions:** Хранит исключения из правила. Например, "встреча 25 декабря отменена" или "встреча 1 января перенесена". Поля: EventId, OriginalDate, NewDate, IsCancelled.
3. **Логика Экспансии:** При запросе "Дай мне расписание на неделю", сервер берет RRule, генерирует виртуальные слоты, накладывает исключения и выдает результат. Это требует использования библиотек для работы с iCal (например, Ical.Net).17

### 3.2. План Практической Работы

Разработка схемы базы данных (Code First) и сервиса экспансии.

**Сущности EF Core:**

* Room: Id, Name, Capacity.
* Booking: Id, RoomId, UserId, Title, StartTime, EndTime (для одиночных).
* RecurringBooking: Id, RoomId, UserId, RRule, TimeZoneId, Duration.
* BookingException: Id, RecurringBookingId, OriginalStart, IsCancelled.

### 3.3. Расписание Дня 2

| **Время** | **Этап** | **Деятельность** | **Детали и Ресурсы** |
| --- | --- | --- | --- |
| **00:00 - 01:00** | **Теория** | Работа со временем. | Изучение стандартов ISO 8601 и IANA Time Zone Database. Чтение 12 ("UTC is enough for everyone... right?"). Разбор формата RRULE. |
| **01:00 - 01:30** | **AI-Промптинг** | Проектирование Схемы. | Генерация и критика ER-диаграммы для поддержки рекуррентных событий. |
| **01:30 - 03:00** | **Практика** | Реализация EF Core. | Создание миграций PostgreSQL. Написание сервиса AvailabilityService, который принимает диапазон дат и возвращает занятые слоты, разворачивая RRULE. |
| **03:00 - 04:00** | **Тестирование** | Unit-тесты логики времени. | Тестирование граничных случаев: переход через полночь, високосные годы, смена DST. |

### 3.4. Протоколы Взаимодействия с AI (Промпты)

**Промпт 1 (Доменное Моделирование):**

"Ты — архитектор баз данных. Я проектирую систему бронирования. Мне нужно хранить повторяющиеся встречи (как в Google Calendar).

1. Предложи схему таблицы PostgreSQL для хранения паттерна повторения (RRULE).
2. Как мне эффективно выполнить SQL-запрос 'Найти все встречи в комнате А на следующую неделю', если у меня есть событие, которое повторяется каждый вторник уже 5 лет? Мне нужно генерировать даты в памяти приложения или можно это сделать в БД?
3. Объясни, как обрабатывать исключения (например, одну встречу из серии удалили). Покажи пример сущностей на C#."

**Промпт 2 (Работа с UTC):**

"У меня есть событие: каждый день в 10:00 утра по Нью-Йорку.

Если я сохраню его просто как 15:00 UTC (зимой), то летом оно станет 11:00 утра по Нью-Йорку, что неверно.

Покажи код на C#, который берет '10:00', 'America/New\_York' и текущую дату, и возвращает правильный DateTimeOffset в UTC для сохранения в базу конкретного инстанса встречи."

## 4. Модуль 3: Распределенная Конкурентность и Redis (День 3)

Временной бюджет: 4 часа

Фокус: Distributed Locks, Race Conditions, Redis, Cache-Aside Pattern.

### 4.1. Теоретический Базис: Управление Состоянием в Распределенной Среде

В Unity состояние (например, позиция игрока) часто защищено тем, что код выполняется в одном потоке. В веб-API тысячи запросов могут прийти одновременно. Классическая проблема "Double Booking":

1. Пользователь А проверяет: "Комната свободна?" -> Да.
2. Пользователь Б проверяет: "Комната свободна?" -> Да.
3. Пользователь А бронирует.
4. Пользователь Б бронирует.  
   В итоге — коллизия. Обычный lock в C# не поможет, если у вас запущено 3 экземпляра сервиса в Docker/Kubernetes, так как у них разная память.19

#### 4.1.1. Распределенные Блокировки (Distributed Locks)

Нам нужен механизм синхронизации, внешний по отношению к приложению. Redis идеально подходит для этого благодаря своей скорости и атомарности операций.

Алгоритм Redlock (или упрощенная версия для одного инстанса Redis) использует команду SET resource\_name my\_random\_value NX PX 30000:

* NX: Записать только если ключа еще нет (атомарная проверка и запись).
* PX: Установить срок жизни (TTL), чтобы в случае падения сервиса блокировка сама снялась через 30 секунд (предотвращение Deadlock).4

Важно понимать ограничения: Redlock не дает 100% гарантии в условиях экстремальных сетевых задержек (проблема "Split Brain"), но для системы бронирования комнат этого более чем достаточно по сравнению с блокировками на уровне БД.19

#### 4.1.2. Паттерн Кэширования: Cache-Aside

Постоянные запросы к PostgreSQL для вычисления расписания (особенно с развертыванием RRULE) дороги. Мы должны кэшировать "вид занятости" на неделю.

Алгоритм:

1. При запросе GET: Проверь Redis. Если есть данные -> верни.
2. Если нет -> Выполни тяжелый запрос к БД -> Сохрани в Redis (на 10 минут) -> Верни.
3. При создании/удалении брони (POST/DELETE): **Инвалидируй** (удали) ключ кэша для этой комнаты. Это гарантирует консистентность.5

### 4.2. План Практической Работы

Интеграция Redis через Docker Compose. Использование библиотек StackExchange.Redis и RedLock.net.

### 4.3. Расписание Дня 3

| **Время** | **Этап** | **Деятельность** | **Детали и Ресурсы** |
| --- | --- | --- | --- |
| **00:00 - 01:00** | **Теория** | Конкурентность. | Различие между Pessimistic (Блокировки) и Optimistic (Versioning) Concurrency.22 Изучение алгоритма Redlock.4 |
| **01:00 - 01:30** | **AI-Промптинг** | Анализ Race Conditions. | Моделирование сценария атаки на бронирование и выбор стратегии защиты. |
| **01:30 - 03:00** | **Практика** | Реализация Distributed Lock. | Внедрение RedLock.net. Обертывание метода CreateBooking в блок using (await redLock.CreateLockAsync(...)). |
| **03:00 - 04:00** | **Оптимизация** | Кэширование. | Реализация паттерна Cache-Aside для метода GetWeeklySchedule. Настройка сериализации (System.Text.Json или MessagePack). |

### 4.4. Протоколы Взаимодействия с AI (Промпты)

**Промпт 1 (Распределенные Блокировки):**

"Ты — Senior Backend Developer. Я реализую метод бронирования переговорки.

1. Объясни, почему использование lock(object) в C# бесполезно, если мое API запущено в нескольких Docker-контейнерах.
2. Напиши пример кода на C# с использованием библиотеки RedLock.net. Логика: Попытаться взять лок на ключ lock:room:{id}. Если успешно — проверить базу на пересечения и сохранить бронь. Если лок занят — вернуть 409 Conflict.
3. Что произойдет, если мой сервер упадет (крашнется) сразу после взятия лока, но до его освобождения? Как TTL решает эту проблему?"

**Промпт 2 (Стратегии Кэширования):**

"Я кэширую расписание комнаты в Redis. Какой подход лучше при создании новой брони:

А) Обновить значение в кэше (Write-Through/Update).

Б) Удалить значение из кэша (Cache Invalidation).

Объясни плюсы и минусы, учитывая риск Race Conditions."

## 5. Модуль 4: Наблюдаемость, Метрики и OpenTelemetry (День 4)

Временной бюджет: 4 часа

Фокус: Observability, Prometheus, Grafana, RED Method, Metrics API.

### 5.1. Теоретический Базис: Отладка vs Мониторинг

Разработчики Unity привыкли "дебажить" глазами или логами. В Enterprise это невозможно. Вы не можете подключить дебаггер к продакшену. Вы должны "видеть" систему через телеметрию. Мы переходим к концепции **Observability** (Наблюдаемость).6

#### 5.1.1. Три Столпа Наблюдаемости

1. **Logs (Логи):** Детальная запись событий (текст). "Что произошло?".
2. **Metrics (Метрики):** Агрегированные числовые данные во времени. "Как система себя чувствует?". (Например, CPU usage, Requests per Second).
3. **Traces (Трассировка):** Путь запроса через микросервисы. "Где тормозит?".

Мы сфокусируемся на **Метриках** и стандарте **OpenTelemetry (OTEL)**. Это вендор-нейтральный стандарт сбора телеметрии. Вместо того, чтобы писать код под конкретный мониторинг (Datadog, NewRelic), мы пишем под OTEL, а он уже экспортирует данные куда угодно (в нашем случае — в Prometheus).23

#### 5.1.2. Метод RED

Для построения дашбордов мы используем метод RED:

* **R (Rate):** Количество запросов в секунду.
* **E (Errors):** Количество ошибок (HTTP 500/4xx).
* **D (Duration):** Длительность обработки (Latency). Нас интересуют не средние значения, а перцентили (P95, P99), так как "среднее" скрывает проблемы.6

### 5.2. План Практической Работы

Инструментация приложения с использованием.NET 8 System.Diagnostics.Metrics. Настройка экспортера Prometheus. Поднятие инфраструктуры мониторинга (Prometheus + Grafana) в Docker Compose.

### 5.3. Расписание Дня 4

| **Время** | **Этап** | **Деятельность** | **Детали и Ресурсы** |
| --- | --- | --- | --- |
| **00:00 - 01:00** | **Теория** | Введение в OpenTelemetry. | Архитектура: App -> OTel Exporter -> Prometheus (Scrape) -> Grafana (Query). Типы метрик: Counter, Histogram, Gauge.7 |
| **01:00 - 01:30** | **AI-Промптинг** | Настройка Инфраструктуры. | Создание конфигураций prometheus.yml и дашбордов Grafana. |
| **01:30 - 03:00** | **Практика** | Инструментация C# кода. | Создание Meter. Добавление счетчика bookings\_created\_total. Добавление гистограммы booking\_processing\_duration\_ms. Настройка endpoint /metrics. |
| **03:00 - 04:00** | **Визуализация** | Grafana Dashboard. | Импорт данных в Grafana. Построение графиков RPS и Latency P95. Настройка алертинга (визуально).24 |

### 5.4. Протоколы Взаимодействия с AI (Промпты)

**Промпт 1 (Инструментация):**

"Ты — DevOps инженер. Я использую.NET 8.

1. Покажи, как зарегистрировать OpenTelemetry метрики в Program.cs, чтобы они экспортировались в формате Prometheus.
2. Напиши код Middleware, который автоматически замеряет время выполнения всех HTTP-запросов и записывает их в Гистограмму с метками (tags) method (GET/POST) и status\_code (200, 500).
3. Почему для измерения латентности (задержки) нужно использовать Гистограмму, а не просто записывать среднее значение?"

**Промпт 2 (Grafana/PromQL):**

"У меня есть метрика http\_request\_duration\_seconds\_bucket (гистограмма).

Напиши запрос на языке PromQL, который покажет 95-й перцентиль (P95) времени ответа за последние 5 минут для моего API. Объясни, как работает функция histogram\_quantile."

## 6. Модуль 5: Резистентность, Нагрузочное Тестирование и Защита (День 5)

Временной бюджет: 4 часа

Фокус: NBomber, Idempotency, Load Testing, Final Report.

### 6.1. Теоретический Базис: Доказательство Надежности

Система не работает, пока вы не доказали это под нагрузкой. "Работает на моем компьютере" — это фраза джуниора. Сеньор показывает отчет NBomber.

Мы будем симулировать Race Condition Attack: 50 виртуальных пользователей пытаются забронировать одну и ту же комнату в одну и ту же миллисекунду.

#### 6.1.1. Идемпотентность (Idempotency)

В распределенной системе сеть ненадежна. Клиент может отправить POST-запрос на создание брони, сервер его обработает, но ответ потеряется. Клиент (или библиотека retry policy) повторит запрос. Без защиты сервер создаст вторую бронь (или попытается).

Решение: Идемпотентность. Клиент генерирует уникальный ключ (Idempotency-Key header) для каждого действия. Сервер сохраняет этот ключ и результат операции в Redis. При повторном запросе с тем же ключом сервер просто возвращает сохраненный результат, не выполняя логику заново.26

#### 6.1.2. Нагрузочное Тестирование (NBomber)

Мы используем **NBomber** — фреймворк на C#, который позволяет писать нагрузочные тесты как код. Это позволяет переиспользовать DTO и логику, а также легко интегрировать тесты в CI/CD.28

### 6.2. План Практической Работы

Реализация Middleware идемпотентности. Написание консольного приложения для атаки на API. Формирование финального отчета.

### 6.3. Расписание Дня 5

| **Время** | **Этап** | **Деятельность** | **Детали и Ресурсы** |
| --- | --- | --- | --- |
| **00:00 - 01:00** | **Практика** | Idempotency Middleware. | Реализация проверки заголовка Idempotency-Key. Кэширование ответов (Status Code + Body) в Redis на 24 часа. |
| **01:00 - 02:00** | **Нагрузка** | Сценарий NBomber. | Написание сценария "The Thundering Herd": 50 потоков атакуют endpoint POST /bookings. Ожидаемый результат: 1 успех (200), 49 конфликтов (409).30 |
| **02:00 - 03:00** | **Анализ** | Валидация метрик. | Запуск теста и наблюдение за Grafana в реальном времени. Проверка, что P99 Latency не улетает в бесконечность, а CPU не достигает 100%. |
| **03:00 - 04:00** | **Итог** | Сборка Документации. | Оформление результатов в Google Doc (структура ниже). Подготовка к "виртуальной защите" (саморефлексия). |

### 6.4. Протоколы Взаимодействия с AI (Промпты)

**Промпт 1 (Идемпотентность):**

"Ты — Solution Architect. Мне нужно реализовать идемпотентность для POST-запросов в API.

1. Напиши Middleware на C#, который:
   * Проверяет наличие заголовка Idempotency-Key.
   * Проверяет Redis: если ключ есть, возвращает сохраненный результат (Json).
   * Если ключа нет: перехватывает поток ответа (Response Body), сохраняет его в Redis после выполнения контроллера, и отдает клиенту.
2. Какие есть подводные камни при кэшировании HttpResponse (например, потоковая передача)?"

**Промпт 2 (Нагрузочное Тестирование):**

"Помоги написать сценарий для NBomber v5 на C#.

Задача: Отправить 100 параллельных запросов на http://localhost:5000/api/bookings. Тело запроса одинаковое (одна и та же комната/время).

Я хочу получить статистику: сколько вернулось 200 OK, сколько 409 Conflict.

Как настроить LoadSimulation, чтобы все запросы ушли максимально одновременно (синхронный старт)?"

## 7. Итоговое Задание и Структура Google Doc

Результатом недели является не просто код, а "Artifact of Knowledge" — структурированный документ, доказывающий вашу компетентность.

### 7.1. Структура Отчета (Шаблон для Google Doc)

**Заголовок:** Enterprise Meeting Room Booking System — Architecture Report

**1. Executive Summary**

* Краткое описание системы и используемых технологий (Stack:.NET 8, PostgreSQL, Redis, OpenIddict, Prometheus).
* Ссылка на GitHub репозиторий.

**2. Архитектурные Решения (Architecture Decision Records - ADR)**

* **Identity:** Почему выбран OpenIddict и PKCE? (Обоснование безопасности).
* **Concurrency:** Как решена проблема Double Booking? (Redlock, диаграмма последовательности).
* **Time:** Как система работает с часовыми поясами и RRULE?

**3. Анализ Производительности (Performance & Reliability)**

* **Load Test Report:** Скриншоты отчета NBomber.
  + *Total Requests:* 1000
  + *RPS:* 500
  + *Successful Bookings:* 1 (Correctness check).
  + *Failed (Conflict):* 999.
* **Observability:** Скриншоты дашборда Grafana под нагрузкой. Анализ "узких мест" (Bottlenecks).

**4. Code Highlights**

* Фрагменты кода с объяснением:
  + Метод расширения рекурсии (RecurringBookingService).
  + Middleware идемпотентности.
  + Настройка распределенного локf.

### 7.2. Критерии Самопроверки (Definition of Done)

Перед завершением недели убедитесь, что вы можете утвердительно ответить на вопросы "Defense Checklist":

1. [ ] **Security:** Могу ли я украсть токен доступа, если перехвачу трафик? (Нет, благодаря HTTPS и PKCE).
2. [ ] **Consistency:** Если я запущу тест NBomber дважды подряд, останется ли база данных в валидном состоянии?
3. [ ] **Observability:** Если БД начнет тормозить, увижу ли я это на графиках до того, как упадет API?
4. [ ] **Recoverability:** Если Redis упадет, перестанет ли работать вся система или только создание брони? (Graceful Degradation).

Этот протокол превращает хаотичные знания в структурированный инженерный опыт, полностью соответствуя уровню Middle+ Backend Developer.

Использованные источники:

1

#### Источники

1. Разработка бэкенда: план обучения C#
2. Community OSS authentication options for ASP.NET Core | Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/community?view=aspnetcore-10.0>
3. IdentityServer4 vs AspNet.Security.OpenIdConnect.Server vs OpenIddict - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/46608920/identityserver4-vs-aspnet-security-openidconnect-server-vs-openiddict>
4. Distributed Locks with Redis | Docs, дата последнего обращения: декабря 6, 2025, <https://redis.io/docs/latest/develop/clients/patterns/distributed-locks/>
5. Caching patterns - Database Caching Strategies Using Redis - AWS Documentation, дата последнего обращения: декабря 6, 2025, <https://docs.aws.amazon.com/whitepapers/latest/database-caching-strategies-using-redis/caching-patterns.html>
6. ASP.NET Core metrics - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/core/log-mon/metrics/metrics?view=aspnetcore-10.0>
7. Example: Use OpenTelemetry with Prometheus, Grafana, and Jaeger - .NET, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/dotnet/core/diagnostics/observability-prgrja-example>
8. Securing Modern .NET Core Apps. OAuth 2.0 | by Soulaiman Ghanem | Code Factory Berlin, дата последнего обращения: декабря 6, 2025, <https://medium.com/code-factory-berlin/securing-modern-net-core-apps-bd18dafe24f8>
9. Interactive Applications With ASP.NET Core - Duende Software Docs, дата последнего обращения: декабря 6, 2025, <https://docs.duendesoftware.com/identityserver/quickstarts/2-interactive/>
10. Understanding IdentityServer4's Removal and Its Alternatives - FeatBit, дата последнего обращения: декабря 6, 2025, <https://www.featbit.co/articles2025/identityserver4-removed-github-alternatives>
11. IdentityServer 4 or OpenIddict? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/42611111/identityserver-4-or-openiddict>
12. UTC is Enough for Everyone, Right? - Zach Holman, дата последнего обращения: декабря 6, 2025, <https://zachholman.com/talk/utc-is-enough-for-everyone-right>
13. Best practices for timestamps and time zones in databases - Tinybird, дата последнего обращения: декабря 6, 2025, <https://www.tinybird.co/blog/database-timestamps-timezones>
14. How should I handle timezones in this context? [duplicate] - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/78024890/how-should-i-handle-timezones-in-this-context>
15. Handling Timezones within Enterprise-Level Applications: UTC vs Local Time - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@20011002nimeth/handling-timezones-within-enterprise-level-applications-utc-vs-local-time-309cbe438eaf>
16. What's the best way to model recurring events in a calendar application? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/85699/whats-the-best-way-to-model-recurring-events-in-a-calendar-application>
17. Recurring Calendar Events — Database Design. | by loribean - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@aureliadotlim/recurring-calendar-events-database-design-dc872fb4f2b5>
18. 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>
19. Why Do You Need Distributed Locking in ASP.NET Core | ABP.IO, дата последнего обращения: декабря 6, 2025, <https://abp.io/community/articles/why-do-you-need-distributed-locking-in-asp.net-core-fx1895hh>
20. Distributed Locking in .NET: Coordinating Work Across Multiple Instances, дата последнего обращения: декабря 6, 2025, <https://www.milanjovanovic.tech/blog/distributed-locking-in-dotnet-coordinating-work-across-multiple-instances>
21. How to do distributed locking - Martin Kleppmann, дата последнего обращения: декабря 6, 2025, <https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html>
22. Handling Concurrency Conflicts - EF Core - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/ef/core/saving/concurrency>
23. OpenTelemetry best practices: A user's guide to getting started with OpenTelemetry | Grafana Labs, дата последнего обращения: декабря 6, 2025, <https://grafana.com/blog/2023/12/18/opentelemetry-best-practices-a-users-guide-to-getting-started-with-opentelemetry/>
24. .NET 8 API with Prometheus and Grafana | by Fauly Coelho - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@faulycoelho/net-8-api-with-prometheus-and-grafana-29003adafd43>
25. .NET Observability with OpenTelemetry – Part 1: Metrics using Amazon Managed Prometheus and Grafana | .NET on AWS Blog, дата последнего обращения: декабря 6, 2025, <https://aws.amazon.com/blogs/dotnet/net-observability-with-opentelemetry-part-1-metrics-using-amazon-prometheus-and-amazon-grafana/>
26. Implement idempotent APIs in ASP.NET Core - C# Corner, дата последнего обращения: декабря 6, 2025, <https://www.c-sharpcorner.com/article/implement-idempotent-apis-in-asp-net-core/>
27. Idempotency in REST APIs in ASP.NET Core | by Daniel Alabuja | Oct, 2025 - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@Alabuja/idempotency-in-rest-apis-in-asp-net-core-4a3f30cb8e94>
28. Hello World Tutorial - NBomber, дата последнего обращения: декабря 6, 2025, <https://nbomber.com/docs/getting-started/hello_world_tutorial/>
29. Load Simulation | NBomber, дата последнего обращения: декабря 6, 2025, <https://nbomber.com/docs/nbomber/load-simulation/>
30. Load Testing HTTP API on C# with NBomber | by KostashOlena | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@OlenaKostash/load-testing-http-api-on-c-with-nbomber-96939511bdab>