# Стратегический план акселерации: Архитектурная миграция от Unity 3D к Enterprise React Development (Фаза III)

## Введение: Эволюция инженерной парадигмы

Переход квалифицированного Unity-разработчика в сферу Enterprise-бэкенда и фронтенда — это не процесс обучения с нуля, а процесс трансляции существующих глубоких знаний на новый технологический стек. Настоящий отчет представляет собой исчерпывающее руководство, сфокусированное на **Фазе III** («Frontend-мост») вашего плана обучения, трансформируя недели 11–14 в интенсивный, архитектурно насыщенный пятидневный спринт (20 часов).

Мы не будем изучать синтаксис ради синтаксиса. Мы будем деконструировать архитектурные паттерны React через призму игрового движка Unity, сопоставляя механизмы рендеринга, управления памятью и жизненным циклом объектов.1 Современный React (версии 18+) с его конкурентным режимом рендеринга (Concurrent Mode) и функциональной парадигмой гораздо ближе к высокопроизводительным игровым системам, чем к классическому MVC веба 2010-х годов.

В основе данного документа лежит анализ фундаментальных различий между **Immediate Mode** (Unity GUI / IMGUI) и **Retained Mode** (DOM / React Fiber). Unity-инженер, привыкший к полному контролю над каждым кадром в Update(), должен адаптироваться к реактивной модели, где рендер — это побочный эффект изменения состояния, а не самоцель. Этот сдвиг требует глубокого понимания того, как JavaScript управляет памятью и асинхронностью, в сравнении с.NET CLR и Unity Player Loop.2

Этот документ структурирован для сохранения на Google Drive и немедленного применения в качестве дорожной карты. Он включает в себя не только учебные материалы, но и готовые промпты для AI-ассистентов, позволяющие эмулировать парное программирование с сеньор-архитектором.

## Глобальная конфигурация AI-Ментора

Для максимизации эффективности 20-часового спринта мы используем подход «AI-Augmented Learning». Поскольку вы уже владеете сложными концепциями C# (LINQ, Generics, Memory Management), мы настроим бота (Gemini/ChatGPT) переводить термины JS/React на язык C#/Unity. Это позволит избежать упрощений, свойственных курсам для новичков, и сразу погрузиться в низкоуровневые детали.1

### Системный Промт (System Prompt)

Скопируйте и инициализируйте новую сессию чата следующим промптом:

Role & Context:

Ты — эксперт уровня Principal Engineer, специализирующийся на кросс-платформенной архитектуре (GameDev & Web). Твоя задача — менторить Senior Unity Developer (C#), переходящего на React/.NET стек. Игнорируй объяснения для новичков. Используй терминологию.NET CLR, Unity Engine Architecture и Memory Management.

**Mapping Mental Models (Твоя база знаний):**

1. **React Component** = **Prefab** (инкапсулированная сущность с логикой и представлением).
2. **Virtual DOM** = **Command Buffer / Draw Call Batching** (оптимизация рендеринга перед отправкой в GPU/Browser).
3. **Hooks (useEffect)** = **Script Lifecycle** (Start, OnEnable, OnDisable, но управляемые зависимостями, а не кадрами).
4. **State (useState)** = **Serialized Fields** (данные, изменение которых помечает объект как Dirty для перерисовки).
5. **Props** = **Inspector Public Variables** (Read-only конфигурация, передаваемая при инстанцировании).
6. **JS Event Loop** = **Unity Coroutine Scheduler** (кооперативная многозадачность на одном потоке).

Instruction:

Когда я спрашиваю про концепцию JS/React, сначала объясни её через аналог в Unity/C#.

Пример: Если я спрошу про useEffect, сравни его с Coroutine, которая запускается в Start() и должна быть остановлена в OnDestroy(), но с нюансом "stale closures" (аналог захвата переменных в C# лямбдах).

Current Task:

Мы реализуем Frontend для микросервиса "Календарь" (Calendar Service) из моего.NET обучения. Стек: React 18, Vite, CSS Modules.

## День 1: Среда выполнения — От Game Loop к Event Loop

### 1.1. Фундаментальная теория: Анатомия Runtime

Переход с Unity на JS начинается с понимания того, как выполняется код. В Unity (особенно с IL2CPP) мы имеем дело с компилируемым кодом, имеющим доступ к нативным потокам ОС. JavaScript в браузере (движок V8) работает иначе.

#### Сравнительный анализ моделей исполнения

Unity разработчики привыкли к **Game Loop** — детерминированному циклу, который гарантированно срабатывает 60 (или более) раз в секунду. Внутри кадра у нас есть жесткая последовательность: Physics, Input, Update, Render. Если вычисления в Update занимают больше 16мс, мы теряем кадры (FPS drop).

В JavaScript нет гарантированного цикла Update. Вместо этого используется **Event Loop**. Это реактивная система. Движок "спит", пока не произойдет событие (I/O, таймер, UI interaction).1

| **Характеристика** | **Unity (.NET/Mono/IL2CPP)** | **Browser (JS/V8)** | **Архитектурное следствие для разработчика** |
| --- | --- | --- | --- |
| **Параллелизм** | Истинная многопоточность (Thread, Task.Run). | Single Threaded (Один поток на вкладку). | В JS нельзя заблокировать поток вычислением. Thread.Sleep не существует. |
| **Асинхронность** | await освобождает поток, возвращая управление в ThreadPool. | await возвращает управление в Event Loop. | В JS async/await — это синтаксический сахар над промисами и очередью микротасков, работающий в *том же* потоке. |
| **Память (GC)** | SOH/LOH, Incremental GC. Проблема фрагментации. | Generational GC (Scavenge/Mark-Sweep). | В JS создание миллионов мелких объектов (например, замыканий) — норма. GC V8 чрезвычайно эффективен для "мусора" нулевого поколения. |
| **Компиляция** | AOT (Ahead-of-Time) в IL2CPP. | JIT (Just-in-Time). | JS код "разогревается". Первый вызов функции медленнее последующих (Inline Caching). |

#### Ментальная модель: Coroutines как основа JS

Представьте, что в Unity убрали Update() и запретили создавать new Thread(). Единственный способ выполнить код — это Coroutines.2

* Вызов fetch() в JS — это как StartCoroutine(UnityWebRequest...).
* Весь JS код — это содержимое одной большой корутины, которая никогда не должна делать while(true) без yield.
* **Call Stack** JS — это стек вызовов вашей корутины.
* **Task Queue** — это список корутин, готовых продолжить выполнение (MoveNext()) на следующем шаге.

#### Область видимости и Замыкания (Closures)

В C# замыкания (в LINQ или лямбдах) создают класс-контейнер для захваченных переменных. В JS замыкания — это фундаментальный блок. Каждая функция "помнит" скоуп, в котором была создана.

* *Опасность для Unity-девелопера:* В C# мы осторожны с замыканиями из-за аллокаций. В JS/React вся архитектура (Hooks) построена на замыканиях. Необходимо понимать "Stale Closures" — когда функция "помнит" старое значение переменной, которое уже изменилось в другом кадре рендера.4

### 1.2. Промт для генерации углубленной теории (День 1)

Используйте этот промт, чтобы получить глубокое техническое сравнение:

Задание для AI:

Давай углубимся в Memory Management и Async Flow.

1. Сравни **V8 Hidden Classes** с **C# Structs/Vtables**. Как динамическая типизация JS оптимизируется под капотом? Почему изменение формы объекта (добавление поля на лету) убивает производительность, похоже ли это на Boxing в C#?
2. Разбери работу **Event Loop** на примере Unity PlayerLoop. Если Microtasks (Promises) — это Update, то Macrotasks (setTimeout) — это FixedUpdate или что-то другое?
3. Напиши код на JS, который эмулирует "зависание" Main Thread (как while(true) в Unity). Покажи, как разбить эту задачу на чанки, используя setTimeout (аналог yield return null).

### 1.3. Практический модуль: "Native JS DOM Manipulation"

**Цель:** Почувствовать "боль" императивного управления состоянием, чтобы оценить декларативность React. Мы создадим прототип списка задач, напрямую манипулируя DOM (аналог прямой работы с мешами и вертексами вместо использования GameObject).

Задание:

Создайте файл index.html и app.js. Реализуйте "ToDo List" без фреймворков.

1. **Data Fetching:**
   * Используйте fetch('https://jsonplaceholder.typicode.com/todos?\_limit=5').
   * *Требование:* Обработайте промис без async/await (через .then), а затем перепишите на async/await, чтобы увидеть разницу (аналог ContinueWith vs await в Task).
2. **DOM Rendering (Императивный подход):**
   * Напишите функцию render(items).
   * Внутри очищайте контейнер div.innerHTML = '' (аналог Destroy всех детей).
   * В цикле создавайте элементы: document.createElement('div') (аналог Instantiate).
   * Добавляйте обработчик клика: el.addEventListener('click',...) (аналог GetComponent<Button>().onClick.AddListener).
3. **State Mutation:**
   * Создайте глобальную переменную let state =.
   * При клике меняйте состояние и вызывайте render(state) вручную.
   * *Наблюдение:* Обратите внимание, что при каждом изменении (например, галочка "выполнено") мы перестраиваем весь DOM. В Unity это эквивалентно пересозданию всей сцены при движении персонажа. Это неэффективно. React решает именно эту проблему (Reconciliation).5

## День 2: React — Компоненты как Префабы

### 2.1. Теоретический блок: Декларативная UI архитектура

В Unity (особенно с UI Toolkit или старым uGUI) мы часто пишем код в стиле: "Найти кнопку, поменять текст, поменять цвет".

button.GetComponent<Image>().color = Color.red;

Это императивный подход. В React мы описываем состояние, а UI — это проекция состояния.

UI = f(State)

#### Компоненты vs Префабы

React Компонент — это функция, которая возвращает описание UI (JSX).

* **Unity Prefab:** Хранит структуру GameObject, компоненты и их начальные настройки.
* **React Component:** То же самое, но описывается кодом, а не YAML файлом.
* **Instantiate:** В Unity мы делаем Instantiate(Prefab). В React мы пишем <MyComponent /> в JSX.

#### Props vs Serialized Fields

В Unity мы настраиваем префаб через Инспектор:

C#

public class PlayerHealth : MonoBehaviour {  
 private int maxHealth = 100; // Это Prop  
}

В React это передается как аргумент функции:

JavaScript

function PlayerHealth({ maxHealth = 100 }) { // Это Prop  
 return <div>HP: {maxHealth}</div>;  
}

**Ключевое отличие:** Props в React **неизменяемы** (read-only) для дочернего компонента. Вы не можете сделать maxHealth = 50. Данные текут только вниз (One-Way Data Flow). Если нужно изменить здоровье, родитель должен передать функцию-колбэк (аналог UnityEvent или Action).

#### Virtual DOM как Оптимизация Draw Calls

Почему React быстрый? Потому что он не трогает "реальный" DOM (который медленный, как вызовы драйвера видеокарты) при каждом чихе.

* **Сценарий:** У вас список из 1000 элементов. Вы меняете цвет одного.
* **Native JS (День 1):** Пересоздает все 1000 элементов.
* **React:**
  1. Запускает render (строит новое дерево объектов в памяти).
  2. Сравнивает новое дерево со старым (Diffing Algorithm).
  3. Видит, что изменился только один атрибут у одного узла.
  4. Точечно меняет этот атрибут в браузере.
* **Аналогия с Unity:** Это похоже на **GPU Instancing** или **Dynamic Batching**. React "батчит" изменения DOM, чтобы минимизировать перерисовки страницы (Reflow/Repaint).

#### Ключи (Keys) в Списках

При рендеринге массивов React требует уникальный key.

* **Аналогия:** В Unity, если вы меняете порядок объектов в иерархии, движок должен знать, кто есть кто. Если key — это индекс массива (0, 1, 2), то при удалении первого элемента React подумает, что элемент 0 изменился, а элемент 1 стал элементом 0. Это приведет к полной перерисовке. key должен быть как InstanceID в Unity — уникальным и постоянным ID сущности (Database ID).5

### 2.2. Промт для генерации углубленной теории (День 2)

Задание для AI:

Объясни архитектуру React Fiber (новый движок рендеринга) через термины Game Loop.

1. Как React разделяет фазу **Render** (вычисление изменений) и фазу **Commit** (применение к DOM)? Сравни это с разделением Update (логика) и Render (отрисовка) в движке.
2. Почему нельзя мутировать пропсы? Объясни это через концепцию **Data Oriented Design** (как в Unity DOTS): данные отделены от логики.
3. Покажи пример компонента DayCell для календаря. Как передать событие клика наверх родителю? Сравни это с GetComponentInParent (плохая практика) и передачей UnityAction (хорошая практика).

### 2.3. Практический модуль: "Архитектура Календаря на Компонентах"

**Цель:** Спроектировать и реализовать структуру UI сервиса Календаря, используя композицию компонентов.7

**Техническое задание:**

1. **Setup:** Инициализируйте проект через Vite (npm create vite@latest calendar-app -- --template react). Структура папок должна напоминать Unity Project (Feature-based grouping): /src/features/calendar/components.
2. **Компонентная иерархия:**
   * CalendarView (Родитель, аналог Canvas/Panel).
   * CalendarHeader (Отображает месяц/год, кнопки навигации).
   * DaysGrid (Контейнер для сетки дней).
   * DayCell (Префаб ячейки дня).
3. **Data Flow:**
   * Создайте статический массив данных (mock data) в CalendarView.
   * Передайте данные вниз через props в DaysGrid, а оттуда в DayCell.
   * В DayCell реализуйте отображение даты и, если есть, списка событий.
   * **Важно:** Используйте .map() для генерации списка дней. Обязательно задайте key (используйте дату ISO string как ключ).

## День 3: Hooks — Управление состоянием и Жизненным циклом

### 3.1. Теоретический блок: Функциональная магия

Хуки (Hooks) — это механизм, позволяющий функциональным компонентам иметь состояние и доступ к жизненному циклу. Для C# разработчика это может выглядеть как магия, так как функции в C# обычно stateless.

#### useState как Сериализация

const [count, setCount] = useState(0);

* **Механика:** React хранит массив состояний для каждого компонента. При рендере он последовательно выдает значения из этого массива.
* **Аналогия:** Представьте, что ваш скрипт — это статический метод, но у него есть доступ к Dictionary<InstanceID, State>, где хранятся его данные. useState достает данные по индексу вызова.
* **Immutability:** В Unity мы часто меняем поля напрямую: transform.position.x += 1. В React стейт иммутабелен (неизменяем). Мы не меняем объект, мы создаем новый. setCount(count + 1). Это похоже на работу со struct в C# или строками: чтобы изменить строку, мы создаем новую. Это критично для определения изменений (Change Detection): React просто сравнивает ссылки (pointers), что очень быстро (O(1)), вместо глубокого сравнения объектов.9

#### useEffect как Script Lifecycle

Это самый мощный и опасный хук. Он заменяет Start, Update, OnDestroy.4

* **Синтаксис:** useEffect(callback, [dependencies]).
* **`` (Empty deps):** Аналог **Start()**. Запускается 1 раз после монтирования.
* **[prop] (Deps present):** Аналог реактивного кода в **Update()**: "Если prop изменился с прошлого кадра, выполни код".
* **Cleanup Function (return () =>...):** Аналог **OnDestroy()** или **OnDisable()**.
  + *Пример:* Вы подписались на событие EventManager.StartListening в Start. Вы *обязаны* отписаться в OnDestroy. В useEffect это делается в возвращаемой функции. Если этого не сделать — утечка памяти.

#### useRef как GetComponent

Иногда нужно сохранить ссылку на DOM элемент (например, чтобы установить фокус или прочитать размер).

* **Аналогия:** useRef — это private поле класса, которое переживает рендеры, но его изменение *не вызывает* перерисовку (в отличие от useState). Это "escape hatch" для императивного кода.11

### 3.2. Промт для генерации углубленной теории (День 3)

Задание для AI:

Разбери паттерн Hooks на атомы.

1. Почему **Rules of Hooks** запрещают вызывать хуки в циклах и условиях? Объясни это через внутреннее устройство React (массивы зависимостей и индексы). Сравни это с чтением бинарного стрима в C# (BinaryReader): что будет, если пропустить байт?
2. Сравни **Stale Closures** в useEffect с захватом переменной цикла в C# (foreach (var i in list) { actions.Add(() => Print(i)); }). Почему useEffect иногда "видит" старое значение стейта?
3. **Custom Hooks:** Объясни, как вынос логики в useCalendar похож на создание ScriptableObject или отдельного класса-контроллера (Composition over Inheritance).

### 3.3. Практический модуль: "Интерактивное Ядро Календаря"

**Цель:** Реализовать навигацию по месяцам и выбор даты, используя Hooks.7

**Техническое задание:**

1. **State Management (CalendarView):**
   * const = useState(new Date()).
   * const = useState(null).
2. **Logic:**
   * Реализуйте функции nextMonth() и prevMonth(). Они должны создавать *новый* объект Date (помните про иммутабельность!). newDate.setMonth(currentDate.getMonth() + 1).
3. **Side Effects (useEffect):**
   * Добавьте логгирование: при смене месяца выводите в консоль "Loaded logic for [Month]".
   * **Advanced:** Реализуйте "Часы" в шапке. Используйте setInterval внутри useEffect с пустым массивом зависимостей ``. *Критически важно:* верните функцию очистки clearInterval. Проверьте, что будет, если не очистить интервал (утечка таймеров при размонтировании компонента).
4. **Integration:** Свяжите кнопки в CalendarHeader с функциями навигации.

## День 4: Архитектура CSS Grid и Визуализация

**(Фокус на аналогии с Unity UI System)**

### 4.1. Теоретический блок: Layout Systems

Верстка в вебе — это, по сути, работа с Layout Groups в Unity, но гораздо мощнее.

#### CSS Grid vs GridLayoutGroup

В Unity компонент GridLayoutGroup достаточно примитивен. CSS Grid — это двумерная система компоновки.12

* **Grid Container:** display: grid.
* **Template Columns:** grid-template-columns: repeat(7, 1fr).
  + 1fr (fraction) — это аналог "Flexible" настройки в Unity Layout Element. Это означает "одна доля свободного пространства". 7 колонок по 1fr автоматически разделят ширину экрана на 7 равных частей.
* **Gap:** gap: 10px. Аналог Spacing в Unity.

#### Flexbox vs Horizontal/Vertical Layout Group

Flexbox (display: flex) — это одномерная система (строка или колонка).

* Используйте Flexbox для выравнивания *контента внутри ячейки* (цифра дня по центру).
* justify-content = Выравнивание по главной оси.
* align-items = Выравнивание по поперечной оси.
* Это полная замена RectTransform якорям (Anchors) и пивотам (Pivots).

#### CSS Modules: Инкапсуляция стилей

В Unity мы настраиваем вид каждого префаба в инспекторе. В React глобальный CSS — это зло (как глобальные статические переменные). Используйте CSS Modules (styles.module.css).

* Это гарантирует, что класс .button в календаре не сломает класс .button в хедере сайта. React сгенерирует уникальные имена: Calendar\_button\_\_x7z9.

### 4.2. Промт для генерации углубленной теории (День 4)

Задание для AI:

Сравни системы верстки Unity uGUI и CSS.

1. Сравни **Draw Calls** в Unity UI и **Reflow/Repaint** в браузере. Почему изменение width элемента в CSS Grid может вызвать пересчет всей страницы, и как этого избегать (аналог Canvas.ForceUpdateCanvases)?
2. Объясни z-index и Stacking Context в CSS через призму **Hierarchy order** и **Canvas Sorting Layer** в Unity.
3. Помоги написать CSS Grid для календаря, который автоматически адаптируется под мобильные устройства (Media Queries), сравнивая это с **Canvas Scaler** в Unity.

### 4.3. Практический модуль: "Pixel-Perfect Верстка"

**Цель:** Создать профессиональную сетку календаря, которая не "разваливается" при ресайзе окна.13

**Техническое задание:**

1. **Grid Layout:** В DaysGrid примените CSS Grid для создания 7 колонок.
2. **Placeholders:** Вычислите, с какого дня недели начинается месяц. Если 1-е число — Среда, добавьте 2 пустых <div> (или дней предыдущего месяца) в начало. Это алгоритмическая задача, схожая с процедурной генерацией тайловой карты.
3. **Styling States:**
   * Текущий день: выделите кружком (аналог спрайта выделения).
   * Выходные: легкий серый фон.
   * События: внутри ячейки дня добавьте список "точек" или полосок событий. Используйте Flexbox для их вертикального выравнивания.
4. **Responsive:** Добавьте @media (max-width: 600px), чтобы на мобильном телефоне названия дней сокращались до 1 буквы (Пн, Вт...), а шрифт уменьшался.

## День 5: Интеграция и Fullstack Архитектура

### 5.1. Теоретический блок: Сетевое взаимодействие и Безопасность

Теперь соединяем Frontend (React) и Backend (ASP.NET Core из Фазы II).1

#### Fetch vs UnityWebRequest

fetch — это нативный браузерный API.

* Он возвращает Promise (аналог Task).
* Отличие от C#: В C# HttpClient блокирует поток при синхронном вызове. В JS синхронных сетевых запросов не существует (они deprecated). Вы *обязаны* работать асинхронно.

#### CORS (Cross-Origin Resource Sharing)

Это стена, с которой сталкивается каждый Unity-разработчик, переходящий в веб. В Unity (Desktop) вы можете послать запрос куда угодно. В браузере действует Same-Origin Policy.

* Если ваш React на localhost:5173, а API на localhost:5000 — это *разные* источники (origins). Браузер заблокирует запрос.
* **Решение:** Настроить Middleware в ASP.NET Core (app.UseCors(...)), разрешив запросы с адреса фронтенда. Это не баг, это защита пользователя от XSS атак на других сайтах.

#### JWT Auth Storage: Security First

Где хранить Access Token?

* **LocalStorage:** Аналог PlayerPrefs. Легко доступен JS-скриптам. Если хакер внедрит JS код на страницу (XSS), он украдет токен.
* **HttpOnly Cookies:** Токен хранится в куках, помеченных флагом HttpOnly. JS не имеет к ним доступа. Браузер сам прикрепляет их к запросам. Это Enterprise стандарт безопасности.15

### 5.2. Промт для генерации углубленной теории (День 5)

Задание для AI:

Я интегрирую React с.NET API.

1. Объясни механику **CORS Preflight Request** (OPTIONS). Почему браузер делает два запроса вместо одного? Сравни это с рукопожатием (Handshake) в сетевых протоколах (TCP/UDP).
2. Сравни **JWT в LocalStorage** vs **HttpOnly Cookies** с точки зрения безопасности. В Unity мы шифруем сейвы, чтобы игрок не читерил. От кого мы защищаемся в вебе? (XSS vs CSRF).
3. Напиши кастомный хук useFetchEvents, который инкапсулирует логику загрузки, состояние isLoading и error. Сравни это с паттерном **Repository** или **Service Layer** в Unity.

### 5.3. Практический модуль: "Интеграция с Реальным API"

**Цель:** Получить данные из вашей БД Postgres и отобразить их в календаре.1

**Техническое задание:**

1. **Backend Config:** В.NET проекте в Program.cs добавьте CORS политику, разрешающую http://localhost:5173 (или ваш порт Vite).
2. **API Client:** Создайте файл api.js (или сервис). Настройте fetch запросы к эндпоинтам:
   * GET /api/events?start=...&end=...
   * POST /api/events (создание)
3. **Data Fetching:** В CalendarView используйте useEffect для загрузки событий при смене месяца.
   * *Важно:* Не забудьте AbortController. Если пользователь быстро переключает месяцы, предыдущие запросы нужно отменять (аналог отмены Task через CancellationToken в C#).
4. **UI Feedback:**
   * Пока данные грузятся, показывайте спиннер (Loader).
   * Если ошибка — показывайте Toast (всплывающее уведомление).

## Итоговые задания (Capstone Projects)

Для подтверждения квалификации Fullstack-инженера необходимо выполнить два архитектурных задания. Эти проекты требуют синтеза знаний C# Backend и React Frontend.

### Задание 1: "Архитектура Виртуализации (Infinite Scroll / Virtual List)"

Контекст: Представьте, что в календаре (в режиме "Повестка дня" / Agenda View) могут быть тысячи событий. Рендерить 5000 <div> элементов — это убийство производительности (аналог спавна 5000 GameObject).

Задача:

1. Реализовать компонент списка событий с **виртуализацией** (Windowing).
2. **Логика:** Рендерить только те элементы, которые видны во viewport + небольшой буфер сверху и снизу.
3. **Сравнение:** Проведите аналогию с **Occlusion Culling** или **LOD** в Unity.
4. **Реализация:**
   * Вычислить высоту скролл-контейнера.
   * На основе позиции скролла (scrollTop) высчитать индексы первого и последнего видимого элемента массива.
   * Рендерить только этот срез (slice) массива, добавляя padding сверху и снизу, чтобы скроллбар оставался правильного размера.
   * *Примечание:* Это классическая задача на оптимизацию, демонстрирующая понимание того, как работает браузерный рендеринг.

### Задание 2: "Менеджер Повторяющихся Событий (Recurring Events Engine)"

Контекст: Бэкенд (Фаза II) хранит повторяющиеся события в формате RRULE (стандарт iCal), например: FREQ=WEEKLY;BYDAY=MO ("Каждый понедельник").

Задача:

1. **Backend:** API возвращает *один* объект события с правилом повторения.
2. **Frontend Logic:** На клиенте (в React) написать движок, который на лету генерирует "виртуальные" инстансы события для отображаемой сетки месяца.
   * Нельзя создавать записи в БД для каждого повторения (это бесконечный объем данных).
3. **State Management:** Использовать useMemo для кэширования вычислений. Пересчитывать инстансы только если изменился месяц или список исходных событий.
4. **Unity-Аналогия:** Это чистый **Procedural Generation**. У вас есть "Seed" (правило повторения) и координаты (даты месяца), и вы процедурно генерируете контент. Это требует глубокого понимания алгоритмической сложности, чтобы календарь не "лагал" при переключении.

## Таблица соответствия компетенций (Unity -> React)

Для быстрого референса в будущем используйте эту таблицу перевода концепций:

| **Концепция Unity (C#)** | **Концепция React (JS/TS)** | **Комментарий** |
| --- | --- | --- |
| **Prefab** | **Component** | Шаблон для создания экземпляров UI/Логики. |
| **Inspector Variable (``)** | **Prop** | Входные параметры. В React всегда read-only. |
| **Private Field** | **State (useState)** | Внутреннее состояние. Изменение вызывает ререндер. |
| **Start() / OnEnable()** | **useEffect(() => {},)** | Инициализация. |
| **Update()** (Reactive logic) | **useEffect(() => {}, [deps])** | Реакция на изменение данных. |
| **OnDestroy()** | **useEffect return () => {}** | Очистка ресурсов. |
| **Coroutine** | **Async/Await + Promises** | Асинхронные операции. |
| **UnityEvent / Delegate** | **Callback Prop** | Передача событий от ребенка к родителю. |
| **GameObject.SetActive(false)** | **Conditional Rendering ({cond && <Comp />})** | Условное отображение. В React элемент удаляется из DOM, а не просто скрывается (обычно). |
| **Object Pooling** | **React Reconciliation (Keys)** | Переиспользование существующих DOM узлов. |
| **Singleton / Static Manager** | **Context API / Redux** | Глобальное состояние (избегайте по возможности). |

Этот план обеспечивает жесткую техническую базу для перехода, игнорируя "игрушечные" примеры и фокусируясь на инженерных принципах, общих для обеих платформ. Удачного кодинга!

#### Источники

1. Разработка бэкенда: план обучения C#
2. Unity coroutine vs threads - Game Development Stack Exchange, дата последнего обращения: декабря 6, 2025, <https://gamedev.stackexchange.com/questions/143454/unity-coroutine-vs-threads>
3. Coroutines OR Async/await? : r/unity - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/unity/comments/zr39d4/coroutines_or_asyncawait/>
4. A Complete Guide to useEffect - Overreacted, дата последнего обращения: декабря 6, 2025, <https://overreacted.io/a-complete-guide-to-useeffect/>
5. Why React Needs a Key for Rendering Lists - DEV Community, дата последнего обращения: декабря 6, 2025, <https://dev.to/jessalejo/why-react-needs-a-key-for-rendering-lists-3ko3>
6. React Keys: A Secret Weapon for Efficient Lists | by Sagar Ninave | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@sagarninave/react-keys-a-secret-weapon-for-efficient-lists-c90425a57d35>
7. React Calendar: Build a Dynamic Calendar from Scratch - YouTube, дата последнего обращения: декабря 6, 2025, <https://www.youtube.com/watch?v=_Og5aDRmWfk>
8. React Tutorial: Creating a Fully Functional Calendar App with React - YouTube, дата последнего обращения: декабря 6, 2025, <https://www.youtube.com/watch?v=wDayVPGWipI>
9. React Hooks Explained Simply - Daily.dev, дата последнего обращения: декабря 6, 2025, <https://daily.dev/blog/react-hooks-explained-simply>
10. React Hooks from a C# Developer's perspective : r/reactjs - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/reactjs/comments/o1swnh/react_hooks_from_a_c_developers_perspective/>
11. All React Hooks Explained - React Hooks Tutorial 2025 - YouTube, дата последнего обращения: декабря 6, 2025, <https://www.youtube.com/watch?v=xfKYYRE6-TQ>
12. CSS Grid vs Flexbox: Which Is Best, When, And Why? | Zero To Mastery, дата последнего обращения: декабря 6, 2025, <https://zerotomastery.io/blog/css-grid-vs-flexbox/>
13. An Interactive Guide to CSS Grid • Josh W. Comeau, дата последнего обращения: декабря 6, 2025, <https://www.joshwcomeau.com/css/interactive-guide-to-grid/>
14. How to create a calendar from scratch in React | by Derrick Otte - Medium, дата последнего обращения: декабря 6, 2025, <https://derrickotte.medium.com/how-to-create-a-calendar-from-scratch-in-react-1f2db197454d>
15. How to securely use JWT in react frontend? : r/reactjs - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/reactjs/comments/1ngq4wj/how_to_securely_use_jwt_in_react_frontend/>
16. Best Practices for Implementing JWT Auth in .NET Core and React - Facile Technolab, дата последнего обращения: декабря 6, 2025, <https://www.faciletechnolab.com/blog/best-practices-for-implementing-jwt-auth-in-net-core-and-react/>