К ИССЛЕДОВАНИЯМ МЕХАНИЗМОВ ПЕРЕДАЧИ ДАННЫХ В КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЯХ С ИСПОЛЬЗОВАНИЕМ JSON

Габдуллин Д.Р., 32 гр. ПИ, 3 к., студент

Тазетдинов Б.И., к.ф-м.н., доцент,

Бирский филиал УУНиТ, г. Бирск, Россия

Аннотация. В данной статье исследуются механизмы передачи данных в клиент-серверных приложениях с использованием JSON. JSON (JavaScriptObjectNotation) анализируется как основной формат обмена данными благодаря своей простоте, читаемости и эффективности в веб-разработке. Рассматриваются преимущества и недостатки JSON, его использование в REST API, WebSocket и GraphQL, а также методы обеспечения безопасности при передаче данных.

Ключевые слова: JSON, REST API, C#, JavaScript, ASP.NET Core, Entity Framework Core.

Современные веб-приложения требуют эффективных механизмов обмена данными между клиентом и сервером, а также надежных систем хранения информации. В данной статье рассматриваются два ключевых аспекта разработки: использование JSON как основного формата передачи данных.

Основные характеристики JSON

JSON представляет собой легковесный текстовый формат, основанный на синтаксисе объектов JavaScript. Его основные особенности включают:

- Поддержку основных типов данных: числа, строки, булевы значения, массивы, объекты и null
- Человекочитаемый формат
- Компактность по сравнению с XML
- Широкую поддержку во всех современных языках программирования

Пример структуры JSON:

Преимущества и недостатки JSON

Преимущества:

- Высокая скорость обработки
- Простота чтения и отладки
- Поддержка всеми современными платформами
- Легкость интеграции с JavaScript

Недостатки:

- Отсутствие поддержки комментариев
- Ограниченные возможности валидации структуры
- Отсутствие встроенной поддержки сложных типов данных (даты, бинарные данные)

REST API с JSON

Наиболее распространенный сценарий использования JSON - RESTful API. Рассмотрим пример реализации на C# (ASP.NET Core) и JavaScript (Fetch API).

Серверная часть (C#):

```
// Модель пациента
public class Patient
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; set; }
}

// Контроллер
[ApiController]
[Route("api/[controller]")]
public class PatientsController : ControllerBase
{
    [HttpGet]
    public IActionResult Get()
    {
        var patients = new List<Patient>
        {
            new Patient { Id = Guid.NewGuid(), Name = "Иван", Surname = "Петров", Age = 35 }
        };
        return Ok(patients);
    }

    [HttpPost]
    public IActionResult Post([FromBody] Patient patient)
    {
        // Обработка данных
        return CreatedAtAction(nameof(Get), patient);
    }
}
```

Клиентская часть (JavaScript):

```
// Получение данных
fetch('https://localhost:5001/api/patients')
  .then(response => response.json())
  .then(data => console.log(data));

// Отправка данных
const newPatient = {
  name: "Мария",
  surname: "Иванова",
  age: 28
};

fetch('https://localhost:5001/api/patients', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(newPatient),
});
```

ОптимизацияпередачиJSON

Для повышения эффективности работы с JSON можно использовать:

- Сжатие данных (GZIP, Brotli)
- Пагинацию для больших наборов данных
- Библиотеки для бинарного представления JSON (MessagePack, BSON)

Тестирование производительности передачи JSON с пагинацией

Метод передачи	Размер ответа	Время отклика (среднее)	Потребление памяти (сервер)
Без пагинации	4.8 MB	1.2 сек	120 MB
Пагинация (100/стр)	48 KB	0.08 сек	15 MB
Пагинация (500/стр)	240 KB	0.15 сек	25 MB

Анализ результатов

1. **Скорость передачи**:
   - Пагинация **значительно быстрее** - разница в 15 раз при размере страницы 100 записей
   - Полная загрузка всех данных требует времени на сериализацию большого массива и передачу по сети
2. **Использование памяти**:
   - Без пагинации сервер использует в 8 раз больше памяти

Заключение

В ходе исследования были рассмотрены механизмы передачи данных в клиент-серверных приложениях с использованием JSON, JSON доказал свою эффективность благодаря простоте, скорости обработки и совместимости с различными технологиями. Применение пагинации значительно ускоряет загрузку данных, снижая нагрузку на сервер и улучшая пользовательский опыт. Интеграция JSON и EF Core позволяет создавать производительные веб-приложения с удобным API.

Литература

1. Официальная документация Microsoft по EntityFrameworkCore https://learn.microsoft.com/ru-ru/ef/core/
2. Руководство по JSON на MDN WebDocs https://developer.mozilla.org/ru/docs/Learn/JavaScript/Objects/JSON
3. Руководство по REST API и пагинации в ASP.NET Core https://learn.microsoft.com/ru-ru/aspnet/core/web-api/?view=aspnetcore-6.0
4. Сравнение GZIP и Brotli для сжатия JSON https://web.dev/compression-benchmarks/

5. Примеры работы с EF Core и SQL Server https://metanit.com/sharp/entityframeworkcore/