Современные платформы прикладной разработки

REST API

Знакомство с REST

REST

REST (*англ. **RE**presentation **S**tate **T**ransfer - передача состояния представления*) - это архитектурный стиль программного обеспечения, предназначенный для распределенных систем и, в частности, для World Wide Web.

REST

Термин «REST» был введён **Роем Филдингом** (англ. *Roy Fielding*), одним из создателей протокола «HTTP» в **2000** году в докторской диссертации *«Архитектурные стили и дизайн сетевых программных архитектур» («Architectural Styles and the Design of Network-based Software Architectures»)* в Калифорнийском университете в Ирвайне

REST

REST не является протоколом или стандартом

REST

Архитектура REST основана на семи **описательных** свойствах

Свойства архитектуры REST

- **Производительность** – то, как взаимодействуют компоненты, влияет на производительность
- **Масштабируемость** - возможность поддержки большого количества компонентов
- **Простота** унифицированного интерфейса

Свойства архитектуры REST

- **Модифицируемость** компонентов для удовлетворения меняющихся потребностей
- **Видимость** - четкая связь между компонентами
- **Переносимость** кода, заполненного данными
- **Надежность** - или сопротивление сбою на уровне системы

REST

Все в архитектуре RESTful связано с ресурсами.

Ресурс - это объект со своими связанными данными.

Ограничения ресурсов

- **Идентификация ресурсов** как «**запросов**», простым способом, который понимается независимо от исходного языка или интерпретации
- **Манипулирование ресурсами**. Когда клиент имеет представление данных, он может затем изменить или удалить этот ресурс.

Ограничения ресурсов

- **Самоописательные сообщения**. Самоописательный ресурс сам по себе; такой обмен сообщениями совершенен как сущность и сам по себе может описывать, как он может быть обработан
- **Гипермедиа** означает, что клиенты изменяют состояние системы только через действия, которые динамически определены в гипермедиа на сервере (к примеру, гиперссылки в гипертексте). Клиент не может предположить что доступна какая-то операция над каким-то ресурсом, если не получил информацию об этом в предыдущих запросах к серверу.

Базовые принципы архитектуры REST

Базовые принципы архитектуры REST

- Разделение клиент-сервер
- Stateless (Без сохранения состояния).
- Cacheable (возможность кэширования)
- Многоуровневые системы.
- Код по требованию

Базовые принципы архитектуры REST

Разделение клиент-сервер

Эти два объекта полностью независимы, что позволяет им разрабатываться и заменяться независимо друг от друга. Клиентский код становится более переносимым, поскольку он отделен от хранилища данных, тогда как сервер становится более масштабируемым, поскольку он не заботится о состоянии пользователя или об интерфейсе.

Базовые принципы архитектуры REST

Без сохранения состояния.

Каждая пара данных запроса и ответа рассматривается как полностью независимая от предыдущих и будущих запросов.

Базовые принципы архитектуры REST

Cacheable

Все в Интернете может быть кэшировано, поэтому ответы должны четко определять, являются ли они кэшируемыми, избегая либо ненадлежащего кэширования или кеширование старой информации.

Базовые принципы архитектуры REST

Многоуровневые системы.

Серверы промежуточного уровня должны быть доступны, чтобы сделать систему более масштабируемой.

Клиент не способен точно определить, взаимодействует ли он напрямую с сервером или же с промежуточным узлом, в связи с иерархической структурой сетей

Базовые принципы архитектуры REST

Код по требованию

Это единственное «необязательное» ограничение REST. Серверы могут временно расширять или настраивать функциональность клиента путем передачи исполняемого кода, например, JavaScript

REST следует основному языку передачи гипертекста HTTP 1.1, который понимает весь Интернет.

Это пояснительные глаголы действия (типа запроса), которые обычно пишутся заглавными буквами

- POST - добавить ресурс
- GET - для извлечения данных, но без изменения, с определенного URL-адреса
- PUT - для сохранения или обновления ресурсов (URI)
- DELETE - удаление указанного ресурса
- PATCH - внести изменения в запрос

REST стал основой, на которой были разработаны стандарты HTTP и URI, которые также были разработаны Филдингом параллельно.

Работа с REST-сервисом

Пример запроса для получения данных

Метод HTTP **Get**

http://localhost:55946/api/cars

http://localhost:55946/api/cars/10

http://localhost:55946/api/cars?model=x3

Пример запроса для отправки данных

Метод HTTP **Post**

http://localhost:55946/api/cars

REST API

REST API

Контроллеры API похожи на обычные контроллеры, за исключением того, что ответы, полученные их методами действия, являются объектами данных, которые отправляются клиенту без разметки HTML.

REST API

Контроллеры API позволяют клиентам получать доступ к данным в приложении.

Не все клиенты являются браузерами, и не все клиенты представляют данные пользователю.

REST API

```
// GET: api/Dogs
[HttpGet]
public IEnumerable<Dog> GetDogs()
{
    return _context.Dogs;
}
```

REST API контроллер

```
[Route("api/[controller]")]
[ApiController]
public class DogsController : ControllerBase
{
    . . .
}
```

REST API контроллер

```
// GET: api/Dogs/5
[HttpGet("{id}")]
public async Task<IActionResult> GetDog([FromRoute] int id)
{
    . . .
}
```

REST API контроллер

```
// PUT: api/Dogs/5
[HttpPut("{id}")]
public async Task<IActionResult> PutDog([FromRoute] int id,
                                        [FromBody] Dog dog)
{
    . . .
}
```

Формат ответа

- Если контроллер возвращает строку, то она передается клиенту без изменения
- Если контроллер возвращает объект C#, то этот объект сериализуется в формат JSON

Формат ответа

StatusCode – Возвращает указанный код состояния клиенту.
Ok – Возвращает клиенту код состояния 200.
Created – Возвращает клиенту код состояния 201.
CreatedAtAction – Возвращает клиенту код состояния 201 с URL заголовке Location.
CreatedAtRoute – Возвращает клиенту код состояния 201 с URL, полученный из маршрута, в заголовке Location.
BadRequest – Возвращает клиенту код состояния 400.
Unauthorized – Возвращает клиенту код состояния 401.
NotFound – Возвращает клиенту код состояния 404.

Minimal API

Minimal API

Minimal APIs — позволяет упростить разработку API, предоставляющих доступ к данным в формате JSON.

Minimal API

Шаги стандартного запроса ASP.NET Core MVC включают маршрутизацию, инициализацию контроллера, выполнение действий с привязкой модели и фильтрами, а также фильтры результатов. Обработка запроса обычно представляет собой 17-шаговый процесс с дополнительными шагами, если вы используете какие-либо механизмы просмотра. При создании API на основе JSON вы можете рассматривать эти шаги как «ненужные» и уменьшать свои возможности для повышения пропускной способности.

https://blog.jetbrains.com/dotnet/2023/04/25/introduction-to-asp-net-core-minimal-apis/ (07.2023)

Minimal API

Было обнаружено, что минимальные API работают немного лучше, чем традиционные API. В одном сравнении минимальный API примерно на 40% быстрее возвращал простой ответ 200 со строкой. Кроме того, объем памяти для минимального API был примерно на 15 МБ меньше.

https://dusted.codes/how-fast-is-really-aspnet-core (07.2023)

Minimal API

Пример минимального API:

```
app.MapGet("/", () => "Hello World!");
```

Minimal API

Пример минимального API:

```
app.MapGet("/api/books/", async (ApplicationDbContext db) =>
{
    return await db.Books.ToListAsync();
})
.WithName("GetAllBooks")
.WithOpenApi();
```

Minimal API

```
public static class BookEndpoints
{
public static void MapBookEndpoints (this IEndpointRouteBuilder routes)
{
    var group = routes.MapGroup("/api/Book").WithTags(nameof(Book));

    group.MapGet("/", async (ApplicationDbContext db) => { . . . });
    group.MapGet("/{id}", async Task<Results<Ok<Book>, NotFound>> (int id,
                                                ApplicationDbContext db) => {. . . });
    group.MapPut("/{id}", async Task<Results<Ok, NotFound>> (int id,
                                            Book book, ApplicationDbContext db) =>
    {
        var affected = await db.Books
            .Where(model => model.Id == id)
            .ExecuteUpdateAsync(setters => setters.SetProperty(m => m.Id, book.Id)
                                . . . );
        return affected == 1 ? TypedResults.Ok() : TypedResults.NotFound();
    });
    . . .
}
```

Minimal API

```
app.MapBookEndpoints();
```