04 BackEnd – WebAPI

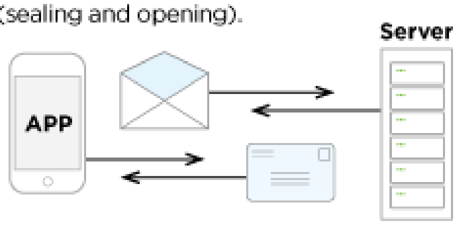
Servidor WEB, Servidor Base de Datos

Web API

SOAP vs. REST APIs

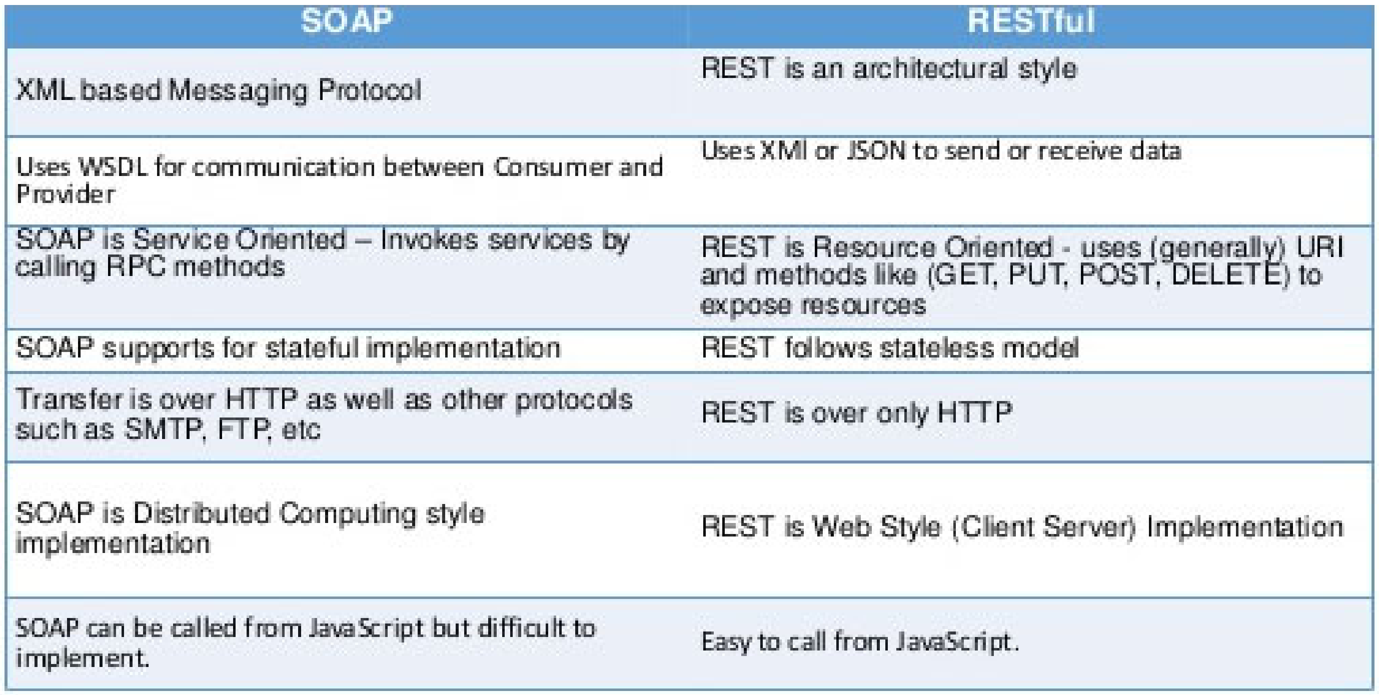
SOAP is like using an envelope

Extra overhead, more bandwidth required, more work on both ends (sealing and opening).



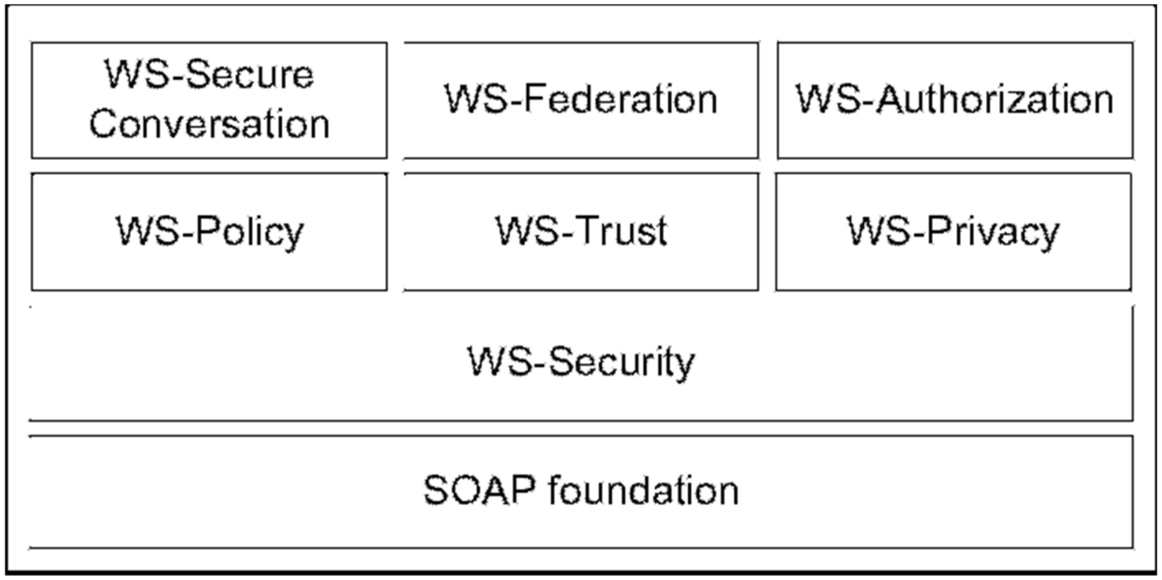
REST is like a postcard

Lighterweight, can be cached, easier to update.



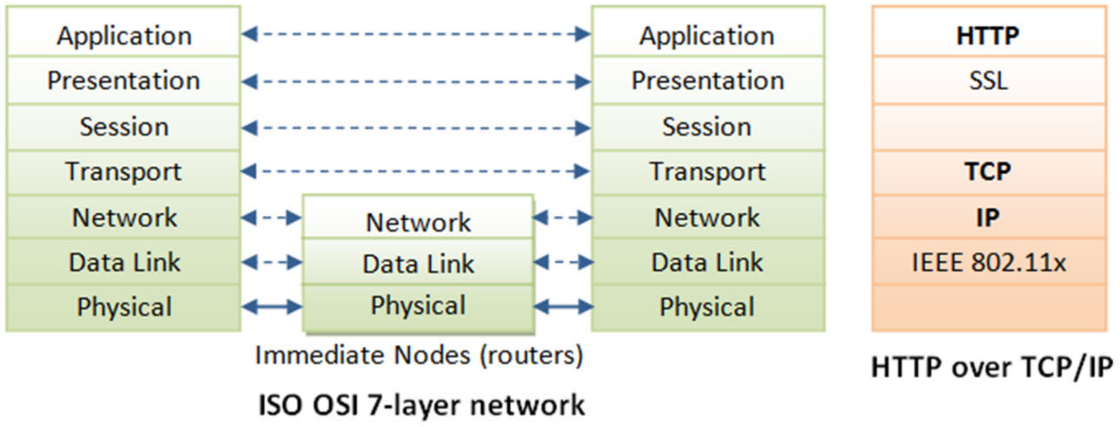
SOAP	RESTful
XML based Messaging Protocol	REST is an architectural style
Uses WSDL for communication between Consumer and Provider	Uses XMl or JSON to send or receive data
SOAP is Service Oriented – Invokes services by calling RPC methods	REST is Resource Oriented - uses (generally) URI and methods like (GET, PUT, POST, DELETE) to expose resources
SOAP supports for stateful implementation	REST follows stateless model
Transfer is over HTTP as well as other protocols such as SMTP, FTP, etc	REST is over only HTTP
SOAP is Distributed Computing style implementation	REST is Web Style (Client Server) Implementation
SOAP can be called from JavaScript but difficult to implement.	Easy to call from JavaScript.

Web API


WS-Secure Conversation
WS-Federation
WS-Authorization
WS-Policy
WS-Trust
WS-Privacy
WS-Security
SOAP foundation

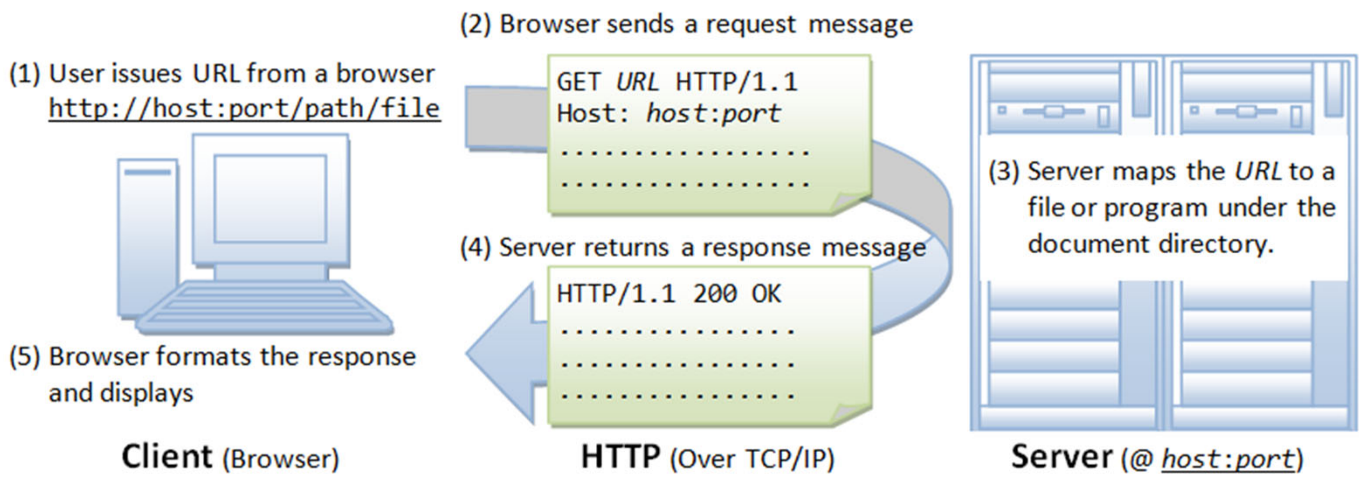
HTTP

- Protocolo Transferencia Hipertexto

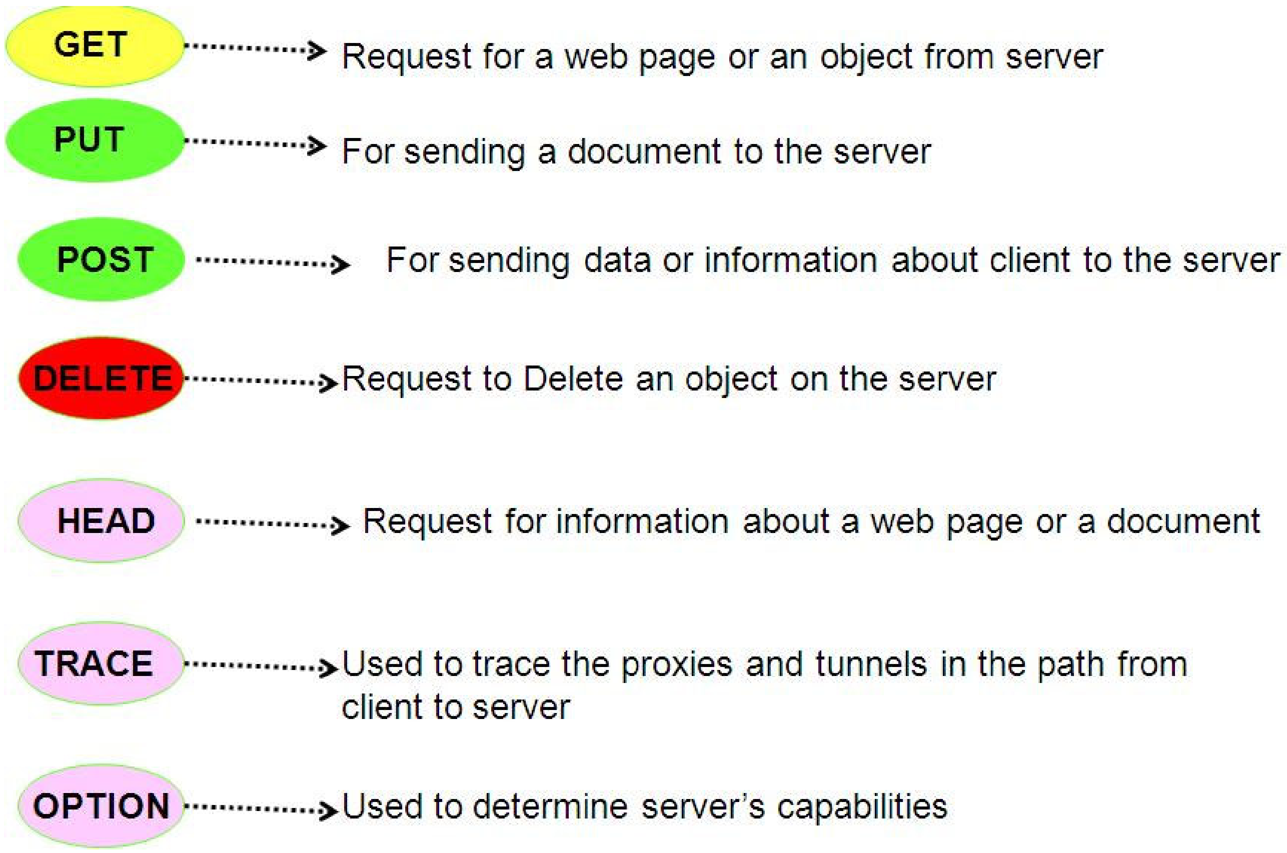


HTTP

- Requerimiento - Respuesta

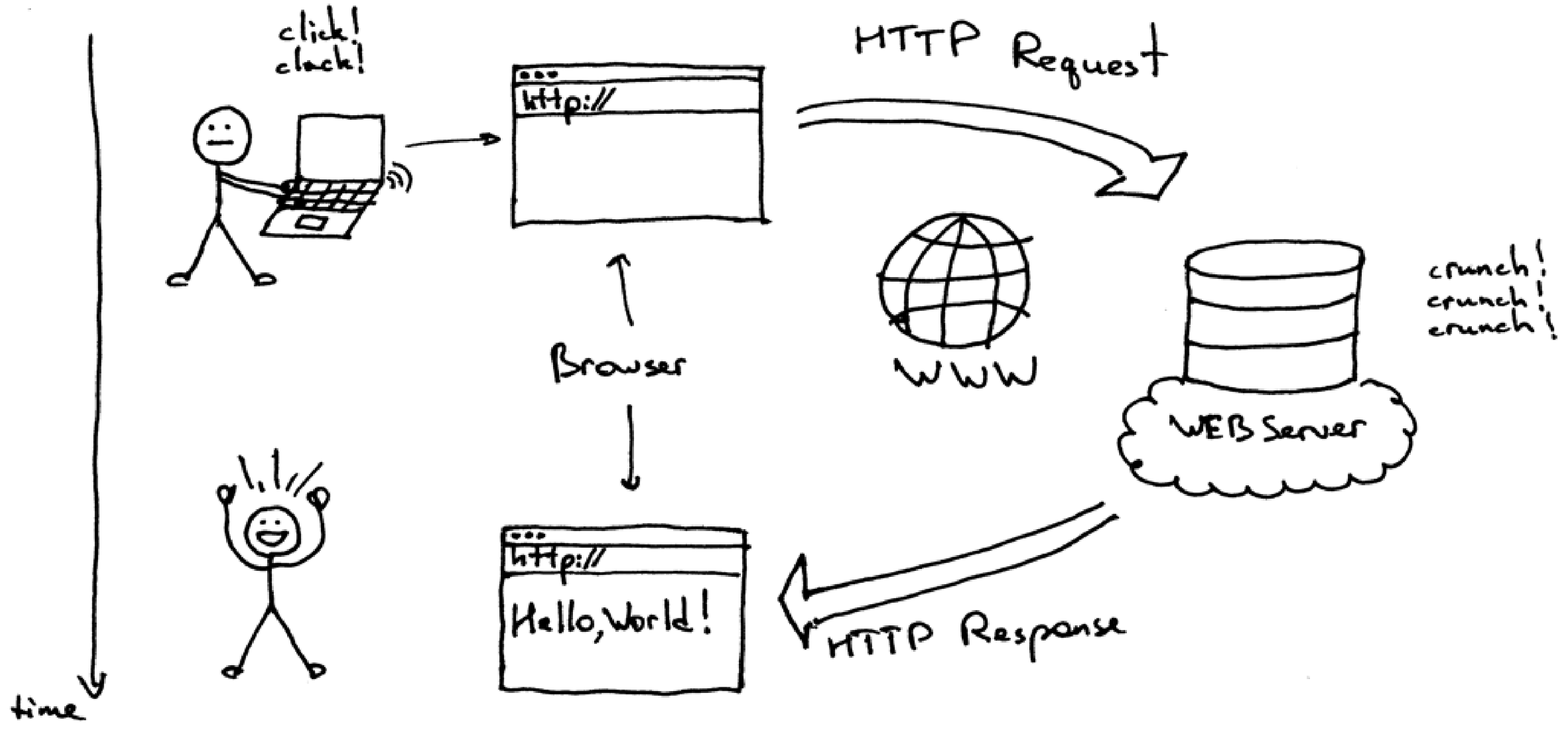

(2) Browser sends a request message
(1) User issues URL from a browser
http://host:port/path/file
GET URL HTTP/1.1
Host: host:port
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

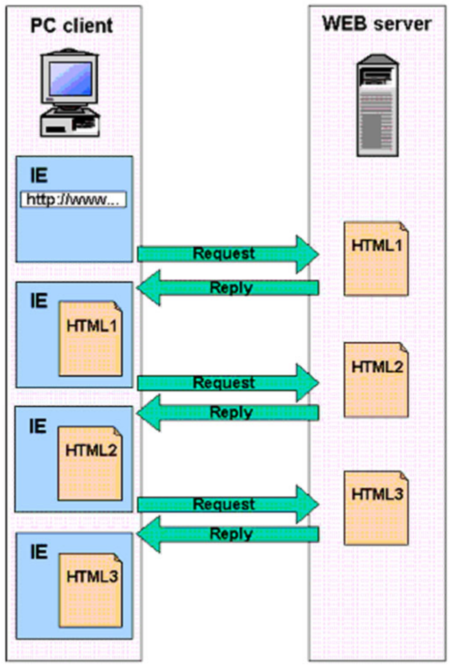
HTTP Métodos

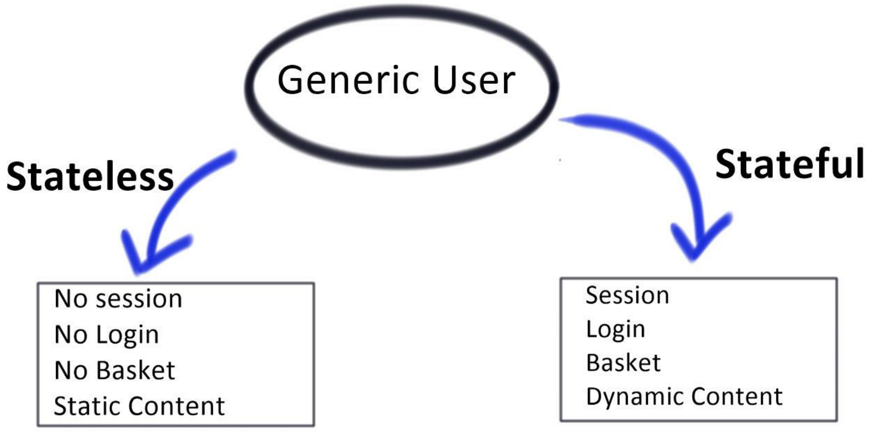


GET	Request for a web page or an object from server
PUT	For sending a document to the server
POST	For sending data or information about client to the server
DELETE	Request to Delete an object on the server
HEAD	Request for information about a web page or a document
TRACE	Used to trace the proxies and tunnels in the path from client to server
OPTION	Used to determine server's capabilities

Servidor WEB – Administra mensajes HTTP

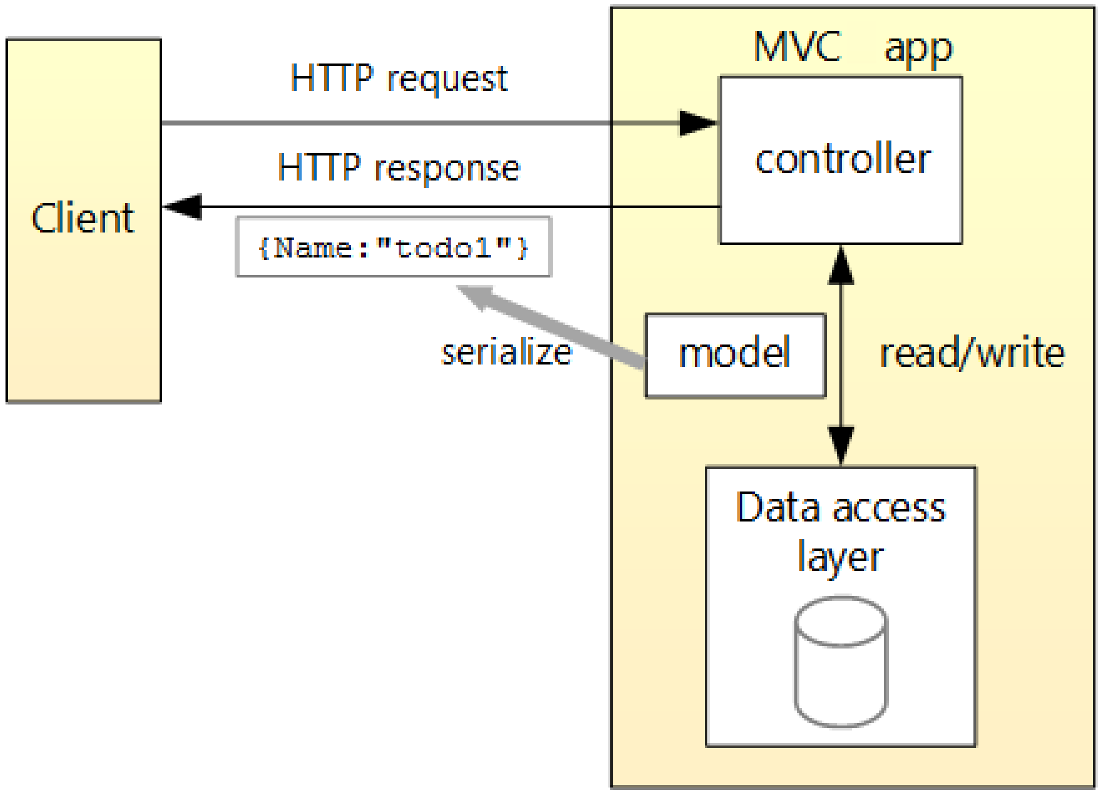




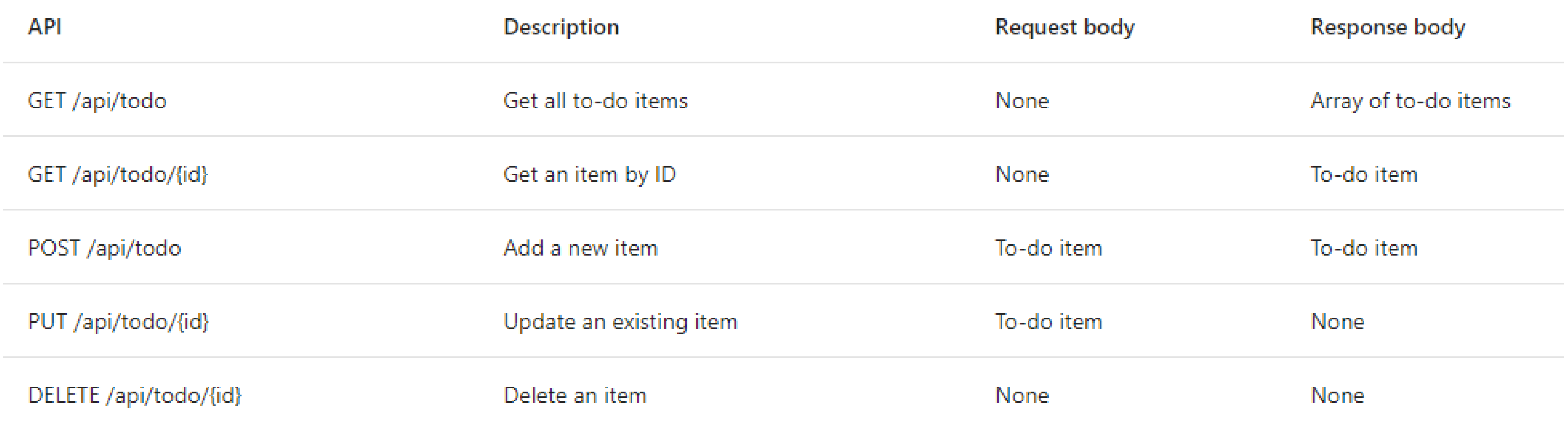


HTTP - Sin Estado

Arquitectura de una Web - Api



Descripción de la API



API	Description	Request body	Response body
GET /api/todo	Get all to-do items	None	Array of to-do items
GET /api/todo/{id}	Get an item by ID	None	To-do item
POST /api/todo	Add a new item	To-do item	To-do item
PUT /api/todo/{id}	Update an existing item	To-do item	None
DELETE /api/todo/{id}	Delete an item	None	None

Web Api - Model

- Modelo – es un objeto que representa a los datos en la aplicación

```
namespace TodoApi.Models
{
    public class TodoItem
    {
        public long Id { get; set; }
        public string Name { get; set; }
        public bool IsComplete { get; set; }
    }
}
```

Web Api – Contexto Base de Datos

- Modelo – es la clase que coordina la funcionalidad entre el EntityFramework y el modelo de datos

```
using Microsoft.EntityFrameworkCore;

namespace TodoApi.Models
{
    public class TodoContext : DbContext
    {
        public TodoContext(DbContextOptions<TodoContext> options): base(options)
        {
         }

        public DbSet<TodoItem> TodoItems { get; set; }
    }
}
```

Web Api – Asociar contexto de Base de Datos

```
using Microsoft.AspNetCore.Builder;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using TodoApi.Models;

namespace TodoApi
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<TodoContext>(opt => opt.UseInMemoryDatabase("TodoList"));
            services.AddMvc();
        }

        public void Configure(IApplicationBuilder app)
        {
            app.UseMvc();
        }
    }
}
```

Web Api – Agregar un controlador

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using TodoApi.Models;
using System.Linq;

namespace TodoApi.Controllers
{
    [Route("api/[controller]")]
    public class TodoController : Controller
    {
        private readonly TodoContext _context;

        public TodoController(TodoContext context)
        {
            _context = context;

            if (_context.TodoItems.Count() == 0)
            {
                _context.TodoItems.Add(new TodoItem { Name = "Item1" });
                _context.SaveChanges();
            }
        }
    }
}
```

Web Api – Agregar un controlador

- Define una clase de controlador vacía. Aquí, agregaremos métodos para implementar la API.
- El constructor utiliza Inyección de dependencia para inyectar el contexto de la base de datos (TodoContext) en el controlador.
- El contexto de base de datos se utiliza en cada uno de los métodos CRUD en el controlador.
- El constructor agrega un elemento a la base de datos en memoria si no existe.

Web Api – Controlador – Métodos Get

```
[HttpGet]
public IEnumerable<TodoItem> GetAll()
{
    return _context.TodoItems.ToList();
}

[HttpGet("{id}", Name = "GetTodo")]
public IActionResult GetById(long id)
{
    var item = _context.TodoItems.FirstOrDefault(t => t.Id == id);
    if (item == null)
    {
        return NotFound();
    }
    return new ObjectResult(item);
}
```

Web Api – Controlador – Métodos Get

- `GET /api/todo`
- `GET /api/todo/{id}`

```
HTTP/1.1 200 OK
   Content-Type: application/json; charset=utf-8
   Server: Microsoft-IIS/10.0
   Date: Thu, 18 Jun 2015 20:51:10 GMT
   Content-Length: 82

   [{"Key":"1", "Name":"Item1","IsComplete":false}]
```

Web Api – Controlador – Create

```
[HttpPost]
public IActionResult Create([FromBody] TodoItem item)
{
    if (item == null)
    {
        return BadRequest();
    }

    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}
```

Web Api – Controlador – Create

```
[HttpPost]
public IActionResult Create([FromBody] TodoItem item)
{
    if (item == null)
    {
        return BadRequest();
    }

    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}
```

Web Api – Controlador – Put

```
[HttpPut("{id}")]
public IActionResult Update(long id, [FromBody] TodoItem item)
{
    if (item == null || item.Id != id)
    {
        return BadRequest();
    }

    var todo = _context.TodoItems.FirstOrDefault(t => t.Id == id);
    if (todo == null)
    {
        return NotFound();
    }

    todo.IsComplete = item.IsComplete;
    todo.Name = item.Name;

    _context.TodoItems.Update(todo);
    _context.SaveChanges();
    return new NoContentResult();
}
```

Web Api – Controlador – Delete

```
[HttpDelete("{id}")]
public IActionResult Delete(long id)
{
    var todo = _context.TodoItems.FirstOrDefault(t => t.Id == id);
    if (todo == null)
    {
        return NotFound();
    }

    _context.TodoItems.Remove(todo);
    _context.SaveChanges();
    return new NoContentResult();
}
```