# Servicios Web

## ▼ Primer Parcial

▼ **08-08-2022**

Docente: Arturo Barajas Saavedra

Vamos a construir tecnologías web con .NET

### Calificaciones

Se evalúa con un repositorio Git con el que se evalúa la materia

▼ **09-08-2022**

¿Qué es SOA?

Arquitectura Orientada a Servicios (SOA del inglés Service Oriented Architecture) es un estilo de arquitectura de TI que se apoya en la orientación a servicios.

La orientación a servicios es una forma de pensar en servicios, su construcción y sus resultados.

Los servicios no solo implican comunicación entre dos sistemas pares, sino que pueden usarse de distintas maneras para diferentes propósitos (ej. bancos)

Investigar: Back-End for Front-End

¿Qué es un servicio?

Es una representación lógica de una actividad de negocio que tiene un resultado de negocio específico (como comprobar el crédito de un cliente, obtener datos de clima, consolidar reportes de perforación)

### Arquitectura SOA

El estilo de arquitectura SOA se caracteriza por:

- Estar basadas en el diseño de servicios que reflejan las actividades del negocio en el mundo real, estas actividades forman parte de los procesos de negocio de la compañía.
- Representar los servicios utilizando descripciones de negocio para asignarles un contexto de negocio.
- Tener requerimientos de infraestructura específicos y únicos para este tipo de arquitectura, en general se recomienda el uso de estándares abiertos para la interoperabilidad y transparencia en la ubicación de servicios.
- Estar implementada de acuerdo con las condiciones específicas de la arquitectura de TI en cada compañía.
- Requerir un gobierno fuerte sobre la representación e implementación de servicios.
- Requerir un conjunto de pruebas que determinen que es un buen servicio.

Aunque ya es muy común usar RESTful aún hay quienes usan SOA.

---

Los 4 pilares de la programación orientada a objetos:

- Abstracción
- Encapsulamiento
- Herencia
- Polimorfismo

Variables para medir calidad de software:

- Tiempo
- Costo

- Alcance

### Manifiesto SOA

El desarrollo e implementación de una arquitectura SOA se rife por los principios descritos en el manifiesto SOA:

http://www.soa-manifesto.org/default.html

Agile manifesto:

https://agilemanifesto.org/

Por otro lado, la aplicación de la orientación a servicios se divide en 2 etapas:

1. Análisis orientado a servicios (Modelado de servicios)
2. Diseño orientado a servicios

## Diseño orientado a servicios

Cuenta con 8 principios de diseño que se aplican sobre cada uno de los servicios modelados, estos principios de diseño son:

- Contrato de servicios estandarizado
- Bajo acoplamiento
- Abstracción
- Reusabilidad
- Autonomía
- Sin estado
- Garantizar su descubrimiento
- Preparado para ser usando en composiciones

Serverless computing es una forma de decir que no se necesita de un servidor como tal para levantar el servicio.

Con Swagger es la forma de mostrar al mundo que yo tengo un servicio

▼ **10-08-2022**

Continuando con ayer

SOA y servicios web no son lo mismo.

Web services engloba varias tecnologías como XML, SOAP, WSDL, etc los cuales permiten construir soluciones de programación para mensajes específicos y para problemas de integración de aplicaciones.

En cambio SOA es una arquitectura de aplicación en la cual todas las funciones están definidas como servicios independientes con interfaces invocables que pueden ser llamados en secuencias bien definidas para tomar los procesos de negocio.

En SOA la clave está en la interfaz puesto que define los parámetros requeridos y la naturaleza del resultado. Esto significa que defina la naturaleza del servicio y no la tecnología utilizada. Esta función permite realizar dos de los puntos críticos: los servicios son realmente independientes y pueden ser manejados.

WS es el estándar apoyado por la industria, por empresas de distintos rubros, no tecnológicos, agrupados en un comité conocido como Web Services Interoperability (WS-I).

Este organismo tiene por principal objetivo asegurar que los grupos de trabajo que definen las especificaciones sobre WS utilizan estándares adecuado, a la vez que monitoriza el avance de sus trabajos, no define ni desarrolla estándares.

## SOA y los microservicios

Los microservicios son un una interpretación moderna de la arquitectura orientada a servicios usada para construir sistemas distribuidos.

Los servicios en una arquitectura de microservicios son procesos que se comunican con otros a través de una red para conseguir el objetivo final.

Estos servicios pueden usar protocolos simples como HTTP o REST.

## Diseño y desarrollo de SOA

La metodología de modelado y diseño para aplicaciones SOA se conoce como análisis y diseño orientado a servicios.

La arquitectura orientada a servicios es tanto un marco de trabajo para el desarrollo de software como un marco de trabajo de implementación

CI/CD (Continuous Integration/Continuous Deployment)

## Diseño y desarrollo de SOA

Cuando se habla de una arquitectura orientada a servicios, se habla de un juego de servicios residentes en internet o intranet. Existen diversas estándares relacionados con los servicios web, incluyendo los siguientes:

- XML
- JSON
- HTTP
- SOAP
- REST
- WSDL
- UDDI

## Beneficios de SOA

- Mejora en los tiempos de realización de cambios en procesos
- Facilidad para evolucionar a modelos de negocios basados en tercerización
- Facilidad para abordar modelos de negocios basados en colaboración con otros entes (socios, proveedores)
- Poder para reemplazar elementos de la capa aplicativa de SOA sin disrupción en el proceso de negocio
- Facilidad para la integración de tecnologías disimiles
- Mejora en la toma de decisiones
- Aplicaciones flexibles
- Aplicaciones reutilizables y adaptables
- Reducción de costes
- Riesgo de migración

## ¿Qué es un servicio web?

Es un sistema de software diseñado para soportar la interacción máquina a máquina a través de una red de forma interoperable. Cuenta con una interfaz descrita en un formato procesable por un equipo informático (específicamente en WSDL (Web Services Description Language)) a través de la que es posible interactuar con el mismo mediante el intercambio de mensajes SOAP, típicamente transmitidos usando serialización XML sobre HTTP conjuntamente con otros estándares web

Es decir un web service es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes y ejecutados sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como internet.

La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios web

▼ **11-08-2022**

## Diferencias entre Web Service y API

Una API puede ser cualquier interfaz de programación que exponga esta interfaz para comunicarse con ellos

## Arquitectura

En la arquitectura de servicios web existen tres partes:

1. El proveedor de servicios web
2. El que pide el servicio web
3. El publicador

Son tres nodos: el service provider, service requester y service provider

## Protocolos y estándares

Web services protocol stack

XML

SOAP

WSDL

UDDI

WS-Security

REST

GraphQL

## Ventajas

Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen

Los servicios web fomentan los estándares y protocolos basados en texto.

## Desventajas

No pueden compararse en su grado de desarrollo con los estándares aiertos de computación distribuida como CORBA

Su rendimiento es bajo si se compara con otros modelos de computación distribuida como RMI, CORBA o DCOM.

Al apoyarse en HTTP pueden esquivar medidas de seguridad basados en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera

---

Por eso surge REST porque SOA era muy pesado y es necesario transferir más información rápidamente

## La pila de protocolos de un servicio web

Los WS que se utilizan para definir, localizar, implementar y hacer que los servicios interactúen entre sí es XML

---

# API y Web API

Una representación gráfica de una API es como un multicontacto pues son muchas interfaces expuestas para que los distintos usuarios o programadores puedan consumir las distintas capacidades de una pieza de código

### API (Interfaz de programación de aplicaciones)

Es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción. Son usadas generalmente en bibliotecas de programación

Cualquier componente que se comunique con otro componente tiene una API.

Por ejemplo las impresoras son puras APIs

Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a servicios desde los procesos y representa un método para conseguir abstracción en la programación.

API consiste en proporcionar un conjunto de funciones de uso general como dibujar ventanas o íconos en pantalla

### Ejemplos de APIs

CORBA

Drupal API

Glibc

Microsoft Win-32 API

Microsoft WMI

ASPI

The Carbon APIs

Symfony

DirectX

OpenGL

▼ **12-08-2022**

Una web API es una API pero publicada en un servidor web que brinda acceso a un conjunto de rutinas que provee acceso a funciones de un determinado software.

Son publicadas por los constructores de software para permitir acceso a características de bajo nivel

Utilizadas por programados para construir sus aplicaciones sin necesidad de volver a programar funciones ya hechas por otros reusando código que se sabe que está probado y funciona correctamente

## Windows Communication Foundation

WCF está basada en SOA y es la plataforma de mensajería que forma parte de la API de la plataforma .NET 3

Fue creada para permitir programación rápida de sistemas distribuidos y el desarrollo de aplicaciones basadas en arquitecturas orientadas a servicios

## SOAP (Simple Object Access Protocol)

Protocolo estándar de acceso a objetos simples. Define cómo comunicarse por medio de intercambio de datos XML. Este protocolo se deriva del XML-RPC (este es la versión de Java y SOAP la versión de Microsoft)

SOAP actualmente está bajo el asupicio de la W3C

SOAP a diferencia de REST tiene más capacidad por usar más protocolos, usa diferentes redes de diferentes naturalezas y REST sólo trabaja con HTTP

## Características de SOAP

Paradigma de mensajería de una dirección sin estado (unidireccional= que puede usarse para formar protocolos más complejos. Pueden formar y construir la capa base de una pila de protocolos web services obreciendo un framework basado en XML con tres partes importantes:

1. Envelope: que define qué hay en el mensaje y cómo procesarlo
2. El conjunto de reglas de codificación para expresar instancias de tipos de datos
3. La convención para representar llamadas a procedimientos y respuestas

Las últimas dos son parte del WSDL

Tiene tres características principales:

- Extensibilidad (seguridad y WS-routing)
- Neutralidad (usando TCP sobre cualquier protocolo como HTTP, SMTP o JMS)
- Independencia (permite cualquier modelo de programación)

## Diferencia entre SOA y SOAP

SOA es una arquitectura de software de la cual derivan todos los servicios actualmente (servicios web, APIs, etc)

SOAP es un protocolo de comunicaciones

## Ventajas de SOAP

- Al usar XML se pueden invocar muchos procedimientos de muchos lenguajes
- Al usar HTTP es fácilmente escalable y ser casi siempre permitido por cortafuegos
- Puede implementarse usando cualquier lenguaje
- Se puede usar mediante usuario anónimo y autentificación
- Se puede transmitir mediante cualquier protocolo de transporte capaz de transmitir texto, típicamente HTTP o SMTP

## Desventajas de SOAP

- Al usar XML SOAP es más lento que otros middlewares como CORBA o como REST (pero en aquel entonces REST no existía) dado que los datos binarios se codifican como texto. Para contrarrestarlo se desarrolló un método optimizado de transmisión de mensajes
- Depende del WSDL (Web Services Description Language)
- Al contrario de Java, PHP o Python, algunos lenguajes no ofrecen apoyo adecuado para uso ya sea a nivel de integración o de soporte de IDE

▼ **16-08-2022**

## Arquitectura de microservicios

Importante ir conociendo qué es un patrón de diseño

Ver videos de YouTube:

- Christopher Okravi - Design Patterns in Object Oriented Programming

Hay una documentación de Microsoft sobre los estilos de arquitectura

Microsoft escribe en esta documentación estilos como Big Compute, Macrodatos, Arquitectura basada en eventos, Microservicios, Aplicación de n niveles y Web-cola trabajo

▼ **17-08-2022**

## Estilo de arquitectura de microservicios (Microsoft)

Una arquitectura de microservicios consta de una colección de servicios autónomoas y pequeños. Cada uno de los servicios es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado. Un contexto delimitado es una división natural de una empresa y proporciona un límite explícito dentro del cual existe un modelo de dominio

Una arquitectura monolítica mantiene una sola pieza que conecta de la UI, la lógica de negocio y la capa de acceso a la base de datos. Mientras que los microservicios permiten que la UI se conecte con distintos microservicios. Cada microservicio puede llamar a otro.

## Microservicio (AWS)

Son un enfoque artquitectónico y organizativo para el desarrollo de software donde está compuesto por pequeños servicios independientes que se comunican a través de la API bien definidas. Los propietarios de estos servicios son equipos pequeños e independientes.

Las arquitecturas de microservicios hacen que las aplicaciones sean más fáciles de escalar y más rápidas de desarrollar. Esto permite la innovación y acelera el tiempo de comercialización de las nuevas características

---

Normalmente se usa una arquitectura llamada Serverless computing. Aquí uno no debe preocuparse por montar un servidor. El servicio lo hará el proveedor y pueden usarse dos tipos de escaladores: Docker y Kubernetes.

Está el scale up y el scale down. Está el scale out y el scale in

---

## Microservicio (Microsoft)

Son pequeños e independientes y están acoplados de forma imprecisa. Un único equipo reducido de programadores puede escribir y mantener un servicio. Son autónomos y especializados.

La política es que tengan un solo origen de datos

Cada servicio es un código base independiente que puede administrarse de esta misma forma. Un equipo puede actualizar un servicio existente sin tener que volver a generar e implementar toda la aplicación

Los servicios son los responsables de conservar sus propios datos o estado externo. Esto difiere del modelo tradicional, donde una capa de datos independiente controla la persistencia de datos

Los servicios se comunican entre sí mediante APIs bien definidas. Los detalles de la implementación interna de cada servicio se ocultan frente a otros servicios

▼ **18-08-2022**

## Componentes que aparecen en una arquitectura típica de microservicios

1. Administración e implementación

   Este componente es el responsable de la colocación de servicios en los nodos, la identificación de errores, el reequilibrio de servicios entre nodos, etc. Normalmente este componente es una tecnología estándar como Kubernetes en lugar de algo creado de forma personalizada.

2. Puerta de enlace de API

   Es el punto de entrada para los clientes hacia los servicios. En lugar de llamar a los servicios directamente, los clientes llaman a la puerta de enlace de API que reenvía a los servicios apropiados en el BackEnd

   Puede usarse un single custom API Gateway Service o también multiple API Gateways / BFF (BackEnds For FrontEnds)

   Hay MVVM (modelo vista vista modelo) y MVC (modelo vista controlador

## Ventajas

- Agilidad

- Equipos pequeños y centrados
- Base de código pequeña
- Mezcla de tecnologías
- Aislamiento de errores
- Escalabilidad
- Aislamiento de los datos

## Desafíos

- Complejidad
- Desarrollo y pruebas
- Falta de gobernanza
- Congestión y latencia de red
- Integridad de datos
- Administración
- Control de versiones
- Serie de aptitudes

▼ **19-08-2022**

Arquitectura de microservicios con Kubernetes en Azure

## REST

Debemos conocer qué vamos a desarrollar. ¿Es realmente necesaria una web api?

¿Estamos construyendo un sitio web?

¿Estamos construyendo una single page app?

¿Estamos construyendo una app móvil?

Si no, ¿qué se está haciendo?

¿Cómo trabaja HTTP?

Una PC cliente pide información a una PC servidor. Eso es una request. Ésta trae un verbo, un header y un contenido. No necesariamente se debe usar el verbo requerido pero es un estándar

El servidor regresa una response compuesta de un status code, headers y contenido.

### El Request deconstruido

Los verbos son las acciones que se realizan en el servidor con los verbos:

- GET: Solicitar un recurso
- POST: Crear un recurso
- PUT: Actualizar un recurso
- PATCH: Actualizar parcialmente un recurso
- DELETE: Borrar un recurso

Después están los headers que son los metadatos del request

- Content type: Formato de contenido
- Content length: Tamaño de contenido

- Authorization: Quién está realizando la llamada
- Accept: Qué tipo(s) se pueden aceptar
- Cookies: Datos pasajeros en el request
- Más encabezados…

El contenido es lo que se leerá del recurso:

- HTML, CSS, JavaScript, XML, JSON
- Content no es válido con algunos verbos
- La información para ayudar a cumplir con el request
- Tipos binarios y blob comunes como jpg

### La response deconstruida

Estado de la operación

- 100: Informativo
- 200: Éxito
- 300: Redirección
- 400: Errores en el cliente
- 500: Errores en el servidor

Los headers igual:

- Content type: Formato de contenido
- Content length: Tamaño de contenido
- Expires: Cuando considerarla no fresca
- Accept: Qué tipo(s) se pueden aceptar
- Cookies: Datos pasajeros en el request
- Más encabezados…

Contenido de la response

- HTML, CSS, JavaScript, XML, JSON
- Tipos binarios y blob comunes como jpg

¿Qué es REST? La Transferencia de Estado Representacional es un estilo de arquitectura de software para sistemas hipermedia. El término se originó en el 2000 por una tesis doctoral sobre la web de Roy Fielding

REST se usa para describir cualquier interfaz que use HTTP sobre los datos en cualquier formato XML o JSON sin abstracciones adicionales como SOAP

RPC no es un ejemplo de REST. Es un antecesor

▼ **22-08-2022**

## REST frente a RPC/SOAP

Una aplicación web REST requiere un enfoque de diseño diferente a una aplicación basada en RPC (llamada de procedimiento remoto), por ejemplo, una aplicación RPC se deberían definir operaciones como:

- getUser()
- addUser()

- removeUser()
- updateUser()
- getLocation()
- addLocation()
- removeLocation()
- updateLocation()
- listUsers()
- listLocations()
- findLocation()
- findUser()

En REST, al contrario, el énfasis se pone en los recursos o sustantivos, especialmente en los nombres que se le asigna a cada tipo de recurso, por ejemplo en un enfoque REST podría definir algunos tipos de recursos asignándoles estos nombres:

- Usuario {}
  - GET https://www.server.com/api/v1/users
- Localización {}

## Conceptos REST

Arquitectura cliente - servidor

Ausencia de estado (el estado se mantiene en el cliente y no en el servidor)

Habilitación y uso de la caché (todas las solicitudes deben declarar si son o no cacheables)

Sistema por capas (tiene relación con la separación de responsabilidades anteriormente mencionada y un cliente debe conocer únicamente la capa a la que le está hablando)

Interfaz uniforme (identificación de recursos en peticiones, manipulación de recursos a través de representaciones, mensajes autodescriptivos e hipermedia como motor del estado de la aplicación usando HATEOAS es decir Hypermedia As The Engine Of Application State)

Demo 1: Una API bien construida: Abrir Postman y ejecutar los métodos GET de los siguientes endpoints:

- api.github.com/users/basaar24
- api.github.com/users/basaar24/repos

## RESTful API

Es una interfaz de programación de aplicaciones creada por Roy Fielding que se ajusta a los límites de la arquitectura REST

Sabemos que una RESTful API es una RESTful API porque está montada en HTTP, regresa JSONs y se enfoca a sustantivos y no verbos en las URIs

## ¿Qué aprendimos?

HTTP es crucial para las Web API

Usar REST es importante pero se debe ser pragmático al respecto

Diseñaremos una API sencilla para aprender los fundamentos

▼ **23-08-2022**

Ejemplo de Postman

La idempotencia es una cualidad de que se ejecute una operación sobre un objeto y éste no cambie

▼ **24-08-2022**

Diseñando una API

Existen las notaciones camelCase, PascalCase, snake_case y kebab-case

Query Parameters

## Formatos de diseño

Formatos más comunes

JSON (application/json)

XML (text/xml)

JSONP (application/javascript)

RSS (application/rss)

ATOM (application/atom)

▼ **25-08-2022**

No fui a clase

▼ **26-08-2022**

Temas de autentificación y autorización

Seguridad de dominio cruzado (ejemplo de cuando se levanta localhost, puese ese es un origen)

Tipos de autentificación: Cookies o OAUTH (Open Autenticator)

▼ **29-08-2022**

Descargar los recursos de Aula Virtual

Comando importante:

```
git remote prune origin
```

▼ **30-08-2022**

Revisión de temas sobre uso de GitHub, clonación de repos entre otras cosas

▼ **31-08-2022**

El primer paso para empezar a trabajar es crear nuestro proyecto de VS

Se creará el proyecto en donde descargamos el repositorio

Se va a buscar “api” y de las opciones que salgan, se elige la opción: “ASP .NET Core Web Api” con C#

Ingresamos nombre del proyecto junto con su dirección

El nombre del proyecto será “AppCitas.Service” y el nombre de la solución será “AppCitas”

Activamos que la solución y el proyecto estén en el mismo directorio

Se elige el Framework 6.0, ninguno en autenticación de campo, se configura para HTTPS, sin Docker, se usan los controladores y se habilita la compatibilidad con OpenAPI. La opción “do not use top level statements” también se marca

Para este punto, el proyecto ya está generado

Una vez ya con el proyecto creado, en la barra de la derecha se le da clic derecho a AppCitas.Service y se clickea sobre “Administrar paquetes NuGet” y ahí en orígenes del paquete se desmarcaría la primera opción y la segunda se deja marcada (en caso de que deje error)

El profesor cambió en launchSettings.json para profiles los puertos a 5001 y a 5000 y en schema lo cambio a 44344. En el puerto SSL se cambia a 44341

Para ejecutarlo se marca AppCitas.Service en el Play

Si se ejecutó bien debería abrir Swagger

Después se hace commit con Git

▼ **01-09-2022**

En la carpeta AppCitas.Service se pegarán los programas “program.cs” y “startup.cs” de los archivos descargados del profesor de Aula o Teams

launchSettings.json de la clase pasada está en la carpeta Properties

▼ **02-09-2022**

Avances en el repositorio de la semana

▼ **05-09-2022**

Avances en el repositorio (visible en el repositorio de GitHub)

Instalar el paquete desde NuGet Microsoft.EntityFrameworkCore.Design

Se instaló con los comandos:

```
cd C:\Users\alexe\OneDrive - Universidad Autónoma de Aguascalientes\Joul\Escolar\3 ICI\9\1 SW\Proyecto-1\AppCitas\AppCitas.Service
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

Luego en comando:

```
cd C:\Users\alexe\OneDrive - Universidad Autónoma de Aguascalientes\Joul\Escolar\3 ICI\9\1 SW\Proyecto-1\AppCitas\AppCitas.Service
dotnet ef migrations add InitialCreate -o Data/Migrations
```

▼ **06-09-2022**

Resolución de dudas

▼ **07-09-2022**

Correr

```
cd C:\Users\alexe\OneDrive - Universidad Autónoma de Aguascalientes\Joul\Escolar\3 ICI\9\1 SW\Proyecto-1\AppCitas\AppCitas.Service
dotnet ef database update
```

Podemos instalar el DB Browser for SQLite o un plugin para VS

Instalaremos el DB Browser:

https://sqlitebrowser.org/dl/

Se instala y se añade a la base de datos:

```
INSERT INTO Users(Id, UserName)
VALUES (1, "Bob");

INSERT INTO Users(Id, UserName)
VALUES (2, "Tom");
```

```
INSERT INTO Users(Id, UserName)
VALUES (3, "Jane");
```

¿Cuáno se justifica la creación de una variable? Cuando vamos a procesarla posteriormente

▼ **08-09-2022**

Haciendo el código asíncrono

▼ **09-09-2022**

No sé qué hicimos hoy D:

▼ **12-09-2022**

Empezamos por arreglar un bug de la clase pasada

▼ **13-09-2022**

Ayuda no sé qué está haciendo :(

El registro ya quedó hecho

Estamos haciendo ahora el login

DTO = Data Transfer Object

▼ **14-09-2022**

Revisión de dudas

▼ **15-09-2022**

Revisión de dudas

# ▼ Segundo Parcial

▼ **19-09-2022**

Vamos a implementar JSON Web Tokens a la aplicación

Junto con toda la escritura de código nos fuimos al administrador de paquetes de nuget y elegimos la librería:

```
system.identitymodel.tokens.jwt
```

Desde consola puede usarse el comando (dentro de la carpeta de AppCitas.Service):

```
dotnet add package System.IdentityModel.Tokens.Jwt --version 6.23.1
```

Más cosas de JWT

▼ **20-09-2022**

Más cambios en código

Se instaló después Microsoft.AspNetCore.Authentication.JwtBearer

▼ **21-09-2022**

Más añadidos a código

▼ **22-09-2022**

Más avances de código

▼ **23-09-2022**

No sé

▼ **26-09-2022**

Página útil:

> https://json-generator.com/

Copiamos 10mil JSONs generados con la herramienta anterior

Damos clic derecho en Data y agregaremos un JSON File: UserSeedData.json

▼ **27-09-2022**

Más avances

▼ **28-09-2022**

Corrección a cosas de código

▼ **29-09-2022**

Más avances de código

▼ **30-09-2022**

Más avances de código

▼ **03-10-2022**

No hubo clases por el congreso de ICI

▼ **04-10-2022**

No hubo clases por el congreso de ICI

▼ **05-10-2022**

No hubo clases por el congreso de ICI

▼ **06-10-2022**

Más avances de código

▼ **07-10-2022**

Más avances de código

▼ **10-10-2022**

Más avances de código

▼ **11-10-2022**

Más avances de código

▼ **12-10-2022**

No hubo clase

▼ **13-10-2022**

No hubo clase

▼ **14-10-2022**

No hubo clase

▼ **17-10-2022**

Más cambios de código

▼ **18-10-2022**

Avances en el código

▼ **19-10-2022**

Avances en el código

▼ **20-10-2022**

Avances en el código

▼ **21-10-2022**

Avances en el código

▼ **24-10-2022**

Avances en el código

▼ **25-10-2022**

Avances en el código

▼ **26-10-2022**

Avances en el código

▼ **27-10-2022**

Avances en el código

▼ **28-10-2022**

Avances en el código

# ▼ Tercer Parcial

▼ **31-10-2022**

Avances en el código

Un comando que se usa en lo que hicimos hoy es:

```
dotnet ef migrations add LikedEntityAdded
```

▼ **01-11-2022**

Avances de código

▼ **03-11-2022**

Avances de código

▼ **04-11-2022**

Avances de código

▼ **07-11-2022**

Avances de código

▼ **08-11-2022**

Más avances de código

▼ **09-11-2022**

Más avances de código

▼ **10-11-2022**

Más avances de código

▼ **11-11-2022**

Más avances de código

▼ **14-11-2022**

No entré a clases

▼ **15-11-2022**

No entré a clases

▼ **16-11-2022**

No entré a clases

▼ **17-11-2022**

No entré a clases

▼ **18-11-2022**

No entré a clases

▼ **22-11-2022**

No entré a clases

▼ **23-11-2022**

No entré a clases

▼ **24-11-2022**

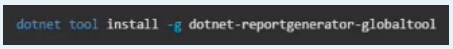
No entré a clases

▼ **25-11-2022**

Comandos proyecto final:

Sobre AppCitas.UnitTests se ejecuta:


Instalamos:




Sobre el archivo xml corremos: