UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Objetivos de la sesión:

- *Aprender a **diferenciar entre servicios REST y servicios SOAP**.*
- ***Crear un proyecto API RESTful** desde cero.*
- *Practicar como crear **entidades JPA e inicializar la Base de datos**.*
- *Practicar como crear un **repositorio JPA**.*
- *Crear la **capa de servicios y del controlador de un servicio RESTFul**.*
- *Aprender a **realizar prueba desde VSCode con REST client**.*

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

¿ Que son los servicios REST?¿Que son los servicios SOAP? ¿Cual es su diferencia?

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Servicios REST (API RESTful) vs servicios SOAP (webservices)

REST es el acrónimo de **Representational State Transfer**. Es un estilo arquitectónico de software que utiliza el protocolo HTTP para obtener datos y/o operar sobre esos datos que cumplen un conjunto de restricciones o reglas que deberíamos seguir para crear una API.

Un **protocolo cliente/servidor sin estado** es un protocolo en el que se realiza una conexión con el servidor, se solicita un dato o se realiza una operación y luego se desconecta del servidor.

Cuando ofrecemos un **API de acceso a datos con un protocolo cliente/servidor sin estado** utilizando HTTP y nos ajustamos a ciertas restricciones y/o reglas, nuestra API podemos considerarla **API RESTful**. Comunmente se suele hacer referencia a las API RESTful como **servicios REST**.

Si utilizamos un **protocolo cliente/servidor en el que no desconectamos** del servidor (tenemos la comunicación abierta para enviar/recibir pero consumimos recursos) hablamos de **servicios SOAP**, comunmente llamados **Webservices**.

¡Los servicios SOAP han sido desplazados en su mayoría por servicios REST!

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Para utilizar las API RESTful nos valemos de los **endpoints**, o lo que es lo mismo, las URLs de un API o un backend que responden a una petición.

¿ Por donde empezamos para crear un API RESTful? ¿Tenemos pasos similares a los de la aplicación de la práctica anterior (CommandLineRunner con CRUD) ?

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Si entendimos como funcionaba la aplicación de la práctica anterior de Acceso a datos , crear un servicio REST no es muy diferente.

Los pasos a seguir para crear un API RESTful son muy similares a la práctica anterior. Marcamos en gris los pasos similares y en negro los nuevos (aunque veremos que no son tan nuevos para nosotros) :

1º Crear el proyecto Spring Boot
2º Configurar la Base de Datos
3º Crear las entidades JPA
4º Inicializar con datos la BD
5º Crear el repositorio para cada entidad JPA
6º Crear la capa de servicios
7º Crear la capa de control (controller)
8º Pruebas

1º Crear el proyecto Spring Boot:

Para empezar vamos a crear nuestro primer servicio REST llamado proyecto "dwes_primer_rest" a imagen y semejanza del proyecto anterior:

Java Projects\Añadir (+) \Spring Boot \Proyecto Maven \ versión 2.7.7 \Java

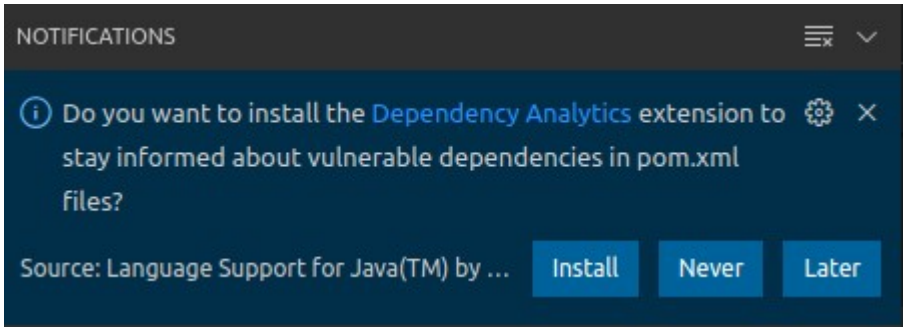
e introducimos los siguientes datos:

Group Id: edu.alumno.NombreAlumno
Artifact Id: dwes_primer_rest
Package: war
Versión de Java: 11

Dependencias:

Deberemos seleccionar "Spring Web" ,"H2 Database" ,"Spring Data JPA", "Lombok" **y añadir "DevTools" y "Validation".**

Añadimos el proyecto al WorkSpace y de manera opcional le podemos decir que instale "Dependency Analytics" si no lo tenemos ya instalado.



UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación 1º Crear el proyecto Spring Boot:

Para continuar deberemos de copiar la dependencia de MapStruct (podemos utilizar la del proyecto de la UD2) en el fichero pom.xml:

pom.xml

dwes_primer_rest > pom.xml > project > dependencies

```
56          </dependency>
57          <dependency>
58              <groupId>org.springframework.boot</groupId>
59              <artifactId>spring-boot-starter-validation</artifactId>
60          </dependency>
61          <!--  DTOs  -->
62          <dependency>
63              <groupId>org.mapstruct</groupId>
64              <artifactId>mapstruct</artifactId>
65              <version>${org.mapstruct.version}</version>
66              <scope>compile</scope>
67          </dependency>
68      </dependencies>
69
70      <build>
```

Fijate que te hará falta añadir la declaración de la propiedad {org.mapstruct.version'}, pero podemos aprovechar para añadir el resto de propiedades y actualizar valores (versión de java 11 y de lombok 1.18.24):

pom.xml

pom.xml

```
14      <packaging>war</packaging>
15      <properties>
16          <java.version>11</java.version>
17          <org.projectlombok.version>1.18.24</org.projectlombok.version>
18          <org.mapstruct.version>1.4.1.Final</org.mapstruct.version>
19      </properties>
```

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación 1º Crear el proyecto Spring Boot:

Y también deberemos añadir el plugin del compilador de maven (podemos utilizar el del proyecto de la UD2) en el fichero pom.xml:

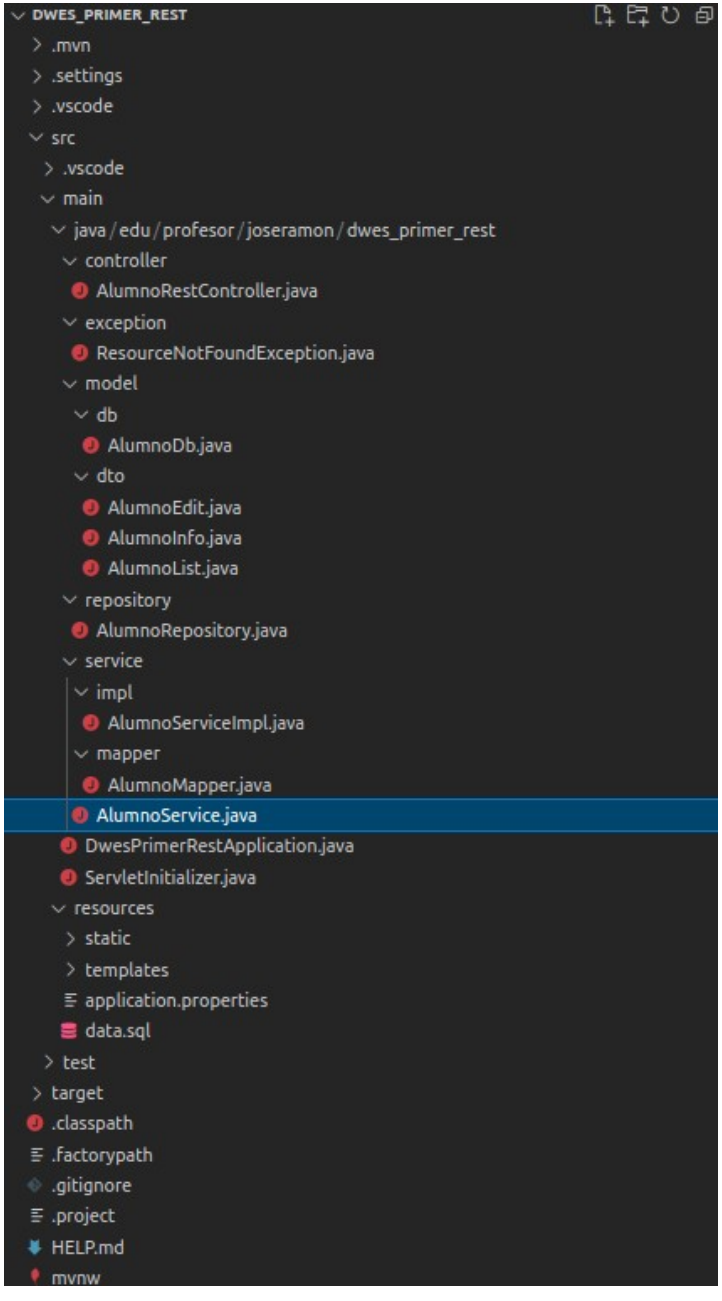
pom.xml

dwes_primer_rest > pom.xml > project > build > plugins > plugin

```
83          </plugin>
84          <plugin>
85              <groupId>org.apache.maven.plugins</groupId>
86              <artifactId>maven-compiler-plugin</artifactId>
87              <version>3.8.1</version>
88              <configuration>
89                  <verbose>true</verbose>
90                  <source>1.8</source>
91                  <target>1.8</target>
92                  <encoding>UTF-8</encoding>
93                  <showWarnings>true</showWarnings>
94                  <annotationProcessorPaths>
95                      <path>
96                          <groupId>org.projectlombok</groupId>
97                          <artifactId>lombok</artifactId>
98                          <version>${org.projectlombok.version}</version>
99                      </path>
100                     <!-- Necesario para que Lombok funcione con MapStruct -->
101                     <path>
102                         <groupId>org.projectlombok</groupId>
103                         <artifactId>lombok-mapstruct-binding</artifactId>
104                         <version>0.2.0</version>
105                     </path>
106                     <path>
107                         <groupId>org.mapstruct</groupId>
108                         <artifactId>mapstruct-processor</artifactId>
109                         <version>${org.mapstruct.version}</version>
110                     </path>
111                     <!-- other annotation processors -->
112                 </annotationProcessorPaths>
113             </configuration>
114         </plugin>
115     </plugins>
```

... continuación 1º Crear el proyecto Spring Boot:

Para tener una visión global de como quedará nuestro proyecto se muestra una captura de pantalla donde veremos las clases que acabaremos creando a continuación:



2º Configurar la Base de Datos:

Podemos copiar y pegar de la práctica anterior la configuración de la base de datos en “application.properties” del proyecto anterior:

application.properties

src > main > resources > application.properties

```
#Activar consola para acceder a la BD H2 via navegador
# localhost:puertoConfigurado/h2-console
spring.h2.console.enabled=true
#Configuración de la BD H2
spring.datasource.url=jdbc:h2:mem:testDwesDb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=dwes
spring.datasource.password=Simarro@1
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

#En la versión 2.4.2 no hace falta, pero en la
# 2.6.2 hace falta para poder hacer inserts en data.sql
spring.jpa.hibernate.ddl-auto=none

#CONFIGURACIÓN SOLO durante las pruebas:
# Habilitar estadísticas hibernate
spring.jpa.properties.hibernate.generate_statistics=true
# Habilitar LOGGER de las estadísticas de hibernate
logging.level.org.hibernate.stat=
# Mostrar que consultas esta realizando Hibernate
spring.jpa.show-sql=true
# Formatear las consultas
spring.jpa.properties.hibernate.format_sql=true
# Mostrar los parametros que estan enviandose a las consultas
logging.level.org.hibernate.type=debug
#FIN CONFIGURACIÓN SOLO durante las pruebas
```

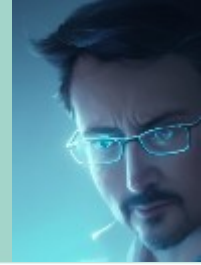
3º Crear las entidades JPA:

En esta sección empezaremos por **crear la entidad JPA "AlumnoDb"**. Puedes copiar y pegar la clase del fichero comprimido del DRIVE con las fuentes de esta práctica. Fijate en las variaciones en algunos atributos y que hay atributos que estaban en la UD2 que de momento no vamos a poner:

AlumnoDb.java

dwes_primer_rest > src > main > java > edu > profesor > joseramon > dwes_primer_rest > model > db > AlumnoDb.java > ...

```
2
3  import java.io.Serializable;…
20
21 @NoArgsConstructor
22 @AllArgsConstructor
23 @Data
24 @Entity
25 @Table(name = "alumnos")
26 public class AlumnoDb implements Serializable{
27     private static final long serialVersionUID = -818542778373595260L;
28     @Id
29     @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener 8 números y u
30     private String dni;
31     @Size(min=5,message="El nombre debe de tener un tamaño mínimo de 5 carácteres")
32     private String nombre;
33     @NotNull(message = "La edad no puede estar vacia")
34     @Range(min = 18, max = 99, message = "La edad debe ser igual o mayor a 18 y menor o
35     @Digits(integer = 2,fraction = 0, message = "La edad no puede tener decimales ni má
36     private Integer edad;
37     private String ciclo;
38     @NotNull(message = "El curso no puede estar vacio")
39     @Digits(fraction = 0, integer = 1,message = "El curso tiene un formato incorrecto")
40     @Range(min = 1, max = 2, message = "El curso solo admite los valores 1 o 2")
41     private Integer curso;
42     private boolean erasmus=false;
43     private String lenguajeFavorito="";
44     private String genero;
45     private String horario;
46     private String pais;
47     private String hobbies;
48 }
```



4º Inicializar con datos la BD:

Al igual que el proyecto anterior, para inicializar la Base de Datos H2 hay que crear el fichero “data.sql” en la carpeta “src\main\resources” y copiar las sentencias siguientes que también están en el DRIVE:

data.sql

src > main > resources > data.sql

```sql
-- Crear la tabla alumnos
DROP TABLE IF EXISTS alumnos;
CREATE TABLE alumnos (
dni VARCHAR(9),
nombre VARCHAR(50) NOT NULL,
edad INT ,
ciclo VARCHAR(5),
curso INT ,
erasmus BOOLEAN DEFAULT FALSE,
lenguaje_Favorito VARCHAR(20),
genero VARCHAR(1),
horario VARCHAR(1),
pais VARCHAR(5),
hobbies VARCHAR(100),
CONSTRAINT pk_alumnos PRIMARY KEY(dni));

--Rellenar la tabla alumnos con algunos valores iniciales
insert into alumnos(dni,nombre,edad,ciclo,curso,erasmus,lenguaje_Favorito,genero,horario,pais,hobbies)
    values('11111111A','Jose Garcia',21,'DAM',1,FALSE,'Java','H','M','ES',' ');
insert into alumnos(dni,nombre,edad,ciclo,curso,erasmus,lenguaje_Favorito,genero,horario,pais,hobbies)
    values('22222222B','Maria Gonzalez',32,'DAW',2,TRUE,'Java','M','T','ES',' ');
insert into alumnos(dni,nombre,edad,ciclo,curso,erasmus,lenguaje_Favorito,genero,horario,pais,hobbies)
    values('33333333C','Pedro Martínez',43,'DAW',2,FALSE,'Java','H','M','ES',' ');
```

5º Crear el repositorio para cada entidad JPA:

Para hacer **CRUD (Create Read Update Delete)** sobre la **tabla "Alumnos"** volvemos a utilizar la interface JpaRepository de Spring esta vez sobre un indice de tipo String:

AlumnoRepository.java

dwes_primer_rest > src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java >

```java
package edu.profesor.joseramon.dwes_primer_rest.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import edu.profesor.joseramon.dwes_primer_rest.model.db.AlumnoDb;

@Repository
public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{

}
```

6º Crear la capa de servicios:

Para crear la capa de servicios vamos a crear primero una interface con los servicios que se ofrecerán y luego implementaremos una clase que rellene dichos métodos de la interface. Se puede copiar esta interface y los datos del DRIVE:

AlumnoService.java

dwes_primer_rest > src > main > java > edu > profesor > joseramon > dwes_primer_rest > service > A

```java
package edu.profesor.joseramon.dwes_primer_rest.service;

import java.util.List;
import java.util.Optional;

import edu.profesor.joseramon.dwes_primer_rest.model.dto.AlumnoEdit;
import edu.profesor.joseramon.dwes_primer_rest.model.dto.AlumnoInfo;
import edu.profesor.joseramon.dwes_primer_rest.model.dto.AlumnoList;

public interface AlumnoService {
    public Optional<AlumnoEdit> getAlumnoEditByDni(String dni);
    public Optional<AlumnoInfo> getAlumnoInfoByDni(String dni);
    public AlumnoEdit save(AlumnoEdit alumnoEdit);
    public String deleteByDni(String dni);
    public AlumnoEdit update(AlumnoEdit alumnoEdit);
    public List<AlumnoList> findAllAlumnoList();
}
```

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación 6º Crear la capa de servicios:

El alumno debe copiar del DRIVE la clase y completar los métodos no implementados:

getAlumnoInfoByDni: Similar a getAlumnoEditByDni

save: Convertir el alumnoEdit a AlumnoDb, guardarlo, convertir el alumnoDb guardado a AlumnoEdit y devolverlo.
Ayuda:
alumnoRepository.save devuelve el AlumnoDb guardado.

update: buscar el AlumnoDb, si existe modificarlo y devolver el AlumnoDb convertido a AlumnoEdit. Si no existe devolver un optional vacio.

findAllAlumnoList: devolver la lista de AlumnoDb convertidos a AlumnoList.

AlumnoServiceImpl.java

```
> dwes_primer_rest > service > impl > AlumnoServiceImpl.java > Language Support for Java(TM) by Red Hat > AlumnoServiceImpl >
16
17      private final AlumnoRepository alumnoRepository;
18
19      @Autowired
20      public AlumnoServiceImpl(AlumnoRepository alumnoRepository) {
21          this.alumnoRepository = alumnoRepository;
22      }
23
24      @Override
25      public Optional<AlumnoEdit> getAlumnoEditByDni(String dni) {
26          Optional<AlumnoDb> alumnoDb=alumnoRepository.findById(dni);
27          if (alumnoDb.isPresent())
28              return Optional.of(AlumnoMapper.INSTANCE.alumnoDbToAlumnoEdit(alumnoDb.get()));
29          else
30              return Optional.empty();
31      }
32
33      @Override
34  >   public Optional<AlumnoInfo> getAlumnoInfoByDni(String dni) {…
41
42      @Override
43  >   public AlumnoEdit save(AlumnoEdit alumnoEdit) {…
47
48      @Override
49      public String deleteByDni(String dni) {
50          return alumnoRepository.findById(dni)
51          .map(a-> {
52                  alumnoRepository.deleteById(dni);
53                  return "Deleted";
54              }).orElse("Not Deleted");
55      }
56
57      @Override
58  >   public Optional<AlumnoEdit> update(AlumnoEdit alumnoEdit) {…
68
69      @Override
70  >   public List<AlumnoList> findAllAlumnoList() {…
73  }
```

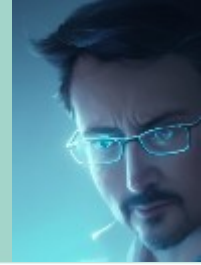
7º Crear la capa de control (controller):

Antes de implementar el controller paremonos a analizar que funcionalidad ofreceremos con el servicio REST:

Path	Método Http	Procedimiento	Código	Descripción
/ api / v1 /alumnos	GET	getAlumnosList	200 (OK)	Devuelve una lista de AlumnosList
/ api / v1 /alumnos	POST	newAlumnoEdit	201 (Creado)	Se ha creado un nuevo Alumno
/ api / v1 /alumnos / {dni}	GET	getAlumnoEditByDni	200 (OK)	Devuelve un alumnoEdit
/ api / v1 /alumnos / {dni}/info	GET	getAlumnoInfoByDni	200 (OK)	Devuelve un alumnoInfo
/ api / v1 /alumnos / {dni}	PUT	updateAlumnoEdit	200 (OK)	Se ha modificado el Alumno
/ api / v1 /alumnos / {dni}	DELETE	deleteByDni	204 (Sin Contenido)	Se ha borrado el Alumno

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



... continuación 7º Crear la capa de control (controller):

Para crear el controller se proporciona una plantilla en el DRIVE en la que se deberá de completar algunos métodos con las pistas de la próxima página.

Antes fijemonos que mediante **@RestController** convertimos nuestro controller en un servicio REST que devuelve JSONs.

El **@RequestMapping** inicial permite indicar el comienzo de la URL para todas las peticiones.

Podemos simplificar código con **@GetMapping, @PostMapping, @PutMapping** y por último **@DeleteMapping**.

AlumnoRestController.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > controller > AlumnoRestController.java > ...

```
@RestController
@RequestMapping("/api/v1/")
public class AlumnoRestController {

    private AlumnoService alumnoService;

    @Autowired
    public AlumnoRestController(AlumnoService alumnoService) {
        this.alumnoService=alumnoService;
    }

    @GetMapping("/alumnos")
    public List<AlumnoList> getAlumnosList() {…

    @PostMapping("/alumnos")
    public AlumnoEdit newAlumnoEdit(@Valid @RequestBody AlumnoEdit alumnoEdit) {
        return alumnoService.save(alumnoEdit);
    }

    @GetMapping("/alumnos/{dni}")
    public ResponseEntity<AlumnoEdit> getAlumnoEditByDni(
    @PathVariable(value = "dni") String dni) throws RuntimeException {
        Optional<AlumnoEdit> alumnoEdit = alumnoService.getAlumnoEditByDni(dni);
        if (alumnoEdit.isPresent())
            return ResponseEntity.ok().body(alumnoEdit.get());
        else throw new ResourceNotFoundException("Alumno not found on :: "+ dni);
    }

    @GetMapping("/alumnos/{dni}/info")
    public ResponseEntity<AlumnoInfo> getAlumnoInfoByDni(…

    @PutMapping("/alumnos/{dni}")
    public ResponseEntity<AlumnoEdit> updateAlumnoEdit(…

    @DeleteMapping("/alumnos/{dni}")
    public String deleteByDni(@PathVariable(value = "dni") String dni) {
        return alumnoService.deleteByDni(dni);
    }
}
```

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación 7º Crear la capa de control (controller):

getAlumnosList:
Utilizamos el servicio para devolver la lista de todos los alumnos (AlumnoList).

getAlumnoInfoByDni:
Muy similar a getAlumnoEditByDni.

updateAlumnoEdit:
Recuperaremos un 'alumnoEditoriginal' para saber si realmente existe o no. Si existe retornamos el alumnoedit modificado utilizando el servicio para realizar la Modificación. Si no existe podemos lanzar una RuntimeException informando del error.

AlumnoRestController.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > controller > AlumnoRestController.java > ...

```
@RestController
@RequestMapping("/api/v1/")
public class AlumnoRestController {

    private AlumnoService alumnoService;

    @Autowired
    public AlumnoRestController(AlumnoService alumnoService) {
        this.alumnoService=alumnoService;
    }

    @GetMapping("/alumnos")
    public List<AlumnoList> getAlumnosList() {…

    @PostMapping("/alumnos")
    public AlumnoEdit newAlumnoEdit(@Valid @RequestBody AlumnoEdit alumnoEdit) {
        return alumnoService.save(alumnoEdit);
    }

    @GetMapping("/alumnos/{dni}")
    public ResponseEntity<AlumnoEdit> getAlumnoEditByDni(
    @PathVariable(value = "dni") String dni) throws RuntimeException {
        Optional<AlumnoEdit> alumnoEdit = alumnoService.getAlumnoEditByDni(dni);
        if (alumnoEdit.isPresent())
            return ResponseEntity.ok().body(alumnoEdit.get());
        else throw new ResourceNotFoundException("Alumno not found on :: "+ dni);
    }

    @GetMapping("/alumnos/{dni}/info")
    public ResponseEntity<AlumnoInfo> getAlumnoInfoByDni(…

    @PutMapping("/alumnos/{dni}")
    public ResponseEntity<AlumnoEdit> updateAlumnoEdit(…

    @DeleteMapping("/alumnos/{dni}")
    public String deleteByDni(@PathVariable(value = "dni") String dni) {
        return alumnoService.deleteByDni(dni);
    }
}
```

8º Pruebas:

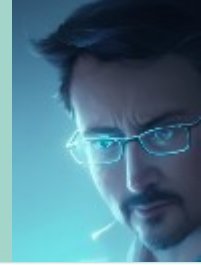
El alumno puede escoger el sistema de pruebas :

- Consulta desde el navegador (Solo para GETs)
- Utilizar Postman
- Utilizar la extensión de VSCode “REST Client”
  (Más información en enlace)

En la próxima página se explica como utilizar “REST Client” y se proporciona una plantilla de peticiones de prueba para testear el servicio REST.

UD 3: Bases de datos y servicios REST
4.- API RESTful

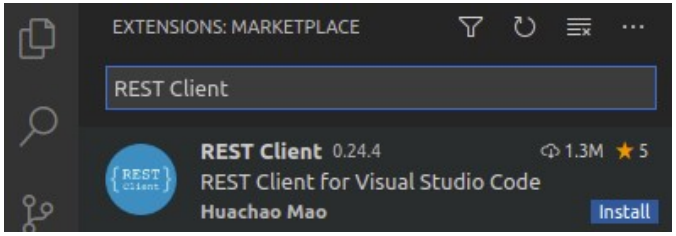
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



… continuación 8º Pruebas:

Vamos a detallar como utilizar la **extensión de VSCODE "REST Client"**:

Debemos instalar REST Client:



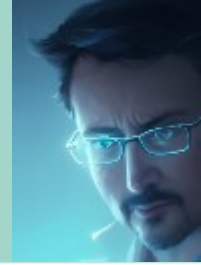

```
test_Rest.http ×
dwes_primer_rest > test_Rest.http > PUT /api/v1/alumnos/44444444A
1   ## getAlumnosList
    Send Request
2   GET http://localhost:8080/api/v1/alumnos HTTP/1.1
3   Content-Type: application/json
4   ###
5   ## newAlumnoEdit
    Send Request
6   POST http://localhost:8080/api/v1/alumnos HTTP/1.1
7   Content-Type: application/json
8
9   {
10      "dni": "44444444A",
11      "nombre": "Alumno añadido",
12      "edad": 44,
13      "ciclo": "DAM",
14      "curso": 2,
15      "erasmus": false,
16      "modificado": null,
17      "erasmusChecked": ""
18  }
19  ###
```

Para realizar pruebas podemos crear un fichero con extensión ".http" en la carpeta raiz de nuestro proyecto (donde esta el pom.xml).

Se puede copiar del DRIVE el fichero "test_Rest.http" :

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



... continuación 8º Pruebas:

Para realizar pruebas tan solo debemos de arrancar la webapp y pulsar sobre Send Request y se ejecutará la petición correspondiente:

test_Rest.http

dwes_primer_rest > test_Rest.http > PUT /api/v1/alumnos/44444444A

```
35  ##  getAlumnoEditByDni
    Send Request
36  GET http://localhost:8080/api/v1/alumnos/11111111A HTTP/1.
37  Content-Type: application/json
38  ###
39  ##  getAlumnoEditByDni: Alumno no existe
    Send Request
40  GET http://localhost:8080/api/v1/alumnos/8A HTTP/1.1
41  Content-Type: application/json
42  ###
43  ##  getAlumnoInfoByDni
    Send Request
44  GET http://localhost:8080/api/v1/alumnos/11111111A/info HT
45  Content-Type: application/json
46  ###
47  ##  getAlumnoInfoByDni: Alumno no existe
    Send Request
48  GET http://localhost:8080/api/v1/alumnos/8A/info HTTP/1.1
49  Content-Type: application/json
50  ###
51  ## updateAlumnoEdit
```

Response(77ms)

```
1  HTTP/1.1 200
2  Content-Type: application/json
3  Transfer-Encoding: chunked
4  Date: Tue, 16 Feb 2021 18:52:52 GMT
5  Connection: close
6
7  {
8    "dni": "11111111A",
9    "nombre": "Jose Garcia",
10   "edad": 21,
11   "ciclo": "DAM",
12   "curso": 1,
13   "erasmus": false,
14   "lenguajeFavorito": "Java",
15   "genero": "H",
16   "horario": "M",
17   "pais": "ES",
18   "hobbies": " "
19 }
```

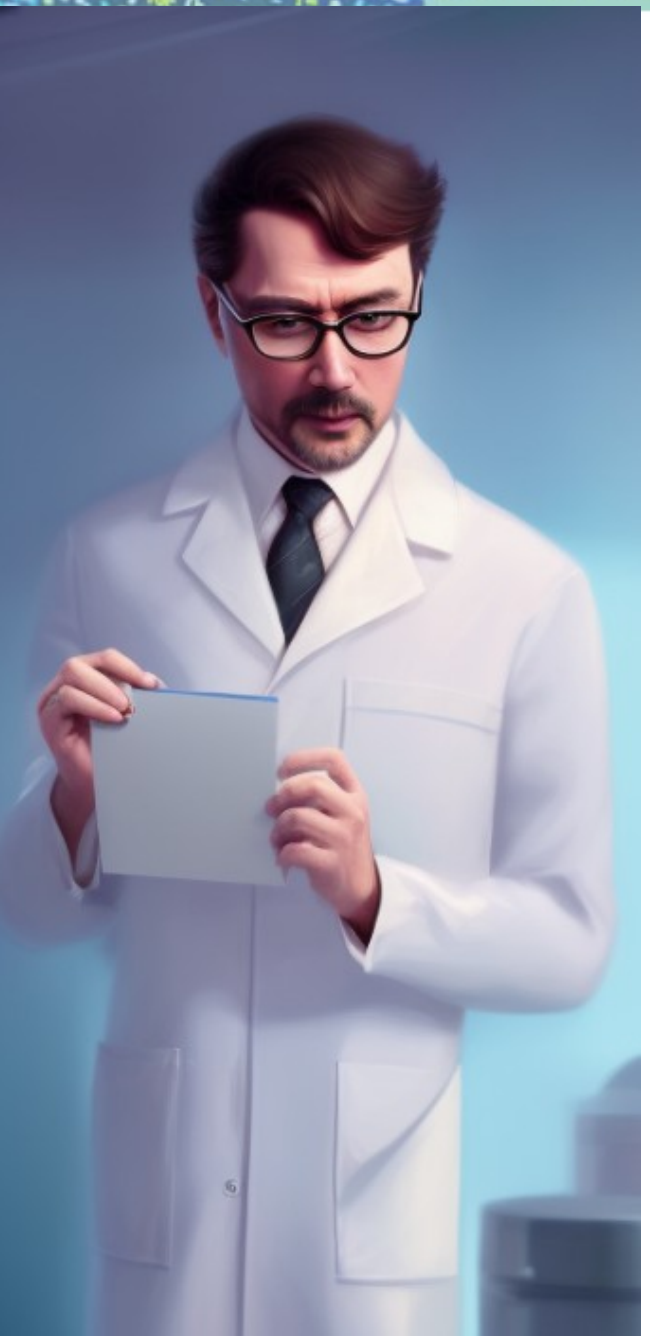
Aclaraciones:

· En un mismo fichero se puede tener más de una petición separadas por el simbolo ###.

· No debemos de preocuparnos por el formateo de los mensajes de error cuando los provocamos. Más adelante veremos como darle forma y como devolver el código de estado correspondiente (200,201,204,...)

UD 3: Bases de datos y servicios REST
4.- API RESTful

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



EJERCICIO:

Sube la aplicación final al moodle.

Para ello:

1º Haz un “Run As \Maven Clean” para dejar solo los fichero fuentes y quitar momentaneamente los necesarios para ejecutar la aplicación (dependencias).

2º Comprime la carpeta de tu aplicación y ponle como nombre UD3_practica3_nombreAlumno.tar.gz al fichero comprimido donde nombreAlumno es el nombre del alumno que entrega la práctica.

3º Súbela al moodle.

IMPORTANTE: No comprimir en RAR, porque Ubuntu no lo lee bien y en clase tenemos Ubuntu. Si tuviesemos Windows, podemos comprimir en ZIP.