

# Trabajo Práctico

## Desarrollo de Aplicaciones Cliente-Servidor

### Historia Clínica

#### Integrantes:

AGUILERA , Diego Leonardo

BENITEZ PERESSI , Gonzalo

ESCALANTE , Salvador Manuel

GAMARRA , María Luciana

VIDAL , Raul Antonio

xgo.237@gmail.com

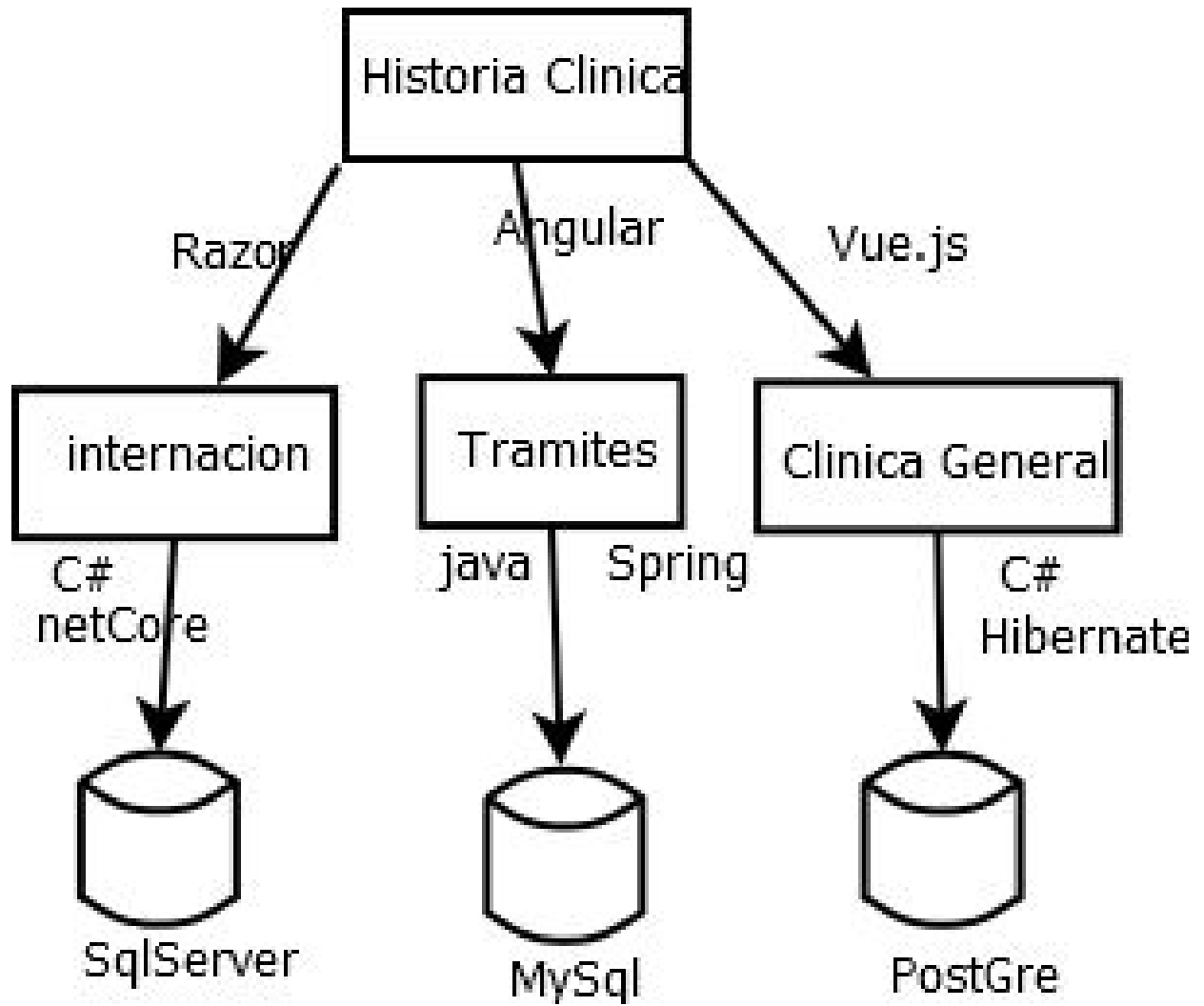
gonzaloperessi@gmail.com

salvamanu12@gmail.com

luchy.isi@gmail.com

tankraul@gmail.com

# Esquema General del Proyecto



## Trámites y Exámenes

Se intentará modelar el camino seguido en el desarrollo de la aplicación, usando los principios recomendados para el desarrollo aprendidos en el transcurso de la materia.

Lenguajes usados: Java, Typescript.

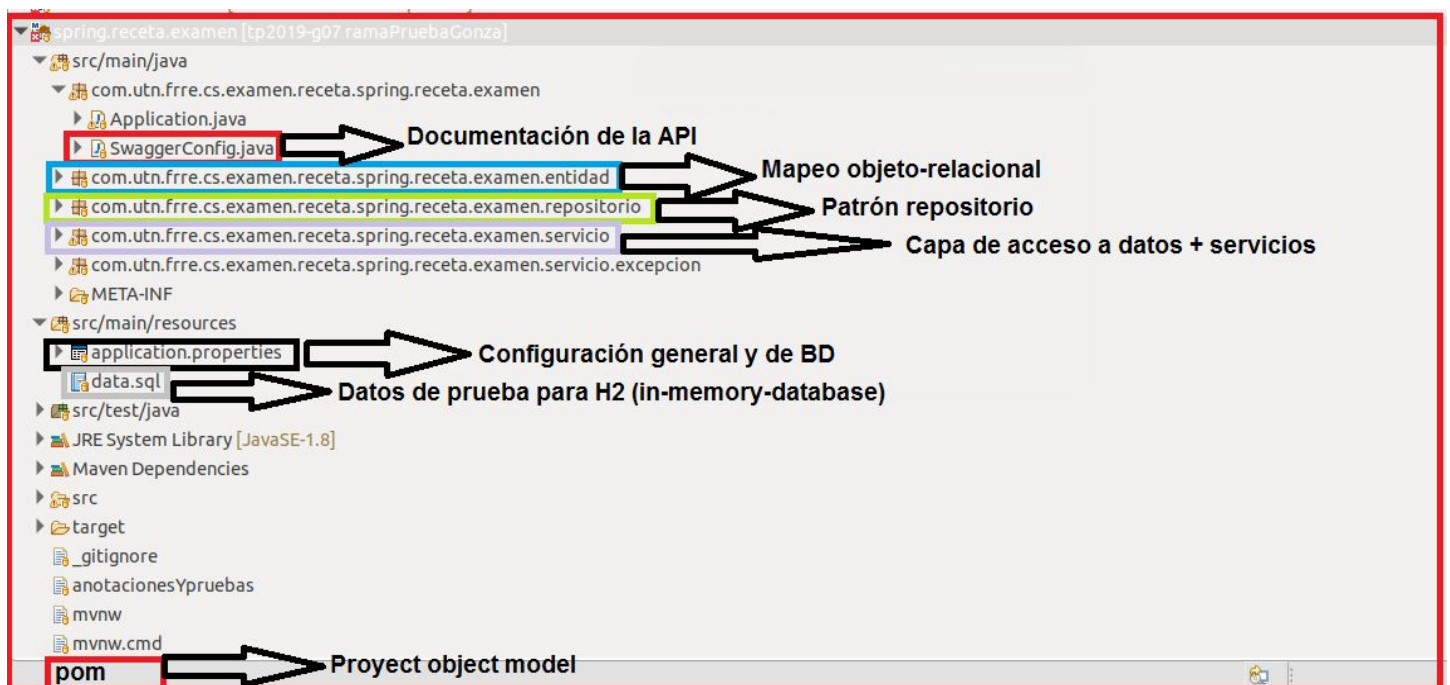
Framework: Spring.

BackEnd: h2-mySql.

FrontEnd: Angular 7.

web server: apache-nginx

### Estructura del proyecto



### Acceso a Datos usando el patrón repositorio:

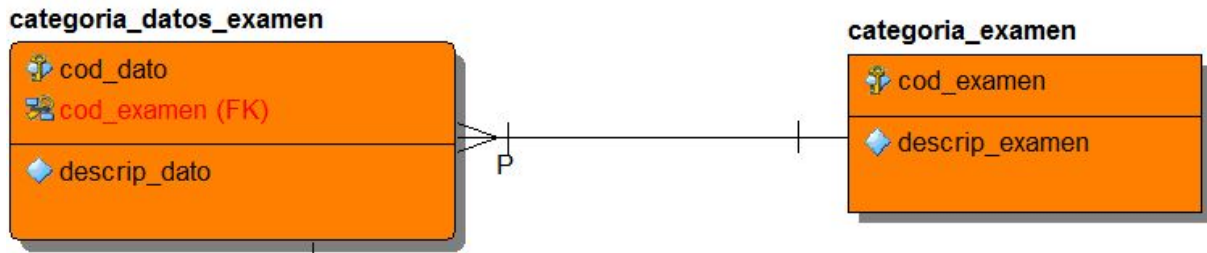
Cabe destacar que dependiendo de la relación

- Uno a muchos
- Uno a uno
- Muchos a muchos

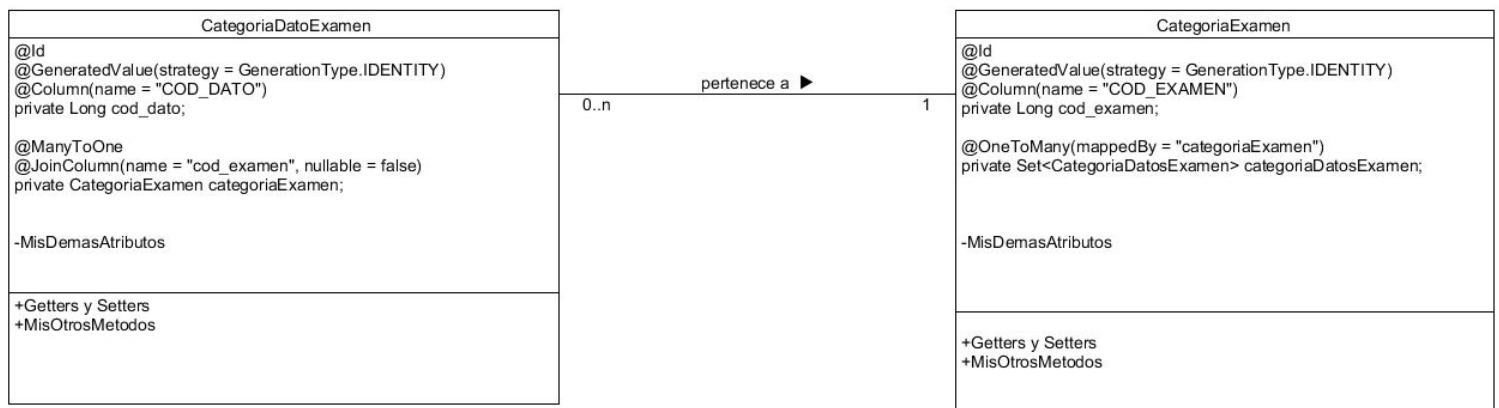
Existen diferentes maneras de mapear dichas asociaciones entre tablas del modelo relacional al modelo orientado a objetos.

A modo de ejemplo veamos cómo se mapean y se usa el patrón repositorio.

- Donde la clase **Categoria\_examen** representa una familia o categoría de exámenes al que puede asociarse multitud de estudios que pertenecen a esa categoría..
- Donde **categoria\_datos\_examen** representa un estudio particular que pertenece a una sola categoría de examen.



Definimos las clases y establecemos las relaciones usando JPA/Hibernate. A grandes rasgos en el modelo orientado objeto para representar dicha relación se realiza lo siguiente:



## Clase CategoriaDatosExamen

```
package com.utn.frre.cs.examen.receta.spring.receta.examen.entidad;
```

```
import java.util.Set;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
```

```

/**
 * CategoriaDatoExamen: Representa un estudio particular sobre una determinada
 *Categoria de Examen
 *
 * @author Gonza
 * @version 1.0
 */

@Entity
@Table(name = "Tipo_examen_estudio")
public class CategoriaDatosExamen {

    /**
     * Es el id que representa un estudio particular sobre una Categoria de Examen
     */

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "COD_DATO")
    private Long cod_dato;

    /**
     * Es la categoria de examen a la que pertenece este estudio particular
     */

    @ManyToOne
    @JoinColumn(name = "cod_examen", nullable = false)
    private CategoriaExamen categoriaExamen;

    /**
     * Descripcion o nombre del estudio en particular
     */

    private String descripcionDato;

    /**
     * Representa que un estudio puede haber sido solicitado en diferentes solicitudes de examen
     *
     */

    @OneToMany(mappedBy = "categoriaDatoExamen")
    private Set<TramiteExamenDatoLinea> tramiteExamenDatoLinea;

    /**
     * Es un constructor por defecto de la clase CategoriaDatosExamen
     */

    public CategoriaDatosExamen() {

    }

    /**
     * Es un constructor de la clase CategoriaDatosExamen necesario para update
     */

```

```

    public CategoriaDatosExamen(Long cod_dato, CategoriaExamen categoriaExamen, String
descripcionDato) {
        super();
        this.cod_dato = cod_dato;
        this.categoriaExamen = categoriaExamen;
        this.descripcionDato = descripcionDato;
    }

    /**
     * Es un constructor de la clase CategoriaDatosExamen necesario para insert particular
     */

    public CategoriaDatosExamen(CategoriaExamen categoriaExamen, String descripcionDato) {
        super();
        this.categoriaExamen = categoriaExamen;
        this.descripcionDato = descripcionDato;
    }

    // Getters y Setters-----

    public Set<TramiteExamenDatoLinea> getTramiteExamenDatoLinea() {
        return tramiteExamenDatoLinea;
    }

    public void setTramiteExamenDatoLinea(Set<TramiteExamenDatoLinea>
tramiteExamenDatoLinea) {
        this.tramiteExamenDatoLinea = tramiteExamenDatoLinea;
    }

    public Long getCod_dato() {
        return cod_dato;
    }

    public void setCod_dato(Long cod_dato) {
        this.cod_dato = cod_dato;
    }

    public CategoriaExamen getCategoriaExamen() {
        return categoriaExamen;
    }

    public void setCategoriaExamen(CategoriaExamen categoriaExamen) {
        this.categoriaExamen = categoriaExamen;
    }

    public String getDescripcionDato() {
        return descripcionDato;
    }

    public void setDescripcionDato(String descripcionDato) {
        this.descripcionDato = descripcionDato;
    }

```

```

// metodo ToString para poder ver el resultado en la consola y hacer pruebas

@Override
public String toString() {
    return "CategoriaDatosExamen [cod_dato=" + cod_dato + ", categoriaExamen=" +
        categoriaExamen + ", descripcionDato=" + descripcionDato + "]";
}

}

```

## Clase CategoriaExamen

```

package com.utn.frre.cs.examen.receta.spring.receta.examen.entidad;

import java.util.Set;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

/**
 * Categoria examen: Representa la categoria a la que puede pertenecer un examen
 *
 * @author Gonza
 * @version 1.0
 */

@Entity
@Table(name = "Categoria_Examen")
public class CategoriaExamen {

    /**
     * Es el id que identifica un tipo examen en particular
     */

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "COD_EXAMEN")
    private Long cod_examen;

    /**
     * Es la descripcion del tipo de examen
     */

    private String descripcionExamen;

```

```

/**
 * Conjunto de estudios que pertenecen a esta categoria de examen
 */

@OneToMany(mappedBy = "categoriaExamen")
private Set<CategoriaDatosExamen> categoriaDatosExamen;

/**
 * Constructor por defecto de CategoriaExamen
 */

public CategoriaExamen() {

}

/**
 * Es un constructor de la clase CategoriaExamen necesario para update particular
 */

public CategoriaExamen(Long cod_examen, String descripcionExamen,
Set<CategoriaDatosExamen> categoriaDatosExamen) {
    super();
    this.cod_examen = cod_examen;
    this.descripcionExamen = descripcionExamen;
    this.categoriaDatosExamen = categoriaDatosExamen;
}

/**
 * Es un constructor de la clase CategoriaExamen necesario para insert particular
 */
public CategoriaExamen(String descripcionExamen, Set<CategoriaDatosExamen>
categoriaDatosExamen) {
    super();
    this.descripcionExamen = descripcionExamen;
    this.categoriaDatosExamen = categoriaDatosExamen;
}

// Getters y Setters-----

public Long getCod_examen() {
    return cod_examen;
}

public void setCod_examen(Long cod_examen) {
    this.cod_examen = cod_examen;
}

public String getDescripcionExamen() {
    return descripcionExamen;
}
}

```



```

    public void setDescripcionExamen(String descripcionExamen) {
        this.descripcionExamen = descripcionExamen;
    }

    public Set<CategoriaDatosExamen> getCategoriaDatosExamen() {
        return categoriaDatosExamen;
    }

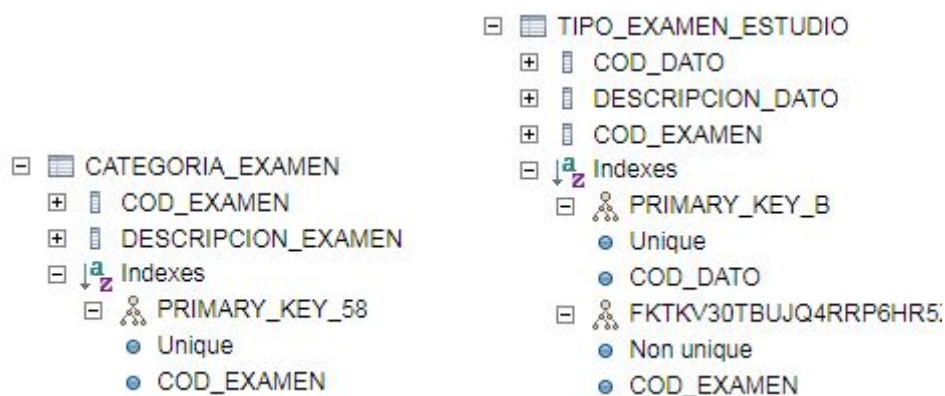
    public void setCategoriaDatosExamen(Set<CategoriaDatosExamen> categoriaDatosExamen) {
        this.categoriaDatosExamen = categoriaDatosExamen;
    }

    // método ToString para poder ver el resultado en la consola y hacer pruebas.

    @Override
    public String toString() {
        return "CategoriaExamen [cod_examen=" + cod_examen + ", descripcionExamen=" +
            descripcionExamen + "]";
    }
}

```

Esto es lo que obtendremos al hacer la migración, cómo definimos estas clases como `@entity` y además definimos las relaciones `@OneToMany` y `@ManyToOne` para representar las relaciones del modelo relacional al modelo orientado a objetos, el resultado ( a una base de datos-H2) es el siguiente:



El nombre de TIPO\_EXAMEN\_ESTUDIO fue cambiado por `@Table(name = "Tipo_examen_estudio")` en la clase CategoriaDatoExamen. Representan lo mismo.

## Migración de la parte Receta-Examen H2 (In memory database)

The screenshot shows the H2 database console interface. On the left, a tree view displays the database schema for 'jdbc:h2:mem:testdb'. The schema includes tables like 'CATEGORIA\_EXAMEN', 'TIPO\_EXAMEN\_ESTUDIO', 'TRAMITE\_EXAMEN', and 'TRAMITE\_RECETA', each with its columns and indexes. On the right, there's a 'SQL statement' input area with buttons for 'Run', 'Run Selected', 'Auto complete', and 'Clear'. Below this, an 'Important Commands' table lists shortcuts like Ctrl+Enter for executing SQL and Ctrl+Space for auto-complete. At the bottom, a 'Sample SQL Script' table provides examples for creating, inserting, querying, and deleting data in a table named 'TEST'.

Icon	Command
?	Displays this Help Page
📜	Shows the Command History
▶	Ctrl+Enter: Executes the current SQL statement
👤	Shift+Enter: Executes the SQL statement defined by the text selection
⌨	Ctrl+Space: Auto complete
🔌	Disconnects from the database

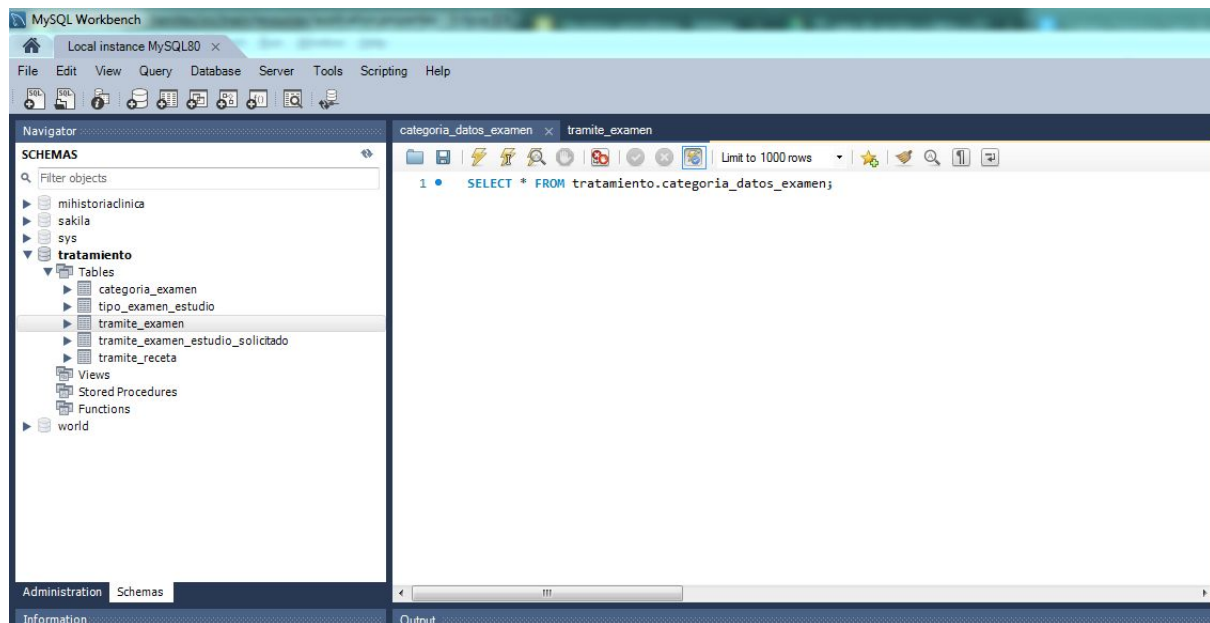
Command	SQL Statement
Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

Creando y cargando el archivo data.sql con datos para la base de datos, SpringBoot se encarga de inicializar H2 con ellos.

The screenshot shows an IDE with a project structure on the left and the content of 'data.sql' on the right. The project structure includes 'src/main/resources' with 'application.properties' and 'data.sql' (highlighted with a red circle). The 'data.sql' file contains SQL statements for inserting data into the H2 database schema.

```
7 INSERT INTO CATEGORIA_EXAMEN (COD_EXAMEN, DESCRIPCION_EXAMEN)
8 VALUES (104, 'Categoria de examen: Sangre');
9 INSERT INTO CATEGORIA_EXAMEN (COD_EXAMEN, DESCRIPCION_EXAMEN)
10 VALUES (105, 'Categoria de examen: Audicion');
11 INSERT INTO CATEGORIA_EXAMEN (COD_EXAMEN, DESCRIPCION_EXAMEN)
12 VALUES (106, 'Categoria de examen: Vision');
13 INSERT INTO CATEGORIA_EXAMEN (COD_EXAMEN, DESCRIPCION_EXAMEN)
14 VALUES (107, 'Categoria de examen: Neurologico');
15
16 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
17 VALUES (55, 'examen: electrocardiograma', 101 );
18 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
19 VALUES (56, 'examen: ecodopler cardiaco', 101 );
20 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
21 VALUES (57, 'examen : examen fisico completo', 102 );
22 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
23 VALUES (58, 'examen: radiografia torax', 102 );
24 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
25 VALUES (59, 'examen: psicoanalisis', 103 );
26 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
27 VALUES (60, 'examen: perdida de nivel auditivo', 105 );
28 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
29 VALUES (61, 'examen: prueba visual', 106 );
30 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
31 VALUES (66, 'examen: reflejos y coordinacion', 107 );
32 INSERT INTO TIPO_EXAMEN_ESTUDIO ( COD_DATO, DESCRIPCION_DATO, COD_EXAMEN )
```

También podemos hacer la migración a MySQL:



Para ello se debe tener las dependencias para MySQL y en el archivo de application properties hacer la configuración correspondiente:

```
Application.java  *application.properties
1
2# Habilitar consola web de administracion de H2-> IN-MEMORY-DATABASE <-MUY UTIL DURANTE DESARROLLO
3# data.sql file lo movi a la parte de test, tiene datos para pre-cargar la H2 con datos y hacer pruebas
4#spring.h2.console.enable=true
5# Habilitar GENERACION de sentencias SQL para ver el codigo que se producen al generar Entidades y relaciones
6spring.jpa.show-sql=true
7#-----
8#Datos de configuracion para la coneccion a la BASE DE DATOS --> mysql
9#-----
10
11# se debe crear una BD en mysql con el nombre de "tratamiento" para que funcione.
12#La parte de timezone fue agregada para evitar conflicto de hora/fecha generada por ide y mysql
13spring.datasource.url=jdbc:mysql://localhost:3306/tratamiento?useTimezone=true&serverTimezone=UTC
14spring.datasource.username=root
15spring.datasource.password=Cuat3rnr1*
16spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
17spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
18
19# update --> can la BD cargada de datos, puedo ver el impacto de los cambios (insert,delete,findBy,update) etc
20# que estan en los metodos que figuran en main
21spring.jpa.hibernate.ddl-auto=update
22
23 # Primera vez-->CREATE-DROP--->al comenzar la aplicacion el esquema del BD sera creado como nuevo
24#(cada vez que haga modificacion en relaciones entre entidades) o mejor usar H2 mas facil y rapido
25spring.jpa.hibernate.ddl-auto=create-drop
26
27#Fuente de datos para cargar la BD que estoy por crear -->dataForMySQLFormat
28spring.datasource.data= PREGUNTAR COMO CARGAR SCRIPT DE DATOS AL INICIO para mysql con H2 todo OK
29#-----
30# JPA y Hibernate Configuracion -->crear BD from entity
31#-----
32
33# estrategia de nombrado
34spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyHbmImpl
35spring.jpa.hibernate.naming.physical-strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
--
```

## Implementación del patrón repositorio:

La clase repositorio contiene todo el código relacionado con la persistencia pero nada de la lógica de negocios. Proporciona métodos para persistir, actualizar y eliminar una entidad o métodos que permiten instanciar o ejecutar consultas específicas. El objetivo de este patrón es separar el código relacionado con la persistencia de aquel relacionado con el de la lógica de negocios. También hacer que código de negocio generado sea más fácil de leer y escribir, ya que puede enfocarse en resolver requisitos de negocio en lugar de interactuar con una base de datos.

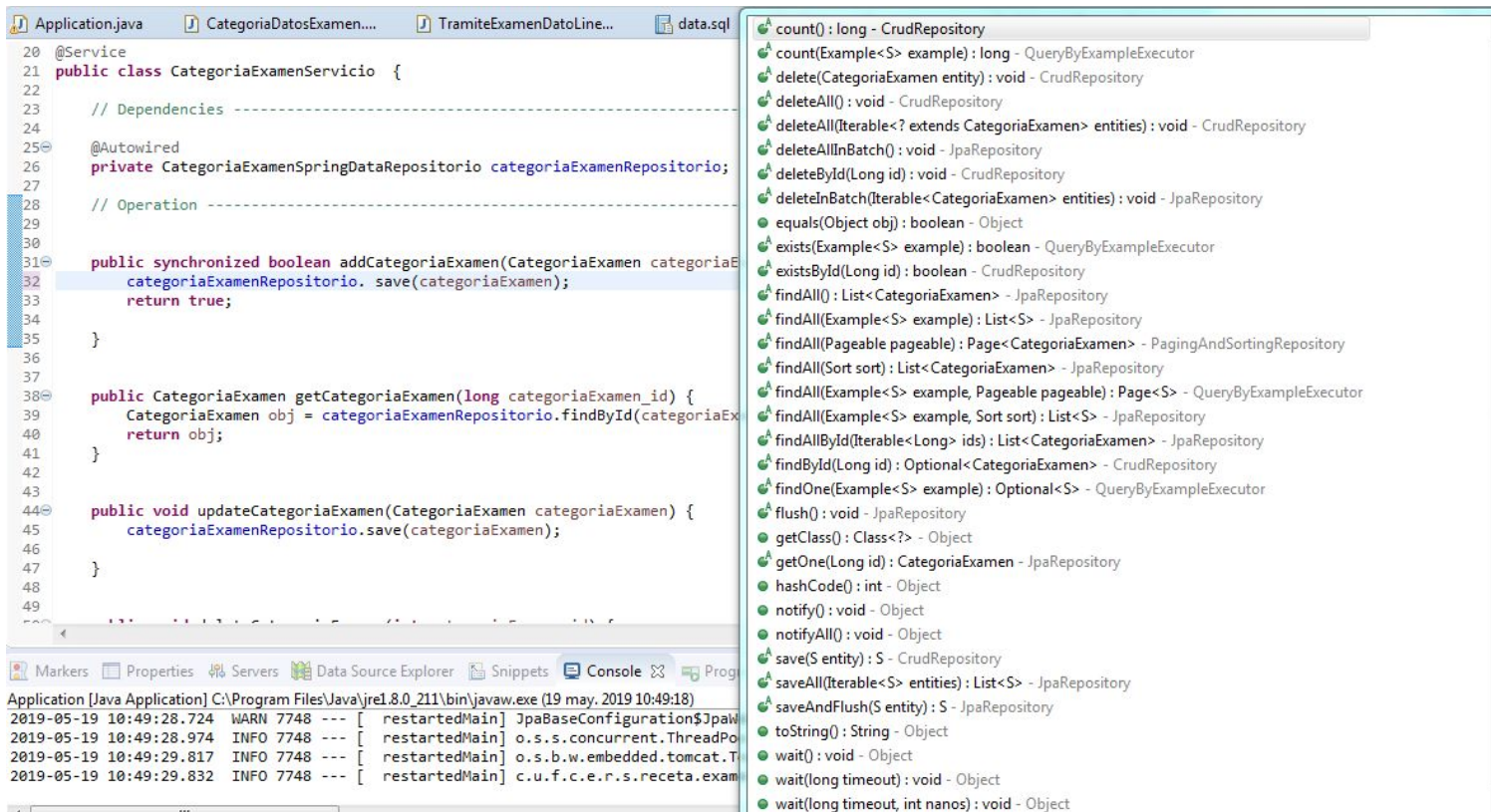
Usando Spring nos permite generar los repositorios fácilmente, para cada una de las entidades. Nos permite generar las operaciones CRUD→ create, read, update, delete más comunes, además de consultas personalizadas basadas en interfaces.

```
1 package com.utn.frre.cs.examen.receta.spring.receta.examen.repositorio;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6 /**
7  * Repositorio de Acceso a Datos para <code>CategoriaExamen</code>.
8  *
9  * @author Gonza
10  * @version 1.0
11  */
12
13 @Repository
14 public interface CategoriaExamenSpringDataRepositorio extends JpaRepository<CategoriaExamen, Long> {
15
16     // mis metodos particulares van aca
17
18 }
19
20
21
22
```

El siguiente fragmento de código define un repositorio al extender la interface a `JpaRepository<NombreDeLaClaseParaGestionar, ID_Clasa_Long>`. Donde `JpaRepository` tiene el comportamiento `CrudRepository` y `PagingAndSortingRepository` por lo que me permite definir un conjunto de métodos para operaciones como guardar(save), borrar(delete), y lecturas (read).

A modo de ejemplo dejo algunas de las operaciones que me permite usar este patrón repositorio al cablear la interface recién creada a la clase `CategoriaExamenServicio` que me facilitan manejar operaciones de la acceso a la capa de datos.





Veamos cómo a partir del repositorio creado con anterioridad se pueden implementar las operaciones básicas para la clase `CategoriaExamen` o sea: `findAll(GET)`, `findById(GET)`, `update(PUT)`, `create(POST)`, `delete(DELETE)`.

```
package com.utn.frre.cs.examen.receta.spring.receta.examen.servicio;
import java.util.Optional;
import javax.validation.Valid;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import com.utn.frre.cs.examen.receta.spring.receta.examen.entidad.CategoriaExamen;
import com.utn.frre.cs.examen.receta.spring.receta.examen.repositorio.CategoriaExamenSpringDataRepositorio;
```

```

/**
 * RestController que implementa las operaciones básicas get,insert,delete,update para
 * <code>CategoriaExamen</code>. Cableo con CategoriaExamenSpringDataRepositorio
 * para ayudarme a definir el comportamiento más adecuado para esta clase
 *
 *
 * @author Gonza
 * @version 1.0
 */
@RestController
@RequestMapping("/api/examen/categoriaExamen")
public class CategoriaExamenServicio {

```

// Dependencias -----

```

@Autowired
private CategoriaExamenSpringDataRepositorio categoriaExamenRepositorio;

```

// Operaciones -----

```

/**
 * GET-->retorna una todas las categoría de examen que están presentes en la BD
 *
 */
@GetMapping()
public Page<CategoriaExamen> getPage(Pageable pageable) {
    return categoriaExamenRepositorio.findAll(pageable);
}

```

```

/**
 * GET-->retorna una categoría de examen segun su id
 *
 */
@GetMapping("/{id}")
public ResponseEntity<CategoriaExamen> findById(@PathVariable Long id) {
    Optional<CategoriaExamen> opt = categoriaExamenRepositorio.findById(id);
    if (opt.isPresent())
        return ResponseEntity.ok(opt.get());
    return ResponseEntity.notFound().build();
}

```

```

/**
 * POST--> crea una categoria de examen
 *
 */
@PostMapping()
public ResponseEntity<CategoriaExamen> create(@Valid @RequestBody CategoriaExamen createRequest){
    return ResponseEntity.ok(categoriaExamenRepositorio.save(createRequest));
}

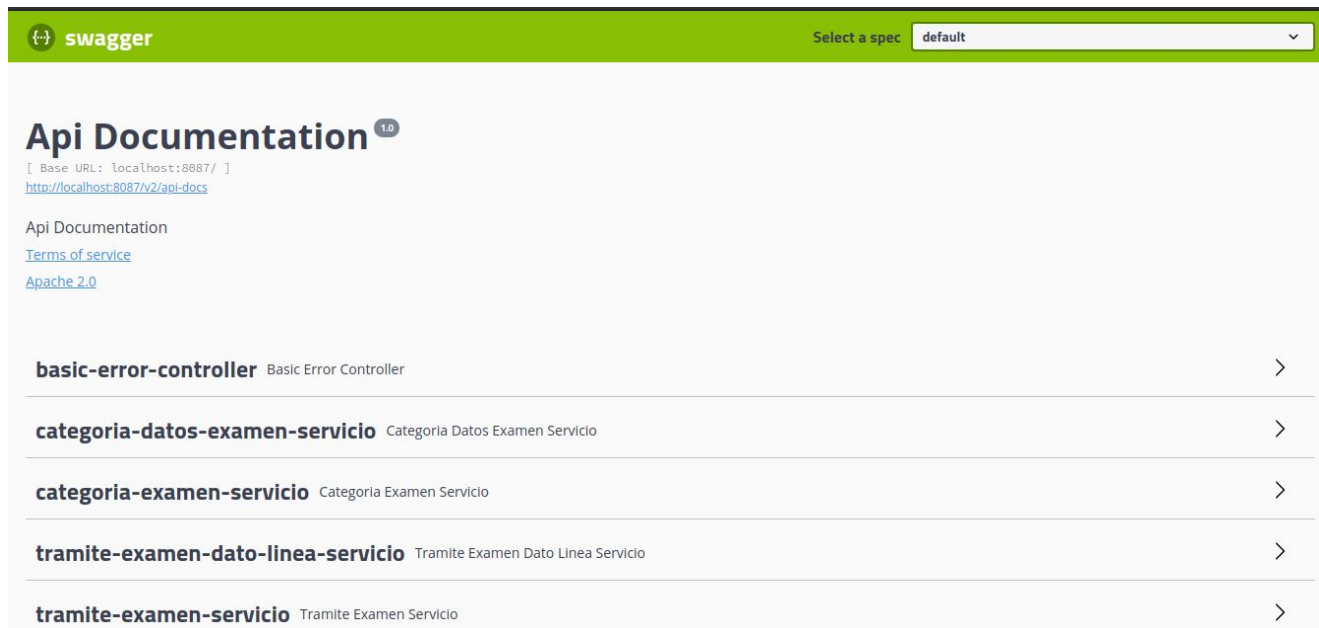
/**
 * PUT-->actualiza una categoria de examen
 *
 */

@PutMapping()
public ResponseEntity<CategoriaExamen> update(@Valid @RequestBody CategoriaExamen updateRequest) {
    boolean exists = categoriaExamenRepositorio.existsById(updateRequest.getCod_examen());
    if (exists) {
        return ResponseEntity.ok(categoriaExamenRepositorio.save(updateRequest));
    }
    return ResponseEntity.notFound().build();
}

/**
 * DELETE-->borra una categoria de examen segun un id
 *
 */
@DeleteMapping("/{id}")
public ResponseEntity<?> delete(@PathVariable Long id) {
    Optional<CategoriaExamen> opt = categoriaExamenRepositorio.findById(id);
    if (opt.isPresent()) {
        categoriaExamenRepositorio.delete(opt.get());
        return ResponseEntity.ok().build();
    }
    return ResponseEntity.notFound().build();
}
}

```

Documentación con swagger:



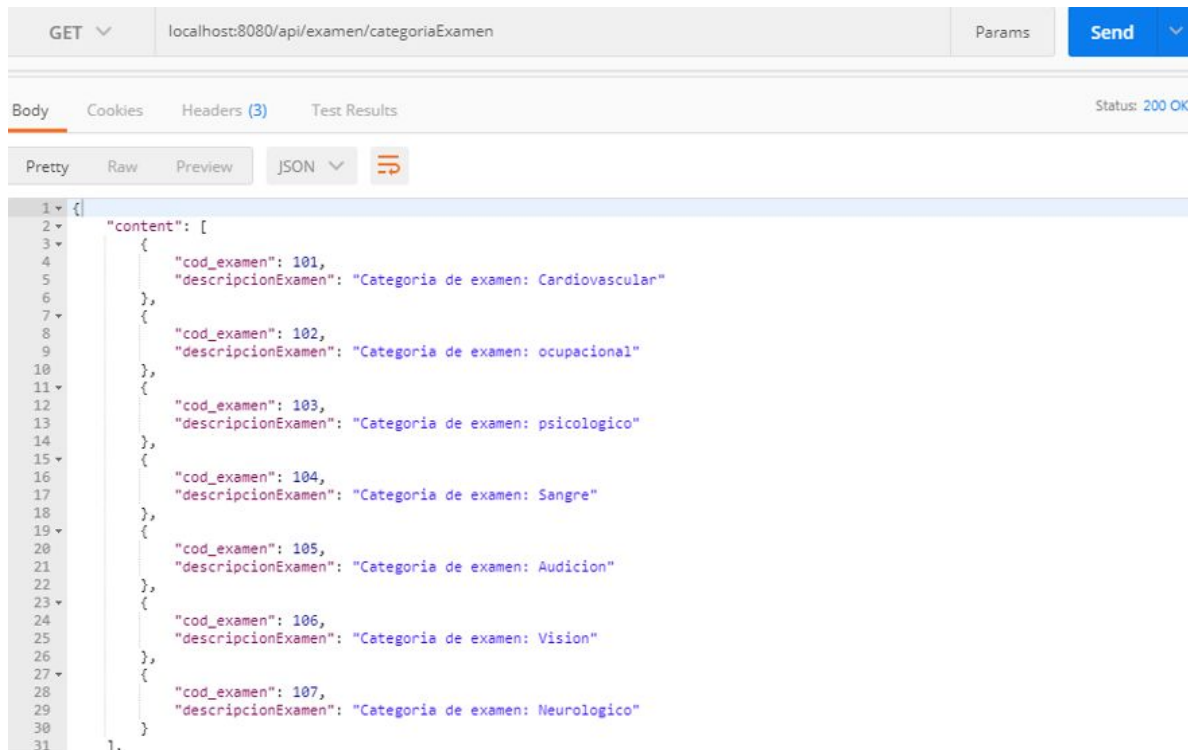
The image shows the Swagger API Documentation interface. At the top, there's a green header with the Swagger logo and a 'Select a spec' dropdown menu set to 'default'. Below the header, the main title 'Api Documentation' is displayed with a version '1.0' badge. Underneath, the base URL is shown as 'localhost:8087/' and a link to 'http://localhost:8087/v2/api-docs' is provided. There are also links for 'Terms of service' and 'Apache 2.0'. A list of API endpoints is shown, each with a name and a description, and a right arrow indicating further details.

Endpoint Name	Description
basic-error-controller	Basic Error Controller
categoria-datos-examen-servicio	Categoria Datos Examen Servicio
categoria-examen-servicio	Categoria Examen Servicio
tramite-examen-dato-linea-servicio	Tramite Examen Dato Linea Servicio
tramite-examen-servicio	Tramite Examen Servicio

## Veamos cómo usar las operaciones con Postman:

**Postman:** Se trata de una herramienta dirigida a desarrolladores web que permite realizar peticiones HTTP a cualquier API. Postman es muy útil a la hora de programar y hacer pruebas, puesto que nos ofrece la posibilidad de comprobar el correcto funcionamiento de nuestros desarrollos.

**Operación(GET): FindAll** las CaterogiaExamen presentes en la base de datos.



The image shows a Postman screenshot of a GET request to 'localhost:8080/api/examen/categoriaExamen'. The status is '200 OK'. The response is in JSON format, showing a list of exam categories with their IDs and descriptions.

```
1 {
2   "content": [
3     {
4       "cod_examen": 101,
5       "descripcionExamen": "Categoria de examen: Cardiovascular"
6     },
7     {
8       "cod_examen": 102,
9       "descripcionExamen": "Categoria de examen: ocupacional"
10    },
11    {
12      "cod_examen": 103,
13      "descripcionExamen": "Categoria de examen: psicologico"
14    },
15    {
16      "cod_examen": 104,
17      "descripcionExamen": "Categoria de examen: Sangre"
18    },
19    {
20      "cod_examen": 105,
21      "descripcionExamen": "Categoria de examen: Audicion"
22    },
23    {
24      "cod_examen": 106,
25      "descripcionExamen": "Categoria de examen: Vision"
26    },
27    {
28      "cod_examen": 107,
29      "descripcionExamen": "Categoria de examen: Neurologico"
30    }
31  ],
32 }
```



Contenido de la base de datos(H2):

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM CATEGORIA_EXAMEN
```

```
SELECT * FROM CATEGORIA_EXAMEN;
```

COD_EXAMEN	DESCRIPCION_EXAMEN
101	Categoria de examen: Cardiovascular
102	Categoria de examen: ocupacional
103	Categoria de examen: psicologico
104	Categoria de examen: Sangre
105	Categoria de examen: Audicion
106	Categoria de examen: Vision
107	Categoria de examen: Neurologico

(7 rows, 0 ms)

Operación(GET): FindById de una CaterogiaExamen presente en la base de datos.

GET	localhost:8080/api/examen/categoriaExamen/101	Params	Send
Body	Cookies	Headers (3)	Test Results
Status: 200 OK			
Pretty Raw Preview JSON			
<pre>1 { 2   "cod_examen": 101, 3   "descripcionExamen": "Categoria de examen: Cardiovascular" 4 }</pre>			

Operación(POST): Create una CaterogiaExamen presente en la base de datos.

- Se debe especificar que el cuerpo está en formato JSON
- Se debe especificar el contenido del cuerpo, con el contenido de la nueva CategoriaExamen. El id(cod\_examen) no es necesario especificar se autogenera.

POST localhost:8080/api/examen/categoriaExamen Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```

1 {
2
3   "descripcionExamen": "Categoria de examen: Patologico"
4 }

```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 57 ms

Pretty Raw Preview

```

{"cod_examen":109,"descripcionExamen":"Categoria de examen: Patologico"}

```

Contenido de la base de datos(H2), luego del create.

Run Run Selected Auto complete Clear SQL statement:  
 SELECT \* FROM CATEGORIA\_EXAMEN|

SELECT \* FROM CATEGORIA\_EXAMEN;

COD_EXAMEN	DESCRIPCION_EXAMEN
101	Categoria de examen: Cardiovascular
102	Categoria de examen: ocupacional
103	Categoria de examen: psicologico
104	Categoria de examen: Sangre
105	Categoria de examen: Audicion
106	Categoria de examen: Vision
107	Categoria de examen: Neurologico
108	Categoria de examen: Toxicologico
109	Categoria de examen: Patologico

Operación(PUT): Update de una CaterogiaExamen presente en la base de datos.

- Se debe especificar que el cuerpo está en formato JSON
- Se debe especificar el contenido del cuerpo, con el contenido de la nueva CategoriaExamen, es necesario especificar el id(cod\_examen) para identificar unívocamente los cambios a realizar en la base de datos.

PUT

localhost:8080/api/examen/categoriaExamen

Params

Send

Authorization

Headers (1)

Body

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

1 {

2 "cod\_examen":109,

3 "descripcionExamen":"Categoria de examen: Cultivo"

4 }

Body

Cookies

Headers (3)

Test Results

Status: 200 OK

Pretty

Raw

Preview

{"cod\_examen":109,"descripcionExamen":"Categoria de examen: Cultivo"}

Contenido de la base de datos(H2), luego del update.

Run

Run Selected

Auto complete

Clear

SQL statement:

SELECT \* FROM CATEGORIA\_EXAMEN|

SELECT \* FROM CATEGORIA\_EXAMEN;

COD_EXAMEN	DESCRIPCION_EXAMEN
101	Categoria de examen: Cardiovascular
102	Categoria de examen: ocupacional
103	Categoria de examen: psicologico
104	Categoria de examen: Sangre
105	Categoria de examen: Audicion
106	Categoria de examen: Vision
107	Categoria de examen: Neurologico
108	Categoria de examen: Toxicologico
109	Categoria de examen: Cultivo

Operación(DELETE): Delete de una CaterogiaExamen presente en la base de datos.

- Se debe especificar que el cuerpo está en formato JSON
- Es necesario especificar el id(cod\_examen) para identificar unívocamente el borrado a realizar en la base de datos.

DELETE

localhost:8080/api/examen/categoriaExamen/109

Params

Send

Authorization

Headers (1)

Body

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

1

{

2

"cod\_examen":109,

3

"descripcionExamen":"Categoria de examen: Cultivo"

4

}

Body

Cookies

Headers (2)

Test Results

Status: 200 OK

Contenido de la base de datos(H2), luego del delete.

Run

Run Selected

Auto complete

Clear

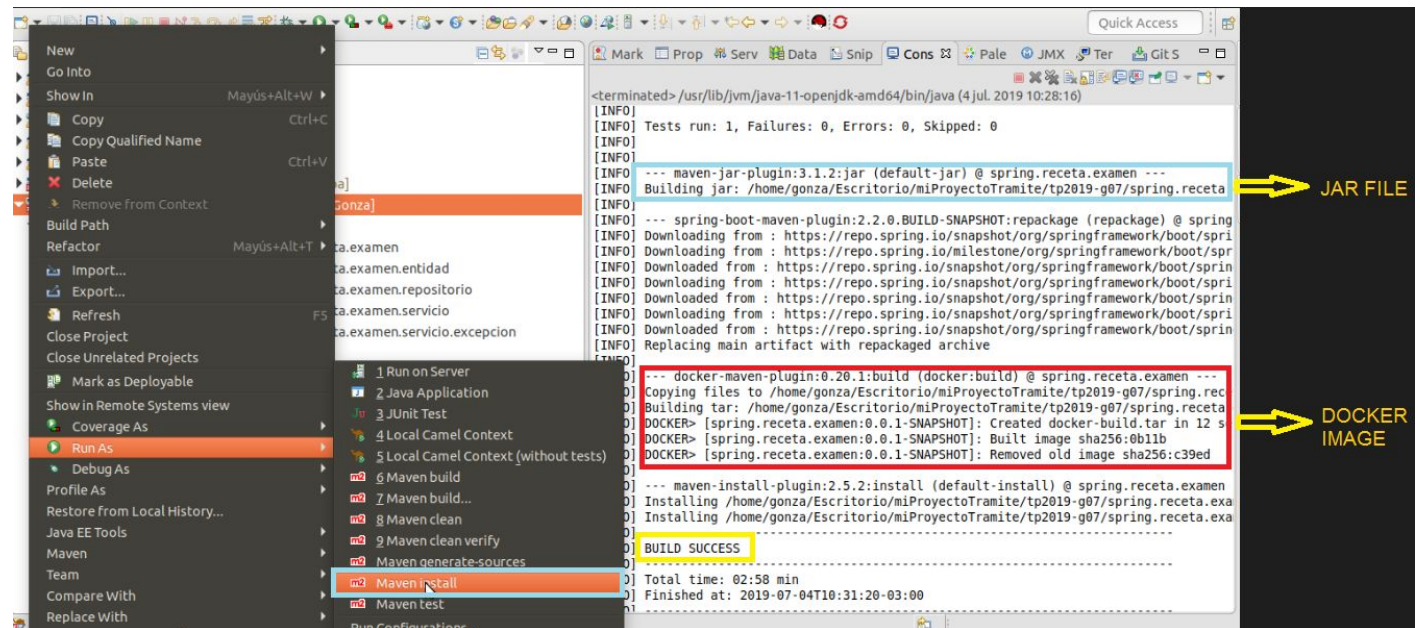
SQL statement:

SELECT \* FROM CATEGORIA\_EXAMEN|

SELECT \* FROM CATEGORIA\_EXAMEN;

COD_EXAMEN	DESCRIPCION_EXAMEN
101	Categoria de examen: Cardiovascular
102	Categoria de examen: ocupacional
103	Categoria de examen: psicologico
104	Categoria de examen: Sangre
105	Categoria de examen: Audicion
106	Categoria de examen: Vision
107	Categoria de examen: Neurologico
108	Categoria de examen: Toxicologico

**MAVEN INSTALL:** Esto compilará si es necesario nuestros fuentes, les pasará los test, generará el **jar** y lo copiará en nuestro repositorio local de jars, en nuestro pc. Además cómo tenemos agregado el plugin de **fabric8**, también se genera una **imagen de docker** del proyecto.



Vemos cómo se creó la imagen de docker: **spring.receta.examen**

```
gonza@gonza-K52JT:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
spring.receta.examen	0.0.1-SNAPSHOT	0b11b459de7e	36 minutes ago
postgres	latest	79db2bf18b4a	12 days ago
<none>	<none>	2ab4d8d98591	13 days ago
frredacs/medicos-frontend	latest	2c8255c8b282	4 weeks ago
frredacs/medicos-backend	latest	c94e244b430a	4 weeks ago

Le damos el ID de la imagen del proyecto, el puerto y lo corremos:





Si quisiéramos compartir nuestra imagen docker, hacemos lo siguiente:

- Nos ubicamos en el directorio en donde guardaremos nuestras imágenes y ejecutamos:

```
sudo docker save spring.receta.examen | gzip > spring.receta.examen.tar.gz
```

Luego de haber guardado un copiar de nuestra imagen podemos copiar a otra pc y restaurarlo:

- Debemos ubicarnos en la directorio que contiene nuestra imagen y ejecutar:

```
sudo docker load -i spring.receta.examen.tar.gz
```

```
sudo docker tag 5948b3144ec4 spring.receta.examen:1.0
```

## Patrón MVC

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Modelos y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación. Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales; la mayoría de los frameworks modernos utilizan MVC (o alguna adaptación del MVC) para la arquitectura.

### Modelo

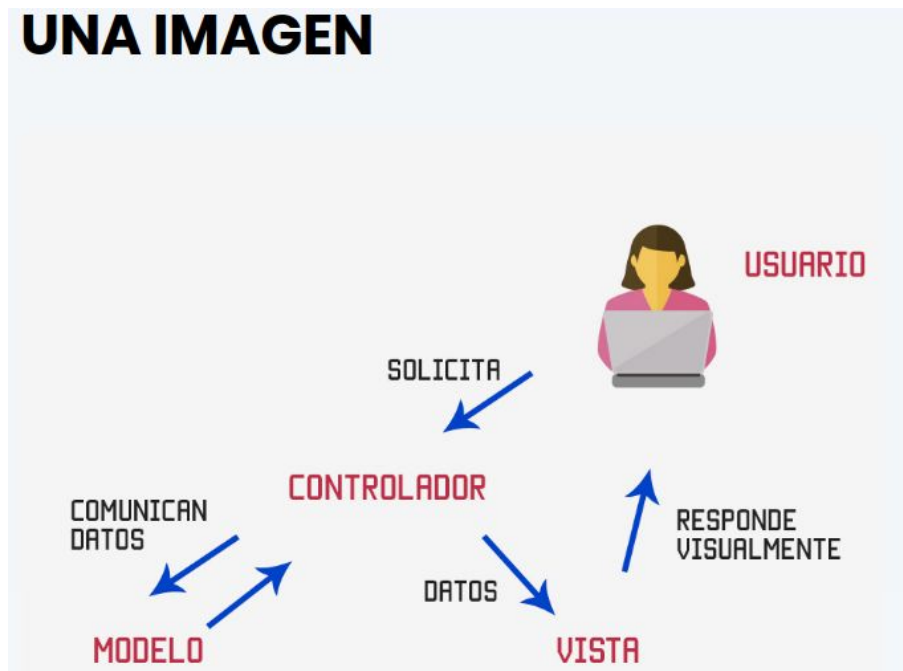
El modelo representa la estructura de datos de una aplicación de software. Es un error común pensar que el modelo es la base de datos que está detrás de la aplicación, sin embargo, es mucho mejor ver el modelo como el cuerpo de código que representan los datos.

### Vista

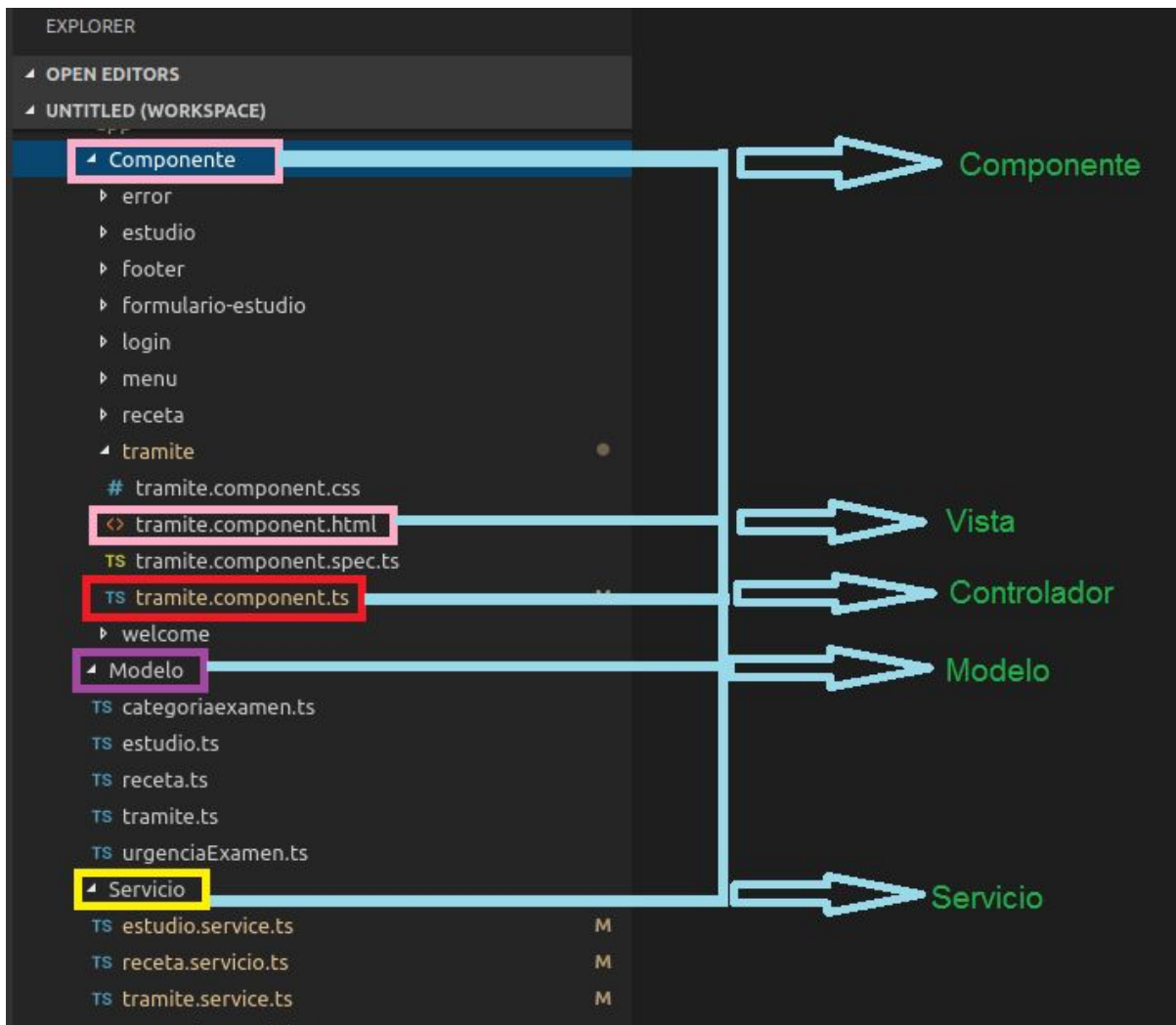
La vista es el cuerpo de código que representa la interfaz del usuario, es decir, todo aquello que el usuario puede ver en la pantalla e interactuar. Una aplicación normalmente tiene múltiples vistas, cada una representa alguna parte del modelo.

### Controlador

El controlador se puede ver como el intermediario entre la vista y el modelo. El modelo y la vista nunca se relacionan directamente.



Estructura del proyecto



El modelo representación de los datos y permite una forma de validación contra el backend, pueden ir constructores y métodos, utilizados en frontend.

```
export class Tramite {  
  ideSolicitudExamen: number;  
  idePersonalMed: number;  
  fecExamen: Date;  
  idInternacion: number;  
}
```

La siguiente parte será la vista, que es la representación de los datos del modelo, en rojo se puede ver un ejemplo de esto. Esto es básicamente código HTML y una expresión en Angular. Aunque hay



que considerar que la vista no se conecta directamente con el modelo sino con una variable creada desde el controlador que se verá más adelante.

```
<h1>Tramites de Examenes Abiertos</h1>
```

```
<div class="container">
```

```
<table class="table">
```

```
<thead>
```

```
<tr>
```

```
<th>Id Tramite</th>
```

```
<th>Id Medico</th>
```

```
<th>Fecha Solicitud Examen</th>
```

```
<th>Id Internacion</th>
```

```
</tr>
```

```
</thead>
```

```
<tbody>
```

```
<tr *ngFor="let t of tramites ">
```

```
<th>{{t.ideSolicitudExamen}}</th>
```

```
<th>{{t.idePersonalMed}}</th>
```

```
<th>{{t.fecExamen | date:"dd/MM/yyyy HH:mm" | uppercase}}</th>
```

```
<th>{{t.id_internacion}}</th>
```

```
<th><a [routerLink]="['misnuevosestudios',t.idePersonalMed,t]">Agregar  
Estudio</a> | <a [routerLink]="['misestudios', t.ideSolicitudExamen]">Detalles  
de Estudio</a></th>
```

```
</tr>
```

```
</tbody>
```

```
</table>
```

```
</div>
```

Ahora lo que realmente importa y hace que funcione todo, el controlador, para ello se requiere importar el modelo de datos(trámite). Básicamente el controlador nos permite mostrar los datos del modelo en la vista, por medio de métodos, objetos y variables que al final me permitirán usarlos en la vista para obtener los información requerida, respetando el modelo de datos .

```
import { Component, OnInit } from '@angular/core';
import {Tramite} from 'src/app/Modelo/tramite'; // importo el modelo
import { TramiteService } from 'src/app/Servicio/tramite.service'; // importo
el servicio de tramite disponibles
import {Observable} from 'rxjs'
```

```
@Component({
  selector: 'app-tramite',
  templateUrl: './tramite.component.html',
  styleUrls: ['./tramite.component.css']
})
```

```
export class TramiteComponent implements OnInit {
```

```
  tramites: Tramite[];
```

```
  var = 300;
```

```
  constructor(private tramiteServicio: TramiteService) { }
```

```
  ngOnInit() {
    this.getAllTramitesMedico(300);
  }
```

```
  getAllTramitesMedico(id: number) {
    this.tramiteServicio.getAllTramitesMedico(id).subscribe(
      response => {
        console.log(response);
        this.tramites = response;
      }
    );
  }
```

```
  getAllTramites() {
    this.tramiteServicio.getAllTramites().subscribe(
      response => {
        console.log(response);
      }
    );
  }
```

```

        this.tramites = response;
    }
};
}
}

```

Por último se agregó un archivo de servicio donde está la clase TramiteService, que me permite recopilar todas las solicitudes (post-put-get-delete) expuesta en el backend para Trámite ( para poder utilizarlas luego en el frontend, por medio de la clase controladora que las ).

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Tramite } from '../Modelo/tramite';
import { Estudio } from '../Modelo/estudio';

@Injectable({
  providedIn: 'root'
})

export class TramiteService {

  API_URI = 'http://10.13.30.2:8087/api/examen';
  // reemplazar ip asignada o por localhost.

  constructor(private httpService: HttpClient) { } // inyeccion de dependencia

  getAllTramites() {
    return this.httpService.get<Tramite[]>(`${this.API_URI}/solicitud`);
  }

  getAllTramitesMedico(id: number) {
    return
    this.httpService.get<Tramite[]>(`${this.API_URI}/solicitud/medico/${id}`);
  }

  getAllEstudioFromTramiteMedico(idTramite: number) {

```

```

        return
this.httpService.get<Estudio[]>(`${this.API_URI}/solicitud/${idTramite}/estudiosSolicitados`);
    }

    getTramite(id: number) {
        return this.httpService.get<Tramite>(`${this.API_URI}/solicitud/${id}`);
    }

    saveTramite(tramite: Tramite) {
        return this.httpService.post(`${this.API_URI}/solicitud`, tramite);
    }

    updateTramite(updateTramite: Tramite) {
        return this.httpService.put(`${this.API_URI}/solicitud`, updateTramite);
    }

    // no creo que sea necesario un DELETE, pero lo dejo por las dudas.
    deleteTramite(id: number){
        return this.httpService.delete(`${this.API_URI}/solicitud/${id}`);
    }

    getDetalleTramite(id: number) {
        return
this.httpService.get<Estudio[]>(`${this.API_URI}/solicitud/${id}/estudiosSolicitados`);
    }

    saveUnEstudioDeTramite(id: number, e: Estudio) {
        return
this.httpService.post(`${this.API_URI}/solicitud/${id}/nuevoEstudioSolicitado`,
e);
    }

}

```

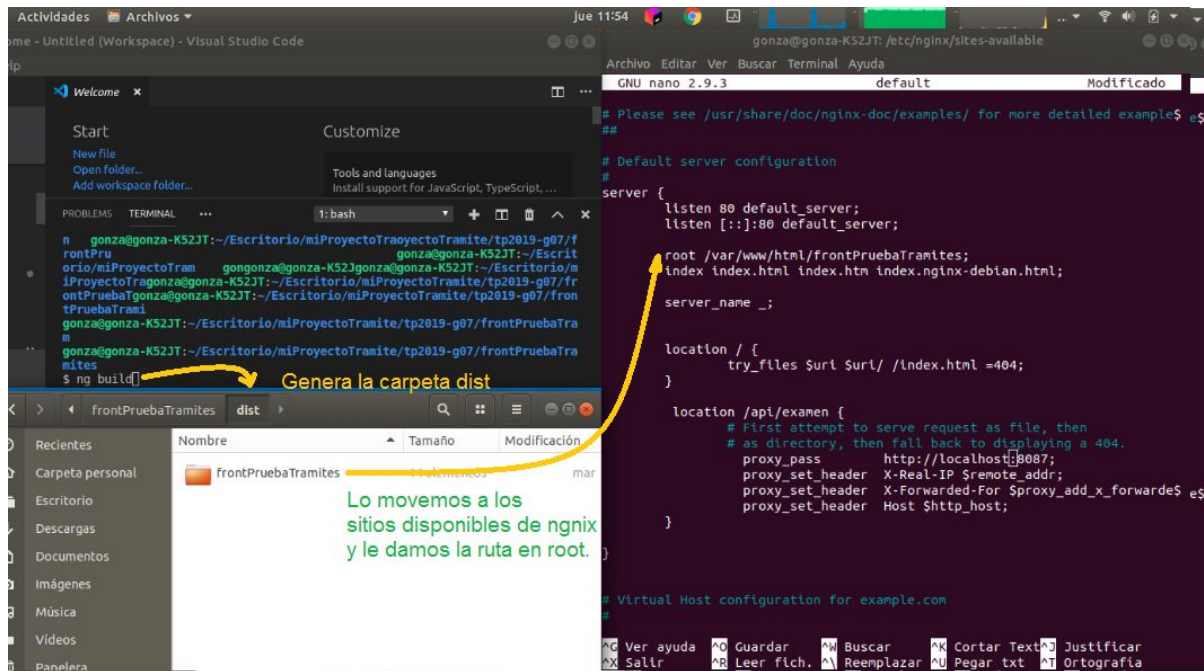
Cuestiones propias de Angular: Podemos mencionar cómo hacer el ruteo y las componentes a los que se llama al acceder a cada ruta.

```
const routes: Routes = [
  { path: '', component: LoginComponent},
  { path: 'login', component: LoginComponent},
  { path: 'welcome/:nombre', component: WelcomeComponent},
  { path: 'mistramites', component: TramiteComponent},
  { path: 'mistramites/misestudios/:id', component: EstudioComponent},
  { path: 'mistramites/misnuevosestudios/:id', component:
FormularioEstudioComponent},
  { path: 'misrecetas', component: RecetaComponent },
  { path: '**', component: ErrorComponent}

];
```

## Implementación básica a un servidor .

Por medio de ng build , se crea una compilación de producción y la cual va a ser copiada en un directorio de nginx (servidor web) .



## Clínica General

### PadecimientoActual

- idPadecimiento (PK)
- idExpediente
- descripcion
- valoracion
- observacion

### ExploracionGeneral

- idExploracionGeneral (PK)
- idExpediente
- EdoConciencia
- marcha
- hidratacion
- coloracion
- valoracion
- observacion

### SintomaGeneral

- idSintomaGeneral (PK)
- idExpediente
- sintoma
- valoracion
- observacion

### ExploracionRegional

- idExploracionRegional (PK)
- idExpediente
- region
- elemento
- estado
- valoracion
- observacion

### SignoVital

- idSignoVital (PK)
- idExpediente
- ta
- fc
- temperatura
- peso
- talla
- resultado
- observacion

Clínica General se encarga de registrar los datos del paciente que concierne a las entidades que se muestran en la imagen anterior. Las cuales explicaremos brevemente.

ExploracionGeneral	contiene los resultados de la exploración general que se realiza sobre el cuerpo humano
idExploracionGeneral	clave primaria

idExpediente	clave foránea que viene de Historia Clínica del Paciente
EdoConciencia	valores que puede tomar: orientado , desorientado
marcha	valores que puede tomar: normal. at.marcha
hidratacion	valores que puede tomar: buena, deshidratado
coloracion	valores que puede tomar: adecuada,palidez,ictérico
valoracion	descripción de resultado
observacion	observaciones adicionales sobre el padecimiento

ExploracionRegional	contiene los resultados de la exploración regional que se realiza sobre el cuerpo humano
idExploracionRegional	clave primaria
idExpediente	clave foránea que viene de Historia Clínica del Paciente
region	valores que puede tomar: cabeza, cuello, tórax,abdomen,miembros,genitales
elemento	valores que puede tomar: pupilas, faringe, superiores, inferiores
estado	valores que puede tomar: estado de cada uno de los elementos, bueno, malo
valoracion	descripción del resultado
observacion	observaciones adicionales

PadecimientoActual	registra el padecimiento actual que describe el paciente
idPadecimiento	clave primaria

idExpediente	clave foránea que viene de Historia Clínica del Paciente
descripcion	breve descripción del padecimiento actual
valoracion	descripción de resultado de padecimiento
observacion	observaciones adicionales sobre el padecimiento

SintomaGeneral	registra síntomas generales del cuerpo humano
idSintomaGeneral	clave primaria
idExpediente	clave foránea que viene de Historia Clínica del Paciente
sintoma	valores que toma: astenia, adinamia, fiebre, pérdida de peso
valoracion	descripción del resultado observado
observacion	observaciones adicionales

SignoVital	registra los signos vitales tomados al paciente
idSignoVital	clave primaria
idExpediente	clave foránea que viene de Historia Clínica del Paciente
ta	unidad mmHg
tc	unidad lpm
temperatura	unidad grados centígrados
peso	unidad kg
talla	unidad cm
resultado	descripción del resultado
observacion	observaciones que presenta los signos vitales



## Implementación de Clínica General

Para la implementación de este proyecto se decidió usar como lenguaje de programación c#, utilizando las tecnologías Fluent Nhibernate y como base de datos Postgres. También se aplicó el Patrón Repository para las operaciones básicas (CRUD).

A continuación, daremos unos ejemplos de cómo se implementaron las entidades, el patrón repositorio y el acceso a la base de datos; usaremos para describir el proceso la entidad SintomaGeneral.

### Definición de Entidad:

```
using System;

namespace WpfApp1.entidades
{
    /// <summary>
    /// Clase que registra síntomas generales del cuerpo humano
    /// </summary>
    5 referencias
    public class SintomaGeneral
    {
        3 referencias
        public virtual int idSintomaGeneral { get; set; } // clave primaria

        3 referencias
        public virtual int idExpediente { get; set; } // clave foranea que viene de Historia Clinica del Paciente
        1 referencia
        public virtual string Sintoma { get; set; } //astenia,adinina,fiebre,perida de peso

        1 referencia
        public virtual string valoracion { get; set; } // descripcion de resultado de padecimiento
        0 referencias
        public virtual string observacion { get; set; } // observaciones adicionales sobre el padecimiento
    }
}
```

### Mapeo de SintomaGeneral:

```
using FluentNHibernate.Mapping;
using WpfApp1.entidades;

namespace WpfApp1.mapeo
{
    2 referencias
    public class SintomaGeneralMap:ClassMap<SintomaGeneral>
    {
        0 referencias
        public SintomaGeneralMap()
        {
            Id(c => c.idSintomaGeneral);

            Map(c => c.idExpediente);
            Map(c => c.Sintoma);
            Map(c => c.valoracion);

            Table("SintomaGeneral");
        }
    }
}
```

## Interface IServerDataRepository

```
1  using System.Collections.Generic;
2
3  namespace webapif.Models.Persistance
4  {
5      public interface IServerDataRepository<T>
6      {
7          IEnumerable<T> Get(int id);
8          IEnumerable<T> GetAll();
9          bool Add(T entidad);
10         void Delete(T entidad);
11         bool Update(T entidad);
12     }
13 }
14
15
```

## Implementación de Interface:

### -Método get() getAll() add()

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using webapif.helper;
5  namespace webapif.Models.Persistance
6  {
7      public class ServerDataRepository<T> : IServerDataRepository<T> where T: class
8      {
9          public IEnumerable<T> Get(int id)
10         {
11             using (var session = NHibernateHelper.OpenSession())
12                 yield return session.Get<T>(id);
13         }
14         public IEnumerable<T> GetAll()
15         {
16             using (var session = NHibernateHelper.OpenSession())
17                 return session.Query<T>().ToList();
18         }
19         public bool Add(T entidad)
20         {
21             using (var session = NHibernateHelper.OpenSession())
22             {
23                 using (var transaction = session.BeginTransaction())
24                 {
25                     session.Save(entidad);
26                     transaction.Commit();
27                 }
28                 return true;
29             }
30         }
31     }
32 }
```

### -Métodos delete() update()

```
31 public void Delete(T entidad)
32 {
33     using (var session = NHibernateHelper.OpenSession())
34     {
35         using (var transaction = session.BeginTransaction())
36         {
37             session.Delete(entidad);
38             transaction.Commit();
39         }
40     }
41 }
42 public bool Update(T entidad)
43 {
44     using (var session = NHibernateHelper.OpenSession())
45     {
46         using (var transaction = session.BeginTransaction())
47         {
48             session.SaveOrUpdate(entidad);
49             try {
50                 transaction.Commit();
51             }
52             catch (Exception) {
53                 throw;
54             }
55         } return true;
56     }
57 }
58
```

Conexión a la Base de Datos:

```
namespace WpfApp1
{
    /// <summary>
    /// Establece la conexion con la base de datos y el mapeo entre las clases y la tablas
    /// </summary>
    8 referencias
    public class SessionFactory
    {
        private static string ConnectionString = "Server=localhost; Port=5432; User Id=postgres; Password=cs2019; Database=ClinicaGeneral";
        private static ISessionFactory session;
        1 referencia
        public static ISessionFactory CriarSession()
        {
            if (session != null)
                return session;
            IPersistenceConfigurer configDB = PostgreSQLConfiguration.PostgreSQL82.ConnectionString(ConnectionString);
            var configMap = Fluently
                .Configure()
                .Database(configDB)
                .Mappings(c => c.FluentMappings.AddFromAssemblyOf<mapeo.ExploracionGeneralMap>())
                .Mappings(c => c.FluentMappings.AddFromAssemblyOf<mapeo.ExploracionRegionalMap>())
                .Mappings(c => c.FluentMappings.AddFromAssemblyOf<mapeo.PadecimientoActualMap>())
                .Mappings(c => c.FluentMappings.AddFromAssemblyOf<mapeo.SignoVitalMap>())
                .Mappings(c => c.FluentMappings.AddFromAssemblyOf<mapeo.SintomaGeneralMap>());

            session = configMap.BuildSessionFactory();
            return session;
        }
        8 referencias
        public static ISession AbrirSession()
        {
            return CriarSession().OpenSession();
        }
    }
}
```

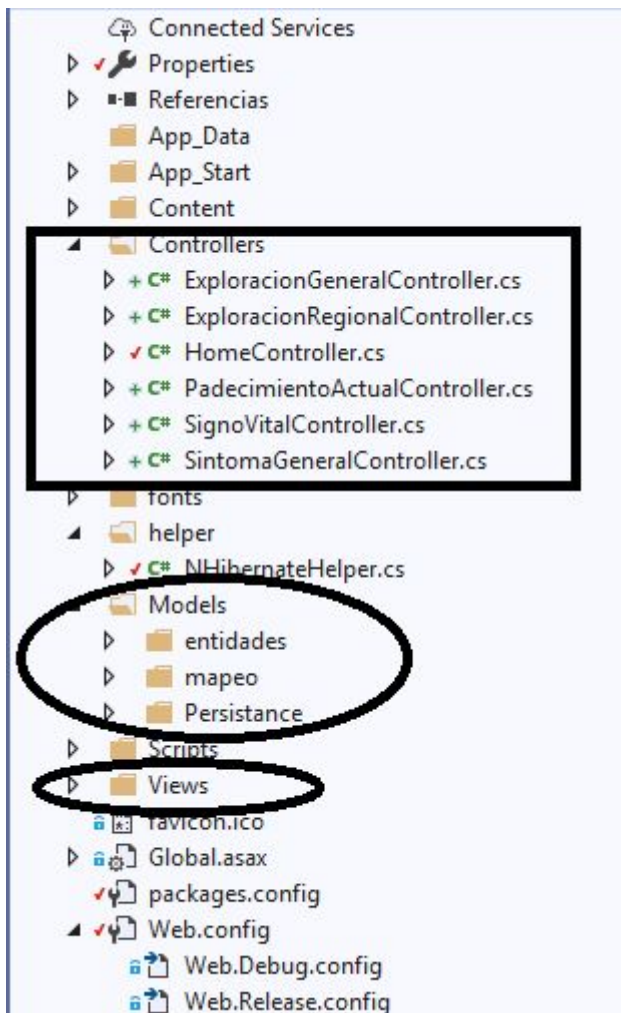
Contenido de la Base de Datos - Tabla SintomaGeneral:

127.0.0.1:52488/browser/
pgAdmin 4
File Object Tools Help
Browser
Databases (3)
ClinicaGeneral
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Schemas (1)
public
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions

sintomageneral
General Columns Constraints Advanced Parameters Security SQL
Inherited from table(s) Select to inherit from...
Columns
+

	Name	Data type	Length	Precision	Not NULL?	Primary key?
<input checked="" type="checkbox"/>	idsintomageneral	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	idexpediente	integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	sintoma	character varying	15		<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	valoracion	character varying	10		<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	observacion	character varying	300		<input type="checkbox"/>	<input type="checkbox"/>

modelo MVC (Back End de Clínica General)



ejemplo de un controlador implementado en c#  
 clase SintomaGeneralController  
 metodos get(), getAll(), post()

```

1  using System.Collections.Generic;
2  using System.Linq;
3  using System.Net;
4  using System.Web.Http;
5  using webapif.Models.entidades;
6  using webapif.Models.Persistencia;
7
8  namespace webapif.Controllers
9  {
10     public class SintomaGeneralController : ApiController
11     {
12         static readonly IServerDataRepository<SintomaGeneral> sgRepository = new ServerDataRepository<SintomaGeneral>();
13
14         [Route("api/sintomageneral/todos")]
15         [HttpGet]
16         public IEnumerable<SintomaGeneral> GetSintomaGeneral()
17         {
18             return sgRepository.GetAll();
19         }
20         [HttpGet]
21         public IEnumerable<SintomaGeneral> GetSintomaGeneralByIdExpediente(int idExpediente)
22         {
23             return sgRepository.GetAll().Where(d => d.idExpediente == idExpediente);
24         }
25         [HttpPost]
26         public void PostSintomaGeneral(SintomaGeneral sg)
27         {
28             if (!sgRepository.Add(sg))
29                 throw new HttpResponseException(HttpStatusCode.NotFound);
30         }
31     }
32 }

```

metodos put(), Delete()



```

29     }
30     [HttpPut]
31     public void PutSintomaGeneral(SintomaGeneral sg)
32     {
33
34         if (!sgRepository.Update(sg))
35             throw new HttpResponseException(HttpStatusCode.NotFound);
36     }
37
38     [HttpDelete]
39     public void DeleteSintomaGeneral(int id)
40     {
41
42         SintomaGeneral sg = new SintomaGeneral()
43         {
44             idSintomaGeneral = id
45         };
46
47         if (sg == null)
48             throw new HttpResponseException(HttpStatusCode.NotFound);
49         sgRepository.Delete(sg);
50     }

```

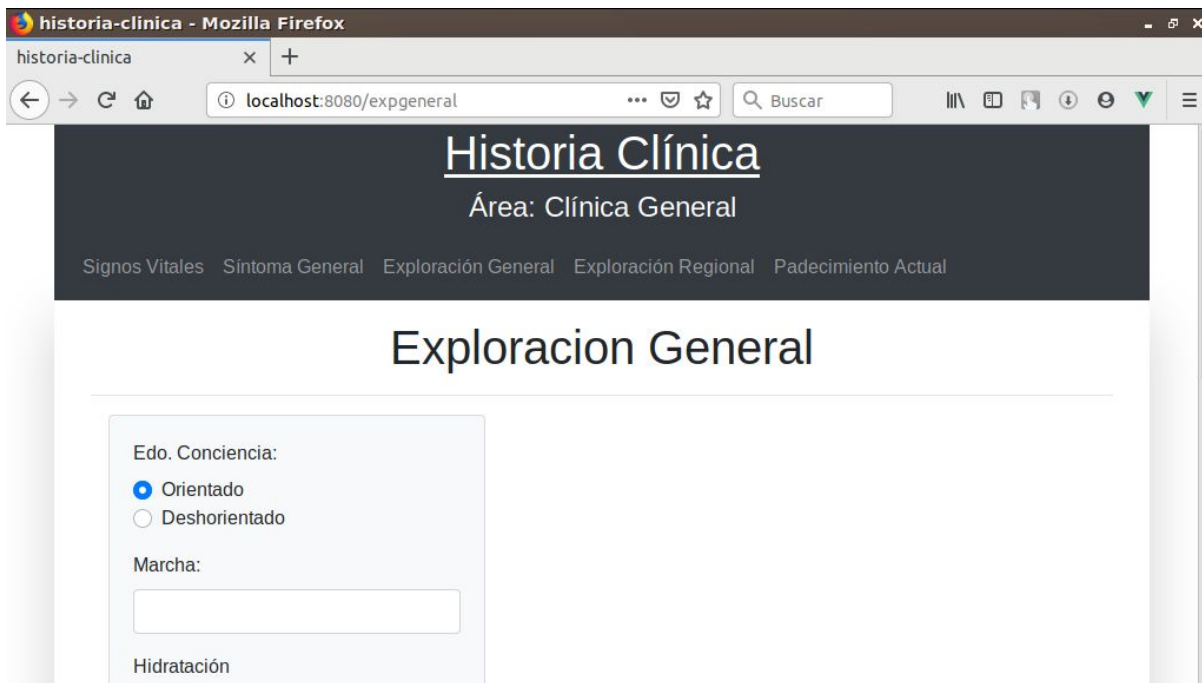
## configuración para llamas a los controladores

```

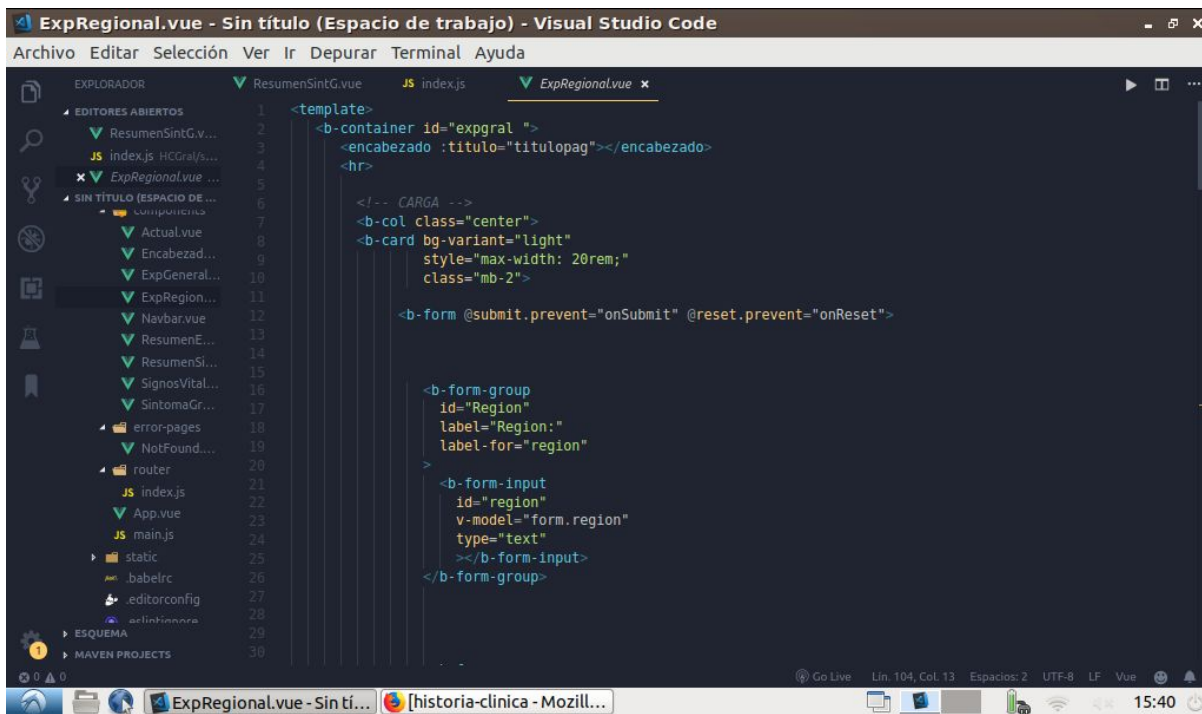
1  using System.Net.Http.Headers;
2  using System.Web.Http;
3
4  namespace webapif
5  {
6      public static class WebApiConfig
7      {
8          public static void Register(HttpConfiguration config)
9          {
10             // Configuración y servicios de API web
11             // Rutas de API web
12             config.MapHttpAttributeRoutes();
13             config.Routes.MapHttpRoute(
14                 name: "DefaultApi",
15                 routeTemplate: "api/{controller}/{id}",
16                 defaults: new { id = RouteParameter.Optional }
17             );
18             //api/signovital/idxpediente/valor
19             config.Routes.MapHttpRoute(
20                 name: "SignovitalByIdExpediente",
21                 routeTemplate: "api/{controller}/idxpediente/{idxpediente}"
22             );
23             //return JSON response by default
24             config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/html"));
25         }
26     }
27
--

```

## vistas (front End Clínica General)



## template de sintomas general en vue.js



## Implementación de Antecedentes clínicos

Para la implementación de esta parte del proyecto fue crear la tabla a través de clases usando .net core con las herramientas ORM de entity framework, usando visual studio 2017 y como motor de base de datos SQL Server 2014.

También se aplicó el Patrón Repository para las operaciones básicas (CRUD).

Dentro de antecedentes clínicos modelamos dos entidades Internaciones y Enfermedades de pacientes.

### Entidad InternacPaciente:

Lo que se hizo fue crear la clase entidad "InternacPacientes" que contiene los datos de la clase en sí de Internación, con sus atributos Id expediente, Id personal, Fecha de ingreso, Impresión diagnóstica, Tratamiento y la clave principal Id Internacion.

```
namespace historiasClinicas.Models
{
    public class InternacPaciente
    {
        [Key]
        [DatabaseGenerated (DatabaseGeneratedOption.Identity)]
        public int Id_internacion { get; set; }

        public int Id_expediente { get; set; }

        public int Id_personal { get; set; }

        [DataType(DataType.Date)]
        public DateTime Fecha_ingreso { get; set; }

        public string Imp_diagnostica { get; set; }
        public string Tratamiento { get; set; }
        public int HistoriaClinicaGral { get; set; }
    }
}
```

### Entidad EnfermedadPaciente:

```
public class EnfermedadPaciente
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int IdEnfermedad { get; set; }

    public string Enfermedad { get; set; }

    public string Edad { get; set; }

    [DataType(DataType.Date)]
    public DateTime Fecha_aprox { get; set; }

    public string Secuela { get; set; }

    public int HistoriaClinicaGral { get; set; }
}
```



### Clase DbContext:

En la clase DbContext se colocan los parámetros referidos a la migración de clase a tabla, como por ejemplo el string de conexión que indica el nombre de la base de datos a la cual se va a conectar.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace historiasClinicas.Models
{
    public class HcContext : DbContext
    {
        public DbSet<EnfermedadPaciente> EnfermedadPacientes { get; set; }
        public DbSet<InternacPaciente> InternacPacientes { get; set; }

        public HcContext(DbContextOptions<HcContext> options)
            : base(options)
        {
        }
    }
}
```

### Capa de acceso a los datos CRUD

Explicamos la implementación del patrón repository y unidad de trabajo con la entidad InternacionPaciente, con enfermedadPaciente hicimos lo mismo. La clase RepositoryInternacion es el patrón usado para poder realizar el CRUD para nuestra tabla de internación, mientras que en la clase InterfaceInternacion definimos como se enuncian dichas funciones para poder llamarlas desde el Program.

### Clase InterfaceInternacion:

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
using System.Linq.Expressions;

namespace Internacion.data
{
    public interface InterfaceInternacion<T> where T : class
    {
        IQueryable<T> GetAll();
        IQueryable<T> Find(Expression<Func<T, bool>> predicate);
        void Add(T entity);
        void Delete(T entity);
        void Edit(T entity);
        void Save();
    }
}

```

## Clase RepositoryInternacion:

Algunos ejemplos

### -Insertar:

```

public void Add(T entity)
{
    using (var dataContext = new InternacContext())
    {
        dataContext.Set<T>().Add(entity);
        dataContext.SaveChanges();
    }
}

```

### -Borrar:

```

public void Delete(T entity)
{
    using (var dataContext = new InternacContext())
    {
        dataContext.Set<T>().Remove(entity);
        dataContext.SaveChanges();
    }
}

```

### -Actualizar:

```

public void Edit(T entity)
{
    using (var dataContext = new InternacContext())
    {
        dataContext.Entry(entity).State = EntityState.Modified;
        dataContext.SaveChanges();
    }
}

```

En la clase programa colocamos el programa principal y las diferentes funciones del CRUD que vamos a poder realizar para simplemente ser enunciadas y ejecutadas dentro del programa principal.

### Programa:

```

using System;
using System.Linq;
using Internacion.data;
using Internacion.Servicio;

namespace Internacion
{
    class Program
    {
        static void Main(string[] args)
        {///En esta parte va el programa principal

            Actualizar();
            Console.ReadLine();

        }
    }
}

```

### Métodos descritos:

-Actualizar:

```

static void Actualizar()
{
    InternacionPacienteServicio Paciente = new InternacionPacienteServicio();
    //RepositoryInternacion<Internac_paciente> paciente = new RepositoryInternacion<Internac_paciente>();
    Internac_paciente inp = new Internac_paciente()
    {
        Id_internacion = 3,
        Id_expediente = 3,
        Id_personal = 4,
        Imp_diagnostica = "cancer",
        Tratamiento = "Paracetamol",
        Fecha_ingreso = DateTime.Now
    };

    Paciente.Edit(inp);
    Console.WriteLine(inp.Id_internacion);
    Console.ReadLine();
}

```

### -Borrar:

```

static void Borrar()
{
    ///Para realizar el borrado dependemos solamente de la clave Id_internacion,
    ///y con el proceso borramos toda la tabla.
    InternacionPacienteServicio Paciente = new InternacionPacienteServicio();
    //RepositoryInternacion<Internac_paciente> paciente = new RepositoryInternacion<Internac_paciente>();
    Internac_paciente inp = new Internac_paciente()
    {
        Id_internacion = 3,
    };

    Paciente.Delete(inp);
    Console.WriteLine(inp.Id_internacion);
    Console.ReadLine();
}

```

### -Insertar:

```

static void Insertar()
{
    InternacionPacienteServicio Paciente = new InternacionPacienteServicio();
    //RepositoryInternacion<Internac_paciente> paciente = new RepositoryInternacion<Internac_paciente>();
    Internac_paciente inp = new Internac_paciente()
    {
        //Id_internacion = 3,
        Id_expediente = 3,
        Id_personal = 4,
        Imp_diagnostica = "cancer",
        Tratamiento = "Paracetamol",
        Fecha_ingreso = DateTime.Now
    };

    Paciente.Add(inp);
    Console.WriteLine(inp.Id_internacion);
    Console.ReadLine();
}

```

## Contenido de la base de datos - Tabla InternacionPacientes

	Id_internacion	Id_expediente	Id_personal	Fecha_ingreso	Imp_diagnostica	Tratamiento
1	1	1	1	2019-05-16 19:33:18.0989444	Estress	Descanso
2	3	3	4	2019-05-16 20:11:16.7482757	cancer	Paracetamol
3	4	3	4	2019-05-16 20:07:41.6339719	Mareos	Paracetamol

## Hacemos la API WEB para Internaciones y Enfermedades

Por cada clase de nuestro modelo implementamos un controlador con los distintos métodos de REST, explicaremos el procedimiento para la clase de Internaciones.

### Controlador de Internaciones

En el definimos los distintos métodos de REST: Get, Post, Put y Delete.

```

namespace historiasClinicas.Controllers
{
    [Route("api/internacPac")]
    [ApiController]
    public class InternacPacienteController : ControllerBase
    {
        private readonly HcContext _context;

        public InternacPacienteController(HcContext context)
        {
            _context = context;

            if (_context.InternacPacientes.Count() == 0)
            {
                // Create a new TodoItem if collection is empty,
                // which means you can't delete all TodoItems.
                _context.InternacPacientes.Add(new InternacPaciente { Tratamiento = "Item1" });
                _context.SaveChanges();
            }
        }
    }
}

```

## GET

```

// GET: api/InternacPac
[HttpGet]
public async Task<ActionResult<IEnumerable<InternacPaciente>>> GetInternacPacientes()
{
    return await _context.InternacPacientes.ToListAsync();
}

// GET: api/InternacPac/{id}
[HttpGet("{id}")]
public async Task<ActionResult<InternacPaciente>> GetInternacPaciente(int id)
{
    var internacPac = await _context.InternacPacientes.FindAsync(id);

    if (internacPac == null)
    {
        return NotFound();
    }

    return internacPac;
}

```

## POST y PUT



```

// POST: api/InternacPac
[HttpPost]
public async Task<ActionResult<InternacPaciente>> PostEltoAntEvaluar(InternacPaciente item)
{
    _context.InternacPacientes.Add(item);
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetInternacPaciente), new { id = item.Id_internacion }, item);
}

// PUT: api/InternacPac/(id)
[HttpPut("{id}")]
public async Task<IActionResult> PutInternacPac(int id, InternacPaciente item)
{
    if (id != item.Id_internacion)
    {
        return BadRequest();
    }

    _context.Entry(item).State = EntityState.Modified;
    await _context.SaveChangesAsync();

    return NoContent();
}

```

## **DELETE**

```

// DELETE: api/InternacPac/(id)
[HttpDelete("{id}")]
public async Task<ActionResult> DeleteInternacPac(int id)
{
    var Item = await _context.InternacPacientes.FindAsync(id);

    if (Item == null)
    {
        return NotFound();
    }

    _context.InternacPacientes.Remove(Item);
    await _context.SaveChangesAsync();

    return NoContent();
}

```

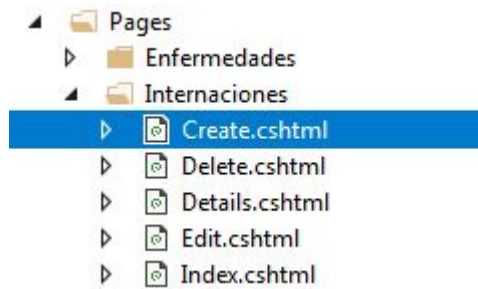
Todos esos métodos nos permiten acceder desde un navegador web o una herramienta como Postman, a la base de datos a la cual se conecta la API y en ella hacer los CRUD.

### **Front End de Internaciones**

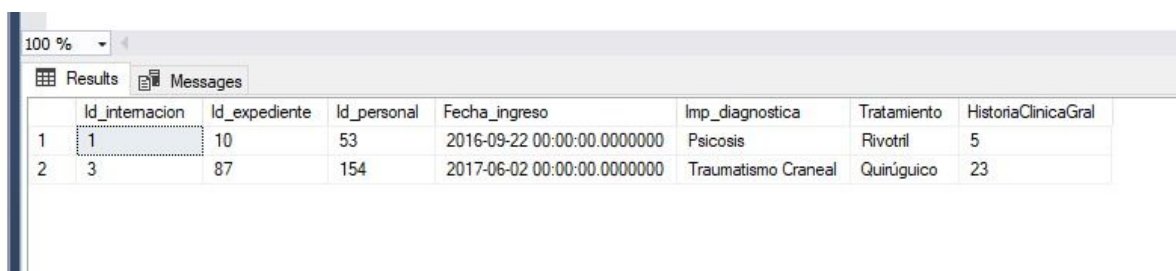
Para finalizar nuestro proyecto utilizando el patrón MVC nos faltaría implementar las vistas ya que tenemos los modelos en la capa de acceso a datos, y los controladores en la api web.

Para las vistas utilizamos Razor web page, y mediante la herramienta Scaffolding generamos la vista.

**Estas son las pantallas que me permiten hacer CRUD:**



**Todos los cambios que hagamos se ven reflejados en nuestra base de datos**



**El FrontEnd se conecta al BackEnd a través de la API**

Hicimos lo mismo con la clase enfermedades.



**Capa de acceso a los datos de Antecedentes**

**Entidad “Tipo de Antecedente”:**

```

namespace CapaAccesoAntecedentes.Models
{
    public class TipoAntecedente
    {
        //clave primaria

        [Key]
        public int IdTipoAnt { get; set; }

        //atributos

        public string NomTipoAnt { get; set; }

        //relacion "1 a muchos" con EltoAntEvaluar

        public ICollection<EltoAntEvaluar> EltoAntEvaluars { get; set; }
    }
}

```

### **Entidad “Elemento de Antecedente a Evaluar”:**

```

public class EltoAntEvaluar
{
    //clave primaria

    [Key]
    public int IdEltoAntEvaluar { get; set; }

    //atributos

    public string NombreEltoEvaluar { get; set; }
    public bool EstadoEltoEvaluar { get; set; }
    public string Observacion { get; set; }
    public int IdInternacion { get; set; }

    //relacion "1 a 1" con TipoAntecedente

    public TipoAntecedente TipoAntecedente { get; set; }
}

```

Para poder utilizar el patrón definimos un servicio para cada una de las entidades que contenga las funcionalidades del CRUD, que están implementadas genéricamente en la clase repository. Además también definimos una INTERFACE, para poder invocarlas y utilizar luego estas funcionalidades.

### Servicio para “Elemento de Antecedente a Evaluar”

```
namespace CapaAccesoAntecedentes.Servicios
{
    public class ServicioEltoAntEval: RepositorioAntecedentes<EltoAntEvaluar>
    {
    }
}
```

### Servicio para “Tipo de Antecedente”

```
namespace CapaAccesoAntecedentes.Servicios
{
    public class ServicioTipoAnt:RepositorioAntecedentes<TipoAntecedente>
    {
    }
}
```

### Interface:

```
namespace CapaAccesoAntecedentes.Repositorio
{
    public interface InterfaceAntecedentes<T> where T : class
    {
        IQueryable<T> GetAll();
        IQueryable<T> Find(Expression<Func<T, bool>> predicate);
        void Add(T entity);
        void Delete(T entity);
        void Edit(T entity);
        void Save();
    }
}
```

### El patrón repository:

```
namespace CapaAccesoAntecedentes.Repositorio
{
    public class RepositorioAntecedentes<T>:InterfaceAntecedentes<T> where T : class
    {
        internal DbContext _context;

        public RepositorioAntecedentes()
        {
            // using (var dbc = new InternacContext()) {
            //     _context = dbc;
            // };
        }
    }
}
```

```

    }

    public RepositorioAntecedentes(DbContext context)
    {
        _context = context;
    }

    public void Add(T entity)
    {
        using (var dataContext = new AntContext())
        {
            dataContext.Set<T>().Add(entity);
            dataContext.SaveChanges();
        }
    }

    public void Delete(T entity)
    {
        using (var dataContext = new AntContext())
        {
            dataContext.Set<T>().Remove(entity);
            dataContext.SaveChanges();
        }
    }

    public void Edit(T entity)
    {
        using (var dataContext = new AntContext())
        {
            dataContext.Entry(entity).State = EntityState.Modified;
            dataContext.SaveChanges();
        }
    }

    public IQueryable<T> Find(Expression<Func<T, bool>> predicate)
    {
        using (var dataContext = new AntContext())
        {
            IQueryable<T> query = dataContext.Set<T>().Where(predicate);
            return query;
        }
    }

    public IQueryable<T> GetAll()
    {
        throw new NotImplementedException();
    }

```

```

    public void Save()
    {
        //dataContext.SaveChanges();
    }
}

```

### El un ejemplo de un ADD, un UPDATE y un DELETE:

```

-----ADD de un NUEVO ELEMENTO DE ANTECEDENTE A EVALUAR
ServicioEltoAntEval eae = new ServicioEltoAntEval();
EltoAntEvaluar e = new EltoAntEvaluar();
EltoAntEvaluar e1 = new EltoAntEvaluar();

e.NombreEltoEvaluar = "Diabetes";
e.IdInternacion = 5;
e.Observacion = "lo tuvo el abuelo";
e.EstadoEltoEvaluar = true;

-----UPDATE de un TIPO DE ANTECEDENTE
ServicioTipoAnt ta = new ServicioTipoAnt();
TipoAntecedente t = new TipoAntecedente();
t.IdTipoAnt = 6;
t.NomTipoAnt = "otra cosa";
ta.Edit(t);
*/
//-----DELETE un TIPO DE ANTECEDENTE-----
ServicioTipoAnt ta = new ServicioTipoAnt();
TipoAntecedente t = new TipoAntecedente();
t.IdTipoAnt = 4;
ta.Delete(t);

```

### Como quedan las tablas en la base de datos:

	IdEltoAntEvaluar	NombreEltoEvaluar	EstadoEltoEvaluar	Observacion	IdInternacion	TipoAntecedenteIdTipoAnt
1	1	Diabetes	1	lo tuvo el abuelo	5	ADD NULL
2	2	Hipertension	1	tuvo su madre	8	NULL

	IdTipoAnt	NomTipoAnt
1	1	Patológicos
2	2	otra cosa
3	5	Quirúrgicos
4	6	otra cosa

UPDATE