ACADEMIA JAVA

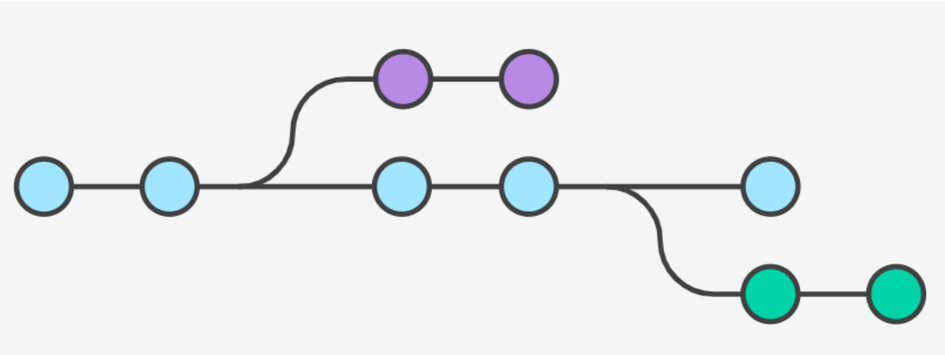
Luis Wonen Olvera Vasquez
Examen Semana 4

Lugar:Ciudad de México
Fecha: 16/12/2022

1. Comandos de Git

GIT

Es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.



Branch

Una rama representa una línea independiente de desarrollo. Las ramas sirven como una abstracción para el proceso de edición/etapa/confirmación.
El comando git branch permite crear, listar, renombrar y eliminar rama. No permite cambiar entre ramas o regresar a un punto en el historial de versiones. Por lo anterior el comando git branch esta sumamente integrado al uso del comando git checkout y git merge.

```
rama git [--color[=<cuando>] | --no-color] [--show-current]
        [-v [--abbrev=<n> | --no-abreviatura]]
        [--columna[=<opciones>] | --sin-columna] [--sort=<clave>]
        [--merged [<commit>]] [--no-merged [<commit>]]
        [--contiene [<commit>]] [--no-contains [<commit>]]
        [--apunta-a <objeto>] [--format=<formato>]
        [(-r | --remotos) | (-a | --todos)]
        [--lista] [<patrón>…]
git branch [--track[=(directo|heredado)] | --sin pista] [-f]
        [--recurse-submodules] <branchname> [<start-point>]
git branch (--set-upstream-to=<upstream> | -u <upstream>) [<branchname>]
 git branch --unset-upstream [<branchname>]
 git branch (-m | -M) [<oldbranch >] <nueva rama>
 git branch (-c | -C) [<oldbranch>] <nueva rama>
 git branch (-d | -D) [-r] <branchname>…
 git branch --edit-description [<branchname >]
```

Usos

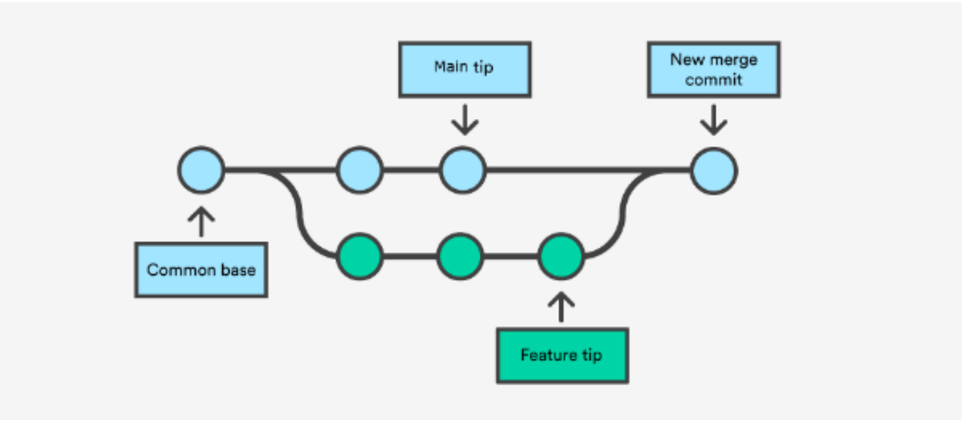
- git branch: Lista todas las ramas del repositorio.
- git branch <nombre_rama>: Crea una nueva rama con el nombre asignado.
- git branch -D <nombre_rama>: Elimina la rama a la que le corresponde el nombre, es una operación segura ya que previene eliminar la rama si tiene cambios que no están fusionados.
- git branch -m <nombre_rama>: Fuerza la eliminación de una rama incluso si los cambios no son fusionados.
- git branch -a: Cambia el nombre de la rama actual.

1.1 Comandos de Git

Merge

El comando git merge permite tomar las líneas independientes de desarrollo creadas por git branch e integrarlas en una sola rama. La rama actual se actualizará para reflejar la fusión, pero la rama de destino no se verá afectada en absoluto.

git merge se suele utilizar junto con git checkout para seleccionar la rama actual y git branch -d para eliminar la rama de destino obsoleta.



Puntos importantes

- La fusión en Git combina secuencias de confirmaciones en un solo historial unificado de confirmaciones.
- En Git hay principalmente dos tipos de fusión: con avance rápido y de tres vías.
- Git permite fusionar las confirmaciones automáticamente salvo que haya cambios que entren en conflicto en ambas secuencias de confirmación.

Usos

- git merge –abort: Cancela el proceso de resolución de conflictos actual e intenta reconstruir el estado anterior a la fusión. Si hay una entrada de autostash, aplíquela al árbol de trabajo.
- git merge –commit: Realiza la fusión y confirma el resultado.
- git merge –edit: Invoca a un editor antes de realizar una fusión para seguir editando el mensaje de combinación generado automáticamente, de modo que el usuario pueda explicar y justificar la combinación.
- git merge –ff: Especifica cómo se maneja una fusión cuando el historial fusionado ya es un descendiente del historial actual.

```
git merge [-n] [--stat] [--no-commit] [--squash] [--[no-]edit]
        [--no-verify] [-s <strategy>] [-X <strategy-option>] [-S[<keyid>]]
        [--[no-]allow-unrelated-histories]
        [--[no-]rerere-autoupdate] [-m <msg>] [-F <file>]
        [--into-name <branch>] [<commit>…]
git merge (--continue | --abort | --quit)
```

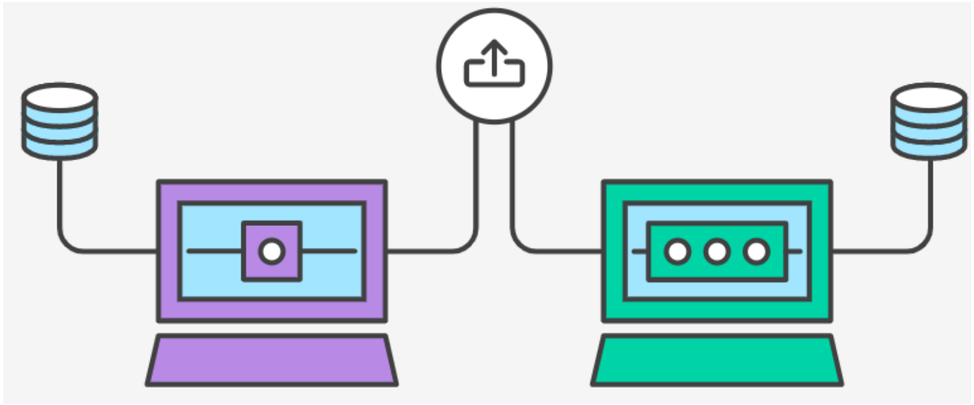
1.2 Comandos de Git

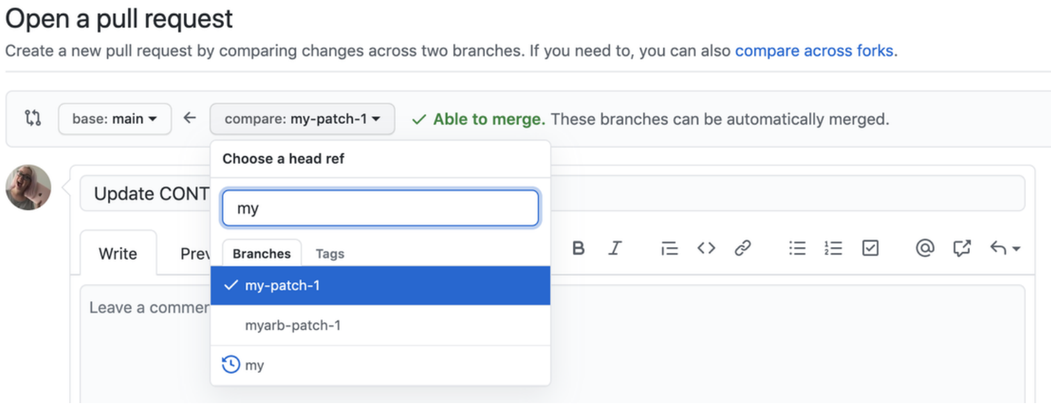
Pull Request

El comando pull request es un mecanismo para notificar a los miembros del equipo que se a completado una tarea y desea que esta sea aprobada por el encargado del repositorio, con el objetivo de fusionar la tarea con el proyecto.
Para comnprender mejor este proceso proporcionare un ejemplo. Supongamos se quiere contribuir a un proyecto opensource en GitHub, por lo tanto hacemos fork en el repositorio que deseamos contribuir.

Después usamos el comando git clone, ya que nos permite tener una copia del proyecto en nuestro equipo local. Creamos una rama con el comando git branch, en esta podemos trabajar en la parte del proyecto que queremos contribuir.

Modificado el código en nuestra rama, realizamos un git commit y después un git push, esto hará los cambios sean accesibles a los colaboradores. Después se realiza el pull request en donde el encargado repositorio revisara y fusionara o no, la característica en la que se trabajó.





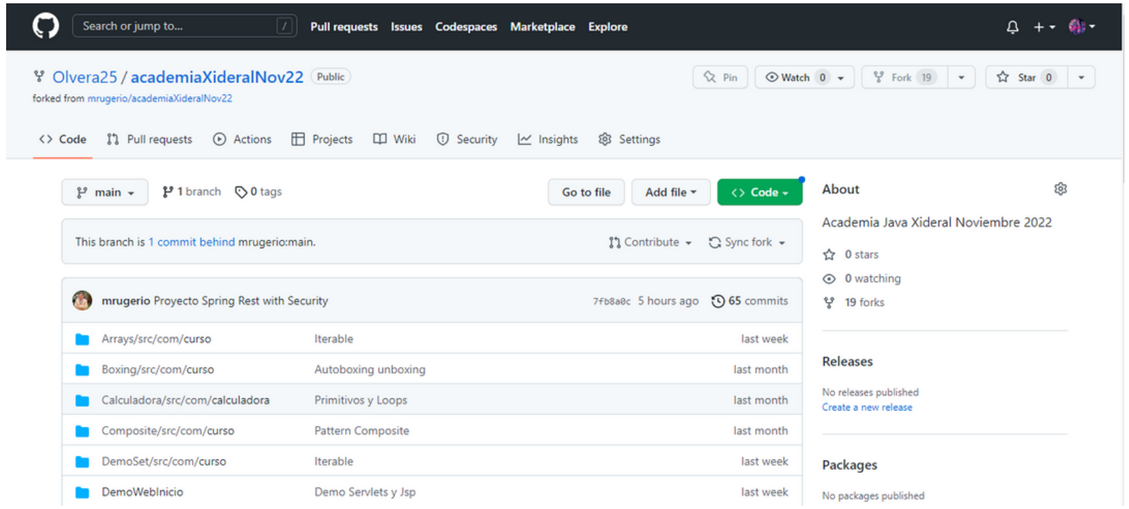
1.3 Comandos de Git

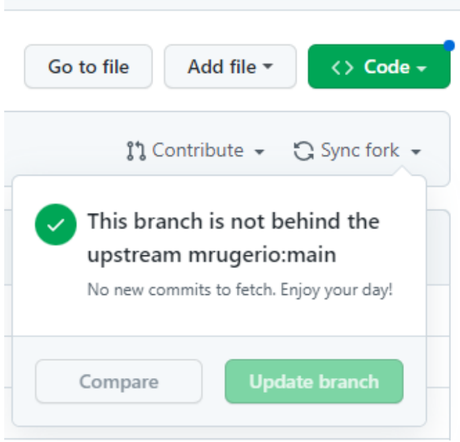
Fork

Un fork consiste en un nuevo repositorio que comparte la configuración de visibilidad y código con el repositorio “ascendente” original. Los Fork se suelen usar para iterar ideas o cambios antes de que se vuelvan a proponer al repositorio ascendente, como en proyectos de código abierto o cuando un usuario no tiene acceso de escritura al repositorio ascendente.

Se puede ramificar un proyecto para proponer cambios en el repositorio ascendente. En este caso, es una buena práctica sincronizar tu fork periódicamente con el repositorio ascendente.

Un ejemplo de este proceso es el que realizamos al inicio de la academia, entramos a la plataforma GitHub, y buscamos el repositorio academiaXideralNov22. El siguiente paso fue ir a la parte derecha de la página y dar click en Fork nos envía a una interface en donde se asigna el Owner y el nombre del repositorio. De la misma forma podemos seleccionar si solo queremos la rama principal o más. Para finalizar damos click en el botón Create fork.





1.4 Comandos de Git

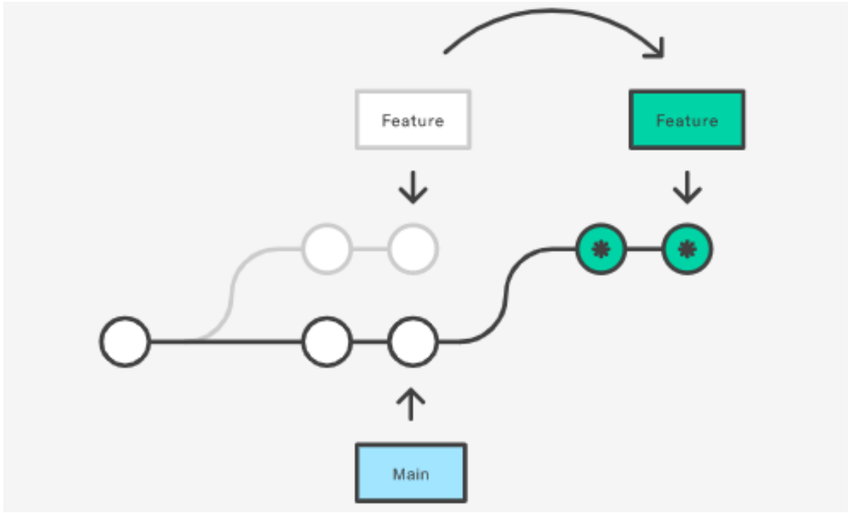
Rebase

El comando git rebase permite realizar cambios de forma más sencilla a commits, modificaciones al historial del repositorio. Permite reordenar, editar o squash commits. Algunos de los usos del git rebase son:

- Editar mensajes previos de commits.
- Combinar multiples commits en uno.
- Elimina o revierte commits que ya no son necesarios.

Usos

- git rebase --p: Usa el commit.
- git rebase --r: usa el commit, pero edita el mensaje.
- git rebase --e: usa el commit, pero hace alto para modificar.
- git rebase --s: usa el commit, pero lo fusiona con el commit anterior.
- git rebase --f: es como el comando –s, pero descarta el mensaje log del commit.
- git rebase --x: ejecuta el comando usando la Shell.




```
git rebase [-i | --interactive] [<options>] [--exec <cmd>]
        [--onto <newbase> | --keep-base] [<upstream> [<branch>]]
git rebase [-i | --interactive] [<options>] [--exec <cmd>] [--onto
<newbase>]
        --root [<branch>]
git rebase (--continue | --skip | --abort | --quit | --edit-todo | --show-
current-patch)
```

1.5 Comandos de Git

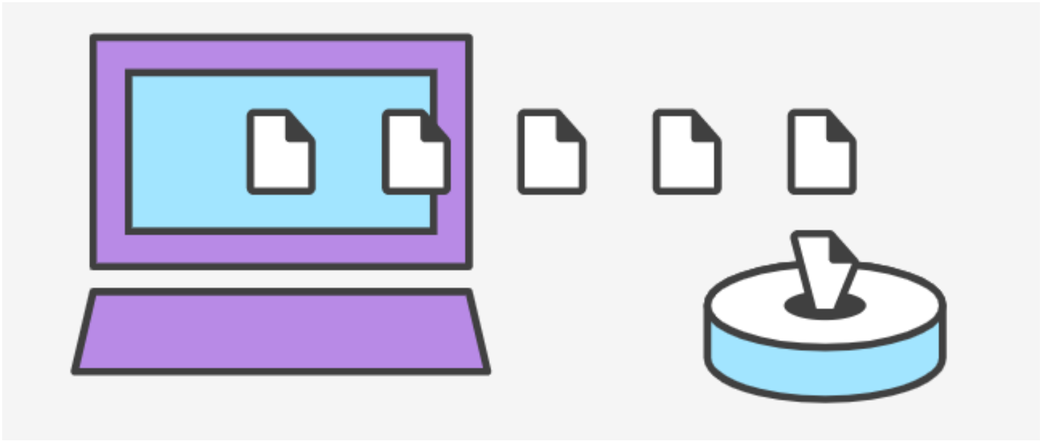
Stash

El comando git Stash guarda los cambios en un directorio de trabajo sucio. El comando git stash almacena temporalmente (o guarda en un stash) los cambios que hayas efectuado en el código en el que estás trabajando para que puedas trabajar en otra cosa y, más tarde, regresar y volver a aplicar los cambios más tarde.

El comando git stash coge los cambios sin confirmar (tanto los que están preparados como los que no), los guarda a parte para usarlos más adelante y, acto seguido, los deshace en el código en el que estás trabajando. Llegados a este punto, tienes libertad para hacer cambios, crear confirmaciones, cambiar de rama y efectuar cualesquiera otras operaciones de Git; y, luego, regresar y volver a aplicar el stash cuando lo tengas todo listo.

Usos

- git stash list: Se utiliza para enumerar las modificaciones guardadas.
- git stash show: Se utiliza para inspeccionar una modificación.
- git stash apply: Se utiliza para restaurar (potencialmente sobre un commit diferente)

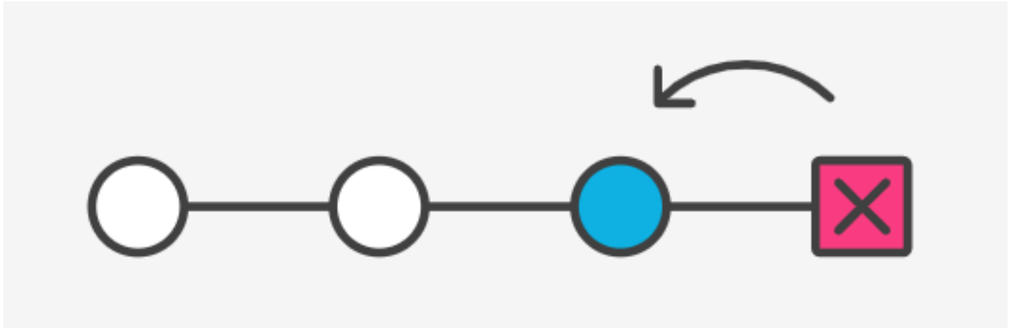


```
git stash list [<opciones de registro>]
 git stash show [-u | --incluir sin seguimiento | --only-untracked]
[<diff-options>] [<stash>]
 git stash drop [-q | --quiet] [<stash>]
 git stash pop [--index] [-q | --quiet] [<stash>]
 git stash apply [--index] [-q | --quiet] [<stash>]
 git stash branch <branchname> [<stash>]
 git stash [push [-p | --parche] [-S | --escenificado] [-k | --
[no-]mantener-índice] [-q | --tranquilo]
             [-tú | --include-sin seguimiento] [-a | --todos] [(-m | --
mensaje) <mensaje>]
             [--pathspec-from-file=<archivo> [--pathspec-file-nul]]
             [--] [<especificación de ruta>…]]
Guardar alijo de git [-p | --parche] [-S | --escenificado] [-k | --
[no-]mantener-índice] [-q | --tranquilo]
             [-tú | --include-sin seguimiento] [-a | --todos] [<mensaje>]
git stash clear
 git stash create [<mensaje>]
 git stash store [(-m | --message) <mensaje>] [-q | --quiet] <compromiso>
```

1.6 Comandos de Git

Clean

El comando git clean remueve archivos sin seguimiento del árbol de trabajo.Los archivos sin seguimiento son archivos que han sido creados dentro del directorio del repositorio de trabajo pero que no han sido añadidos al índice de seguimiento del repositorio mediante el comando git add.

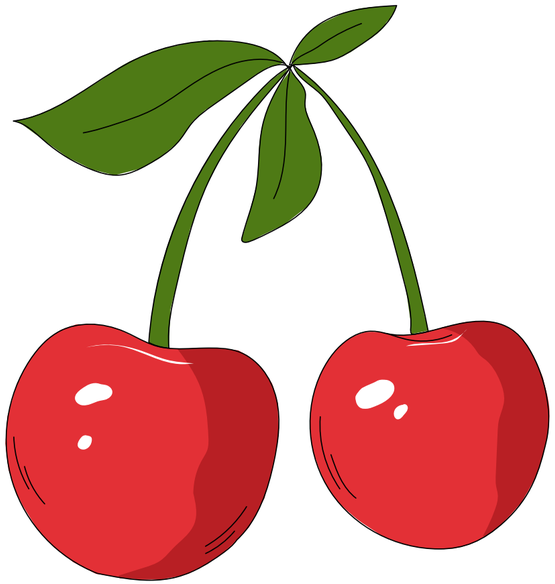


Usos

- git clean -n: realizará una ejecución en seco, por lo que mostrará que archivos se eliminarán sin eliminarlos realmente.
- git clean -f: inicia la eliminación real de archivos sin seguimiento del directorio actual.
- git clean -d: Normalmente, cuando no se especifica <path>, git clean no recurrirá a directorios sin seguimiento para evitar eliminar demasiado. Especifique -d para que también se repita en dichos directorios.
- git clean -i: Muestra lo que se haría y limpia los archivos de forma interactiva.
- git clean -e: Usa el patrón de exclusión dado, además de las reglas estándar de ignore.

```
git clean [-d] [-f] [-i] [-n] [-q] [-e <patrón>] [-x | -X] [--]
[<especificación de ruta>…]
```

1.7 Comandos de Git



Cherry Pick

git cherry-pick es un comando poderoso que permite seleccionar confirmaciones de Git arbitrarias por referencia y agregarlas al HEAD de trabajo actual. La selección de cherry es el acto de seleccionar una confirmación de una rama y aplicarla a otra. git cherry-pick puede ser útil para deshacer cambios.

git cherry-pick es una herramienta útil pero no siempre una buena práctica. La selección selectiva puede causar compromisos duplicados y muchos escenarios en los que la selección selectiva funcionaría, en su lugar se prefieren las fusiones tradicionales.

Usos

- git cherry-pick -edit: con esta opción hará que git solicite un mensaje de confirmación antes de aplicar la operación de selección selectiva.
- git cherry-pick: ejecutará la selección del cherry, pero en lugar de realizar una nueva confirmación, moverá el contenido de la confirmación de destino al directorio de trabajo de la rama actual.
- git cherry-pick --signoff: agregará una línea de firma de 'aprobación' al final del mensaje de confirmación seleccionado.

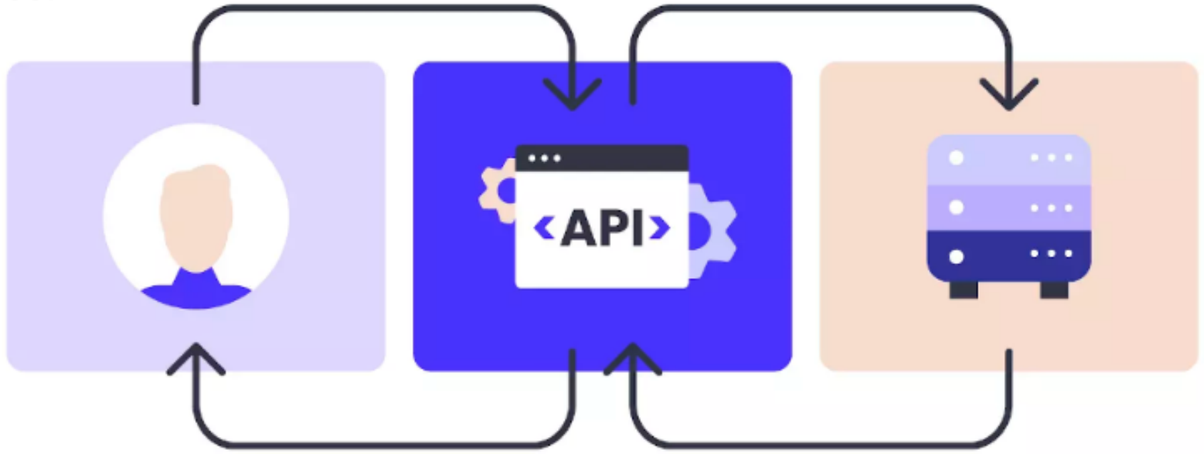
```
git cherry-pick [--edit] [-n] [-m <parent-number>] [-s] [-x] [--ff]
                [-S[<keyid>]] <commit>…
git cherry-pick (--continue | --skip | --abort | --quit)
```

2. Mejores prácticas de un API REST

API

Una interfaz de programa de aplicación (API) define las reglas que se deben seguir para comunicarse con otros sistemas de software. Los desarrolladores exponen o crean API para que otras aplicaciones puedan comunicarse con sus aplicaciones mediante programación. Se puede pensar en una API web como una puerta de enlace entre los clientes y los recursos de la Web.

Los clientes son usuarios que desean acceder a información desde la Web. El cliente puede ser una persona o un sistema de software que utiliza la API. Los recursos son la información que diferentes aplicaciones proporcionan a sus clientes. Los recursos pueden ser imágenes, videos, texto, números o cualquier tipo de datos.



2.1 Mejores prácticas de un API REST

REST

La transferencia de estado representacional (REST) es una arquitectura de software que impone condiciones sobre cómo debe funcionar una API. En un principio, REST se creó como una guía para administrar la comunicación en una red compleja como Internet.

Es posible utilizar una arquitectura basada en REST para admitir comunicaciones confiables y de alto rendimiento a escala. Puede implementarla y modificarla fácilmente, lo que brinda visibilidad y portabilidad entre plataformas a cualquier sistema de API.

Métodos HTTP

- **GET:** Los clientes utilizan GET para acceder a los recursos que están ubicados en el URL especificado en el servidor. Pueden almacenar en caché las solicitudes GET y enviar parámetros en la solicitud de la API REST para indicar al servidor que filtre los datos antes de enviarlos.
- **POST**: Los clientes usan POST para enviar datos al servidor. Incluyen la representación de los datos con la solicitud. Enviar la misma solicitud POST varias veces produce el efecto secundario de crear el mismo recurso varias veces.

2.2 Mejores prácticas de un API REST

Métodos HTTP

- **PUT**: Los clientes utilizan PUT para actualizar los recursos existentes en el servidor. A diferencia de POST, el envío de la misma solicitud PUT varias veces en un servicio web RESTful da el mismo resultado.

- **DELETE**: Los clientes utilizan la solicitud DELETE para eliminar el recurso. Una solicitud DELETE puede cambiar el estado del servidor. Sin embargo, si el usuario no cuenta con la autenticación adecuada, la solicitud fallará.

- **HEAD**: El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

- **CONNECT**: El método CONNECT establece un túnel hacia el servidor identificado por el recurso.

- **TRACE**: El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

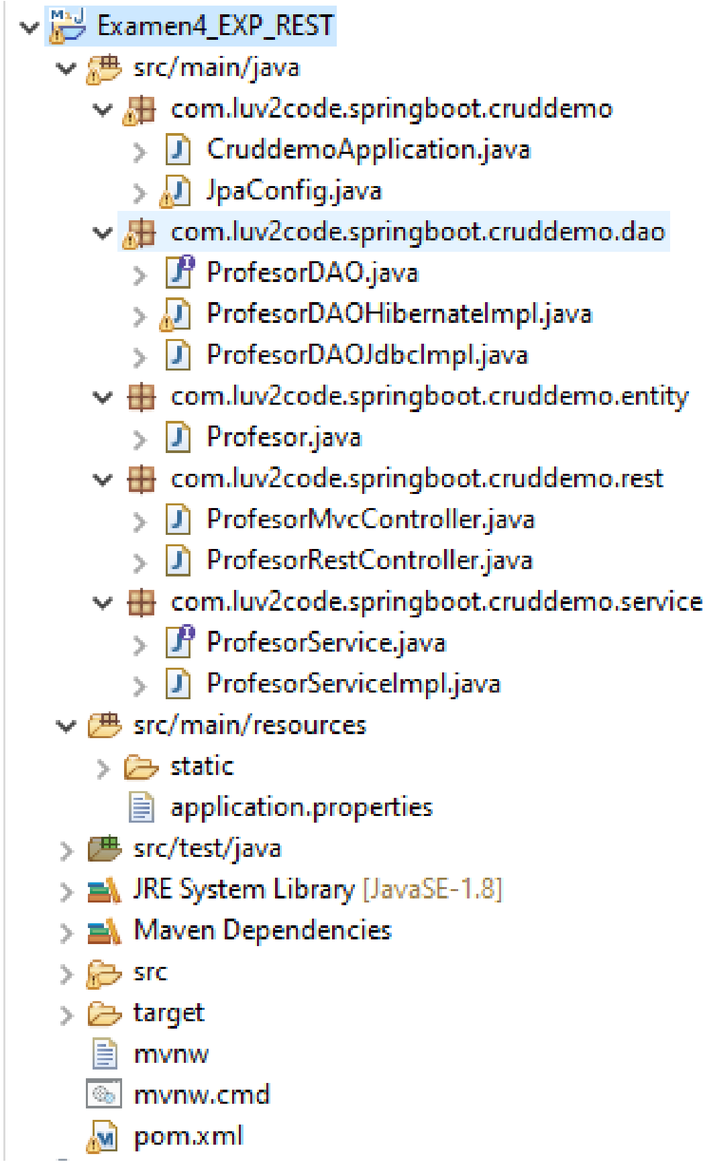
- **PATCH**: El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

3. Exponer un servicio REST

El objetivo de este proyecto es exponer un servicio REST, por lo cual dentro de la carpeta Examenes/Semana 4 creamos el proyecto Examen4_EXP_REST. El proyecto consiste en exponer el servicio REST devolviendo información en formato JSON, el proyecto tendrá conexión a una base de datos MySQL mediante hibernate asi como también JDBC.

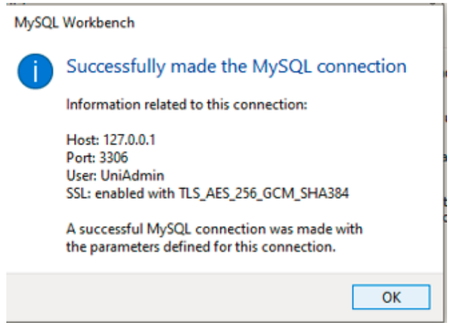
El proyecto Examen4_EXP_REST es un proyecto spring boot por lo cual tiene el servidor embebido, esto significa que al momento de ejecutarse el proyecto ocuparemos la clase CruddemoApplication y la ejecutaremos como Java Application. Este proyecto tiene la amabilidad de contener mvc y jsp aunque no son necesarios ya que el único proposito es exponer el servicio REST.

Dentro del proyecto debemos conocer como estamos haciendo la conexión, si es mediante Hibernate o JDBC, en este caso seleccionamos JDBC. Como podemos comprobar lo anterior, bueno debemos ir a la clase ProfesorServiceImpl, después del @Qualifier y comprobamos si implementa la clase profesorDAOJdbcImpl o profesorDAOHibernateImpl.

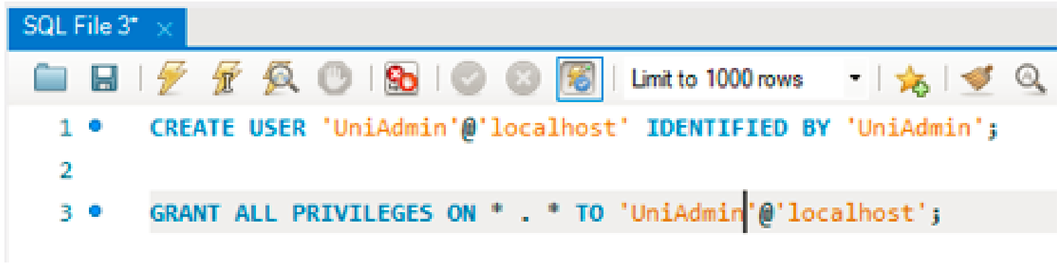


3.1 Exponer un servicio REST

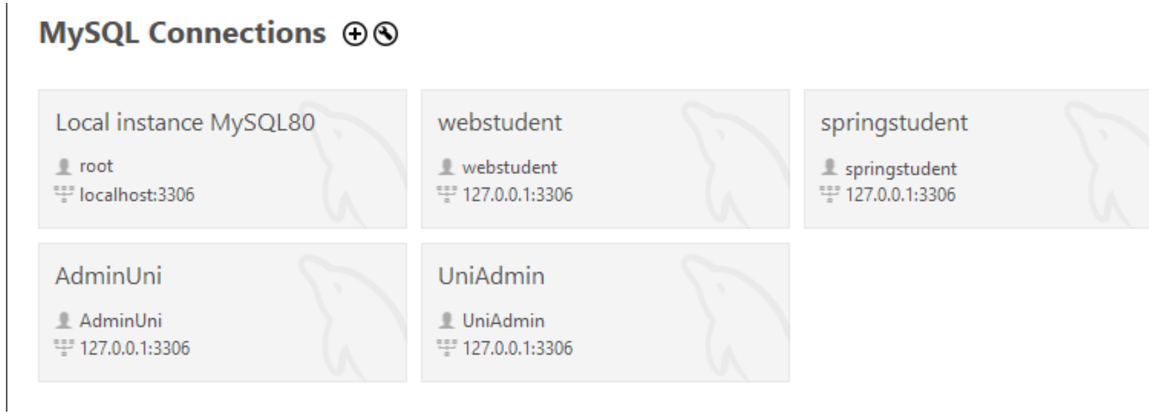
A continuación usaremos la herramienta MySQL Workbech para crear un usuario, en este caso UniAdmin después se aplicarán los permisos necesarios. El siguiente paso es entrar al usuario UniAdmin y asignar una contraseña después probar la conexión.

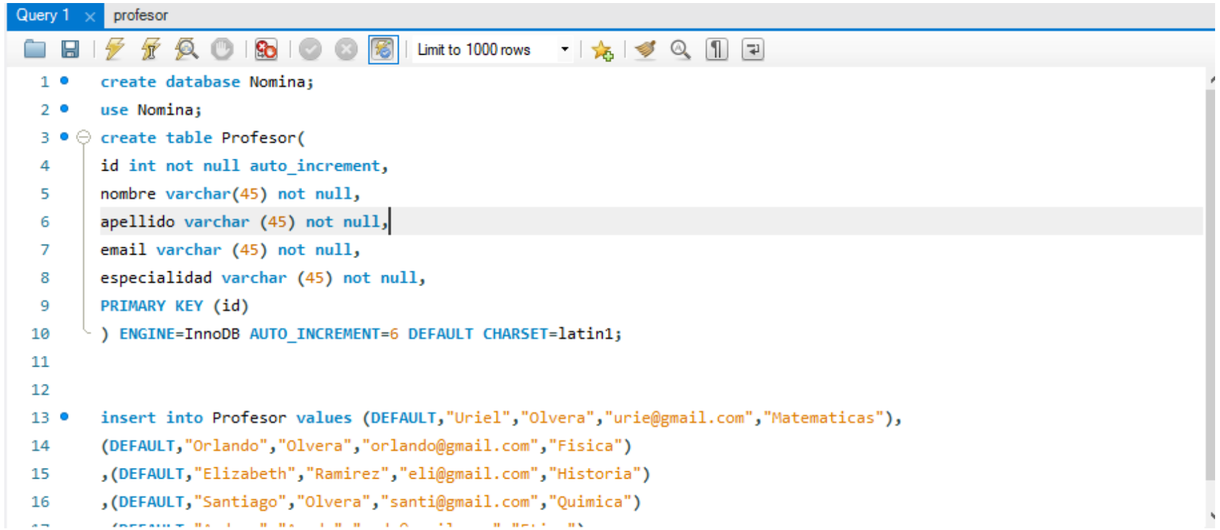


Una vez realizado lo anterior, creamor una base datos llamada Nomina la cual tendrá una tabla llamada Profesor que contiene los elementos: id auto incrementable, nombre, apellido, email y especialidad.



Una vez creada la tabla añadimos algunos elementos, con el propósito de utilizarlos posteriormente con el uso de los métodos http.





3.2 Exponer un servicio REST

Gracias a la clase ProfesorServiceImpl sabemos que implementaremos JDBC. Ahora teniendo la base de datos que utilizamos, necesitamos saber en que archivo realizamos la conexión a la base de datos. Para JDBC realizamos la conexión mediante el archivo JpaConfig mientras que si usáramos Hibernate lo haríamos en el archivo Application properties.

El archivo appplication properties también nos sirve para determinar el puerto, en este caso ocupamos el puerto 7777. Dentro de la carpeta entity tenemos el archivo Profesor aquí declaramos los atributos que contiene la tabla nomina a la que accedemos, en nuestro caso es id, nombre, apellido, email y especialidad.

Un aspecto importante a considerar es la dirección url, ya que esta cambiará dependiendo la actividad que deseamos realizar, en nuestro caso tenemos el archivo ProfesorRestController, este contiene las adiciones necesarias para la url.

Para esto es importante considerar las anotaciones @GetMapping, @PostMapping, @PutMapping y @DeleteMapping.

ProfesorServiceImpl.java

```java
package com.luv2code.springboot.cruddemo.service;

import java.util.List;

@Service
public class ProfesorServiceImpl implements ProfesorService {


    private ProfesorDAO profesorDAO;

    @Autowired
    public ProfesorServiceImpl(@Qualifier("profesorDAOJdbcImpl") ProfesorDAO theProfesorDAO) {
        profesorDAO = theProfesorDAO;
    }
```

JpaConfig.java

```java
package com.luv2code.springboot.cruddemo;

import javax.sql.DataSource;

@Configuration
public class JpaConfig {

    //https://howtodoinjava.com/spring-boot2/datasource-configuration/
    @Bean(name = "mySqlDataSource")
    @Primary
    public DataSource mySqlDataSource()
    {
        DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();
        dataSourceBuilder.url("jdbc:mysql://localhost:3306/nomina?useSSL=false&serverTimezone=UTC");
        dataSourceBuilder.username("UniAdmin");
        dataSourceBuilder.password("UniAdmin");
        return dataSourceBuilder.build();
    }
}
```

3.3 Exponer un servicio REST

Para comprobar que nuestro proyecto funciona debemos ejecutar la clase CruddemoApplication, cuando el proyecto termine de ejecutarse entramos a el navegador de preferencia y escribimos localhost:7777/rest/profesors, esto porque ocupamos el puerto 7777 y el resto porque nos basamos en el archivo ProfesorRestController.

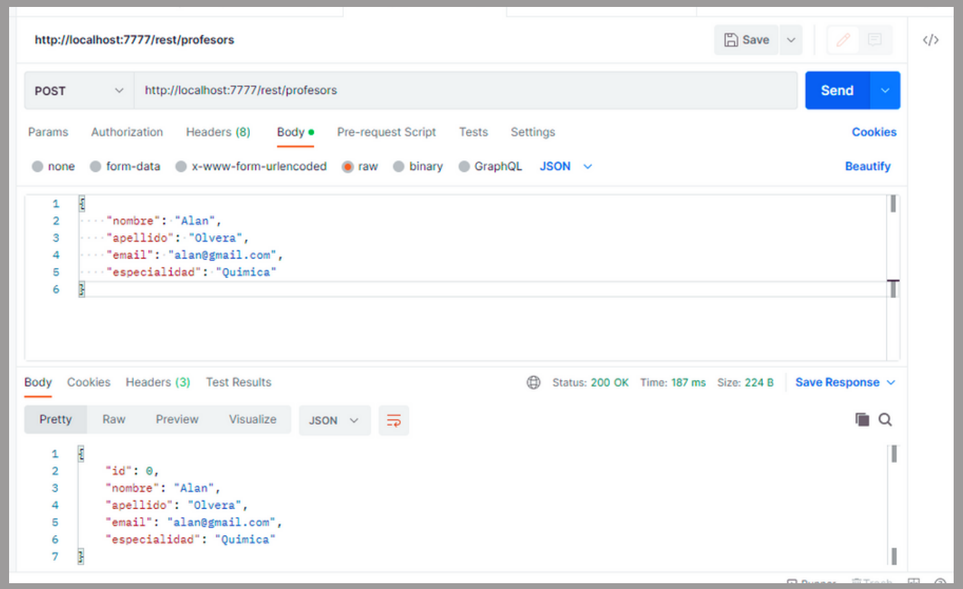
Ocuparemos la herramienta Postman, la cual nos ayuda mediante mpetodos HTTP a obtener, actualizar, agregar o eliminar elementos de nuestra base de datos. Por lo anterior Postman nos ayudará a comprobar que nuestro proyecto funcione de manera correcta.

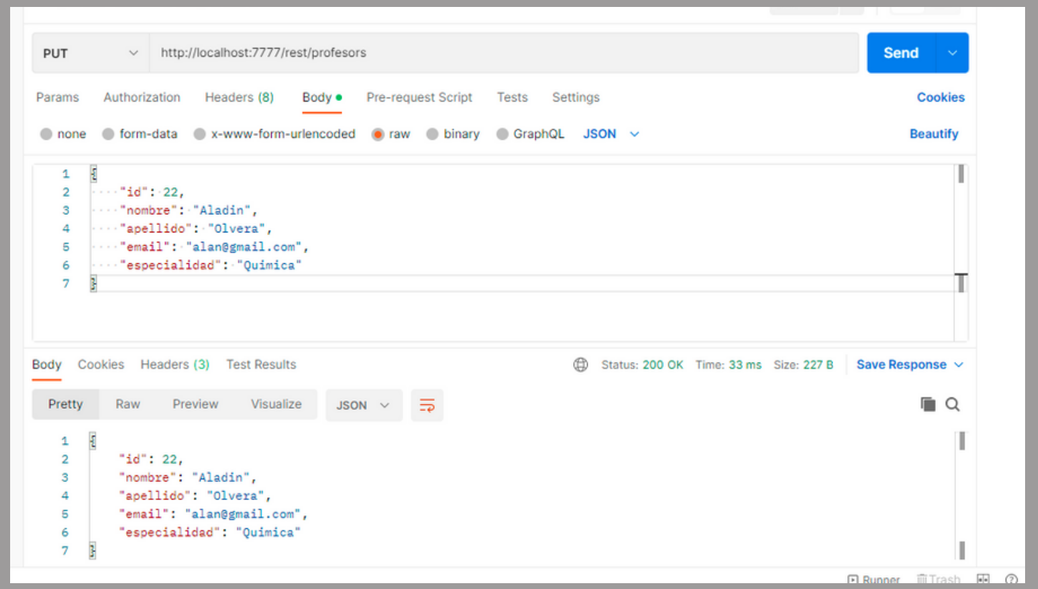
ProfesorRestController.java

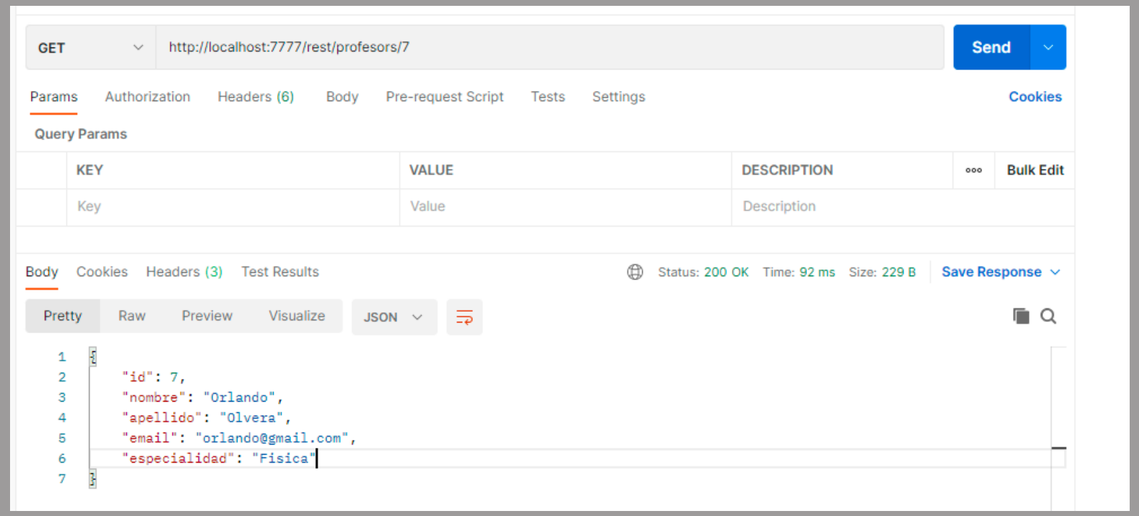
```java
package com.luv2code.springboot.cruddemo.rest;

import java.util.List;

@RestController
@RequestMapping("/rest")
public class ProfesorRestController {

    private ProfesorService profesorService;

    @Autowired
    public ProfesorRestController(ProfesorService theProfesorService) {
        profesorService = theProfesorService;
    }

    // expose "/profesors" and return list of profesors
    @GetMapping("/profesors")
    public List<Profesor> findAll() {
        return profesorService.findAll();
    }

    // add mapping for GET /profesors/{profesorId}

    @GetMapping("/profesors/{profesorId}")
    public Profesor getProfesor(@PathVariable int profesorId) throws Exception {

        Profesor theProfesor = profesorService.findById(profesorId);
```

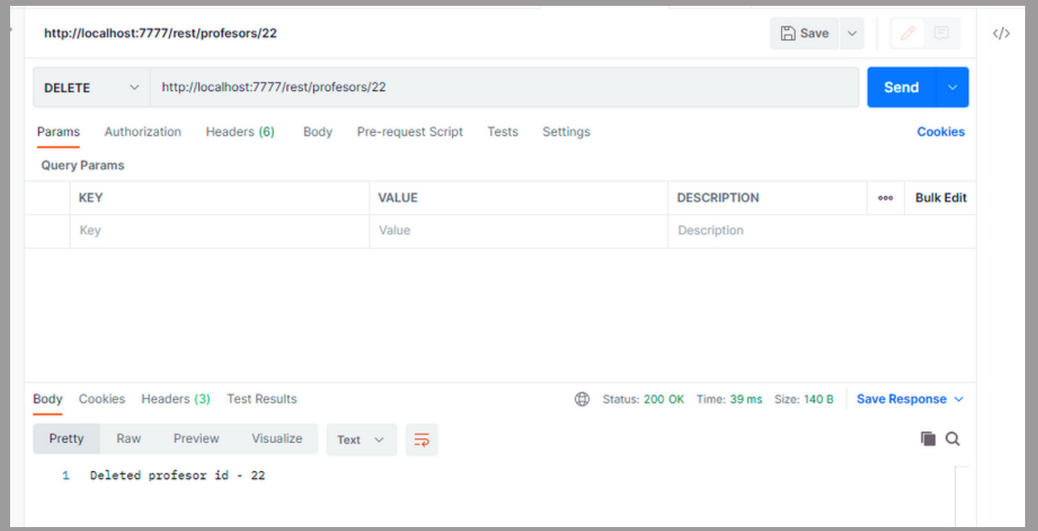

```
[{"id":6,"nombre":"Uriel","apellido":"Olvera","email":"urie@gmail.com","especialidad":"Matematicas"},
{"id":7,"nombre":"Orlando","apellido":"Olvera","email":"orlando@gmail.com","especialidad":"Fisica"},
{"id":8,"nombre":"Elizabeth","apellido":"Ramirez","email":"eli@gmail.com","especialidad":"Historia"},
{"id":10,"nombre":"Andrea","apellido":"Agudo","email":"andy@gmail.com","especialidad":"Etica"},
{"id":12,"nombre":"Alondra","apellido":"Jimenez","email":"alo@gmail.com","especialidad":"Ciencias"},
{"id":19,"nombre":"Estefani","apellido":"Ramirez","email":"estef@gmail.com","especialidad":"Ciencias"},
{"id":21,"nombre":"Oscar","apellido":"Daniel","email":"oscar21@gmail.com","especialidad":"Derecho"}]
```

3.4 Exponer un servicio REST







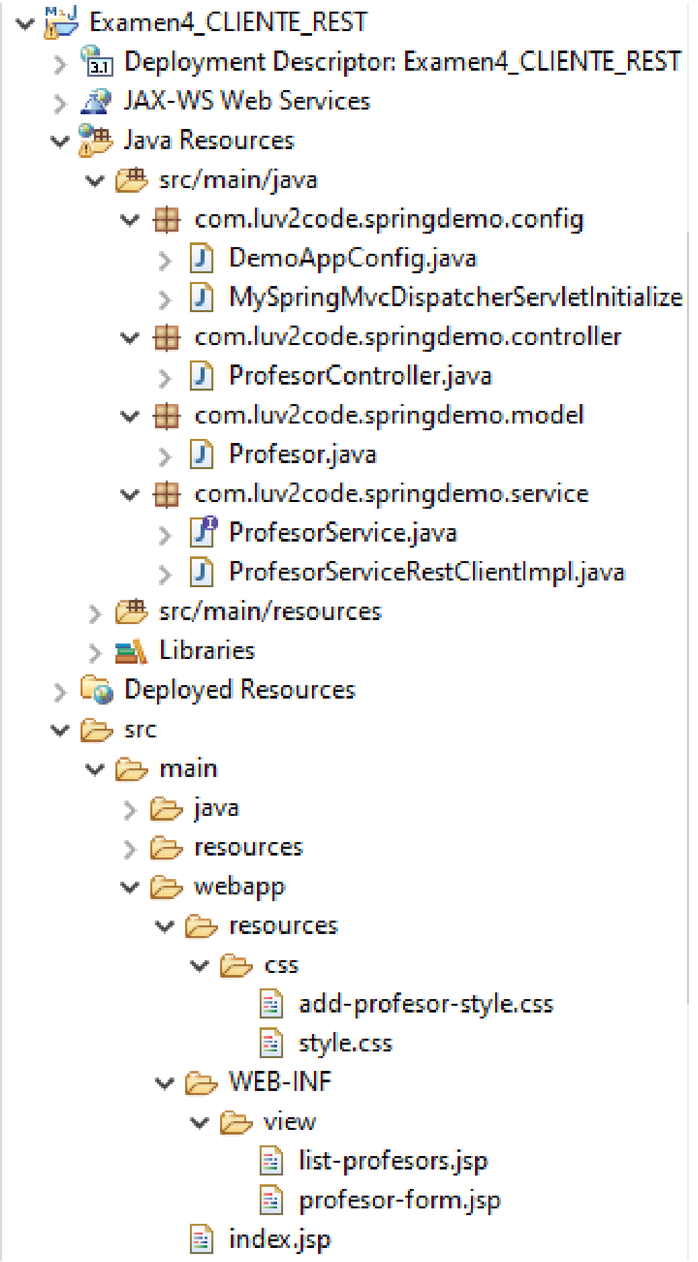


4. Exponer un Cliente REST

El objetivo de este proyecto es consumir un servicio REST, por lo cual dentro de la carpeta Examenes/Semana 4 creamos el proyecto Examen4_CLIENTE_REST. El proyecto cliente consiste en consumir el servicio REST del proyecto Examen4_EXP_REST. Por lo tanto asumimos que el cliente no tiene acceso directo a la base de datos sino que en lugar de conectarse a la misma, se conecta al proyecto que expone el servicio.

Este proyecto se ejecuta a través del server Apache Tomcat 9.0, por lo tanto para saber en que puerto se ejecutará, revisamos el servidor en este caso será en el puerto 8080. Por lo tanto para ejecutar este proyecto es necesario que primero se este ejecutando el proyecto que expone el servicio y después dar click derecho al proyecto cliente y dar click en Run on Server.

Este proyecto tiene el propósito de proporcionarle la vista al usuario, esto a través de diversos archivos JSP y se proporciona el estilo a partir de CSS. Una coincidencia que tendremos con el proyecto que expone el servicio es la clase Profesor.



4.1 Exponer un Cliente REST

Donde realizamos la conexión al proyecto que expone el servicio, esto lo encontramos en el documento index. Además es importante tener en cuenta que el ProfesorServiceClientImpl nos hace cargar el resto de la url.

Como entramos a la vista es otra pregunta que nos debemos hacer, sabemos que el proyecto se ejecuta de forma local por lo que utilizaremos el localhost, y como revisamos anteriormente se utiliza apache tomcat el cual se ejecuta en el puerto 8080. La pregunta ahora es que pasa con el resto de la url, bueno para seguir necesitamos revisar la clase ProfesorController.

Dentro de la clase ProfesorController debemos poner atención a las anotaciones @RequestMapping y @GetMapping ya que estas serán la clave para acceder a nuestra página principal además del contexto de nuestro programa. Por lo tanto la url correcta es localhost:8080/nomina/profesors/list.

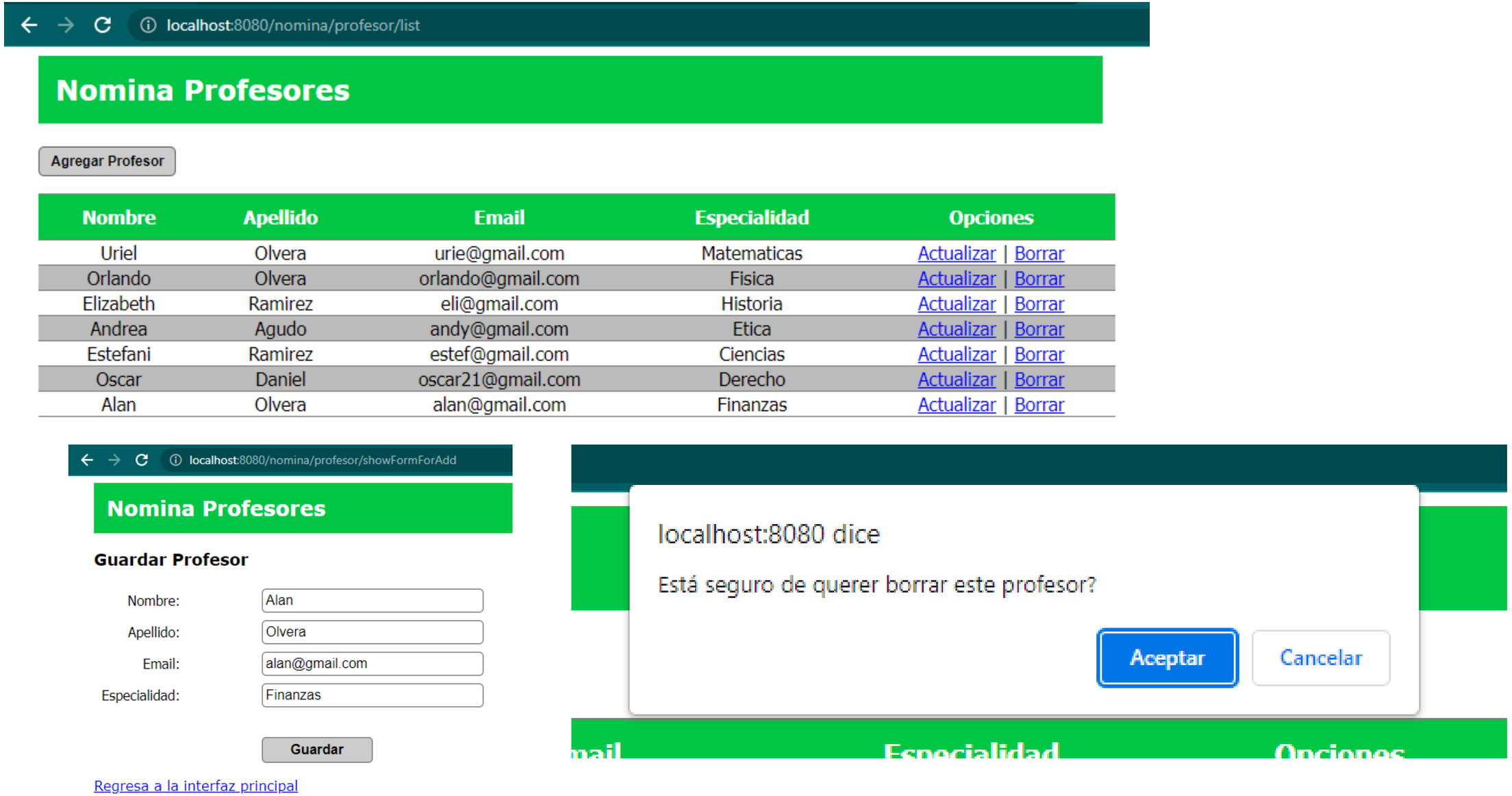
ProfesorServiceRestClientImpl.java

```java
package com.luv2code.springdemo.service;

import java.util.List;

@Service
public class ProfesorServiceRestClientImpl implements ProfesorService {

    private RestTemplate restTemplate;

    private String crmRestUrl;

    private Logger logger = Logger.getLogger(getClass().getName());

    @Autowired
    public ProfesorServiceRestClientImpl(RestTemplate theRestTemplate,
                                         @Value("${crm.rest.url}") String theUrl) {

        restTemplate = theRestTemplate;
        crmRestUrl = theUrl;

        logger.info("Loaded property:  crm.rest.url=" + crmRestUrl);
    }
```

ProfesorController.java

```java
package com.luv2code.springdemo.controller;

import java.util.List;

@Controller
@RequestMapping("/profesor")
public class ProfesorController {

    // need to inject our profesor service
    @Autowired
    private ProfesorService profesorService;

    @GetMapping("/list")
    public String listProfesors(Model theModel) {

        // get profesors from the service
        List<Profesor> theProfesors = profesorService.getProfesors();

        // add the profesors to the model
        theModel.addAttribute("profesors", theProfesors);

        return "list-profesors";
```

4.2 Exponer un Cliente REST



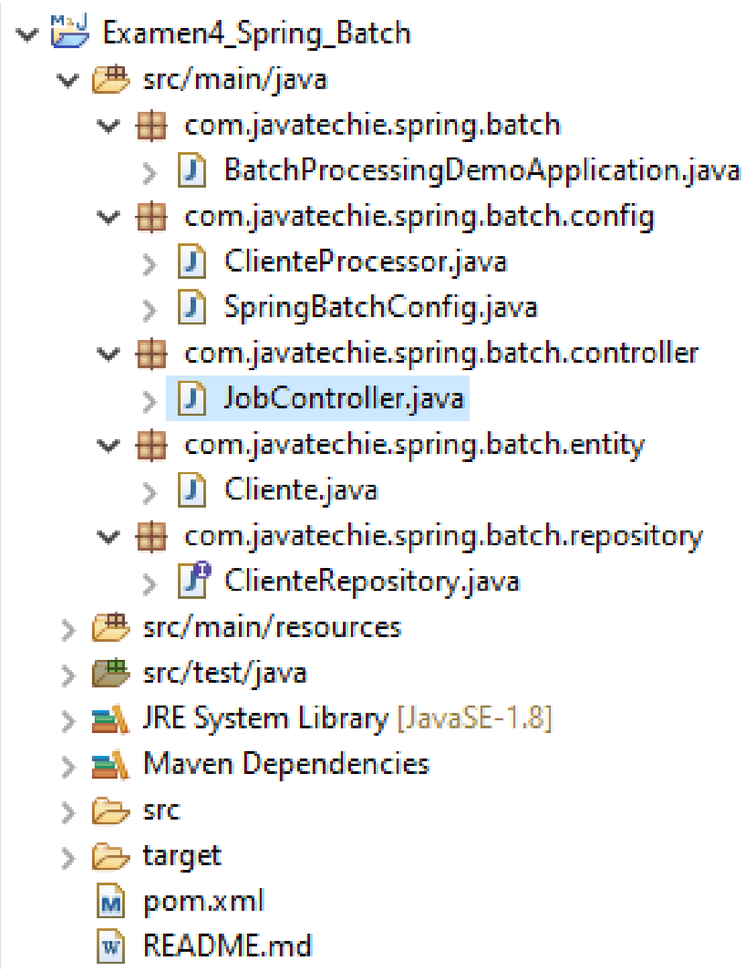
5. Proyecto Spring Batch

El objetivo principal de este proyecto es la comprensión del ciclo spring batch, ya que este proyecto lo implementa. Por lo tanto dentro de la carpeta Examenes/Semana 4 encontraremos el proyecto Examen4_Spring_Batch.

Una de las clases principales y donde podemos observar el proceso spring batch es la clase SpringBatchConfig, ya que en esta podemos ver el proceso Job el cual puede contener uno o muchos Steps, además el Step tiene al menos dos procesos entrada y salida. Mientras que en la clase JobController tenemos el proceso JobLauncher.

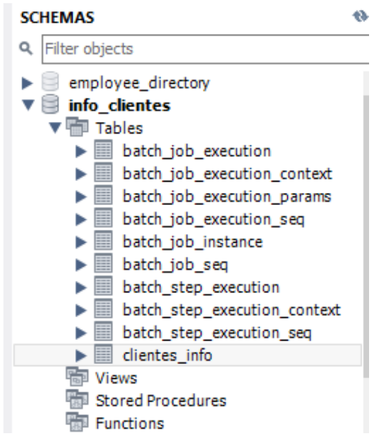
Podemos observar que en este proyecto no tenemos la implementación del DAO con Hibernate o JDBC ya que este proyecto lo hacemos con el uso de JPA, implementado en la clase ClienteRepository.

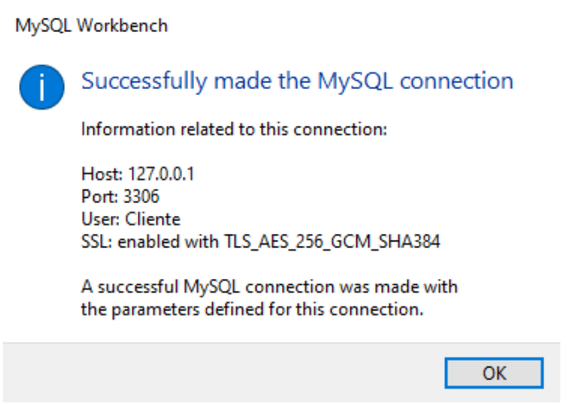
JPA nos ayuda mucho ya que con solo la clase Cliente realiza todo el proceso que antes involucraba clases y código que ahora nos ahorramos. La clase Cliente contiene los atributos de la tabla que se creara al ejecutar el método POST.



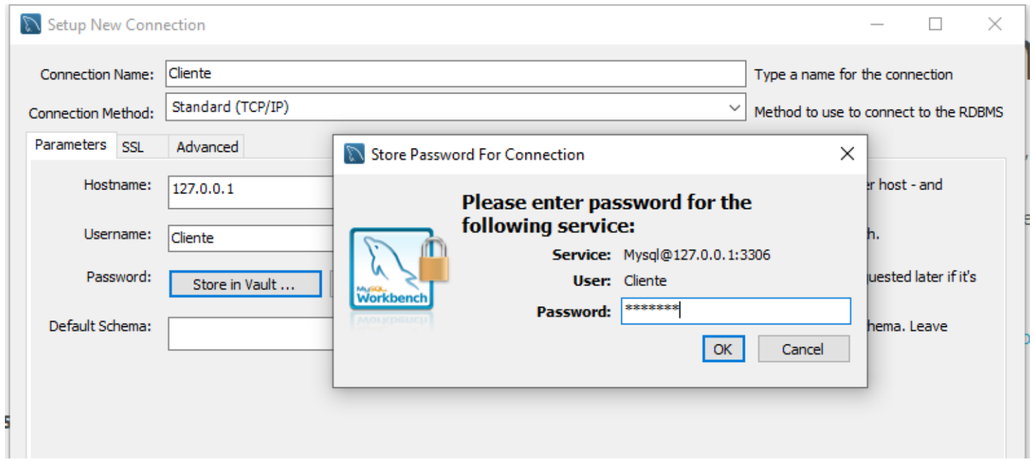
5.1 Proyecto Spring Batch

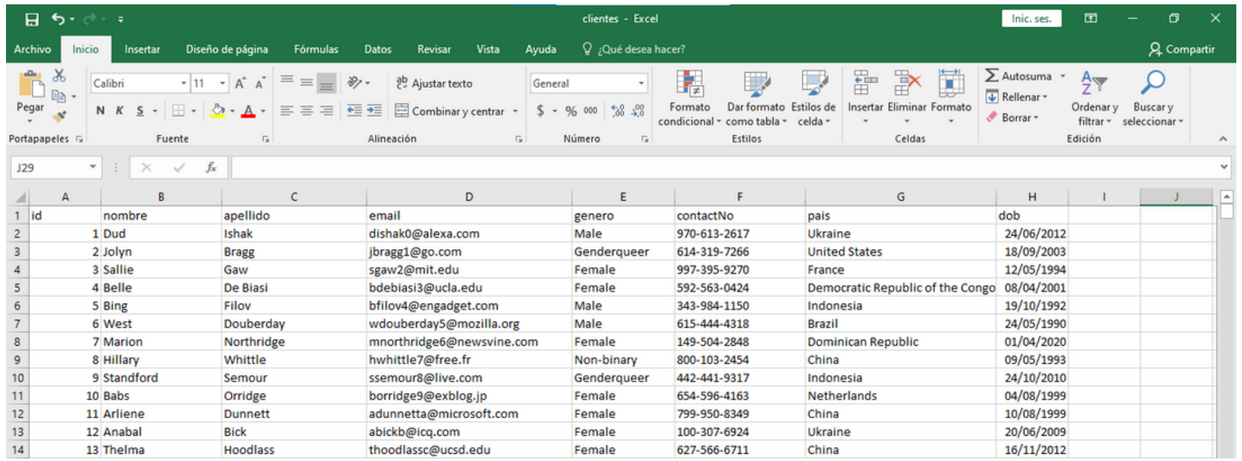
Para la base de datos utilizaremos la herramienta MySQL Workbech, en este proyecto lo único que necesitaremos, es crear un usuario y una base de datos. Para realizar la conexión a la base de datos ocuparemos el archivo application properties, en donde pondremos la url de la base de datos, el nombre del usuario, la contraseña y el puerto donde funcionara el proyecto.





Cuando ejecutemos el proyecto, ¿Cómo importaremos los datos del documento clientes.csv ?. Es una buena pregunta esto lo resolveremos con la clase JobController ya que revisando las anotaciones nos damos cuenta que para que los datos se manden a la base de datos debemos ejecutar el método Post. Por lo que utilizaremos la herramienta Postman indicando la siguiente url localhost:9191/jobs/importClientes con el método POST.



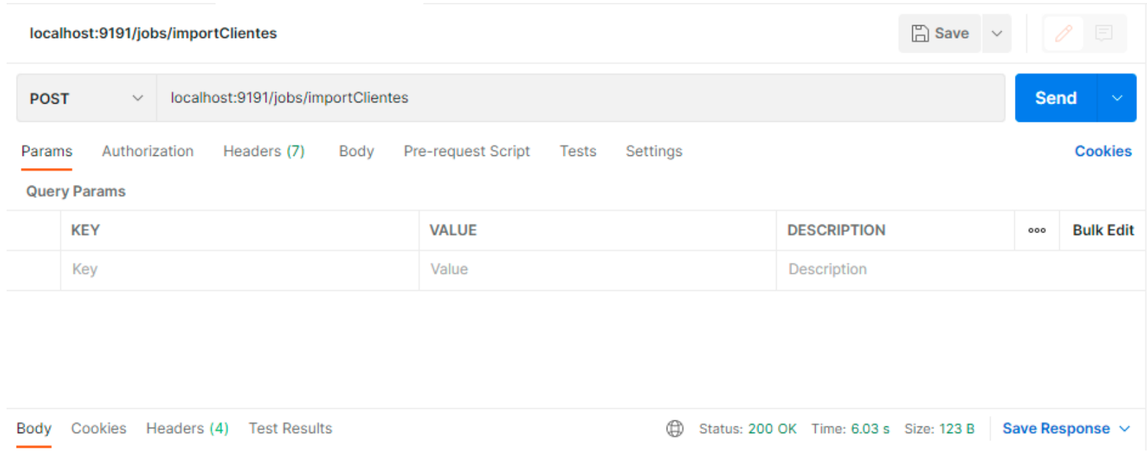


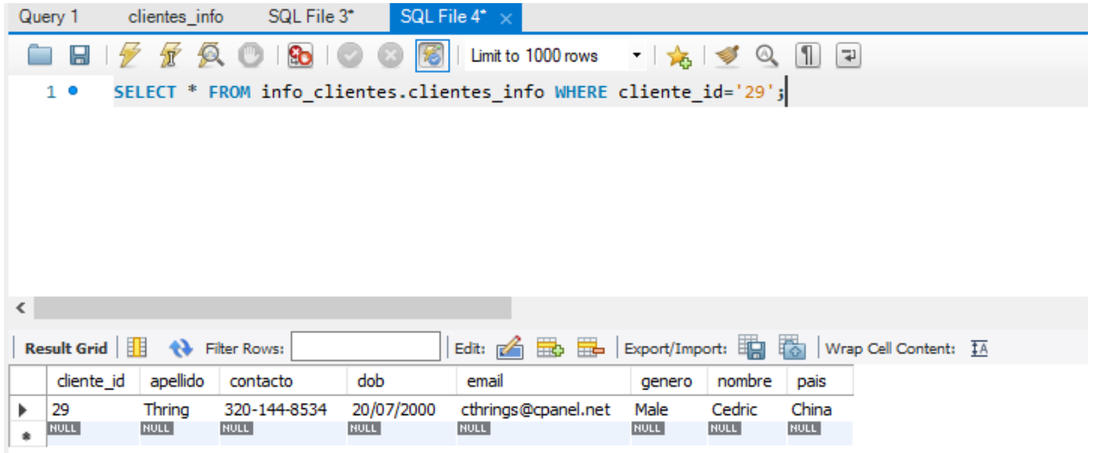
clientes - Excel

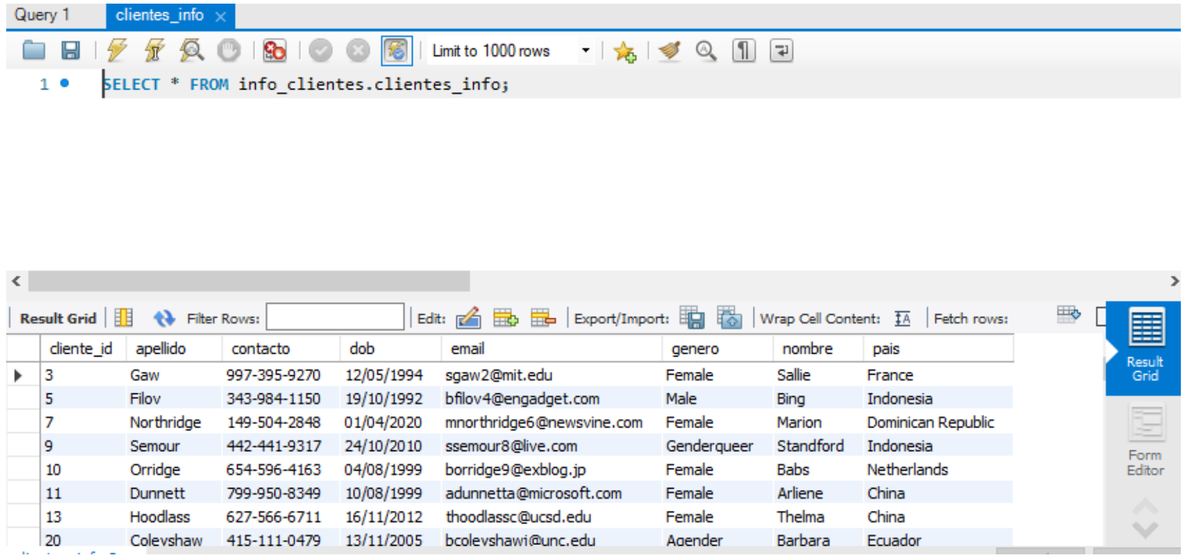
id	nombre	apellido	email	genero	contactNo	pais	dob
1	Dud	Ishak	dishak0@alexa.com	Male	970-613-2617	Ukraine	24/06/2012
2	Jolyn	Bragg	jbragg1@go.com	Genderqueer	614-319-7266	United States	18/09/2003
3	Sallie	Gaw	sgaw2@mit.edu	Female	997-395-9270	France	12/05/1994
4	Belle	De Biasi	bdebiasi3@ucla.edu	Female	592-563-0424	Democratic Republic of the Congo	08/04/2001
5	Bing	Filov	bfilov4@engadget.com	Male	343-984-1150	Indonesia	19/10/1992
6	West	Douberday	wdouberday5@mozilla.org	Male	615-444-4318	Brazil	24/05/1990
7	Marion	Northridge	mnorthridge6@newsvine.com	Female	149-504-2848	Dominican Republic	01/04/2020
8	Hillary	Whittle	hwhittle7@free.fr	Non-binary	800-103-2454	China	09/05/1993
9	Standford	Semour	ssemour8@live.com	Genderqueer	442-441-9317	Indonesia	24/10/2010
10	Babs	Orridge	borridge9@exblog.jp	Female	654-596-4163	Netherlands	04/08/1999
11	Arliene	Dunnett	adunnetta@microsoft.com	Female	799-950-8349	China	10/08/1999
12	Anabal	Bick	abickb@icq.com	Female	100-307-6924	Ukraine	20/06/2009
13	Thelma	Hoodlass	thoodlassc@ucsd.edu	Female	627-566-6711	China	16/11/2012

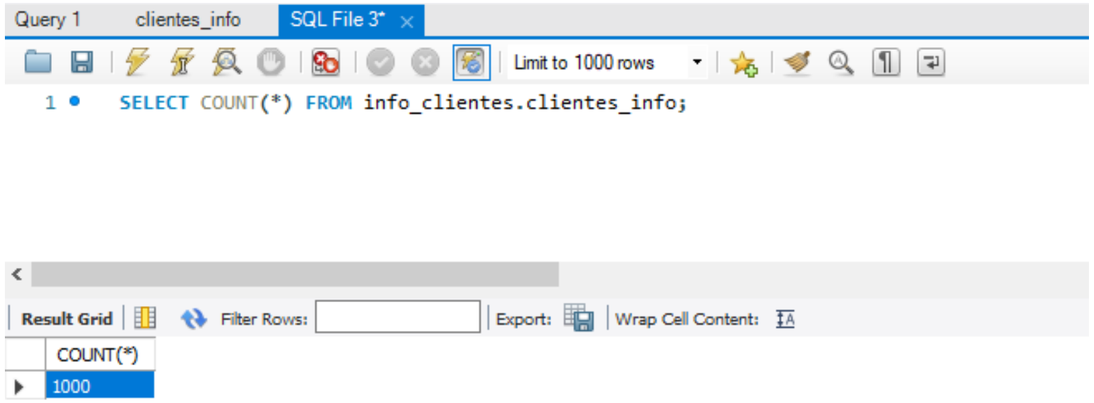
5.2 Proyecto Spring Batch

Después de haber enviado desde Postman la url adecuada con el médoto POST. Comprobamos en la base de datos para saber si los registros fueron insertados de manera correcta.









6. Referencias

- https://git-scm.com/docs/git-branch
- https://www.atlassian.com/git/tutorials/using-branches
- https://git-scm.com/docs/git-merge
- https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts#:~:text=Git%20can%20handle%20most%20merges,working%20in%20a%20team%20environment.
- https://www.atlassian.com/git/tutorials/making-a-pull-request
- https://www.gitkraken.com/learn/git/tutorials/what-is-a-pull-request-in-git#:~:text=commonly%20get%20wrong.-,What%20is%20a%20pull%20request%20in%20Git%3F,remote%20hosting%20service%2C%20like%20GitHub.
- https://docs.github.com/es/get-started/quickstart/fork-a-repo
- https://aprendegit.com/fork-de-repositorios-para-que-sirve/
- https://docs.github.com/en/get-started/using-git/about-git-rebase
- https://www.atlassian.com/git/tutorials/undoing-changes/git-clean
- https://git-scm.com/docs/git-clean
- https://git-scm.com/docs/git-cherry-pick
- https://www.atlassian.com/git/tutorials/cherry-pick
- https://aws.amazon.com/es/what-is/restful-api/
- https://developer.mozilla.org/es/docs/Web/HTTP/Methods
- https://git-scm.com/docs/git-rebase
- https://www.atlassian.com/es/git/tutorials/saving-changes/git-stash#:~:text=El%20comando%20git%20stash%20almacena,aplicar%20los%20cambios%20m%C3%A1s%20tarde.
- https://git-scm.com/docs/git-stash