Back-End Coding Challenge

Proyecto creado en Spring Tool Suite 4.

Dependencias seleccionadas:

- Spring Web
- Spring Batch
- Strping Data JPA
- H2 Database

Archivo .CVS con los códigos postales, obtenido de:

- https://www.correosdemexico.gob.mx/SSLServicios/ConsultaCP/Descarga.aspx

En este proyecto se usó la información del Estado de México, se extrajo la información y se creó un sub archivo llamado Mexico.csv.

```
d_codigo,d_asenta,d_tipo_asenta,D_mnpio,d_estado,d_ciudad,d_CP,c_estado,c_oficina,c_CP,c_tipo_asenta,c_mnpio,id_asenta_cpcons,d_zona,c_cve_ciudad
50000,Toluca de Lerdo Centro,Colonia,Toluca,México,Toluca de Lerdo,50091,15,50091,,09,106,0160,Urbano,20
50010,Celanese,Colonia,Toluca,México,Toluca de Lerdo,50091,15,50091,,09,106,0163,Urbano,20
50010,Club Jardín,Colonia,Toluca,México,Toluca de Lerdo,50091,15,50091,,09,106,0164,Urbano,20
50010,Guadalupe,Colonia,Toluca,México,Toluca de Lerdo,50091,15,50091,,09,106,0165,Urbano,20
```

Para cargar los datos del archivo Mexico.csv necesitaremos crear un objeto con los mismos campos.

```java
@Entity
@Table(name = "POJO_ZIPCODE")
public class Pojo_ZipCode implements Serializable {

    private static final long serialVersionUID = 2188053831729437529L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer d_codigo;
    private String d_asenta;
    private String d_tipo_asenta;
    private String D_mnpio;
    private String d_estado;
    private String d_ciudad;
    private Integer d_CP;
    private Integer c_estado;
    private Integer c_oficina;
    private Integer c_CP;
    private Integer c_tipo_asenta;
    private Integer c_mnpio;
    private Integer id_asenta_cpcons;
    private String d_zona;
    private Integer c_cve_ciudad;
```

Este objeto nos servirá para insertar los registros obtenidos del archivo Mexico.csv en nuestra base de datos (H2), de una forma muy eficiente.

Batch

El proyecto se planeó para que, al iniciar, se inserten los registros del archivo Mexico.csv, para eso haremos uso de Spring Batch, un framework destinado al proceso de grandes lotes de información en “modo batch”. La ejecución de procesos batch está enfocada en resolver el procesamiento sin intervención del usuario y de forma periódica. Entonces par nosotros será transparente la creación de la base de datos.

Crearemos una clase llamada BatchConfig.java

```
@Configuration
@EnableBatchProcessing
public class BatchConfig {
```

Con los siguientes atributos:

```
@Autowired
private JobBuilderFactory jobBuilderFactory;

@Autowired
private StepBuilderFactory stepBuilderFactory;

@Autowired
public DataSource dataSource;
```

Para dar inicio con el proceso crearemos nuestro Job y el step:

```
@Bean
public Job readCSVFileJob() {
    return jobBuilderFactory
            .get("readCSVFileJob")
            .incrementer(new RunIdIncrementer())
            .start(step())
            .build();
}

@Bean
public Step step() {
    return stepBuilderFactory
            .get("step")
            .<Pojo_ZipCode, Pojo_ZipCode>chunk(5)
            .reader(reader())
            .writer(writer())
            .build();
}
```

El método reader() se encargará de leer la información del archivo Mexico.csv

```
@Bean
public FlatFileItemReader<Pojo_ZipCode> reader() {
    FlatFileItemReader<Pojo_ZipCode> itemReader = new FlatFileItemReader<Pojo_ZipCode>();
    itemReader.setLineMapper(lineMapper());
    itemReader.setLinesToSkip(1);
    itemReader.setResource(new ClassPathResource("Mexico.csv"));
    return itemReader;
}
```

Para leer el archivo necesitaremos un FlatFileItemReader, primero asignamos el LineMapper, el cual nos servirá para mappear los registros, agregamos los campos requeridos y asignamos la clase que creamos anteriormente Pojo_ZipCode.class.

```
@Bean
public LineMapper<Pojo_ZipCode> lineMapper() {
    DefaultLineMapper<Pojo_ZipCode> lineMapper = new DefaultLineMapper<Pojo_ZipCode>();
    DelimitedLineTokenizer lineTokenizer = new DelimitedLineTokenizer();
    lineTokenizer.setNames(new String[]{ "d_codigo", "d_asenta", "d_tipo_asenta", "D_mnpio", "d_estado", "d_ciudad", "d_CP",
            "c_estado", "c_oficina", "c_CP", "c_tipo_asenta", "c_mnpio", "id_asenta_cpcons", "d_zona", "c_cve_ciudad"});
    BeanWrapperFieldSetMapper<Pojo_ZipCode> fieldSetMapper = new BeanWrapperFieldSetMapper<Pojo_ZipCode>();
    fieldSetMapper.setTargetType(Pojo_ZipCode.class);
    lineMapper.setLineTokenizer(lineTokenizer);
    lineMapper.setFieldSetMapper(fieldSetMapper);
    return lineMapper;
}
```

Le asignamos la ruta del archivo de donde se leerá la información, mediante un ClassPathResource.

Y con eso tenemos listo nuestro reader() y ya podremos insertar nuestra información, para eso el método writer().

En este método debemos asignarle el atributo dataSource y el query para insertar los datos.

```
@Bean
public JdbcBatchItemWriter<Pojo_ZipCode> writer() {
    JdbcBatchItemWriter<Pojo_ZipCode> itemWriter = new JdbcBatchItemWriter<Pojo_ZipCode>();
    itemWriter.setDataSource(dataSource);
    itemWriter.setSql("INSERT INTO Pojo_ZipCode(d_asenta, d_tipo_asenta, D_mnpio, d_estado, d_ciudad, d_CP, c_estado, c_oficina, "
            + "c_CP, c_tipo_asenta, c_mnpio, id_asenta_cpcons, d_zona, c_cve_ciudad) "
            + "VALUES(:d_asenta, :d_tipo_asenta, :D_mnpio, :d_estado, :d_ciudad, :d_CP, :c_estado, :c_oficina, "
            + ":c_CP, :c_tipo_asenta, :c_mnpio, :id_asenta_cpcons, :d_zona, :c_cve_ciudad)");
    itemWriter.setItemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider<Pojo_ZipCode>());
    return itemWriter;
}
```

Una vez implementados los métodos, agregamos la configuración de nuestra base de datos en el archivo application.properties

```
spring.datasource.url=jdbc:h2:mem:javatpoint;DB_CLOSE_DELAY=-1
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create
#enabling the H2 console
spring.h2.console.enabled=true
```

Spring Data

Para poder consultar los datos que se insertan en nuestra base de datos, haremos uso de Spring Data, facilitándonos el acceso a nuestra información.

Lo primero será crear una Interface con el nombre de ZipCodesRepository.java

```
public interface ZipCodesRepository extends CrudRepository<Pojo_ZipCode, Integer> {

}
```

Al extender nuestra interface con CrudRepository, obtenemos acceso a distintos métodos genéricos, los cuales nos podrían ser útiles.

Ahora creamos una Interface llamada ZipCodesService que nos servirá como servicio y su implementación llamada ZipCodesServiceImp para los métodos que necesitamos, otorgados por CrudRepository

```
public interface ZipCodesService {
    public Pojo_ZipCode findById(Integer zip_code);
}
```

```
@Service
public class ZipCodesServiceImp implements ZipCodesService{

    @Autowired
    private ZipCodesRepository zipCodes;

    @Override
    @Transactional(readOnly = true)
    public Pojo_ZipCode findById(Integer zip_code) {
        return zipCodes.findById(zip_code).orElse(null);
    }
}
```

El método findById nos servirá para encontrar el zip-code que estemos buscando, únicamente debemos enviarle el id en cuestión. Si el método encuentra el id lo regresa, si no lo hace regresa null.

Para poder consumir estos métodos necesitaremos una clase Controller, creando un API REST EndPoint, al cual accederemos mediante un buscador o PostMan.

Creamos nuestra clase llamada Controller_ZipCode, con el método getZipCode y mediante el GetMapping le asignamos la ruta “/zip-codes/{zip_code}”

- {zip_code} : Parametro que estemos buscando.

El método getZipCode lo único que hace es consumir el método findById que creamos en la clase ZipCodesService, mandándole como parametro el zip_code que se recibe de la petición.

```
@Controller
public class Controller_ZipCode {

    @Autowired
    private ZipCodesService zipCodes;

    @GetMapping("/zip-codes/{zip_code}")
    public ResponseEntity<Pojo_ZipCode> getZipCode(@PathVariable Integer zip_code) {
        Pojo_ZipCode zipCode = zipCodes.findById(zip_code);
        if (zipCode != null)
            return new ResponseEntity<Pojo_ZipCode>(zipCode, HttpStatus.OK);
        else
            return new ResponseEntity<Pojo_ZipCode>(zipCode, HttpStatus.NOT_FOUND);
    }
}
```

Pruebas

Una vez llegados a este punto, solo quedaría levantar nuestro servicio.

- Clic derecho al proyecto

- run as -> spring boot app

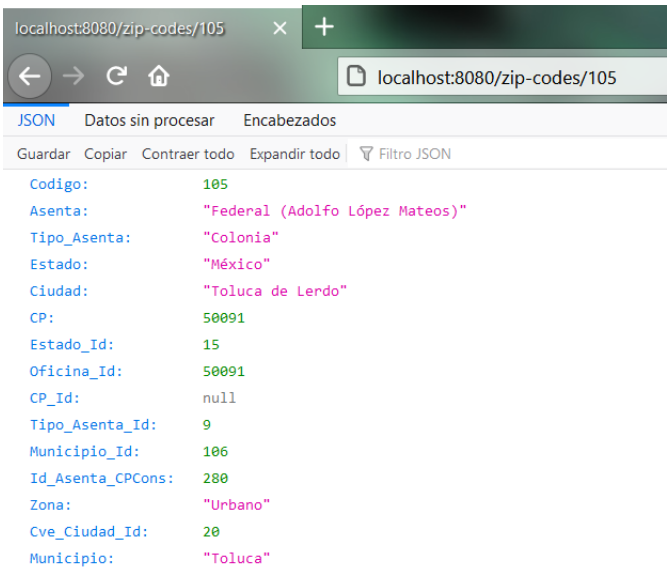
- El servicio cargará automáticamente la información, mediante el proceso Batch que creamos.

Navegador:

Si accedemos desde nuestro navegador a la ruta:

http://localhost:8080/zip-codes/105

Podremos ver que nos carga los datos de ese Código.

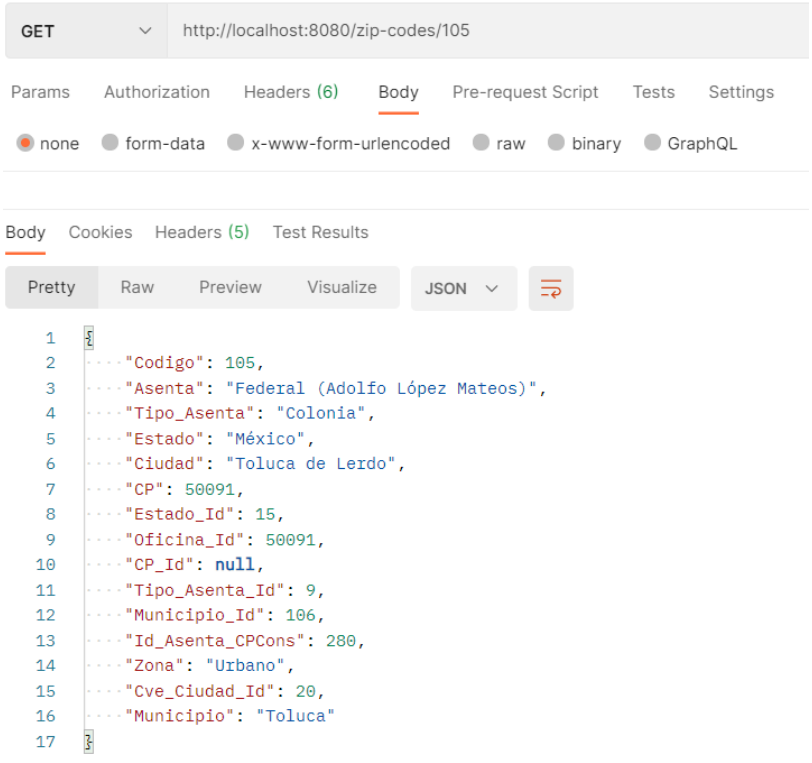


Postman:

De igual forma, si accedemos desde PostMan con un Get a la ruta:

http://localhost:8080/zip-codes/105

Podremos ver que nos carga la misma información de ese Código.



Google Cloud:

El proyecto se subió a GoogleCloud y generó una url a la cual podremos acceder en cualquier momento, sin necesidad de levantar nuestro servicio localmente

Si accedemos desde nuestro navegador a la ruta:

https://zipcode-jleon.wn.r.appspot.com/zip-codes/105

Podremos ver que nos carga la misma información de ese Código.