SPRING BATCH

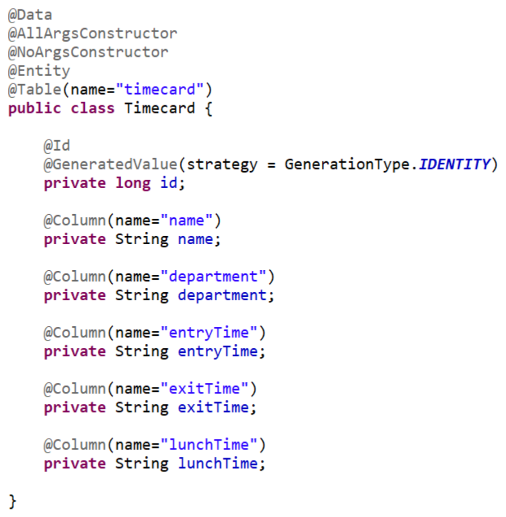
HP

Introduction

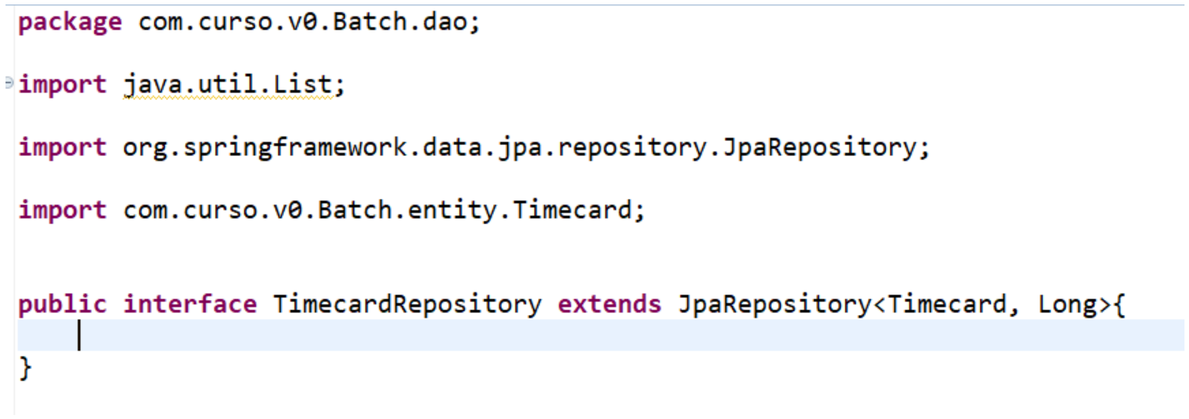
Spring batch is a framework built to handle the requiriements of bash processing in java applications, which are a series of tasks that process large volumes of data, with spring batch, these processes can be scheduled and automated, and thanks to its chunk-oriented processing it doesn’t affect the performance or scallability of the application.

Implementation

To show the capabilities of spring batch the following entites and repository were created using a combination of lombok and spring data:

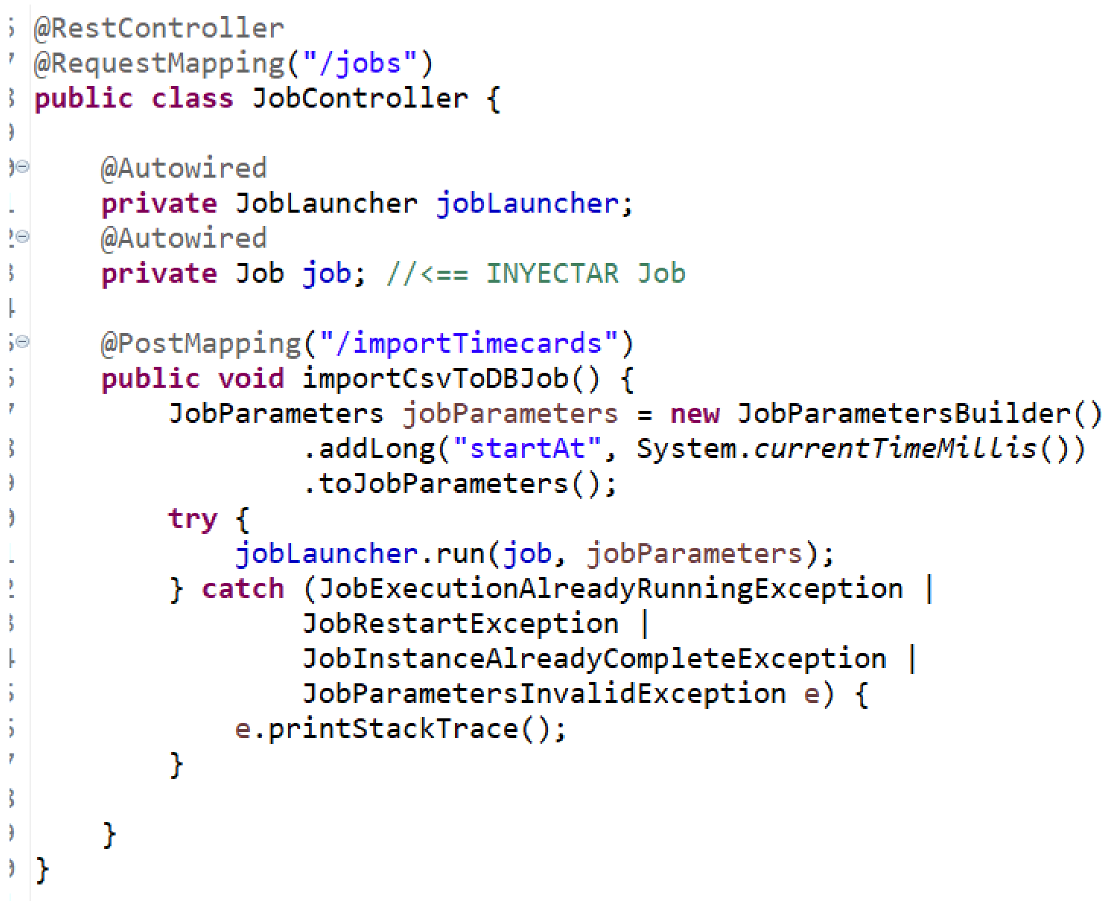


```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name="timecard")
public class Timecard {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name="name")
    private String name;

    @Column(name="department")
    private String department;

    @Column(name="entryTime")
    private String entryTime;

    @Column(name="exitTime")
    private String exitTime;

    @Column(name="lunchTime")
    private String lunchTime;

}
```

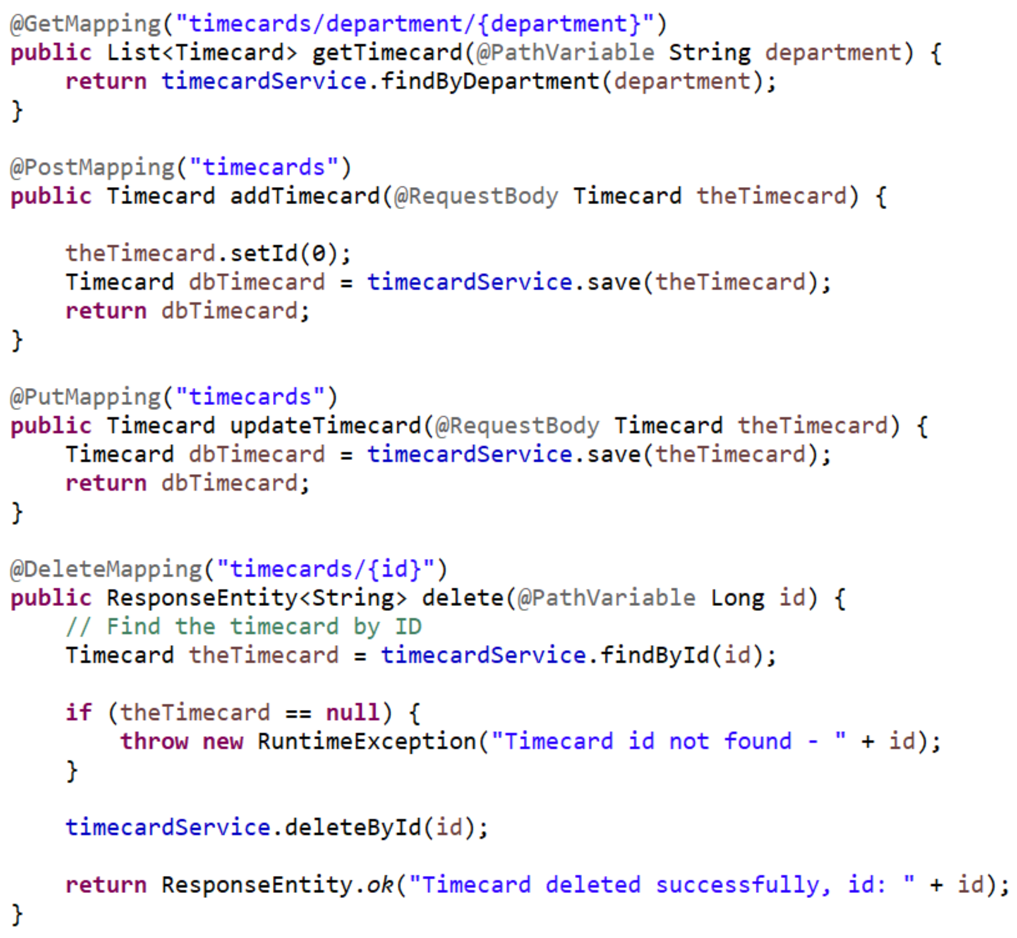


```java
package com.curso.v0.Batch.dao;

import java.util.List;

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

import com.curso.v0.Batch.entity.Timecard;


public interface TimecardRepository extends JpaRepository<Timecard, Long>{

}
```

Next an endpoint is exposed to actívate the job, which is the process that will handle a file of data and tranlsate it and put i ton the database:



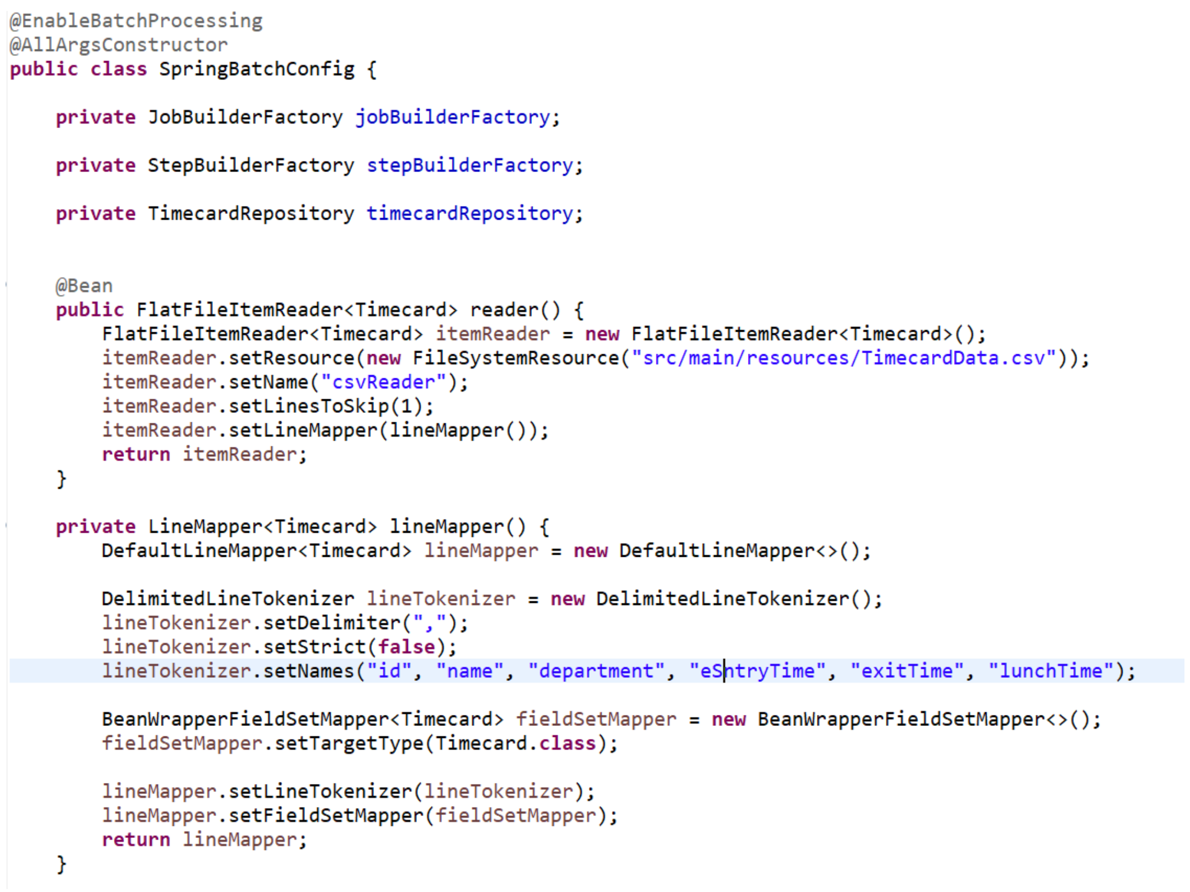
```java
@RestController
@RequestMapping("/jobs")
public class JobController {

    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private Job job; //<== INYECTAR Job

    @PostMapping("/importTimecards")
    public void importCsvToDBJob() {
        JobParameters jobParameters = new JobParametersBuilder()
                .addLong("startAt", System.currentTimeMillis())
                .toJobParameters();
        try {
            jobLauncher.run(job, jobParameters);
        } catch (JobExecutionAlreadyRunningException |
                JobRestartException |
                JobInstanceAlreadyCompleteException |
                JobParametersInvalidException e) {
            e.printStackTrace();
        }

    }
}
```



```
@GetMapping("timecards/department/{department}")
public List<Timecard> getTimecard(@PathVariable String department) {
    return timecardService.findByDepartment(department);
}

@PostMapping("timecards")
public Timecard addTimecard(@RequestBody Timecard theTimecard) {

    theTimecard.setId(0);
    Timecard dbTimecard = timecardService.save(theTimecard);
    return dbTimecard;
}

@PutMapping("timecards")
public Timecard updateTimecard(@RequestBody Timecard theTimecard) {
    Timecard dbTimecard = timecardService.save(theTimecard);
    return dbTimecard;
}

@DeleteMapping("timecards/{id}")
public ResponseEntity<String> delete(@PathVariable Long id) {
    // Find the timecard by ID
    Timecard theTimecard = timecardService.findById(id);

    if (theTimecard == null) {
        throw new RuntimeException("Timecard id not found - " + id);
    }

    timecardService.deleteById(id);

    return ResponseEntity.ok("Timecard deleted successfully, id: " + id);
}
```

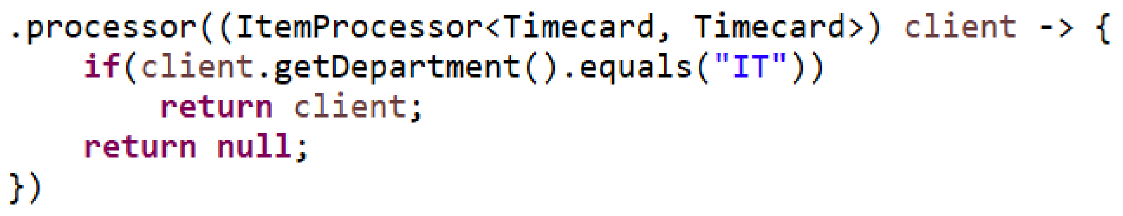
To define the operations the batch job will do we need to define its configuration with a Reader to get the data to use on the operation, a processor to apply the bussines logic to the read data and a writer to save the processed data.

First the writer takes the file TimecardData and maps it to the timecard class:



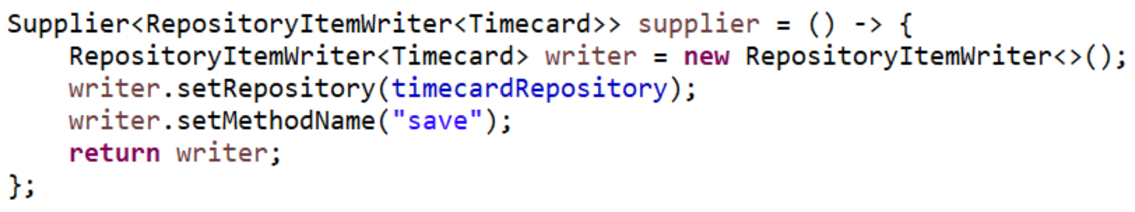
```java
@EnableBatchProcessing
@AllArgsConstructor
public class SpringBatchConfig {

    private JobBuilderFactory jobBuilderFactory;

    private StepBuilderFactory stepBuilderFactory;

    private TimecardRepository timecardRepository;


    @Bean
    public FlatFileItemReader<Timecard> reader() {
        FlatFileItemReader<Timecard> itemReader = new FlatFileItemReader<Timecard>();
        itemReader.setResource(new FileSystemResource("src/main/resources/TimecardData.csv"));
        itemReader.setName("csvReader");
        itemReader.setLinesToSkip(1);
        itemReader.setLineMapper(lineMapper());
        return itemReader;
    }

    private LineMapper<Timecard> lineMapper() {
        DefaultLineMapper<Timecard> lineMapper = new DefaultLineMapper<>();

        DelimitedLineTokenizer lineTokenizer = new DelimitedLineTokenizer();
        lineTokenizer.setDelimiter(",");
        lineTokenizer.setStrict(false);
        lineTokenizer.setNames("id", "name", "department", "eSntryTime", "exitTime", "lunchTime");

        BeanWrapperFieldSetMapper<Timecard> fieldSetMapper = new BeanWrapperFieldSetMapper<>();
        fieldSetMapper.setTargetType(Timecard.class);

        lineMapper.setLineTokenizer(lineTokenizer);
        lineMapper.setFieldSetMapper(fieldSetMapper);
        return lineMapper;
    }
```

Then a processor iterates on the data and filters those who aren’t form the IT department in this case the function was declares as a lambda on the step:



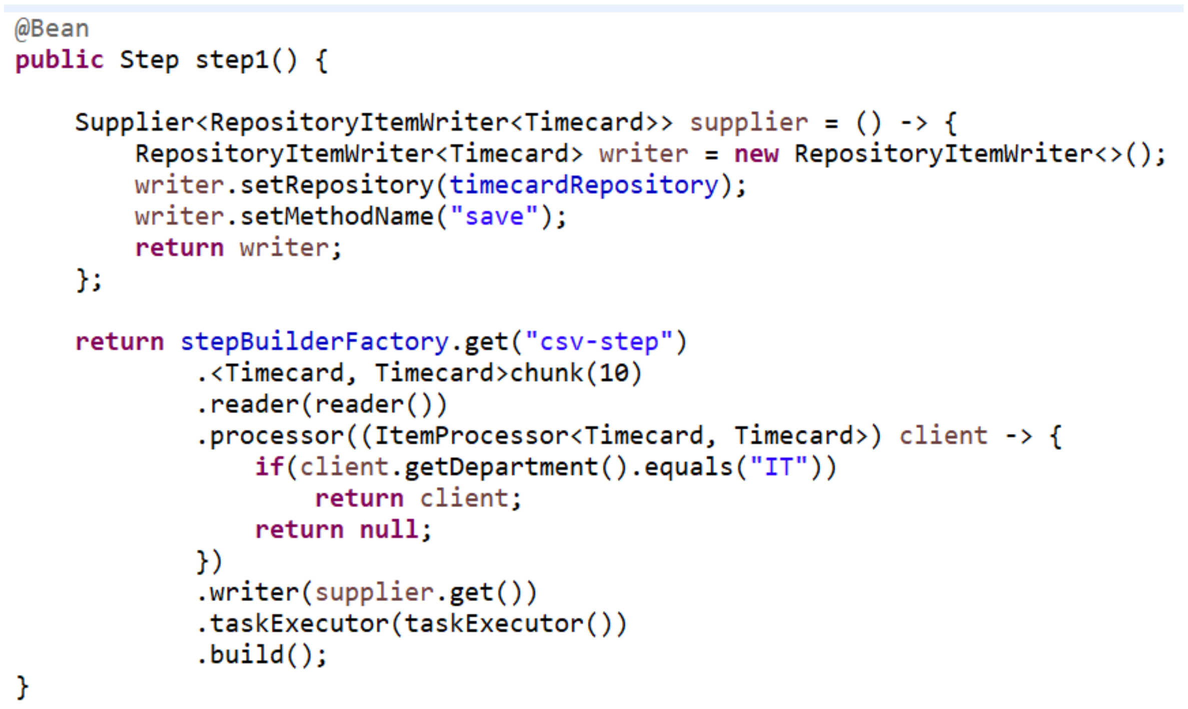
```java
.processor((ItemProcessor<Timecard, Timecard>) client -> {
    if(client.getDepartment().equals("IT"))
        return client;
    return null;
})
```

Finally a supplier calls the method to save the processed data in a repository:



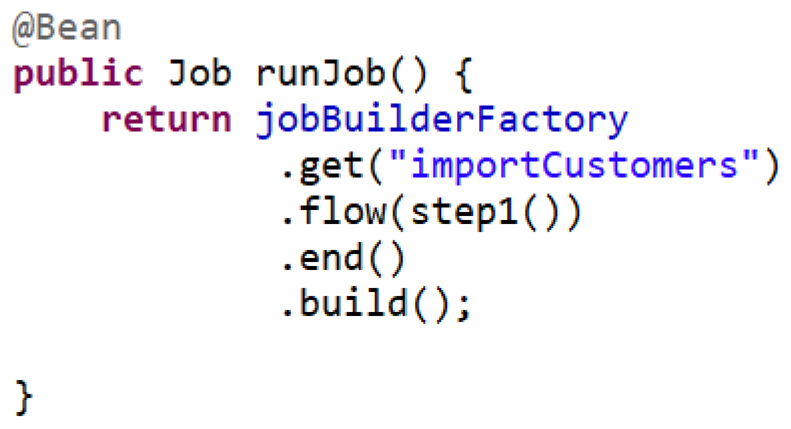
```java
Supplier<RepositoryItemWriter<Timecard>> supplier = () -> {
    RepositoryItemWriter<Timecard> writer = new RepositoryItemWriter<>();
    writer.setRepository(timecardRepository);
    writer.setMethodName("save");
    return writer;
};
```

These are the components of a single step in the job, other steps can be applied which also need a reader, processor and writer.



```java
@Bean
public Step step1() {

    Supplier<RepositoryItemWriter<Timecard>> supplier = () -> {
        RepositoryItemWriter<Timecard> writer = new RepositoryItemWriter<>();
        writer.setRepository(timecardRepository);
        writer.setMethodName("save");
        return writer;
    };

    return stepBuilderFactory.get("csv-step")
            .<Timecard, Timecard>chunk(10)
            .reader(reader())
            .processor((ItemProcessor<Timecard, Timecard>) client -> {
                if(client.getDepartment().equals("IT"))
                    return client;
                return null;
            })
            .writer(supplier.get())
            .taskExecutor(taskExecutor())
            .build();
}
```

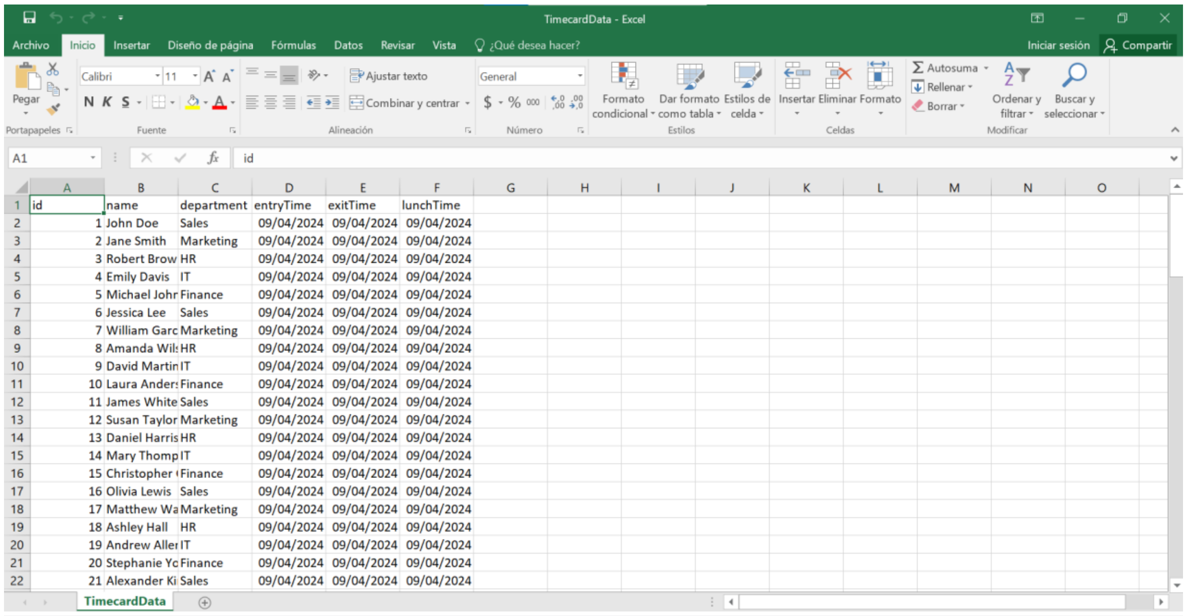
The job is defined with the previously built step:



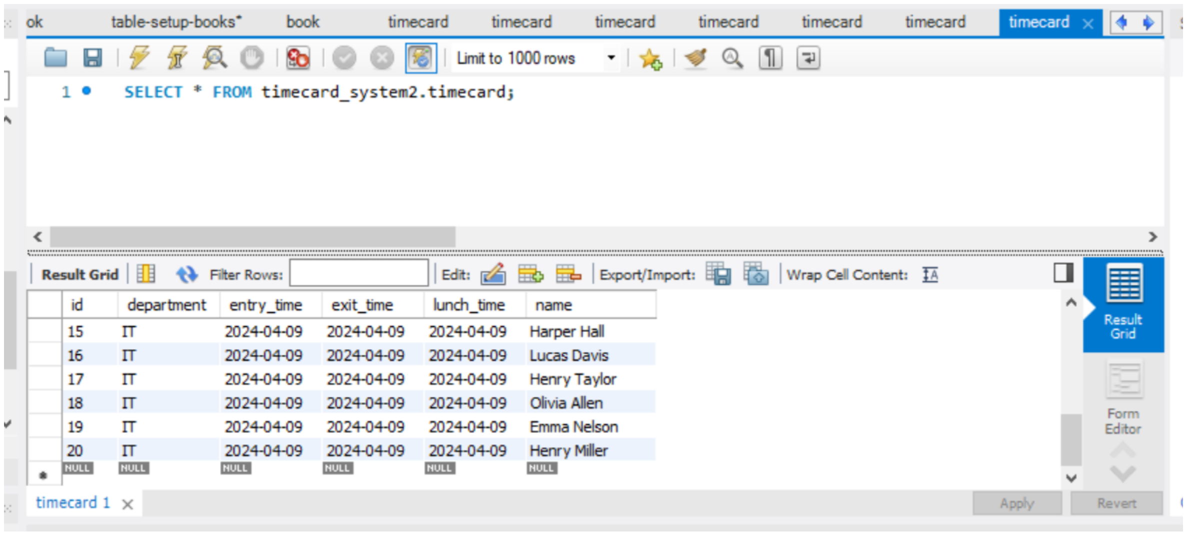
```java
@Bean
public Job runJob() {
    return jobBuilderFactory
            .get("importCustomers")
            .flow(step1())
            .end()
            .build();

}
```

Results

When the application is running we can use the specified URL to actívate the batch job, saving the contents of an Excel file on a database, but only the regiters that match the filters:



	A	B	C	D	E	F
1	id	name	department	entryTime	exitTime	lunchTime
2	1	John Doe	Sales	09/04/2024	09/04/2024	09/04/2024
3	2	Jane Smith	Marketing	09/04/2024	09/04/2024	09/04/2024
4	3	Robert Brow	HR	09/04/2024	09/04/2024	09/04/2024
5	4	Emily Davis	IT	09/04/2024	09/04/2024	09/04/2024
6	5	Michael Johr	Finance	09/04/2024	09/04/2024	09/04/2024
7	6	Jessica Lee	Sales	09/04/2024	09/04/2024	09/04/2024
8	7	William Garc	Marketing	09/04/2024	09/04/2024	09/04/2024
9	8	Amanda Wil:	HR	09/04/2024	09/04/2024	09/04/2024
10	9	David Martin	IT	09/04/2024	09/04/2024	09/04/2024
11	10	Laura Ander:	Finance	09/04/2024	09/04/2024	09/04/2024
12	11	James White	Sales	09/04/2024	09/04/2024	09/04/2024
13	12	Susan Taylor	Marketing	09/04/2024	09/04/2024	09/04/2024
14	13	Daniel Harris	HR	09/04/2024	09/04/2024	09/04/2024
15	14	Mary Thomp	IT	09/04/2024	09/04/2024	09/04/2024
16	15	Christopher (	Finance	09/04/2024	09/04/2024	09/04/2024
17	16	Olivia Lewis	Sales	09/04/2024	09/04/2024	09/04/2024
18	17	Matthew Wa	Marketing	09/04/2024	09/04/2024	09/04/2024
19	18	Ashley Hall	HR	09/04/2024	09/04/2024	09/04/2024
20	19	Andrew Aller	IT	09/04/2024	09/04/2024	09/04/2024
21	20	Stephanie Yo	Finance	09/04/2024	09/04/2024	09/04/2024
22	21	Alexander Ki	Sales	09/04/2024	09/04/2024	09/04/2024



```
SELECT * FROM timecard_system2.timecard;
```

id	department	entry_time	exit_time	lunch_time	name
15	IT	2024-04-09	2024-04-09	2024-04-09	Harper Hall
16	IT	2024-04-09	2024-04-09	2024-04-09	Lucas Davis
17	IT	2024-04-09	2024-04-09	2024-04-09	Henry Taylor
18	IT	2024-04-09	2024-04-09	2024-04-09	Olivia Allen
19	IT	2024-04-09	2024-04-09	2024-04-09	Emma Nelson
20	IT	2024-04-09	2024-04-09	2024-04-09	Henry Miller
NULL	NULL	NULL	NULL	NULL	NULL