



[Home](#) > [Frameworks](#)

### Learn Spring framework:

- [Learn Spring on YouTube](#)
- [Understand the core of Spring](#)
- [Understand Spring MVC](#)
- [Understand Spring AOP](#)
- [Understand Spring Data JPA](#)
- [Spring Dependency Injection \(XML\)](#)
- [Spring Dependency Injection \(Annotations\)](#)
- [Spring Dependency Injection \(Java config\)](#)
- [Spring MVC beginner tutorial](#)
- [Spring MVC Exception Handling](#)
- [Spring MVC and log4j](#)
- [Spring MVC Send email](#)
- [Spring MVC File Upload](#)
- [Spring MVC Form Handling](#)

- [Spring MVC Form Validation](#)
- [Spring MVC File Download](#)
- [Spring MVC JdbcTemplate](#)
- [Spring MVC CSV view](#)
- [Spring MVC Excel View](#)
- [Spring MVC PDF View](#)
- [Spring MVC XstlView](#)
- [Spring MVC + Spring Data JPA + Hibernate - CRUD](#)
- [Spring MVC Security \(XML\)](#)
- [Spring MVC Security \(Java config\)](#)
- [Spring & Hibernate Integration \(XML\)](#)
- [Spring & Hibernate Integration \(Java config\)](#)
- [Spring & Struts Integration \(XML\)](#)
- [Spring & Struts Integration \(Java config\)](#)
- [14 Tips for Writing Spring MVC Controller](#)

## Spring Boot Export Data to Excel Example

Exporting data to Excel documents is a common feature of almost any applications. Through this article, I'm very glad to share with you guys my experience in implementing Excel export function in a Spring Boot application with the help of Apache POI Excel library.

Suppose that we have an existing Spring Boot project using Spring Data JPA and Hibernate to access data, Thymeleaf to render the view and MySQL as the database.

The code examples below demonstrate how to retrieve information about users from the database, and generate an Excel file which the users can download onto their computers.

## 1. Code of Entity Classes and Repositories Interfaces

We have the `User` entity class that maps to the `users` table in the database, as shown below:

```
1  package net.codejava;
2
3  import java.util.*;
4
5  import javax.persistence.*;
6
7  @Entity
8  @Table(name = "users")
9  public class User {
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private Integer id;
13
14     private String email;
15
16     private String password;
17
18     @Column(name = "full_name")
19     private String fullName;
20
21     private boolean enabled;
22
23     @ManyToMany(cascade = CascadeType.PERSIST, fetch = FetchType.EAGER)
24     @JoinTable(
25         name = "users_roles",
26         joinColumns = @JoinColumn(name = "user_id"),
27         inverseJoinColumns = @JoinColumn(name = "role_id")
28     )
29     private Set<Role> roles = new HashSet<>();
30
31     // constructors, getter and setters are not shown for brevity
32 }
```

And the `Role` entity class that maps to the `roles` table in the database:

```
1 package net.codejava;
2
3 import javax.persistence.*;
4
5 @Entity
6 @Table(name = "roles")
7 public class Role {
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private Integer id;
11
12    private String name;
13
14    private String description;
15
16    // constructors, getter and setters are not shown for brevity
17 }
```

The fields will be included in the generated Excel document are: User ID, E-mail, Full Name, Roles and Enabled.

And code of the respective repository interfaces looks like this:

```
1 package net.codejava;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface UserRepository extends JpaRepository<User, Integer> {
6
7 }
8
9
10 public interface RoleRepository extends CrudRepository<Role, Integer> {
11
12 }
```

These are simple, typical repositories as required by Spring Data JPA.

## 2. Declare Dependency for Excel Library

To generate Excel file, we need to use an external library and [Apache POI](#) is one of the most popular ones. So we need to declare the following dependency to use Apache POI for Excel in the Maven's project file:

```
1 <dependency>
2   <groupId>org.apache.poi</groupId>
3   <artifactId>poi-ooxml</artifactId>
4   <version>4.1.0</version>
5 </dependency>
```

## 3. Code for the Service Class

In the service layer, we may have the `UserServices` class as follows:

```
1 package net.codejava;
2
3 import java.util.List;
4
5 import javax.transaction.Transactional;
6
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.data.domain.Sort;
9 import org.springframework.stereotype.Service;
10
11 @Service
12 @Transactional
13 public class UserServices {
14
15     @Autowired
16     private UserRepository repo;
17
18     public List<User> listAll() {
19         return repo.findAll(Sort.by("email").ascending());
20     }
21
22 }
```

As you can see, the `listAll()` method delegates the call to the `findAll()` method of the `UserRepository` interface, which is implemented by Spring Data JPA (extended from `JpaRepository`). The `listAll()` method will be invoked to get data about users from the database.

## 4. Code Excel Exporter Class

Next, code a separate class that is responsible to generate an Excel document based on the input is a `List` collection of `User` objects, as shown below:

```
1 package net.codejava;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 import javax.servlet.ServletOutputStream;
7 import javax.servlet.http.HttpServletResponse;
8
9 import org.apache.poi.ss.usermodel.Cell;
10 import org.apache.poi.ss.usermodel.CellStyle;
11 import org.apache.poi.ss.usermodel.Row;
12 import org.apache.poi.xssf.usermodel.XSSFFont;
13 import org.apache.poi.xssf.usermodel.XSSFSheet;
14 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
15
16 public class UserExcelExporter {
17     private XSSFWorkbook workbook;
18     private XSSFSheet sheet;
19     private List<User> listUsers;
20
21     public UserExcelExporter(List<User> listUsers) {
22         this.listUsers = listUsers;
23         workbook = new XSSFWorkbook();
24     }
25
26
27     private void writeHeaderLine() {
28         sheet = workbook.createSheet("Users");
```

```
28     sheet = workbook.createSheet( "Users" );
29
30     Row row = sheet.createRow(0);
31
32     CellStyle style = workbook.createCellStyle();
33     XSSFFont font = workbook.createFont();
34     font.setBold(true);
35     font.setFontHeight(16);
36     style.setFont(font);
37
38     createCell(row, 0, "User ID", style);
39     createCell(row, 1, "E-mail", style);
40     createCell(row, 2, "Full Name", style);
41     createCell(row, 3, "Roles", style);
42     createCell(row, 4, "Enabled", style);
43
44 }
45
46 private void createCell(Row row, int columnCount, Object value, Cell
47     sheet.autoSizeColumn(columnCount);
48     Cell cell = row.createCell(columnCount);
49     if (value instanceof Integer) {
50         cell.setCellValue((Integer) value);
51     } else if (value instanceof Boolean) {
52         cell.setCellValue((Boolean) value);
53     } else {
54         cell.setCellValue((String) value);
55     }
56     cell.setCellStyle(style);
57 }
58
59 private void writeDataLines() {
60     int rowCount = 1;
61
62     CellStyle style = workbook.createCellStyle();
63     XSSFFont font = workbook.createFont();
64     font.setFontHeight(14);
65     style.setFont(font);
66
67     for (User user : listUsers) {
68         Row row = sheet.createRow(rowCount++);
69         int columnCount = 0;
70
71         createCell(row, columnCount++, user.getId(), style);
72         createCell(row, columnCount++, user.getEmail(), style);
73         createCell(row, columnCount++, user.getFullName(), style);
74         createCell(row, columnCount++, user.getRoles().toString(), s
75         createCell(row, columnCount++, user.isEnabled(), style);
76
77     }
78 }
79
80 public void export(HttpServletResponse response) throws IOException
81     writeHeaderLine();
82     writeDataLines();
83
84     ServletOutputStream outputStream = response.getOutputStream();
85     workbook.write(outputStream);
86     workbook.close();
87
88     outputStream.close();
89
90 }
91 }
```

This class will create an Excel document with one sheet containing a header row and rows for the data. The header row consists of these columns: User ID, E-mail, Full Name, Roles and Enabled.

Pay attention to the `export()` method that takes an `HttpServletResponse` as the argument, because it will write the content of the Excel file into the output stream of the response, so the clients (web browsers) will be able to download the exported Excel file.

## 5. Code Handler method in the Controller Class

Next, implement a handler method in a Spring MVC controller class – `UserController` – as follows:

```
1  package net.codejava;
2
3  import java.io.IOException;
4  import java.text.DateFormat;
5  import java.text.SimpleDateFormat;
6  import java.util.Date;
7  import java.util.List;
8
9  import javax.servlet.http.HttpServletResponse;
10
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.stereotype.Controller;
13 import org.springframework.web.bind.annotation.GetMapping;
14
15 @Controller
16 public class UserController {
17
18     @Autowired
19     private UserServices service;
20
21
22     @GetMapping("/users/export/excel")
23     public void exportToExcel(HttpServletResponse response) throws IOException {
24         response.setContentType("application/octet-stream");
25         DateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-dd_HH:mm:ss");
26         String currentDateTime = dateFormatter.format(new Date());
27
28         String headerKey = "Content-Disposition";
29         String headerValue = "attachment; filename=users_" + currentDateTime;
30         response.setHeader(headerKey, headerValue);
31
32         List<User> listUsers = service.listAll();
33
34         UserExcelExporter excelExporter = new UserExcelExporter(listUsers);
35
36         excelExporter.export(response);
37     }
38
39 }
```

As you can see the `exportToExcel()` method will serve HTTP GET request with the URI `/users/export/excel`. It will use the `UserServices` class to get data of users from

the database, and use the `UserExcelExporter` class to write an Excel document to the response.

Also notice name of the generated Excel file is appended with the current date time, making it's easier for the users to track multiple versions of files downloaded.

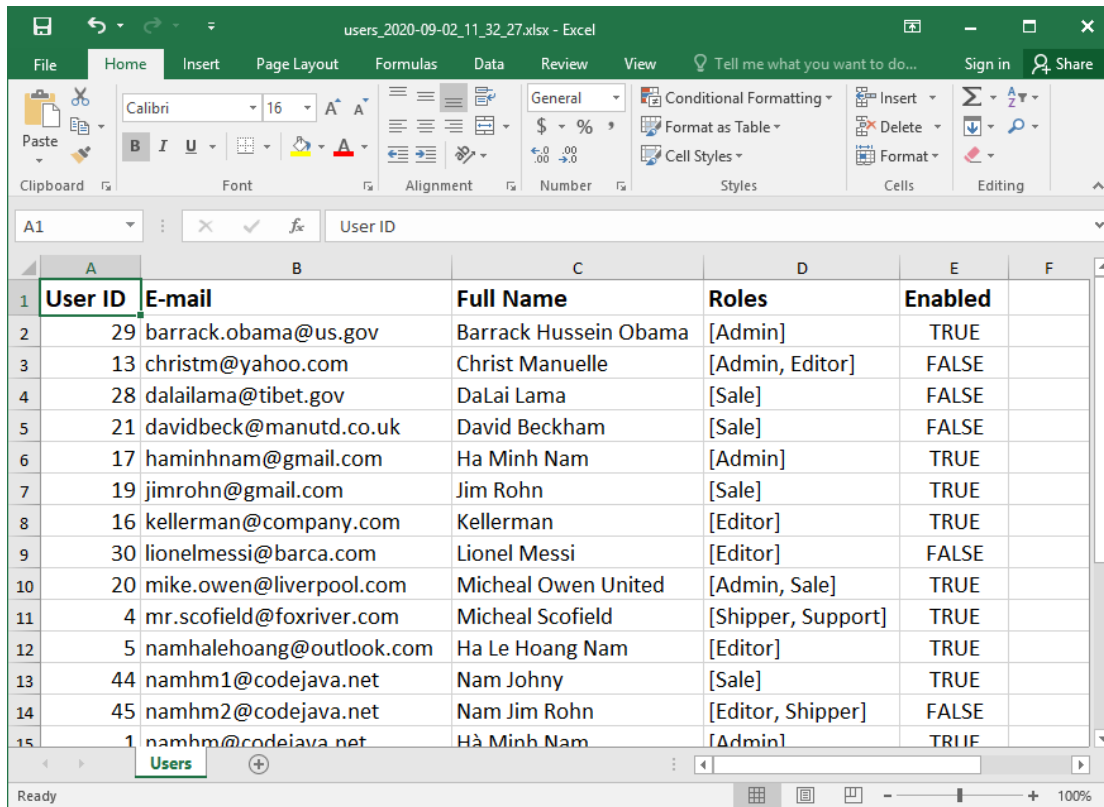
## 6. Add Export Excel Link in the View Page

We use HTML and Thymeleaf to create a hyperlink that allows the user to click to export data to Excel as follows:

```
1 <a th:href="@{/users/export/excel}">Export to Excel</a>
```

## 7. Test Export and Download Excel file

Click the hyperlink Export to Excel, the Spring Boot application will generate an Excel file and the browser will automatically download that file. The file name is something like this: `users_2020-09-02_11-30-06.xlsx`. Open this file using Microsoft Excel application, you would see the following screen:



The screenshot shows the Microsoft Excel application with a file named 'users\_2020-09-02\_11\_32\_27.xlsx'. The table contains the following data:

User ID	E-mail	Full Name	Roles	Enabled
29	barrack.obama@us.gov	Barrack Hussein Obama	[Admin]	TRUE
13	christm@yahoo.com	Christ Manuelle	[Admin, Editor]	FALSE
28	dalailama@tibet.gov	DaLai Lama	[Sale]	FALSE
21	davidbeck@manutd.co.uk	David Beckham	[Sale]	FALSE
17	haminhnam@gmail.com	Ha Minh Nam	[Admin]	TRUE
19	jimrohn@gmail.com	Jim Rohn	[Sale]	TRUE
16	kellerman@company.com	Kellerman	[Editor]	TRUE
30	lionelmessi@barca.com	Lionel Messi	[Editor]	FALSE
20	mike.owen@liverpool.com	Micheal Owen United	[Admin, Sale]	TRUE
4	mr.scofield@foxriver.com	Micheal Scofield	[Shipper, Support]	TRUE
5	namhalehoang@outlook.com	Ha Le Hoang Nam	[Editor]	TRUE
44	namhm1@codejava.net	Nam Johny	[Sale]	TRUE
45	namhm2@codejava.net	Nam Jim Rohn	[Editor, Shipper]	FALSE
1	namhm@codejava.net	Hà Minh Nam	[Admin]	TRUE

## Conclusion

So far you have learned how to code Excel export function for a Spring Boot web application. You see, Spring Data JPA makes it easy to get data from the database, and



Apache POI makes it easy to generate documents compatible with Microsoft Excel format.

For video version of this tutorial, watch the video below:

## Spring Boot Export Data to Excel with Apache POI Example



### Related Tutorials:

- [How to Write Excel Files in Java using Apache POI](#)
- [Java code example to export from database to Excel file](#)
- [How to Read Excel Files in Java using Apache POI](#)
- [Java code example to import data from Excel to database](#)

### Other Spring Boot Tutorials:

- [Spring Boot Export Data to CSV Example](#)
- [Spring Boot Export Data to PDF Example](#)
- [Spring Boot Hello World Example](#)
- [Spring Boot automatic restart using Spring Boot DevTools](#)
- [Spring Boot Form Handling Tutorial with Spring Form Tags and JSP](#)
- [How to create a Spring Boot Web Application \(Spring MVC with JSP/ThymeLeaf\)](#)
- [Spring Boot - Spring Data JPA - MySQL Example](#)
- [Spring Boot Hello World RESTful Web Services Tutorial](#)
- [How to use JDBC with Spring Boot](#)
- [Spring Boot CRUD Web Application with JDBC - Thymeleaf - Oracle](#)
- [Spring Boot RESTful CRUD API Examples with MySQL database](#)
- [How to package Spring Boot application to JAR and WAR](#)
- [Spring Boot Security Authentication with JPA, Hibernate and MySQL](#)
- [Spring Data JPA Paging and Sorting Examples](#)
- [Spring Boot Error Handling Guide](#)

### About the Author:

[Nam Ha Minh](#) is certified Java programmer (SCJP and SCWCD). He started programming with Java in the time of Java 1.4 and has been falling in love with Java since then. Make friend with him on [Facebook](#) and watch [his Java videos](#) you YouTube.



### Add comment

☐ Notify me of follow-up comments

Tôi không phải là người  
máy

reCAPTCHA  
Bảo mật - Điều khoản

### Comments

## See All Java Tutorials

CodeJava.net shares Java tutorials, code examples and sample projects for programmers at all levels.  
CodeJava.net is created and managed by Nam Ha Minh - a passionate programmer.

[Home](#) [About](#) [Contact](#) [Terms of Use](#) [Privacy Policy](#) [Facebook](#) [Twitter](#) [YouTube](#)

Copyright © 2012 - 2020 CodeJava.net, all rights reserved.