Spring Boot: Upload & Read CSV file into MySQL Database | Multipart File

bezkoder.com/spring-boot-upload-csv-file

bezkoder Last modified: February 16, 2022

A CSV (comma-separated values) file is a plain text file that contains data which format is described in RFC4180. Each row has a number of fields, separated by commas. Each line should contain the same number of fields throughout the file. In this tutorial, I will show you how to upload, read CSV file data and store into MySQL Database using Spring Boot & Apache Commons CSV.

Related Posts:
– Spring Boot Multipart File upload example
– How to upload multiple files in Java Spring Boot
– Spring Boot Download CSV file from Database example

Excel file instead:
Spring Boot: Upload/Import Excel file data into MySQL Database

Deployment:
– Deploy Spring Boot App on AWS – Elastic Beanstalk
– Docker Compose: Spring Boot and MySQL example

Spring Boot Rest APIs for uploading CSV Files

Assume that we have a **.csv** file that contains Tutorial data as following:

```
Id,Title,Description,Published
1,Spring Boot Tut#1,Tut#1 Description,FALSE
2,Spring Data Tut#2,Tut#2 Description,TRUE
3,MySQL Database Tut#3,Tut#3 Description,TRUE
4,Hibernate Tut#4,Tut#4 Description,FALSE
5,Spring Cloud Tut#5,Tut#5 Description,TRUE
6,Microservices Tut#6,Tut#6 Description,FALSE
7,MongoDB Database Tut#7,Tut#7 Description,TRUE
8,Spring Data JPA Tut#8,Tut#8 Description,TRUE
```

We’re gonna create a Spring Boot Application that provides APIs for:

- uploading CSV File to the Spring Server & storing data in MySQL Database
- getting list of items from MySQL table
- downloading CSV file that contains MySQL table data

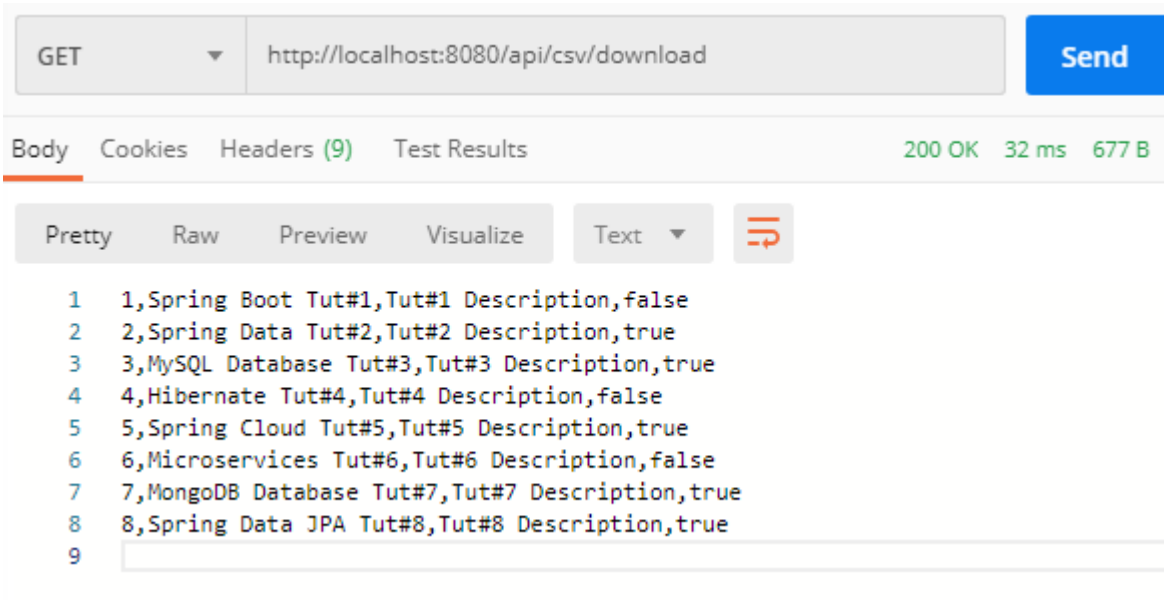
After the CSV file is uploaded successfully, tutorials table in MySQL database will look like this:

If we get list of Tutorials, the Spring Rest Apis will return:

id	description	published	title
1	Tut#1 Description	0	Spring Boot Tut#1
2	Tut#2 Description	1	Spring Data Tut#2
3	Tut#3 Description	1	MySQL Database Tut#3
4	Tut#4 Description	0	Hibernate Tut#4
5	Tut#5 Description	1	Spring Cloud Tut#5
6	Tut#6 Description	0	Microservices Tut#6
7	Tut#7 Description	1	MongoDB Database Tut#7
8	Tut#8 Description	1	Spring Data JPA Tut#8

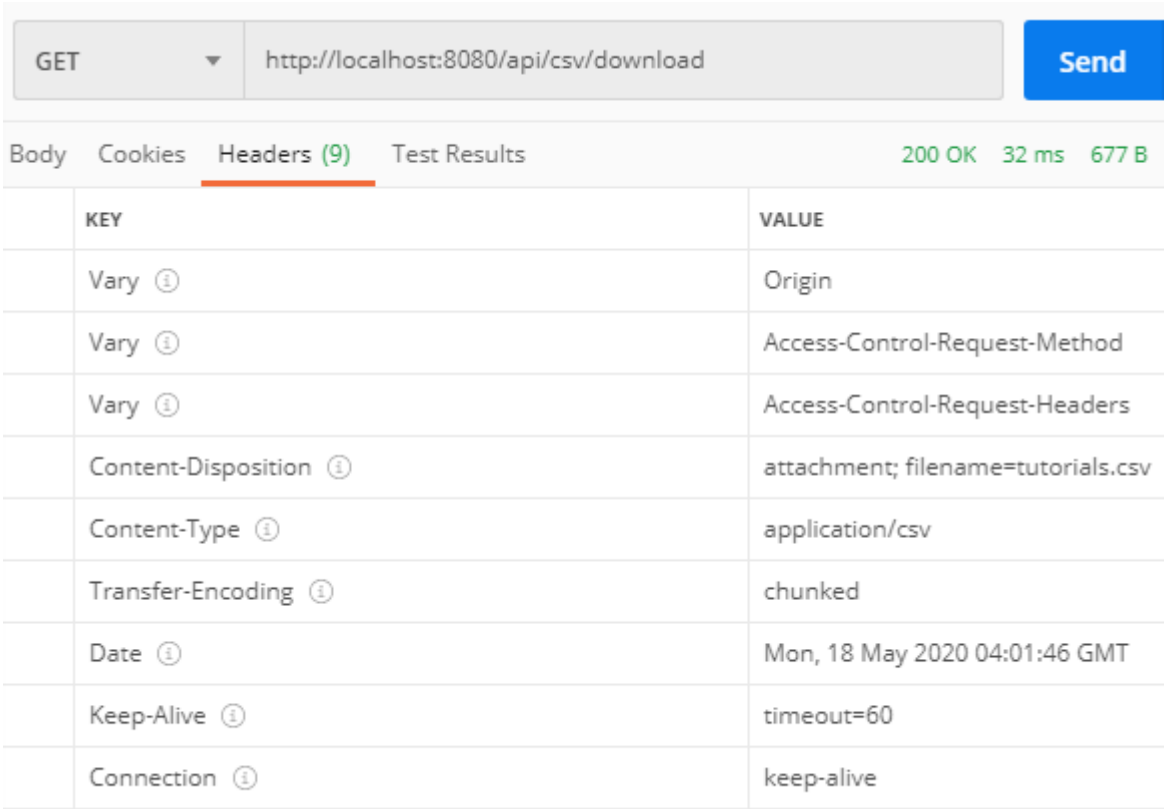
Spring Boot Rest API returns CSV File

If you send request to `/api/csv/download`, the server will return a response with a CSV file **tutorials.csv** that contains data in MySQL table:



How to do this?
You need to set the HTTP header:

```
"Content-disposition" : "attachment; filename=[yourFileName]"
"Content-Type" : "application/csv"
```



You can find step by step for downloading CSV file in the tutorial:
Spring Boot Download CSV file from Database example

These are APIs to be exported:

Methods	Urls	Actions
POST	/api/csv/upload	upload a CSV File
GET	/api/csv/tutorials	get List of items in db table
GET	/api/csv/download	download db data as CSV file

Technology

- Java 8
- Spring Boot 2 (with Spring Web MVC)
- Maven 3.6.1
- Apache Commons CSV 1.8

Project Structure

This is the project directory that we're gonna build:

– `CSVHelper` provides functions to read/write to CSV file.

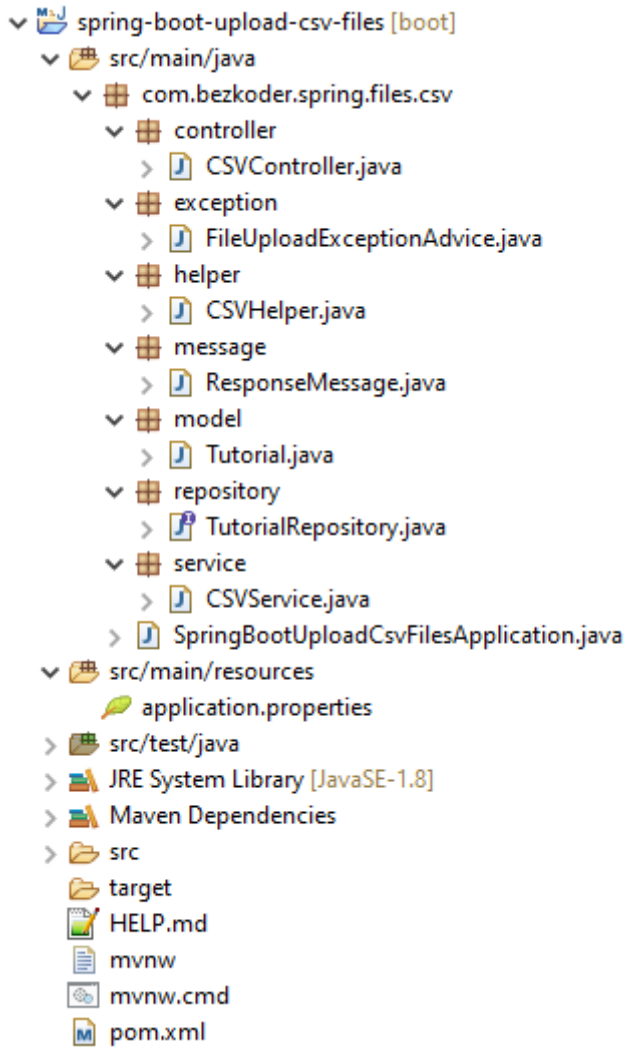
– `Tutorial` data model class corresponds to entity and table **tutorials**.
– `TutorialRepository` is an interface that extends `JpaRepository` for persisting data.

– `CSVService` uses `CSVHelper` and `TutorialRepository` methods to save CSV data to MySQL, load data to export CSV file, or get all Tutorials from MySQL table.

– `CSVController` calls `CSVService` methods and export Rest APIs: upload CSV file, get data from MySQL database.

– `FileUploadExceptionAdvice` handles exception when the controller processes file upload.

– *application.properties* contains configuration for Spring Data and Servlet Multipart file.



– *pom.xml* for Spring Boot, MySQL connector, Apache Commons CSV dependencies.

Setup Spring Boot CSV File Upload/Download project

Use Spring web tool or your development tool (Spring Tool Suite, Eclipse, Intellij) to create a Spring Boot project.

Then open **pom.xml** and add these dependencies:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-csv</artifactId>
        <version>1.8</version>
</dependency>
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
</dependency>
```

Configure Spring Datasource, JPA, Hibernate

Under **src/main/resources** folder, open *application.properties* and write these lines.

```
spring.datasource.url= jdbc:mysql://localhost:3306/testdb?useSSL=false
spring.datasource.username= root
spring.datasource.password= 123456
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto= update
```

- `spring.datasource.username` & `spring.datasource.password` properties are the same as your database installation.
- Spring Boot uses Hibernate for JPA implementation, we configure `MySQL5InnoDBDialect` for MySQL database
- `spring.jpa.hibernate.ddl-auto` is used for database initialization. We set the value to `update` value so that a table will be created in the database automatically corresponding to defined data model. Any change to the model will also trigger an update to the table. For production, this property should be `validate`.

Define Data Model

Our Data model is Tutorial with four fields: id, title, description, published.
In **model** package, we define `Tutorial` class.

model/Tutorial.java

```
package com.bezkoder.spring.files.csv.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name = "tutorials")
public class Tutorial {
  @Id
  @Column(name = "id")
  private long id;
  @Column(name = "title")
  private String title;
  @Column(name = "description")
  private String description;
  @Column(name = "published")
  private boolean published;
  public Tutorial() {
  }
  public Tutorial(long id, String title, String description, boolean published) {
    this.id = id;
    this.title = title;
    this.description = description;
    this.published = published;
  }
  public long getId() {
    return id;
  }
  public void setId(long id) {
    this.id = id;
  }
  public String getTitle() {
    return title;
  }
  public void setTitle(String title) {
    this.title = title;
  }
  public String getDescription() {
    return description;
  }
  public void setDescription(String description) {
    this.description = description;
  }
  public boolean isPublished() {
    return published;
  }
  public void setPublished(boolean isPublished) {
    this.published = isPublished;
  }
  @Override
  public String toString() {
    return "Tutorial [id=" + id + ", title=" + title + ", desc=" + description +
", published=" + published + "]";
  }
}
```

– `@Entity` annotation indicates that the class is a persistent Java class.
– `@Table` annotation provides the table that maps this entity.
– `@Id` annotation is for the primary key.
– `@Column` annotation is used to define the column in database that maps annotated field.

Create Data Repository for working with Database

Let's create a repository to interact with Tutorials from the database.
In **repository** package, create `TutorialRepository` interface that extends `JpaRepository`.

repository/TutorialRepository.java

```
package com.bezkoder.spring.files.csv.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.bezkoder.spring.files.csv.model.Tutorial;
public interface TutorialRepository extends JpaRepository {
}
```

Now we can use JpaRepository's methods: `save()`, `findOne()`, `findById()`, `findAll()`, `count()`, `delete()`, `deleteById()` ... without implementing these methods.

The quantity of rows in CSV file (also **tutorials** table) could be large, so you may want to get only several at once by modifying this Repository to work with Pagination, the instruction can be found at:
Spring Boot Pagination & Filter example | Spring JPA, Pageable

You also find way to write Unit Test for this JPA Repository at:
Spring Boot Unit Test for JPA Repositiory with @DataJpaTest

Implement Read/Write CSV Helper Class

We're gonna use Apache Commons CSV classes such as: `CSVParser`, `CSVRecord`, `CSVFormat`.
Let me summarize the steps for reading CSV file:

- create `BufferedReader` from `InputStream`
- create `CSVParser` from the `BufferedReader` and CSV format
- iterate over `CSVRecord` s by `Iterator` with `CsvParser.getRecords()`
- from each `CSVRecord`, use `CSVRecord.get()` to read and parse fields

```
BufferedReader fileReader = new BufferedReader(new InputStreamReader(is, "UTF-
8"));
CSVParser csvParser = new CSVParser(fileReader, CSVFormat.DEFAULT...);
Iterable<CSVRecord> csvRecords = csvParser.getRecords();
for (CSVRecord csvRecord : csvRecords) {
  Tutorial tutorial = new Tutorial(
        Long.parseLong(csvRecord.get("Id")),
        csvRecord.get("Title"),
        csvRecord.get("Description"),
        Boolean.parseBoolean(csvRecord.get("Published"))
      );
  tutorialsList.add(tutorial);
}
```

Under **helper** package, we create `CSVHelper` class with 3 methods:

- `hasCSVFormat()` : check if a file has CSV format or not
- `csvToTutorials()` : read `InputStream` of a file, return a list of Tutorials

Here is full code of *helper/CSVHelper.java*:

```
package com.bezkoder.spring.files.csv.helper;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVRecord;
import org.springframework.web.multipart.MultipartFile;
import com.bezkoder.spring.files.csv.model.Tutorial;
public class CSVHelper {
  public static String TYPE = "text/csv";
  static String[] HEADERs = { "Id", "Title", "Description", "Published" };
  public static boolean hasCSVFormat(MultipartFile file) {
    if (!TYPE.equals(file.getContentType())) {
      return false;
    }
    return true;
  }
  public static List<Tutorial> csvToTutorials(InputStream is) {
    try (BufferedReader fileReader = new BufferedReader(new InputStreamReader(is,
"UTF-8"));
        CSVParser csvParser = new CSVParser(fileReader,

CSVFormat.DEFAULT.withFirstRecordAsHeader().withIgnoreHeaderCase().withTrim());) {
      List<Tutorial> tutorials = new ArrayList<Tutorial>();
      Iterable<CSVRecord> csvRecords = csvParser.getRecords();
      for (CSVRecord csvRecord : csvRecords) {
        Tutorial tutorial = new Tutorial(
              Long.parseLong(csvRecord.get("Id")),
              csvRecord.get("Title"),
              csvRecord.get("Description"),
              Boolean.parseBoolean(csvRecord.get("Published"))
            );
        tutorials.add(tutorial);
      }
      return tutorials;
    } catch (IOException e) {
      throw new RuntimeException("fail to parse CSV file: " + e.getMessage());
    }
  }
}
```

Create CSV File Service

`CSVService` service class will be annotated with `@Service` annotation, it uses `CSVHelper` and `TutorialRepository` for 2 functions:

- `save(MultipartFile file)` : store CSV data to database
- `getAllTutorials` (): read data from database and return `List<Tutorial>`

service/*CSVService.java*

```
package com.bezkoder.spring.files.csv.service;
import java.io.IOException;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import com.bezkoder.spring.files.csv.helper.CSVHelper;
import com.bezkoder.spring.files.csv.model.Tutorial;
import com.bezkoder.spring.files.csv.repository.TutorialRepository;
@Service
public class CSVService {
  @Autowired
  TutorialRepository repository;
  public void save(MultipartFile file) {
    try {
      List<Tutorial> tutorials = CSVHelper.csvToTutorials(file.getInputStream());
      repository.saveAll(tutorials);
    } catch (IOException e) {
      throw new RuntimeException("fail to store csv data: " + e.getMessage());
    }
  }
  public List<Tutorial> getAllTutorials() {
    return repository.findAll();
  }
}
```

Define Response Message

The `ResponseMessage` is for message to client that we're gonna use in Rest Controller and Exception Handler.

message/*ResponseMessage.java*

```
package com.bezkoder.spring.files.csv.message;
public class ResponseMessage {
  private String message;
  public ResponseMessage(String message) {
    this.message = message;
  }
  public String getMessage() {
    return message;
  }
  public void setMessage(String message) {
    this.message = message;
  }
}
```

Create Controller for Upload CSV Files

In **controller** package, we create `CSVController` class for Rest Apis.
– `@CrossOrigin` is for configuring allowed origins.
– `@Controller` annotation indicates that this is a controller.
– `@GetMapping` and `@PostMapping` annotation is for mapping HTTP GET & POST requests onto specific handler methods:

- POST /upload: `uploadFile()`
- GET /tutorials: `getAllTutorials()`

– We use `@Autowired` to inject implementation of `CSVService` bean to local variable.

controller/*CSVController.java*

```
package com.bezkoder.spring.files.csv.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import com.bezkoder.spring.files.csv.service.CSVService;
import com.bezkoder.spring.files.csv.helper.CSVHelper;
import com.bezkoder.spring.files.csv.message.ResponseMessage;
import com.bezkoder.spring.files.csv.model.Tutorial;
@CrossOrigin("http://localhost:8081")
@Controller
@RequestMapping("/api/csv")
public class CSVController {
  @Autowired
  CSVService fileService;
  @PostMapping("/upload")
  public ResponseEntity<ResponseMessage> uploadFile(@RequestParam("file")
MultipartFile file) {
    String message = "";
    if (CSVHelper.hasCSVFormat(file)) {
      try {
        fileService.save(file);
        message = "Uploaded the file successfully: " + file.getOriginalFilename();
        return ResponseEntity.status(HttpStatus.OK).body(new
ResponseMessage(message));
      } catch (Exception e) {
        message = "Could not upload the file: " + file.getOriginalFilename() +
"!";
        return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body(new
ResponseMessage(message));
      }
    }
    message = "Please upload a csv file!";
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new
ResponseMessage(message));
  }
  @GetMapping("/tutorials")
  public ResponseEntity<List<Tutorial>> getAllTutorials() {
    try {
      List<Tutorial> tutorials = fileService.getAllTutorials();
      if (tutorials.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
      }
      return new ResponseEntity<>(tutorials, HttpStatus.OK);
    } catch (Exception e) {
      return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }
}
```

Configure Multipart File for Servlet

Let's define the maximum file size that can be uploaded in *application.properties* as following:

```
spring.servlet.multipart.max-file-size=2MB
spring.servlet.multipart.max-request-size=2MB
```

- `spring.servlet.multipart.max-file-size` : max file size for each request.
- `spring.servlet.multipart.max-request-size` : max request size for a multipart/form-data.

Handle File Upload Exception

This is where we handle the case in that a request exceeds Max Upload Size. The system will throw `MaxUploadSizeExceededException` and we're gonna use `@ControllerAdvice` with `@ExceptionHandler` annotation for handling the exceptions.

exception/FileUploadExceptionAdvice.java

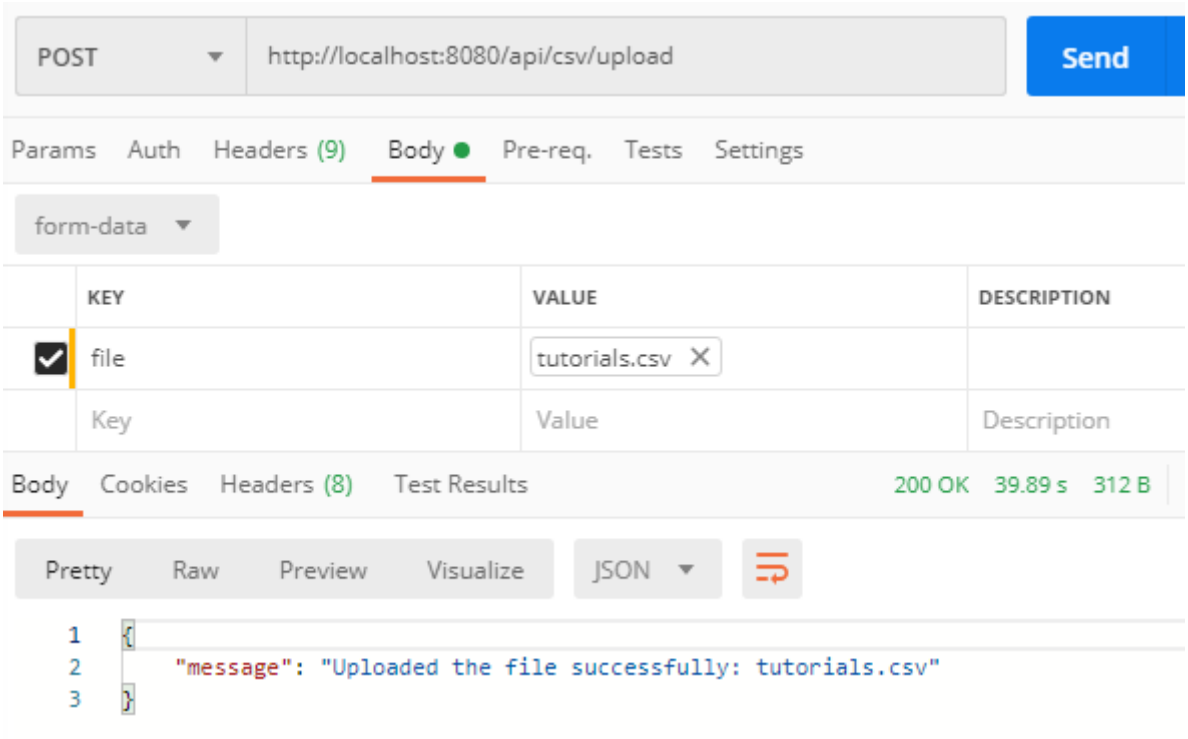
```
package com.bezkoder.spring.files.csv.exception;
import org.springframework.web.multipart.MaxUploadSizeExceededException;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandle

import com.bezkoder.spring.files.csv.message.ResponseMessage;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
@ControllerAdvice
public class FileUploadExceptionAdvice extends ResponseEntityExceptionHandler {
  @ExceptionHandler(MaxUploadSizeExceededException.class)
  public ResponseEntity handleMaxSizeException(MaxUploadSizeExceededException exc)
{
    return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body(new
ResponseMessage("File too large!"));
  }
}
```

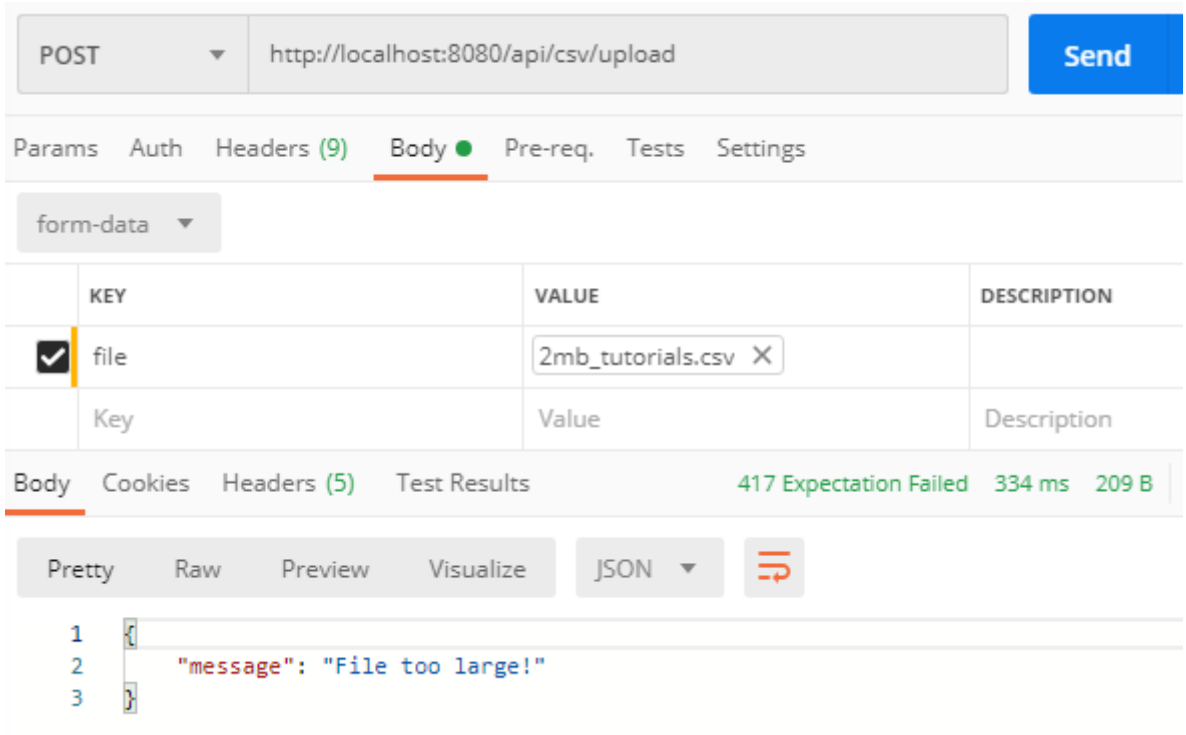
Run & Check

Run Spring Boot application with command: `mvn spring-boot:run` .

Let's use **Postman** to make some requests.



If you upload a file with size larger than max file size (2MB):



Conclusion

Today we've built a Rest CRUD API using Spring Boot to upload and read CSV file, then store data in Mysql database.

We also see how to use Apache Commons CSV to read/write data with CSV file, `JpaRepository` to retrieve items in database table without need of boilerplate code.

If you want to add Pagination to this Spring Application, you can find the instruction at:
Spring Boot Pagination & Filter example | Spring JPA, Pageable

For downloading CSV file:
Spring Boot Download CSV file from Database example

Upload Excel file instead:
Spring Boot: Upload/Import Excel file data into MySQL Database

Or upload files to database as BLOB:
Spring Boot Upload/Download File to/from Database example

id	data	name	type
5d71322e-a954-4d7a-b0e6-7c799b5aae5f	BLOB	bezkoder.png	image/png
6ba3578c-ce22-4dd7-999e-72192bf31b53	BLOB	bezkoder.doc	application/msword
88108ee4-5354-4041-bfc6-2965fc8af4f4	BLOB	bezkoder.jpg	image/jpeg

Happy learning! See you again.

Further Reading

Deployment:
– Deploy Spring Boot App on AWS – Elastic Beanstalk
– Docker Compose: Spring Boot and MySQL example

Source Code

You can find the complete source code for this tutorial on Github.