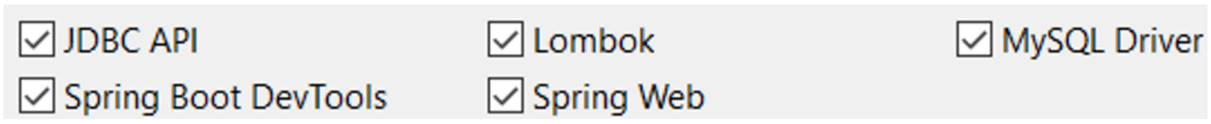
REST API Code Build Guide :

Step 1:

Create a SPRING STARTER PROJECT.

- Add Dependencies/features:

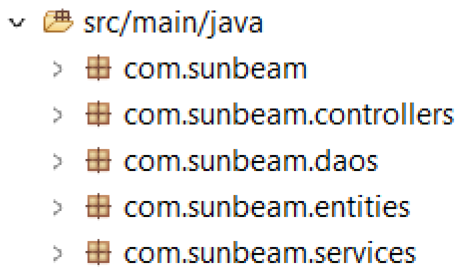


✓ JDBC API	✓ Lombok	✓ MySQL Driver
✓ Spring Boot DevTools	✓ Spring Web	

Next > Finish.

Step 2:

Create a Parent Package, and inside it, all your sub-packages will coincide.



```
src/main/java
    com.sunbeam
    com.sunbeam.controllers
    com.sunbeam.daos
    com.sunbeam.entities
    com.sunbeam.services
```

Step 3: Create a sub-package of 'ENTITIES'

- Inside, it create POJO classes for all tables in your Database. (It contains variable declarations similar to column names and matching data types.)

Give annotation:

```
@RequiredArgsConstructor
@AllArgsConstructor
@Data
```

- Lombok Annotation is used by Spring to create helper methods (getters, setters, and parameterless/parameterized constructors).

Step 4: Create Subpackage 'DAOS'.

- Inside it, there will be DAO interfaces (only method declarations of those that we want to implement as functionalities of the Backend).

    - Every Dao should be marked as @Component (bean can be created, but ONLY reference is possible for interface)
    - And then inside the same package, IMPLEMENT every DAO by inheriting its respective DAO Interface. Mark it with @Repository(it deals with the database)
    - Be assured, you implement the right methods in the right way to get the correct output.
    - Implement these DAOs and use JdbcTemplate and RowMapper. Implement methods with String SQL.
    - Inside Dao's Sub package: Create Row Mapper for each entity. Implemented by RowMapper<Entity>.
    - Inside it, create an object of identity type(get data by result set) and return it(object of ENTITY).
- Make it marked with an annotation: @Component (It will be auto-wired in future use).

Step 5: Create a Service sub-package, and create a Service interface for each DAO of the entity, and each service implemented is to be annotated by @Service (it is also to be Autowired). Call methods by @Autowired Dao.

Step 6: Create a Controller sub-package, write methods Of Mapping with URL (

URL: http://localhost:8080/@RequestMapping/@(GET/POST/PUT/PATCH/DELETE)Mapping

Annot methods with proper mapping as per req res you will be generating.

Application Properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/mobiles_db
spring.datasource.username=W2_89944_Prathamesh
spring.datasource.password=manager

imagesFolderPath = P:/PG-DMC/05WEBJAVA/ClassWork/Day7/images
```

- DataSource: here depicts the route to the Data from where we are doing data handling(Database).

imagesFolderPath: Check forward slashes (it must be forward slashes without “ ”).

    - Will be used by @Value("${ imagesFolderPath}"). Return the String to below variable or field below.
- It will be used where any data handling will be carried out.

Response Util:

```
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonInclude.Include;

import lombok.AllArgsConstructor;
import lombok.Data;


@JsonInclude(Include.NON_NULL)
@AllArgsConstructor
@Data
public class ResponseUtil<T> {
            private String status;
            private String message;
            private T data;

            public static <T> ResponseUtil<T> apiSuccess(T data){

                return new ResponseUtil<T>("success", null, data);
            }
            public static <T> ResponseUtil<T> apiError(String message) {
                ResponseUtil<T> result = new ResponseUtil<T>("error", message, null);
                return result;
            }
}
```

JsonInclude(Include.NON_NULL) // non null values are ignored in Output

    - T is a Generic Type.
    - <T> ResponseUtil<T> is a generic method and can be of the type of 'data' object we will send.
    - API error will be called when your output is not desired, and our custom message will be returned in the message body.
- It returns data (our output OBJECT as a response).

Exception Handling:

```
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class ExceptionHandeling {

    @ExceptionHandler
    public ResponseUtil<?> handleException(Throwable ex) {
        return ResponseUtil.apiError(ex.getMessage());
    }

}
```

@RestControllerAdvice: it annotates the controller to use the resources of this class (ExceptionHandler in our case)

@ExceptionHandler: it indicates that every error handling is being handled by the class below.

Image Downloader:

```
@RestController
public class ImageDownloader {

        @Value("${imagesFolderPath}")
        private String imageFolderPath;

        @GetMapping("/images/{imageName}")
        public void downloadImage(@PathVariable("imageName") String imgName, HttpServletResponse resp) throws Exception {
            String filePath = imageFolderPath + "/" + imgName;
            try(FileInputStream fin = new FileInputStream(filePath)) {
                FileCopyUtils.copy(fin, resp.getOutputStream());
            }
        }
}
```

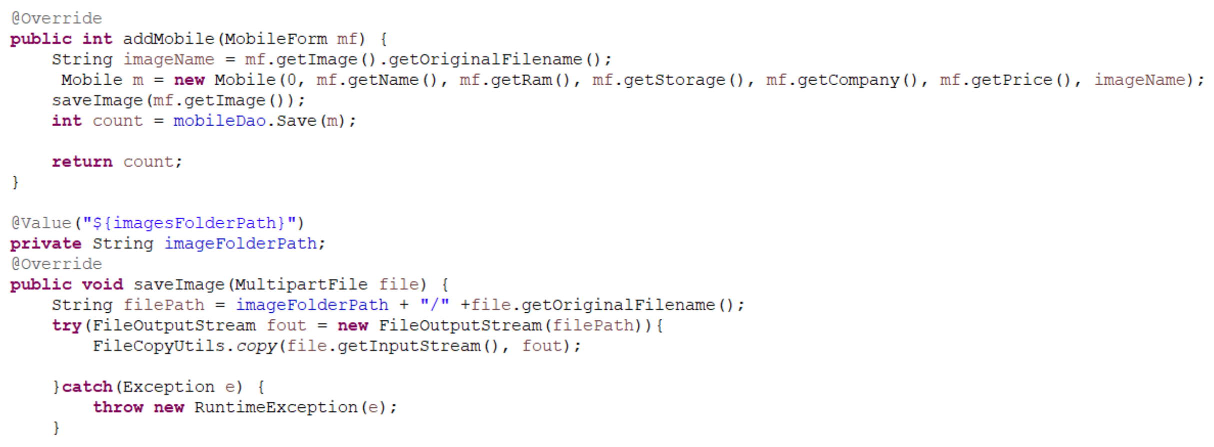
- The 'String imageFolderPath ' gets the value of the actual path of the folder where the data (images) is present.
- Filepath has the absolute path as it attaches = imageFolderPath + "/" + imageName;
- And then we open an input stream in try with a resource block, where it reads from(filePath(Absolute Path of given image)). And write on HttpServletResponse resp (i.e, Screen [std.out]).

  - FileCopyUtils.copy(From, Destination); //Function copies from source and writes to Destination.

Save (Upload):

```
@PostMapping("")
public ResponseUtil<?> add(MobileForm mf){
    mobileServices.addMobile(mf);
    return ResponseUtil.apiSuccess("Mobile FROM updated.");
}
```

- We go for the default mapping URL (RequestMapping).
- We receive MobileForm(a model created for a DATA came from data-form in the request).
- Calls addMobile(mf) by @Autowired mobileServices
- The return statement is executed if it is executed successfully.
- But a call is made to MobileServiceImpl >> addMobile, and the mf object is passed.



```java
@Override
public int addMobile(MobileForm mf) {
    String imageName = mf.getImage().getOriginalFilename();
     Mobile m = new Mobile(0, mf.getName(), mf.getRam(), mf.getStorage(), mf.getCompany(), mf.getPrice(), imageName);
    saveImage(mf.getImage());
    int count = mobileDao.Save(m);

    return count;
}

@Value("${imagesFolderPath}")
private String imageFolderPath;
@Override
public void saveImage(MultipartFile file) {
    String filePath = imageFolderPath + "/" +file.getOriginalFilename();
    try(FileOutputStream fout = new FileOutputStream(filePath)){
        FileCopyUtils.copy(file.getInputStream(), fout);

    }catch(Exception e) {
        throw new RuntimeException(e);
    }
```

Here, we are in the MobileServiceImpl class:

- We take String imageName = will store the image name of the MultipartFile file.
- Mobile m is an object that sets all parameters and specifically the image name as a string.
- SaveImage(mf.getImage()) // sends actual file(multipartfile file)
- In SaveImage :
    - We have String filePath = Absolute file path.
    - We open fout(file output stream, which is a file path).
    - We open the input stream on the file we got as input from the add method.
    - FileCopyUtils.copy(From, Destination) //copy from stream into Destination.