IMPRESSICO BUSINESS SOLUTIONS

your technology partner

**Created by- Denis Victor**

**Trainer- Mr. Durgesh**

**Manager - Mr. Neeraj Arora**

# Spring Boot

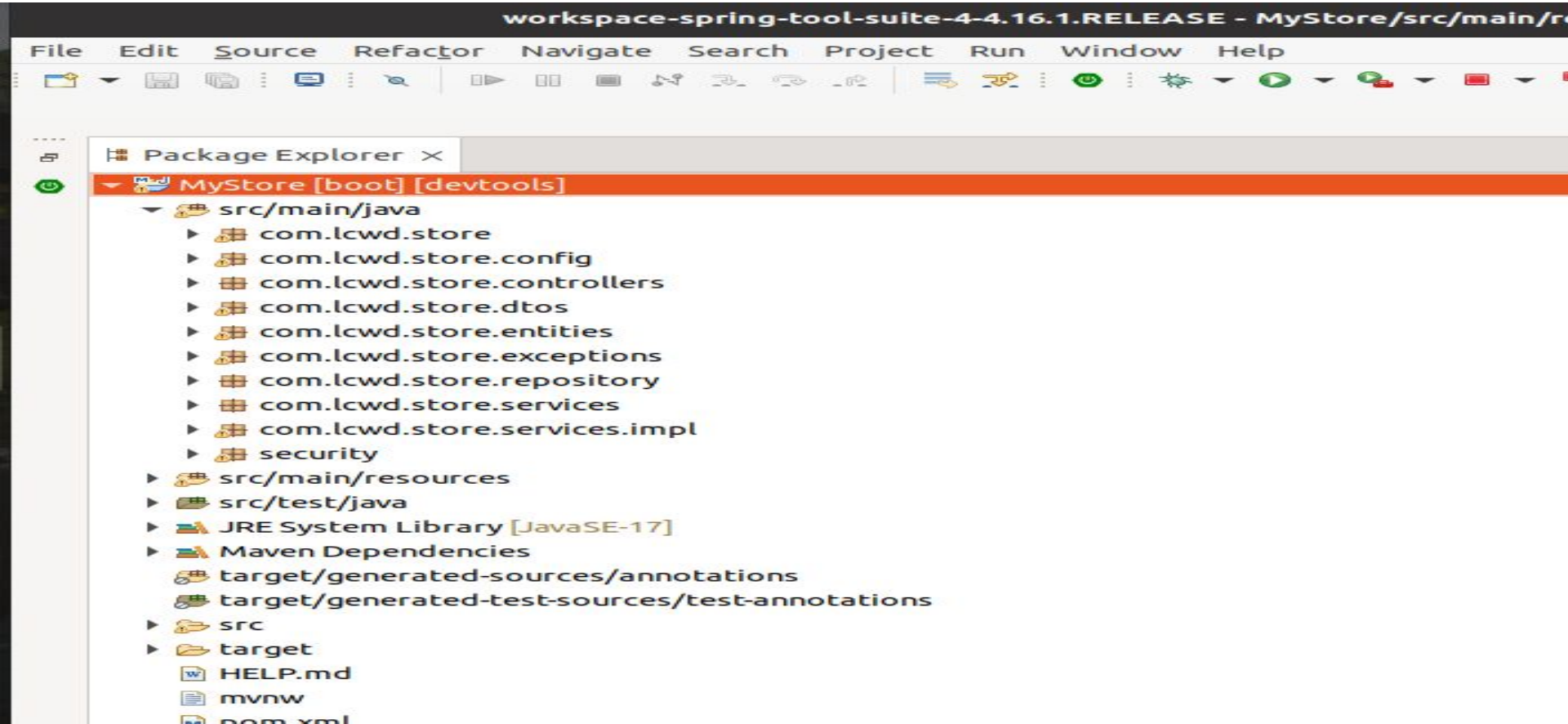
- Create stand-alone Spring applications
- Inbuilt Embed Tomcat  etc
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Absolutely no code generation and no requirement for XML configuration

# MyStore Application

It’s an E-Commerce application that lets you create an account, add products to the cart and order items in a safe, secure and reliable way using spring-boot technology.

Walk through of the project and various technologies used.

# Project Architecture



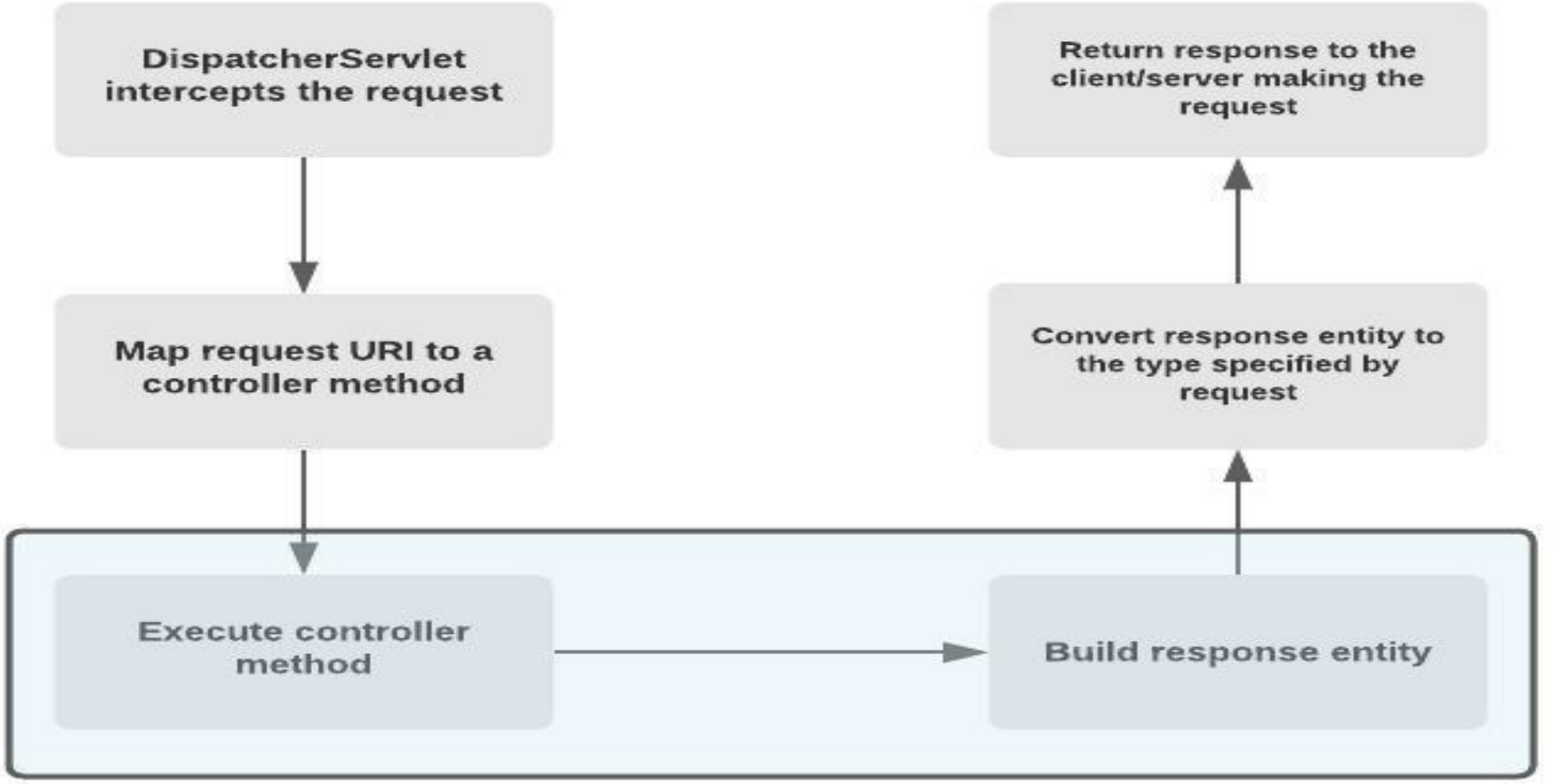
# Annotations

**Spring Annotations** are a form of metadata that provides data about a program. Annotations are used to provide supplemental information about a program. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.

# Annotations Used

- @SpringBootApplication
- @Configuration
- @RestController
- @RequestMapping
- @Entity
- @RestControllerAdvice
- @Service

# Controller


DispatcherServlet intercepts the request
Map request URI to a controller method
Execute controller method
Build response entity
Convert response entity to the type specified by request
Return response to the client/server making the request

```java
package com.lcwd.store.controllers;

import java.util.List;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private AuthenticationManager manager;

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private JwtTokenHelper helper;


    private Logger logger = LoggerFactory.getLogger(JwtAuthenticationFilter.class);

    //create
    @PostMapping
    public ResponseEntity<UserDto> createUser(@Valid @RequestBody UserDto userDto){
        userDto.setPassword(passwordEncoder.encode(userDto.getPassword()));
        UserDto userDto2 = userService.addUser(userDto);
        return new ResponseEntity<UserDto>(userDto2,HttpStatus.CREATED);
```

# Service

- The business logic resides within the service layer so we use the **@Service Annotation** to indicate that a class belongs to that layer.

- @Service Annotation is it can be applied only to classes.

- Spring context will autodetect these classes when annotation-based configuration and classpath scanning is used.

# Repository

JPA Repository is mainly used for managing the data in a Spring Boot Application.

```
public interface UserRepository extends JpaRepository<User, String> {

    // custom finder methods

}
```

# Global Exception Handling

In Java, exception handling is done by try, catch blocks but spring boot also allows us to provide customized global exception handling where we need not to add try catch block everywhere, we can create a separate class for handling exceptions and it also separates the exception handling code from bussinesss logic code.

```java
package com.lcwd.store.exceptions;

import java.util.HashMap;

@RestControllerAdvice
public class GlobalExceptionHandler {

    private Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.class);

    // resource not found exception
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ApiResponse> handleRunTimeException(RuntimeException e){
        logger.info("RuntimeException generated : {}", e.getMessage() );
        return new ResponseEntity<ApiResponse>(ApiResponse.builder().message(e.getMessage()).success(false).build(

    }

    //handling for invalid exception
    @ExceptionHandler(InvalidAgeException.class)
    public ResponseEntity<ApiResponse> handleRuntimeException(InvalidAgeException e){
        logger.info("Invalid age generated : {} ", e.getMessage());
        return new ResponseEntity<ApiResponse>(ApiResponse.builder().message(e.getMessage()).success(false).build(
    }

    //method for handling validation exception
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>> handelValidationException(MethodArgumentNotValidException e){
        Map<String, String> responseMap = new HashMap<>();

        List<ObjectError> allErrors = e.getBindingResult().getAllErrors();
        allErrors.forEach(error ->{
            String defaultMessage = error.getDefaultMessage();
```

# Java Persistence API (JPA)

It is a Java specification that gives some functionality and standard to ORM tools. It is used to examine, control, and persist data between Java objects and relational databases. It is observed as a standard technique for Object Relational Mapping.

# Hibernate

The main feature of **Hibernate** is to map the Java classes to database tables. Following are some key features of Hibernate :

- Hibernate is an implementation of JPA guidelines.
- It helps in mapping Java data types to SQL data types.
- It is the contributor of JPA.

# JPA vs Hibernate

*The major difference between Hibernate and JPA is that Hibernate is a framework while JPA is API specifications. Hibernate is the implementation of all the JPA guidelines.*

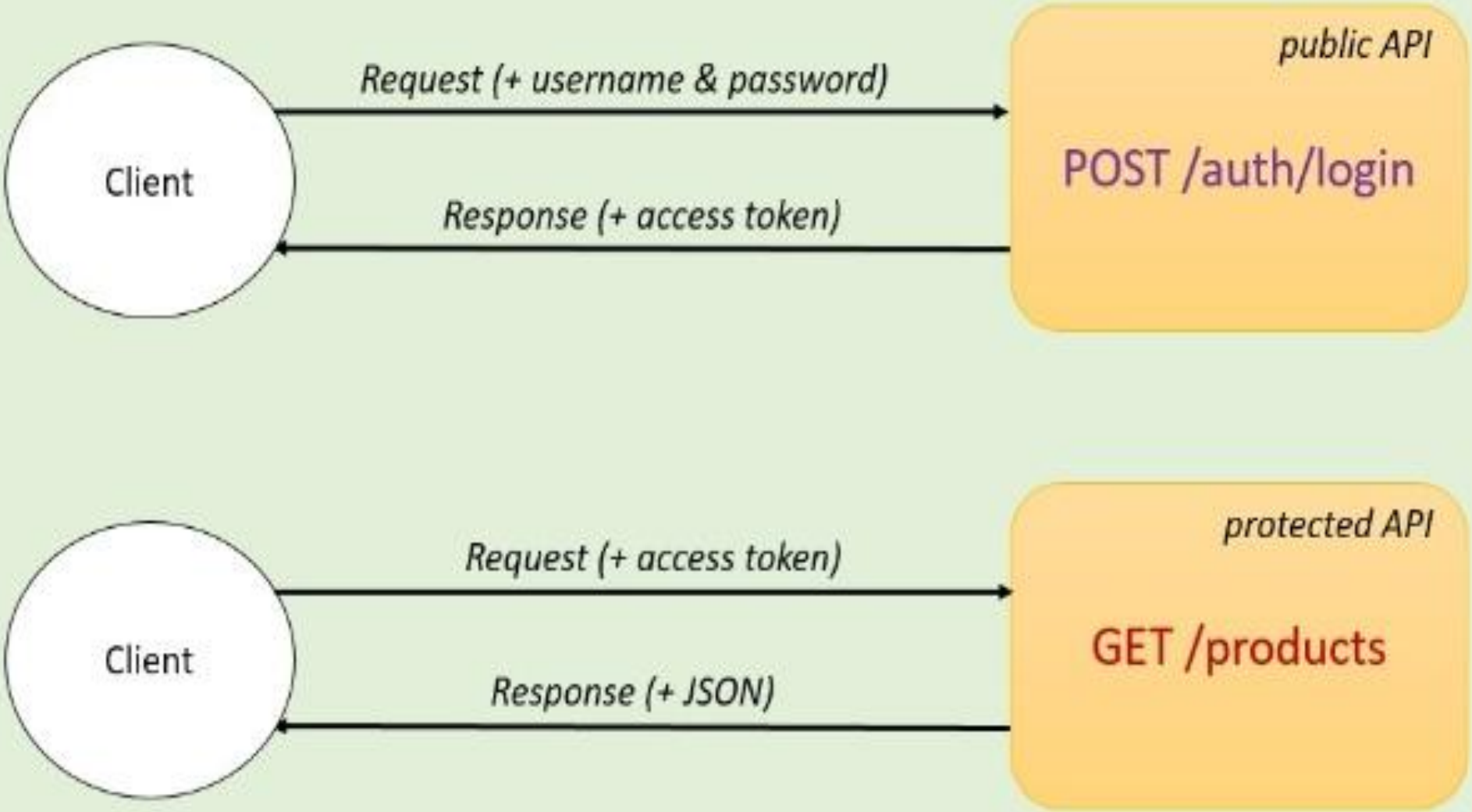
# **D**ata **T**ransfer **O**bject (DTO)

In Spring Framework, **D**ata **T**ransfer **O**bject (DTO) is an object that carries data between processes. When you're working with a remote interface, each call is expensive. As a result, you need to reduce the number of calls. The solution is to create a Data Transfer Object that can hold all the data for the call.

```java
@Setter
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Builder
public class UserDto {

    private String id;

    @NotBlank(message = "Name is required")
    @Size(min=5,max=15, message = "Username must be between 5 to 15 chars")
    private String name;

    @Email(message = "valid email is required")
    private String email;


    @Pattern(regexp = "(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{8,}", message = "pass shoudl conta
    private String password;

    private String about;

    @NotBlank(message = "gender is required")
    private String gender;


    @JsonFormat(pattern = "dd/MM/yyyy")
    private Date dob;

    private Set<RoleDto> roles = new HashSet<>();
}
```

# Spring Security and JWT

JSON Web Token or JWT, as it is more commonly called, is an open Internet standard (RFC 7519) for securely transmitting trusted information between parties in a compact way. The tokens contain claims that are encoded as a JSON object and are digitally signed using a private secret or a public key/private key pair.


Client
Request (+ username & password)
Response (+ access token)
public API
POST /auth/login
Client
Request (+ access token)
Response (+ JSON)
protected API
GET /products

# Swagger

Swagger is an open source project used to generate the REST API documents for RESTful web services. It provides a user interface to access our RESTful web services via the web browser.

## cart-controller Cart Controller

## category-controller Category Controller

GET /category getAll

POST /category addEntity

GET /category/{categoryId} getCategory

PUT /category/{categoryId} updatEntity

DELETE /category/{categoryId} deletEntity

POST /category/{categoryId}/products createProuct

POST /category/{categoryId}/products/{productId} updateCategory

Parameters

Try it out

# Thank You