Introduction to Spring Framework

George Michas
Team : Dataurus
Software Technology 2017-2018

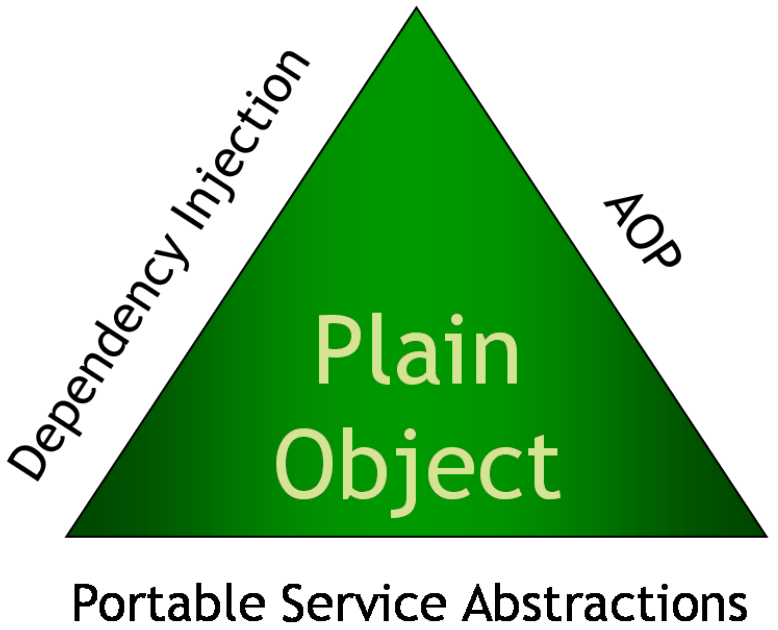
What we will cover in this talk ?

- What is Spring Framework ? (Captain Obvious)
- Important Features 
- Spring Architecture
- Benefits
- Spring Boot
- Bonus Round : Step by Step Spring usage in our Scrumit Project

What is Spring Framework ?

- The most popular application development framework for enterprise Java
- First version was written by Rod Johnson, the year 2002
- Open source Java Platform since 2003 (Release Date)
- Spring is a framework that aims to reduce the complexity and helps to make a lot of things simpler to develop

Important Features



Dependency Injection (DI)

- Do not create or instantiate an object inside another Java class
- Objects "talk" to each other only as arguments via constructors, factory methods, setters
- Rely on Spring's IoC (Inversion of Control) module to create the object for you
- Help us :
  - Keep classes as independent as possible (Greater Modularity)
  - Easy Testing
  - More understandable code
  - ~~“Spaghetti” coding~~
  - Reusable code
  - S.O.L.I.D

Simple example of DI

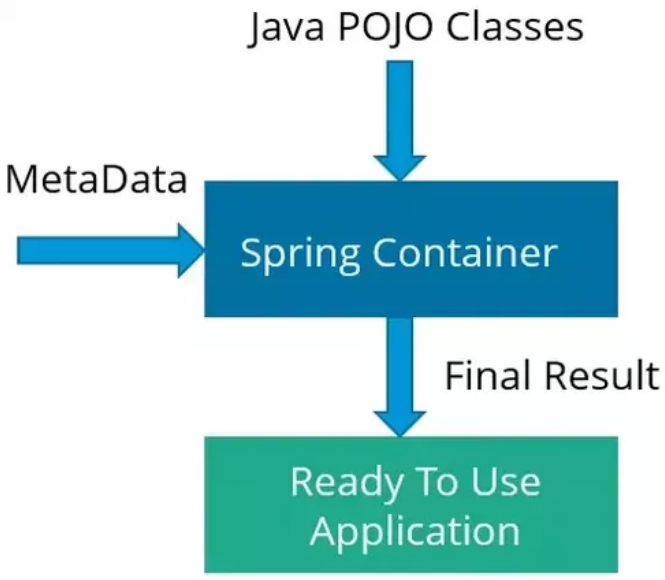
```java
public class Store {
    private Item item;

    public Store() {
        item = new ItemImpl1();
    }
}
```

```java
public class Store {
    private Item item;
    public Store(Item item) {
        this.item = item;
    }
}
```

Inversion of Control

- Inversion of Control is a principle in software engineering by which the control of objects or portions of a program is transferred to a container or framework. It’s most often used in the context of object-oriented programming.
- Dependency Injection is a pattern through which to implement IoC, where the control being inverted is the setting of object’s dependencies.
- IoC container is the core of the spring framework
  - Create Objects
  - "Put" them together
  - Configure
  - Manage lifecycle

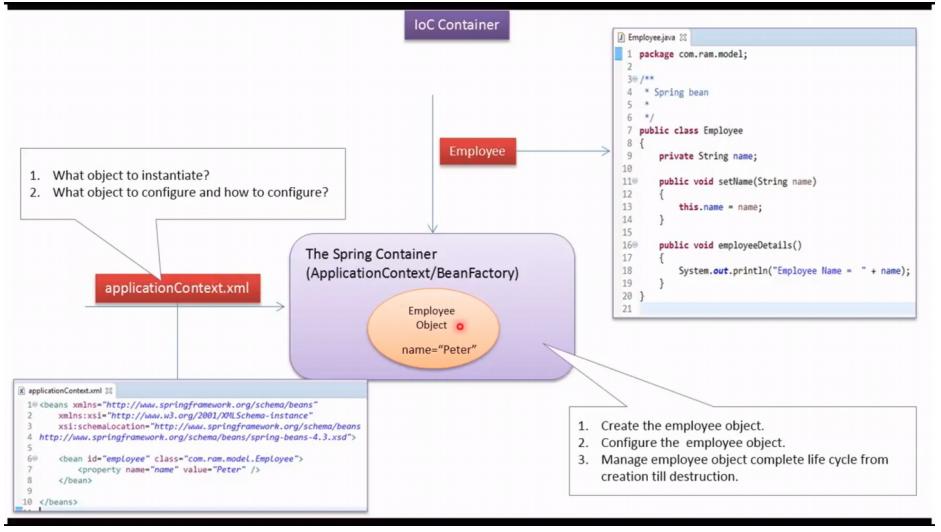
How IoC works ?

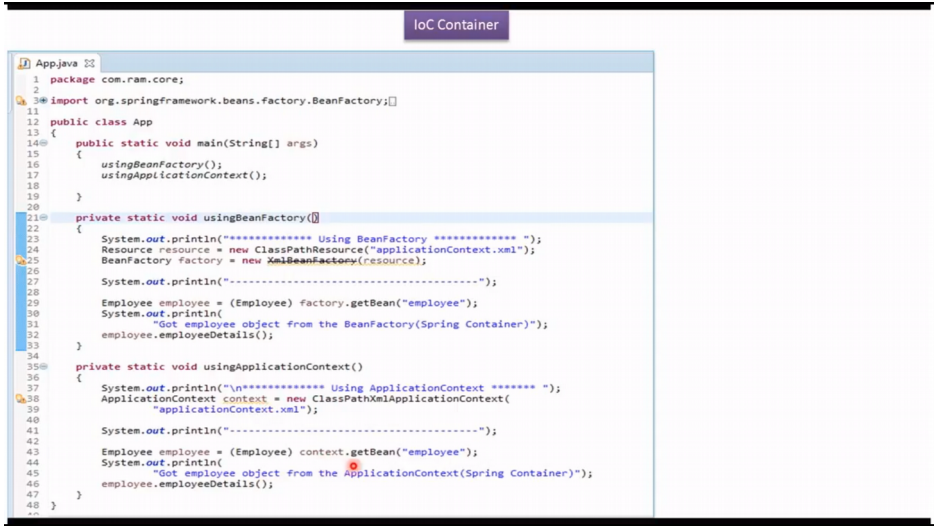


- Container reads MetaData (Beans)
  - Instantiates
  - Configures
  - Assembles

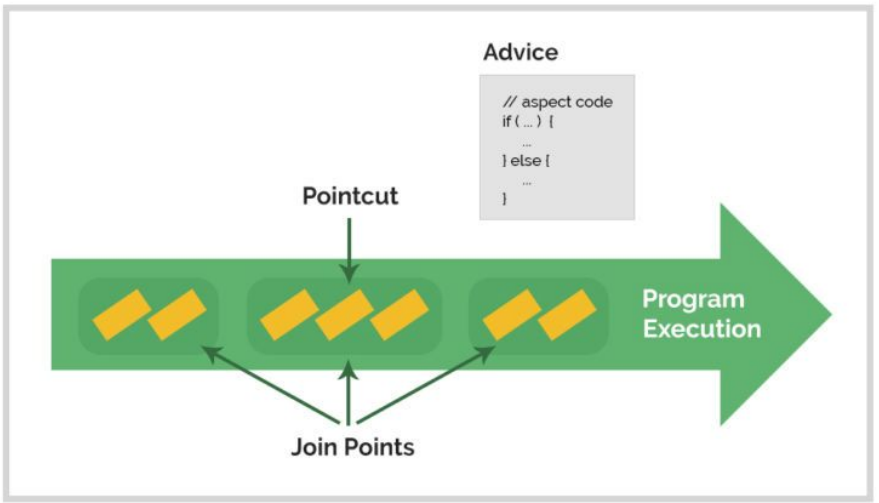
- Configuration MetaData
  - Annotation Based : By using @Service or @Component
  - XML Based : Configure the beans. If you are using Spring MVC framework, the xml based configuration can be loaded automatically by writing some boiler plate code in web.xml file
  - Java Based : Configure Spring beans using java programs. Some important annotations @Configuration, @ComponentScan and @Bean

More analytic Example of IoC Container (Old)





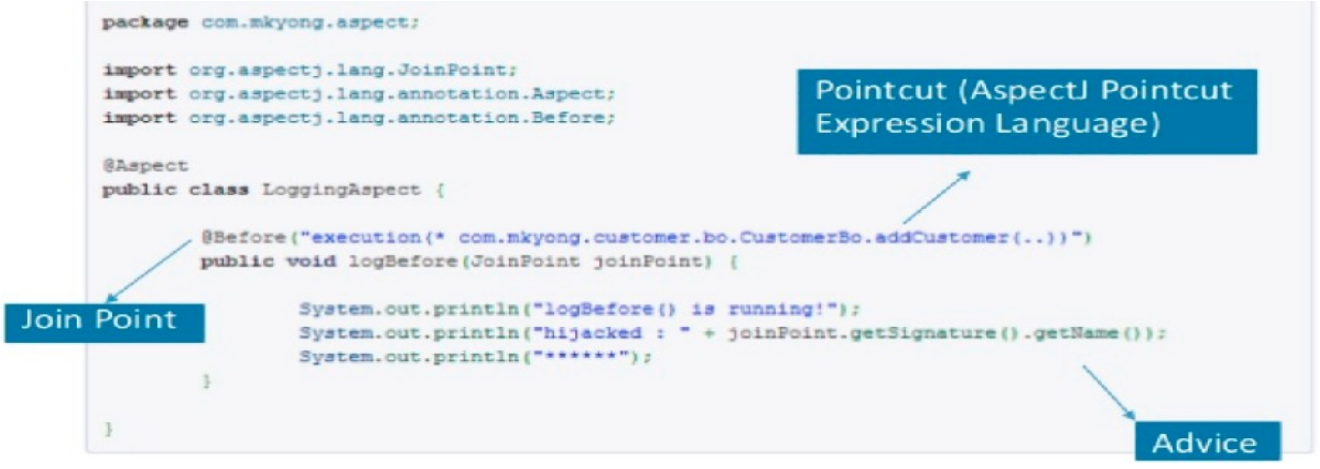
Aspect Oriented Programming (A.O.P)



- Do not mix non-Business Logic with Business Logic
- An easy way for adding behavior to existing code without modifying that code

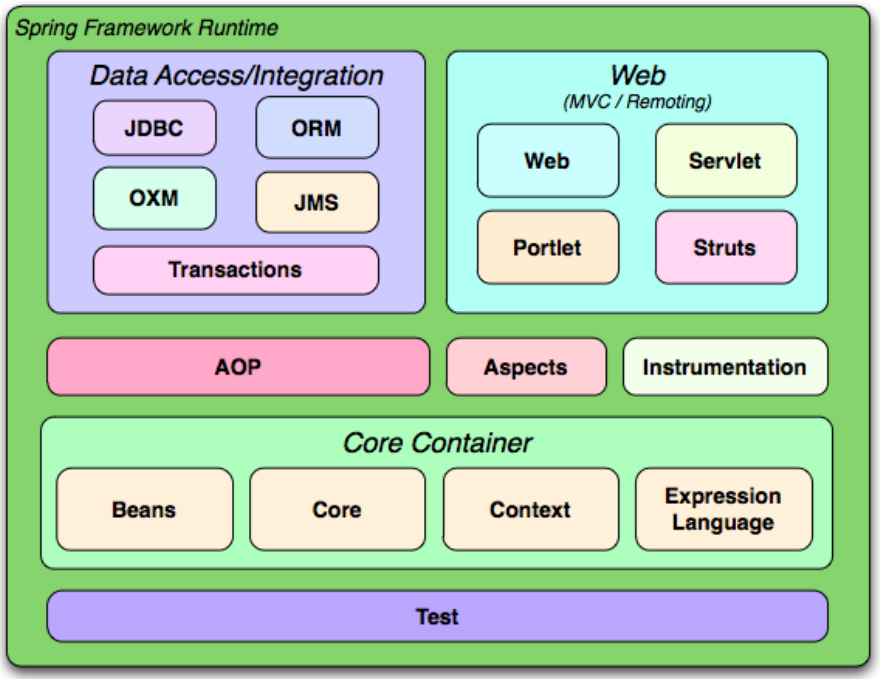
- Ingredients
  - Joinpoint : Point during the execution of a program, such as execution of a method or the handling of an exception
  - Pointcut : Predicate that helps match an Advice to be applied by an Aspect at a particular JoinPoint.
  - Advice : Action taken by an aspect at a particular Joinpoint.

A.O.P Example



- **@Before** – Run before the method execution
- **@After** – Run after the method returned a result
- **@AfterReturning** – Run after the method returned a result, intercept the returned result as well.
- **@Around** – Run around the method execution, combine all three advices above.
- **@AfterThrowing** – Run after the method throws an exception

Spring Architecture



The Spring Framework provides about 20 modules which can be used based on an application requirement.

Core Container

- Core Module
  - Provides the fundamental parts of the framework, including the IoC and Dependency Injection features
- Bean Module
  - Provides BeanFactory (container which instantiates, configures, and manages a number of beans), which is a sophisticated implementation of the factory pattern
- Context Module
  - Builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module. (Bean factory methods for accessing application components, publish events to registered listeners etc)
- Expression Language Module
  - Provides a powerful expression language for querying and manipulating an object graph at RunTime

Data Access/Integration

- JDBC Module
  - Provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding
- ORM Module
  - Provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis
- OXM Module
  - Provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream
- JMS Module (Java Messaging Service)
  - Contains features for producing and consuming messages
- Transaction Module
  - supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs

Web

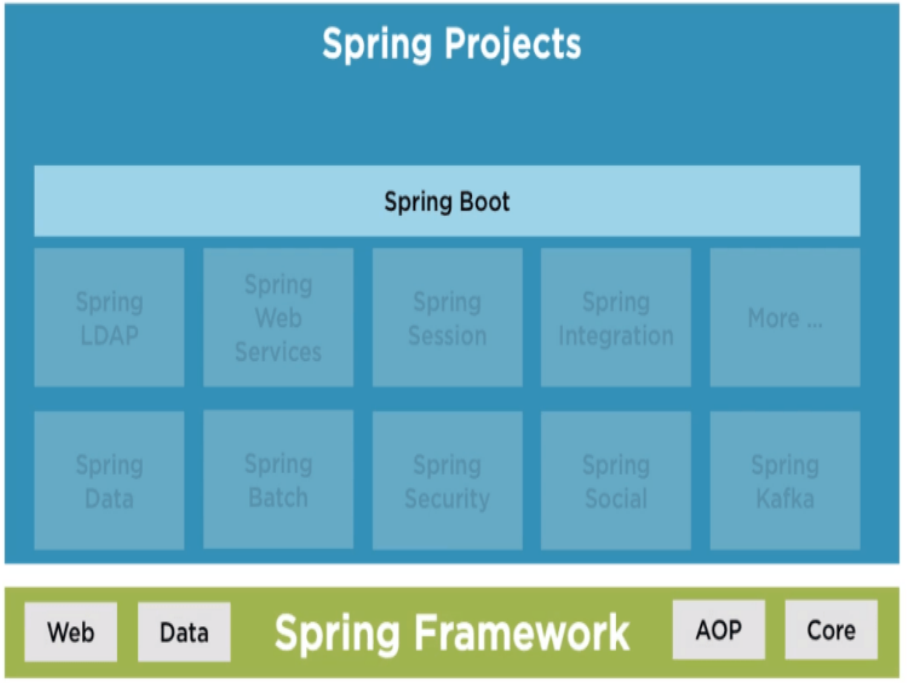
- Web Module
  - Provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context
- Web-MVC Module
  - Contains Spring's Model-View-Controller (MVC) implementation for web applications
- Web-Socket Module
  - Provides support for WebSocket-based, two-way communication between the client and the server in web applications
- Web-Portlet Module
  - Provides the MVC implementation to be used in a portlet (fragments) environment and mirrors the functionality of Web-Servlet module

Benefits

- Modularity
- Open Source
- A.O.P
- Reusable
- Extendable
- Simple Coding
- Easy Testing
- Security Framework

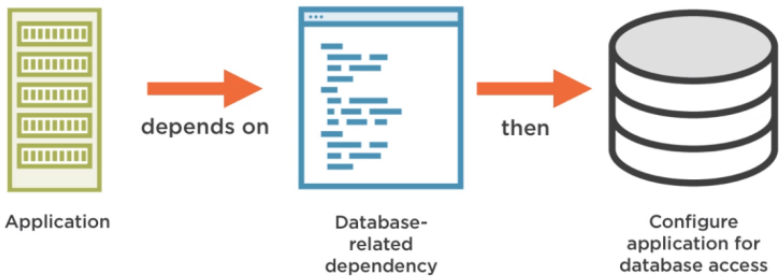
Spring Boot

- MAGIC :)
- Spring Boot is a project built on the top of the Spring framework. It provides a simpler and faster way to set up, configure, and run both simple and web-based applications



- In the Spring core framework, you need to configure all the things for yourself. Hence, you can have a lot of configuration files, such as XML descriptors. That's one out of the main problems that Spring Boot solves for you
- It smartly chooses your dependencies, auto-configures all the features you will want to use, and you can start your application with one click. Furthermore, it also simplifies the deployment process of your application

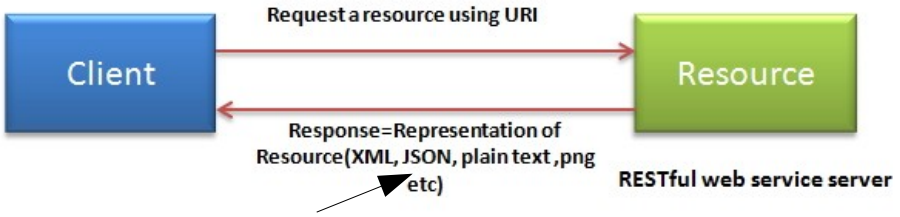
Configuration

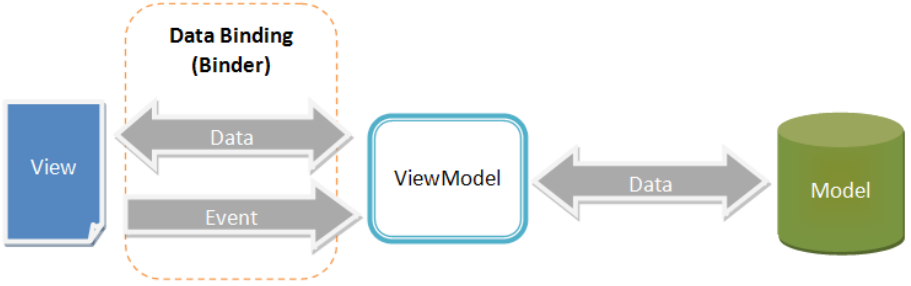


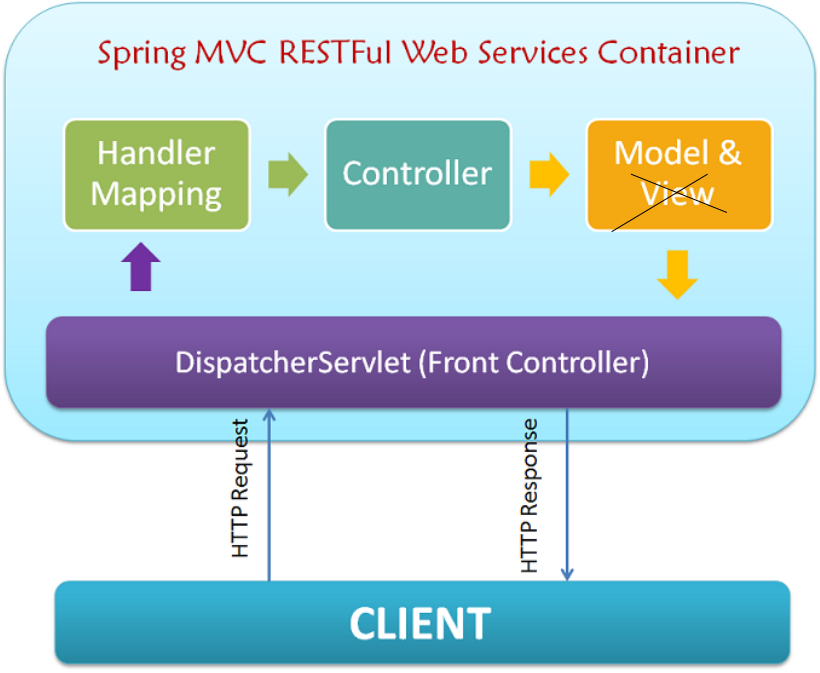

```
@EnableAutoConfiguration
public class DemoApplication {
    ...
}
```

- Add a dependency to the pom.xml/build.gradle, which relates to a database, the framework assumes that you probably would like to use a database
- Then, it auto-configures your application for database access
- Package your application
- Run it with some simple command like java -jar my-application.jar
- Spring Boot takes care of the rest by starting and configuring an embedded web server and deploys your application there

{Rest | MVVM |Spring MVC}







Step by Step example of Spring usage (POST/{GET(via ID)} of a Project)

```java
@Entity
@EntityListeners(AuditingEntityListener.class)
@Table(name = "Projects", schema = "scrumit")
public class Project {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

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

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

    @CreatedBy
    @Column(name = "owner", updatable = false)
    private Long owner;

    @CreatedDate
    @Column(name = "creation_time", updatable = false)
    private Timestamp creationTime;

    @ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinColumn(name = "team_id", nullable = false)
    private Team team;

    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL)
    private List<Sprint> sprints;

    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL)
    private List<Epic> epics;

    @JsonIgnore
    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL)
    private List<Story> stories;
```

Rest Controller for Project

```java
@RestController
@RequestMapping("/api/projects")
public class ProjectController {

    @Autowired
    private ProjectService projectService;

    @Autowired
    private TeamService teamService;

    @Autowired
    private NotificationService notificationService;

    private static final Logger logger = LoggerFactory.getLogger(ProjectController.class);

    @PostMapping
    @PreAuthorize("hasRole('USER')")
    public ResponseEntity<?> createProject(@Valid @RequestBody ProjectRequest projectRequest,
                                           @Valid @CurrentUser UserDetailsImpl currentUser) {
        Team team = teamService.createTeam(projectRequest.getTeamName(), currentUser);
        Project project = projectService.createProject(projectRequest, team);

        URI location = ServletUriComponentsBuilder
                .fromCurrentRequest().path("/{projectId}")
                .buildAndExpand(project.getId()).toUri();

        return ResponseEntity.created(location).body(new ApiResponse(true, "Project Created Successfully.", project));
    }
```

```java
@GetMapping("/{projectId}")
@PreAuthorize("hasRole('USER')")
public Project getProjectById(@PathVariable(value = "projectId") Long projectId,
                              @Valid @CurrentUser UserDetailsImpl currentUser) {
    return projectService.getProjectById(projectId, currentUser);
}
```

Service for Project

```java
@Service
public class ProjectService {

    @Autowired
    private ProjectRepository projectRepository;

    @Autowired
    private AuthorizationService authorizationService;

    @Autowired
    private ValidatePathService validatePathService;

    @Autowired
    private ValidatePageParametersService validatePageParametersService;

    private static final Logger logger = LoggerFactory.getLogger(ProjectService.class);

    public Project createProject(ProjectRequest projectRequest, Team team) {
        Project project = new Project();
        project.setName(projectRequest.getName());
        project.setDescription(projectRequest.getDescription());
        project.setTeam(team);

        return projectRepository.save(project);
    }
```

```java
// Returns a specific project.
public Project getProjectById(Long projectId, UserDetailsImpl currentUser) {
    Project project = validatePathService.validatePathAndGetProject(projectId);

    // Check if the user is authorized for this action.
    // Only members of a project's team can get its contents.
    authorizationService.isMemberOfProject(projectId, currentUser.getUsername());

    return project;
}
```

```java
// Checks if a given username is in the list of the project's team members.
public void isMemberOfProject(Long projectId, String username) {
    if (!projectRepository.findUsernamesOfProjectMembersById(projectId).contains(username)) {
        throw new NotAuthorizedException("Sorry, You're not authorized to perform this action. " +
                "You are not a member of the project's team.");
    }
}
```

Repository for the Project

```java
@Repository
public interface ProjectRepository extends PagingAndSortingRepository<Project, Long> {

    Optional<Project> findById(Long projectId);

    // Returns the id of a project's owner.
    @Query("select p.owner from Project p where p.id = :projectId")
    Long findOwnerByProjectId(@Param("projectId") Long projectId);

    // Returns a list of projects in which the currently logged in user
    // participates either as an owner or a developer.
    @Query("select tm.projects from User u inner join u.teams as tm where u.id = :userId")
    Page<Project> findProjectsByUserId(@Param("userId") Long userId, Pageable pageable);

    // Returns a list with the usernames of the members of a project.
    @Query("select ptm.username from Project p inner join p.team as pt inner join pt.members as ptm where p.id = :projectId")
    List<String> findUsernamesOfProjectMembersById(@Param("projectId") Long projectId);
}
```

Simple Example of Spring Security Config

```java
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }

    @Bean
    @Override
    public UserDetailsService userDetailsService() {
        UserDetails user =
             User.withDefaultPasswordEncoder()
                .username("user")
                .password("password")
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(user);
    }
}
```


Thank You!