PHASE 3 TOPICS

**SPRING CORE**

**20 March 2023 – DAY 1**

1. Spring is basically built on 3 core principles
   1. AOP – Aspect Oriented Programming  
      is used to provide common functionalities within your spring application context
   2. DI – dependency Injection  
      is used to inject any dependencies that your application may have
   3. Abstraction layer over the existing API’s
2. Create a spring quickstart maven project
   1. Add dependency in pom.xml file spring-context
3. Also create a core java project and
   1. add dependencies downloading from  
      <https://repo.spring.io/ui/native/release/org/springframework/spring/>
   2. Add dependencies in the build path
4. Created our java classes -> every java class in spring is called as a bean
5. Create spring.xml => spring configuration file to add spring related configurations
   1. If maven create within src/main/resources folder
   2. If normal java project then add in src folder
6. Dependency injection
   1. XML based approach
      1. Configure java classes using <bean class=”fully qualified clas name”> . id is optional but it is mandatory when there is ambiguity of the beans i.e more than 1 instance of same bean is configured
      2. Constructor injection  
         <constructor-arg>
      3. Property / setter injection  
         <property>
      4. Scoping in spring
         1. Singleton : which is default as only 1 instance per application is created
         2. Prototype : which creates different instances everytime getBean is called
         3. Request and session used in web applicaitons
   2. Annotation based approach
      1. @Component => is added on the class name to tell spring that this bean or class needs to be loaded and instantiated.
      2. @Value => used for injecting primitive type dependencies
      3. @Autowired => used for injecting reference type dependencies
      4. @Configuration => used as an alternative for xml based configuration. It is used on the class
      5. @ComponentScan => will tell spring to look up for all the beans with spring annotations [ Component, Repository, Service, Controller, RestController, Bean ] and load them. It default looks in the current / root package and the sub packages
      6. @Primary => can be used on the class to be primary bean injected for ambiguity
      7. @Qualifier => can be used to override the @Primary
7. Load the configuration file , xml  
   ClassPathXmlApplicaitonContext(“spring.xml”)
8. Load the configuration file , annotation  
   AnnotationConfigApplicationContext(AppConfig.class)
9. BeanFactory vs ApplicationContext
10. Event Handling in spring
    1. Create an event of type ApplicationEvent
    2. Create a listener using @EventListener
    3. Create a Publisher using ApplicationPublisherEvent

**21 March 2023 – DAY 2**

1. Collections Mapping
   1. @Autowired
   2. @Bean : can be used on any method within the configuration file.

**SPRING JDBC**

1. Create a quickstart maven project
2. Added context, jdbc and mysql dependencies
3. Create beans for which database table and columns will be mapped
4. Database connection parameters:
   1. Provides an abstraction layer over JDBC API
   2. JdbcTemplate provides with skeleton to connect and execute queries for database
      1. Reference of type DataSource : DriverManagerDataSource [ connection parameters ]
      2. Inject JdbcTemplate with DataSource reference
5. Inject JdbcTemplate in any DAO layer which needs to execute queries for DB
6. Insert / update / delete => jdbcTemplate.update(sql, …)
7. Select by id => queryForObject(sql, RowMapper, ..)
8. Select all => query (sql, RowMapper)
9. Configuration for DB connection parameters can be externalize as follows:
   1. Create properties file under src/main/resources folder and add the db connection parameters as key=value pairs
   2. Use @PropertySource to load the properties file
10. **PASSWORDS SHOULD BE ENCRYPTED - JASYPT**  
    <https://www.baeldung.com/spring-boot-jasypt>

**22 March 2023 – DAY 3**

**SPRING MVC**

1. When the request comes to a spring web application
   1. Goes to Front Controller -> DispatcherServlet [ DS ]
   2. DS delegates the request to BeanNameUrlHandlerMapping : that provide with mapping of the url with the respective bean method
   3. The bean is loaded, method is processed and response along with viewname is sent to DS
   4. DS delegates the request to ViewResolver that looks up for the view name configured as prefix and suffix. If success it returns the view content to DS
   5. Then DS sends the response back to the client
2. Create spring MVC Application
   1. Create a maven webapp project
   2. Add dependencies: context. Web-mvc, servlet-api, jsp-api, Lombok, jdbc, mysql
   3. Configuring build path for maven and java version
   4. Create the configuration classes
      1. Load the dispatcher servlet : implements WebApplicationInitializer interface
      2. To configure for beans : Annotated with @Configuration and @ConponentScan. Provide the DB or ViewResolver configurations
   5. Create a class as follows:
      1. annotated with @Controller annotation
      2. Add methods with different mapping like @GetMapping , @PostMapping for diff HTTP requests
      3. Controller return the viewname
      4. To pass data from controller to the view, use Map [ Model or ModelAndView ]
3. SPRING MVC PROJECT STRUCTURE
   1. config : it has the spring configuration files
   2. com.vodafone :
      1. HelloController : login, logout, register and admin
      2. DashboardController : dashboard
   3. com.vodafone.database :
      1. CustomerDb : for all customer related CRUD operations
      2. LoginDB : to validate the user
   4. com.vodafone.service :
      1. LoginService : that calls validate method of LoginDB
   5. com.vodafone.entity : Customer class which is mapped with database table
   6. com.vodafone.dto : CustomerDTO class to return only specific fields of Customer
   7. src/main/webapp/WEB-INF/views folder: which consists of all the jsp files

**23 March 2023 – DAY 4**

**SPRING BOOT**

**https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods**

1. opinionated framework: it generates opinions based on the libraries in the build path
2. it is not replacing spring
3. it provides a mechanism to create production ready spring application with no or less configuration
4. It takes care of vanilla code / many of the common functionalities required in a particular application
5. It provides with various starter-XXX dependencies to add different modules to integrate different functionalities
6. To create a spring boot project :
   1. Go to start.spring.io or Use STS
      1. Java version
      2. Maven or gradle
      3. Boot version >= 3 requires java 17 onwards
      4. Groupid: artifact id: project name
      5. Add dependencies : mysql, jdbc, data-jpa, Lombok
   2. @SpringBootApplication :
      1. @SpringBootConfiguration
      2. @ComponentScan
      3. @EnableAutoConfiguration
   3. VVIMP to follow the package structure
   4. Database:
      1. Added DB connection parameters : in application.properties file
      2. Autowire the JdbcTemplate and DONE
      3. Create an interface that extends crudRepository interface which provides with all basic CRUD operations
      4. Can create custom queries as well

**24 March 2023 – DAY 5**

**SPRING BOOT MVC**

1. Steps to configure MVC in spring boot
   1. No need to configure dispatcher servlet as it is configured by spring boot based on the web-mvc dependency in the build path
   2. View Resolver : suffix and prefix in application.properties file
   3. Add tomcat-embed-jasper dependency to parse jsp files
   4. Follow the package structure
   5. Then create controller and add jsp files within the webapp folder

**SPRING BOOT REST**

1. Create a class and annotate with @RestController annotation
2. @RestController => @Controller + @ResponseBody
3. REST API leverages the existing HTTP protocol
   1. HTTP methods
   2. HTTP status codes
   3. HTTP headers
4. @RequestMapping : generalized annotation for GET POST PUT DELETE etc
5. Specific annotations for diff HTTP methods
   1. @GetMapping
   2. @PostMapping
   3. @PutMapping
   4. @DeleteMapping
   5. @PathVariable => http://localhost:8081/rest/{id}/comment/{name}  
      <http://localhost:8081/rest/689979/comment/shalini>
   6. @RequestParam => <http://localhost:8081/rest>?id=101 [ also be used to access form parameters ]
   7. @MatrixParam => [http://localhost:8081/rest/{name};id=101](http://localhost:8081/rest/%7bname%7d;id=101)

**27 March 2023 – DAY 6**

1. ResponseEntity is used to send the response back to the user as it gives flexibility to send different responses along with metadata like status codes etc
2. Exception handling in spring boot REST API
   1. @ControllerAdvice annotation on the class (global exception handler ) for generalized exceptions ,means common exceptions across your applications
   2. @ExceptionHandler annotation is used on the respective methods that handles the specific exceptions and they can be defined either within the global exception handler or within the restcontroller class as well.
3. @RequestBody : Is used when the REST API is consuming json or xml data
4. Content negotiation : consumes and produces along with HTTP headers like content-type and accept. Add jackson-xml libraries for XML content

**SPRING BOOT SWAGGER**

1. Swagger library:
   1. Used to document the REST API for the consumers to understand how to make a request, input and output parameters
   2. Add opendoc-swagger librbary
   3. [http://localhost:<portno>/swagger-ui.html](http://localhost:%3cportno%3e/swagger-ui.html)
   4. Different annotations can be used for adding more information about your REST API

**SPRING BOOT SECURITY**

1. Add spring-boot-starter-security as a dependency
2. Spring security generates a dynamic password, default username is user and by default all the urls are now protected
3. It also exposes login and logout pages
4. To provide custom authentication
   1. Create a class that extends WebSecurityConfigurerAdapter and annotate with @Configuration and @EnableWebSecurity
   2. Then override a method configure(AuthenticationManagerBuilder auth) to provide authencticaiton credentials
   3. Override configure(WebSecurity web) to provide custom security