**Ինչ է Spring MVC – ն (v6․0․4 02․02․2023 դրությամբ)**

Spring Framework – ի կոմպոնենտ է որը թույլ է տալիս ստեղծել web app – ներ Java – ով։

App – ները ստեղծվում են Modev-View-Controller կոնցեպտի հիման վրա։

MVC – ն պատտերն է որը ունի 3 կոմպոնենտ՝

Model – տվյալների հետ աշխատելու լոգիկա

View – user -ի համար տեսանելի հատվածի լոգիկա (օր HTML), ինտերֆեյս

Controller – նավիգացիայի լոգիկա, request – ների մշակում

Spring MVC- ում կիրառելի են Spring Core – ի բոլոր հնարավորությունները։

Spring MVC – ն կազմված է Java կլասսներից և դրանցում ակտիվ օգտագործվող անոտացիաներից, HTML – ներից JS կոդի կիրառմամբ, կոնֆիգուրացիոն ֆայլերից Java/Anotation/Xml

Http request – ի և Controller – ների միջև կան նաև DispatcherServlet որը հանդիսանում է Spring MVC app – ի մուտքի կետը և որը Spring տրամադրում է ավտոմատ։ Request – ը սերվերին փոխանցվելուց հետո փոխանցվում է DispatcherServleti – ի որը այն ուղարկում է ճիշտ controller – ին։ Նշված մոդելը կոչվում է առաջնային controller – ի շաբլոն որտեղ կենտրոնական սերվլետը DispatcherServlet – ը տրամադրում է ընդհանուր ալգորիթմ request – ների մշակման համար իսկ ավելի նեղ մշակումը կատարում են առանձին կոմպոնոնտները (controller - ները)։ Սերվլետների նման DispatcherServlet – ի համար նույնպես պետք է mapping` Java – ի կամ web.xml – ի միջոցով:

AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext();

context.register(AppConfig.class);

// Create and register the DispatcherServlet

DispatcherServlet servlet = new DispatcherServlet(context);

ServletRegistration.Dynamic registration = servletContext.addServlet("app", servlet);

registration.setLoadOnStartup(1);

registration.addMapping("/app/\*");

web.xml – ի դեպքում՝

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/app-context.xml</param-value>

</context-param>

<servlet>

<servlet-name>app</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value></param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>app</servlet-name>

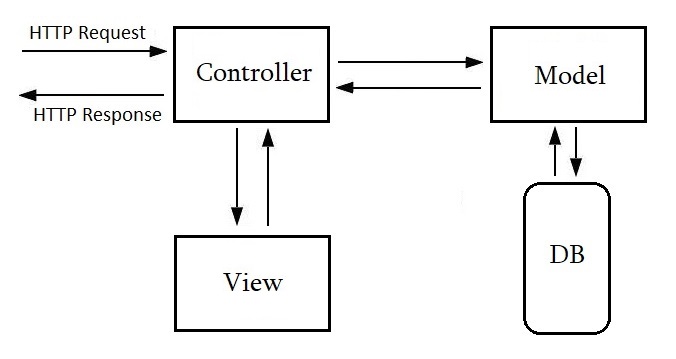
<url-pattern>/app/\*</url-pattern>

</servlet-mapping>

Controller- ը մշակում է request – ը, տվյալներով փոխանակվում model – ի հետ, user – ին ցույց է տալիս ճիշտ view, ըստ անհրաժեշտության վերահացեագրում է դեպի այլ ռեսուրսներ։

Model – ը պահում է իր մեջ տվյալներ, աշխատում է DB – ի հետ, controller – ին է տրամադրում տվյալներ։

View – ն տվյալները ստանում է controller – ից և արտապատկերում է դրանք բրաուզերում, HTML – ի դինամիկ գեներացման համար կարող են օգտագործվել շաբլոնիզաորներ օր՝ Thymeleaf, Freemarker, Velocity:

****

**@Controller** – ժառանգում է @Component անոտացիայից, այսինքն @ComponentScan – ը Controller – ի հետ աշխատում է նույն մեթոդով, եթե գտնում է անոտացիան ստեղծում է դրա bean – ը։ @Controller կլասսը կարող է իր մեջ պարունակել մեթոդներ ցանկացած անունով, որոնք համապատասխանում են 1 url – ի և սովորաբար վերադարձնում են String (պարտադիր չէ)։ Թե որ մեթոդը պետք է մշակի կոնկրետ request – ը որոշվում է mapping – ի միջոցով, որը սահմանվում մեթոդի վրա է HTTP request մեթոդներին համապատասխան անոտացիաներով՝ **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping:** Երբեմն կարելի է հանդիպել հին սիտաքսին՝ **@RequestMapping(value = “/url” method = RequestMethod.GET):**

@RequestMapping – ը կարող է լինել նաև կլասսի վրա՝

@Controller

@RequestMapping(“/someUrl”)

public class MyController {}

Այս դեպքում մնացած բոլոր mapping մեթոդներըի հասցեները պետք է իրենց մեջ պարունակեն RequestMapping – ում նշված արժեքը։

Mapping – ը կապում է controller – ը հասցեի հետ որին կարելի է դիմել։

Mapping մեթոդները String վերադարձնելիս default – ով փորձում են գտնել համապատասխան view – ն։ Օր Thymeleaf շաբլոնիզատորից օգտվելու համար պետք է հայտարարել համապատասխան bean – ները և web.xml – ում պետք է նշվի նաև DispatcherServlet – ի mapping – ը և այն xml – ը որից պետք է վերցվեն մնացած config – ները (bean - երը)։ Նույն գործողությունը առանց xml և web.xml օգտագործելու կարելի կատարել միայն Java կոդով։ Այս դեպքում պետք է ժառանգել AbstractAnnotationConfigDispatcherServletInitializer աբստրակտ կլասսից որի մեթոդների միջ համապատասխան փոփոխություններից հետո այն կփոխարինի web.xml – ին։

Իսկ ուղղակի String ստանալու համար Mapping մեթոդին պետք է ավելացվի @RequestBody անոտացիան։

Get – ի միջոցով փոխանցված պարրամետրերի մշակումը Controller – ում կարող է կատարվել HttpServetRequest օբյեկտի getParameter(“name”) մեթոդի կամ @RequestParam(“name”) անոտացիայի միջոցով`

@RequestParam(“name”) String name

Այս դեպքում Spring – ը ավտոմատ կվերցնի պարամետրը url – ից և կվերագրի փոփոխականին։ HttpServetRequest օբյեկտ օգտագործելիս եթե նշված անունով պարամետրը չի փոխանցվում ապա կունենանք null իսկ @RequestParam անոտացիայի դեպքում` 400 error: Եթե չենք ուզում նման վարքագիծ ապա կարելի է՝ @RequestParam(value=”name”, required = false):

View – ին (Html - ին) կարելի է տվյալներ փոխանցել նաև Model օբյեկտի addAttribute(key, value) միջոցով: շաբլոնիզատորները հնարավորություն են տալիս Html – ում expression language – ի միջոցով ստանալ ատրիբուտները։

**CRUD, REST, DAO**

**CRUD (**create, read, update, delete) սրանք տվյալների հետ աշխատելու բազային ֆունկցիաներն են։ Այսինքն CRUD application – ը հնարավորություն է տալիս ստեղծել, կարդալ, փոխել, ջնջել տվյալներ DB – ում։ Crud application – ները որպես կանոն նաև web application – ներ են, այսինքն նշված գործողությունները իրականացվում են HTTP – ի միջոցով՝ create – POST, read – GET, update – PATCH, delete - DELETE։

CRUD app – ների համար կան convension – ներ՝

GET /entities – բոլորը ստանալու համար (read)

POST /entities – նորը ստեղծելու համար (create)

GET /entities/new – նորը ստեղծելու HTML ֆորման ստանալու համար

GET /entities/:id/edit – կոնկրետ մեկի փոփոխման ֆորման ստանալուհամար

GET /entities/:id – կոնկրետ մեկը ստանալու համար

PATCH /entities/:id – կոնկրետ մեկը փոփոխելու համար

DELETE /entities/:id – կոնկրետ մեկը ջնջելու համար

Այս ամենը նկարագրում է REST – ը, որը իրենից ներկայացնում է web ծրագրավորման պատտերն։ REST – ի միջոցով նկարագրվում է թե ինչպես պետք է փոխազդեն կլիենտը և սերվերը՝ HTTP – ի միջոցով։ Փոխազդեցությունը որպես կանոն տեղի է ունենում վերոհիշյալ 4 գործողությունների միջոցով։

CRUD – ը հիմնական մեթոդներն են որոնք օգտագործվում են տվյալների հետ աշխատելիս իսկ REST – ը պատտերն է որը նկարագրում է թե ինչ HTTP մեթոդներով են փոխազդելու սերվերը և կլիենտը այդ գործողությունները կատարելու համար։

Այսինքն CRUD – ը պարտադիր չէ որ լինի web app այն կարող է լինելե պարզապես app որը աշխատում է տվյալների հետ, իսկ REST – ի դեպքում արդեն խոսքը վերաբերում է HTTP պրոտոկոլով աշխատանքին՝ այսինքն web app – ին։

Նպատակահարման չէ DB – ի հետ աշխատանքի լոգիկան գրել ամբողղջությամբ model – հատվածում, այդ պատճառով օգտագործվում է DAO (data acsess object) պատտերնը։ DAO պատտերնը ենթադրում է կոնկրետ entity – ի համար DB – ի հետ աշխատանքի լոգիկան դուրս բերել առանձին կլասսի մեջ։ DAO – կլասսները սովորաբար պարունակում են sql կոդ։

@ModelAttribute անոտացիան կարող է օգտագործվել մեթոդների կամ արգումենտների հետ շաբլոնիզատորների հետ աշխատանքի ժամանակ։

Մեթոդի հետ այս անոտացիան նշանակում է որ model տեսակի օբյեկտի մեջ պետք է ներդրվի անոտացիայի արգումենտում նշված key – ով այն արժեքը որը վերադարձնում է անոտավորված մեթոդը։

Մեթոդի արգումենտի հետ կարող է կիրառվել օրինակ ֆորմաներից ստացով տվյալների հիման վրա օբյեկտ ստեղծելիս։

Վալիդացիաների համար կարող է կիրառվել hibernet validator dependency – ն։ Որը string – երի համար ունի օր` @NotEmpty(message = “error message”), @Size(min = 2, max = 30, message = “error message”), @Email(message = “error”)։ Թվերի համար օր՝ @Min(value = 0, message = “error message”):

Այլ վալիդացիաներ՝ <https://alexkosarev.name/2018/07/30/bean-validation-api/>

Սրանից հետո @Valid անոտացիան դրվում է համապատասխան օբյեկտի հետ արգումենտում, իսկ error – ը գրվում է BindingResult օբյեկտի մեջ (spring validation – ի նման)։

**JdbcTemplate**

JdbcTemplate – ը wrap է անում JDBC Api – ը և հեշտացնում է աշխատանքը DB – ի հետ։ Jdbc Api – ի խնդիրներից կարելի է նշել մեծ ծավալի կոդը (connection, statement, resultset), կոդի կրկնությունը (օր preparedStatement – ում անընդհատ set անելը), SQLException – ը որը քիչ ինֆորմատիվ է throw է արվում ցանկացած խնդրի դեպքում և պետք է այն միշտ catch անել։

Template – ով աշխատելու համար պետք է spring-jdbc dependency – ն։ Օգտագործելու համար պետք է ստեղծել դրա bean – ը, որի համար անհրաժեշտ է նաև ստեղծել DataSource bean: Վեջինի միջոցով նշվում են DB – ի config -ները՝ setDriverClassName, setUrl, setUsername, setPassword։ Դրանից հետո արդեն ստեղծել JdbcTemplate bean – ը DataSource – ը փոխանցելով որպես արգումենտ։ JdbcTemplate – ի (կախված sqlQuery – ի տեսակից) query կամ update մեթոդին փոխանցում է sql query – ն և որպես 2-րդ արգումենտ այսպես կօչված row mapper որը տվյալների բազայում աղյուսակի պարունակությունը վերագրում է օբյեկտին։ Row mapper – ը կլասս է որը իմպլեմենտացնում է RowMapper ինտերֆեյսը և override անում mapRow(ResultSet rs, int i) մեթոդը (generic է, նշվում է աղյուսակին համապատասխան մոդելի օբյեկտը, Comparator – ի նման): Նշված մեթոդից պետք է վերցնել resultSet օբյեկտը և այն վերագրել մոդել կլասսին։

JdbcTemplate – ը ավտոմատ օգտագործում է prepared statement: Պարրամետրերը փոխանցվում են sqlQuery – ին հաջորդող արգումենտով որը օբյեկտների մասիվ է օր՝

query(“SELECT \* FROM Person WHERE id=?”, new Object [] {id}, new MyRowMapper);

query – ն վերադարձնում է list:

update – ի դեպքում ի տարբերություն query - ի պարրամետրերի արգումենտը Object … args է։ Պատճառը այն է որ query – ի ժամանակ փոխանցվում է նաև 3 – րդ արգումենտ։

Եթե մոդել օբյեկտի դաշտերի անունները համընկնում են աղյուսակի սյուների անունների հետ ապա custom row mapper գրելու կարիք չկա, փոխարենը կարելի է օգտագործել spring – ի տրամադրած BeanPropertyRowMapper<>() օբյեկտը որին որպես արգումենտ փոխանցվում է մոդել կլասսը MyModelClass.class։