# Lab: Creating Custom-Made MVC Framework using Servlets

This **tutorial** provides step-by-step guidelines to build a **MVC framework** in Java, Servlets and Tomcat. The framework should implement **@Controller, @GetMapping, @PostMapping, @PathVariable, @RequestParam** annotations.

## Project Specification

Design and implement a **MVC framework** in Java, Servlets and Tomcat. Create 5 **annotations** with the following functionality:

* **@Controller**
  + Indicates that any class will be used as a controller and it will contain mapping actions
* **@GetMapping**
  + Provides a route to any method within a controller for **get requests**
* **@PostMapping**
  + Provides a route to any method within a controller for **post requests**
* **@PathVariable**
  + A variable within a method action that indicates **dynamic path**
  + /edit/{id} – **{id}** will be **dynamic** and should be contained in the **annotated parameter**
* **@RequestParam** 
  + Indicates any **parameter** coming from a **post or get request**

## MVC Spring-Like Architecture



* **DispatcherServlet** receives the request.
* **DispatcherServlet** dispatches the task of selecting an appropriate controller to HandlerMapping. HandlerMapping selects the controller which is mapped to the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet.
* **DispatcherServlet** dispatches the task of executing of business logic of Controller to HandlerAdapter.
* **HandlerAdapter** calls the business logic process of Controller.
* Controller executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter.
* **DispatcherServlet** dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolver returns the View mapped to View name.
* **DispatcherServlet** dispatches the rendering process to returned View.
* View renders Model data and returns the response.

## Project Setup

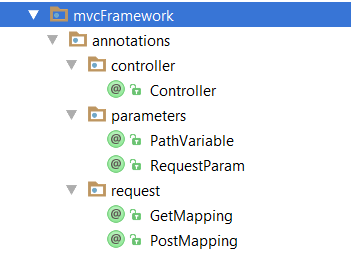
Create a new maven project called Bookhut. Add Java EE Web Application Framework support to the project. This is a recommended pom file:

|  |
| --- |
| **pom.xml** |
| <**properties**>  <**maven.compiler.source**>1.8</**maven.compiler.source**>  <**maven.compiler.target**>1.8</**maven.compiler.target**> </**properties**>  <**dependencies**>  <**dependency**>  <**groupId**>javax</**groupId**>  <**artifactId**>javaee-api</**artifactId**>  <**version**>7.0</**version**>  </**dependency**>  <**dependency**>  <**groupId**>jstl</**groupId**>  <**artifactId**>jstl</**artifactId**>  <**version**>1.2</**version**>  </**dependency**> </**dependencies**> |

## Create Annotations

Create 5 main annotations for the framework:

* **@Controller**
  + Applicable on classes only
* **@GetMapping**
  + Applicable on methods only
  + Has value element
* **@PostMapping**
  + Applicable on methods only
  + Has value element
* **@PathVariable**
  + Applicable on parameters only
  + Has value element
* **@RequestParam** 
  + Applicable on methods only
  + Has value element



## Create ControllerActionPair Class

Create a class called ControllerActionPair. It will be used as data traveler between different components.

It should have the **class, the method and the path variables** of the found **controller**:

* **Class controllerClass;**
* **Method method;**
* **Map<String, String> pathVariables;**

## Create Dispatcher Servlet

You would need a **dispatcher servlet** that will take care of the incoming **requests**:

|  |
| --- |
| **Dispatcher.java** |
| **public interface** Dispatcher {   ControllerActionPair dispatchRequest(HttpServletRequest request);   String dispatchAction(HttpServletRequest request, ControllerActionPair controllerActionPair); } |

|  |
| --- |
| **DispatcherServlet.java** |
| @WebServlet(**"/"**) **public class** DispatcherServlet **extends** HttpServlet **implements** Dispatcher {   **private** HandlerMapping **handlerMapping**;   **private** HandlerAction **handlerAction**;   **public** DispatcherServlet() {  **this**.**handlerMapping** = **new** HandlerMappingImpl();  **this**.**handlerAction** = **new** HandlerActionImpl();  }  @Override  **protected void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  *//****TODO Handle the requests***  }   @Override  **protected void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  *//****TODO Handle the requests***  }   @Override  **public** ControllerActionPair dispatchRequest(HttpServletRequest request) {  *//****TODO Send the request to the Handle Mapping***  }   @Override  **public** String dispatchAction(HttpServletRequest request, ControllerActionPair controllerActionPair) {  *//****TODO Send the ControllerActionPair to the Handle Action***  }   **private void** handleRequest(HttpServletRequest request, HttpServletResponse response){  *//****TODO Handle the request and return the controller view to the web client***  } } |

## Create Handler Mapping

The Handler Mapping is responsible to find the **controller** listening to the incoming **request route**.

|  |
| --- |
| **HandlerMapping.java** |
| **public interface** HandlerMapping {   ControllerActionPair findController(HttpServletRequest request) **throws** IOException, ClassNotFoundException, InstantiationException, IllegalAccessException; } |

|  |
| --- |
| **HandlerMappingImpl.java** |
| **public class** HandlerMappingImpl **implements** HandlerMapping {    @Override  **public** ControllerActionPair findController(HttpServletRequest request) **throws** IOException, ClassNotFoundException, InstantiationException, IllegalAccessException {  String urlPath = request.getRequestURI();  String projectPath = URLDecoder.*decode*(request.getServletContext().getResource(**"/WEB-INF/classes"**).getPath(), **"UTF-8"**);  *//****TODO Find all controllers***  **for** (Class controller : controllers) {  Method[] methods = controller.getDeclaredMethods();  **for** (Method method : methods) {  *//****TODO Find method path***  **if** (methodPath == **null**) {  **continue**;  }   *//****TODO If paths are matching return a new ActionPairController with the appropriate class, method and path variables***  }  }   **return null**;  }   **private void** addPathVariables(ControllerActionPair controllerActionPair, String urlPath, String methodPath) {  String[] uriTokens = urlPath.split(**"/"**);  String[] methodTokens = methodPath.split(**"/"**);  *//****TODO If there is path variable add it to the ControllerActionPair***  }    **private boolean** isPathMatching(String urlPath, String methodPath) {  **boolean** isPathMatching = **true**;  String[] uriTokens = urlPath.split(**"/"**);  String[] methodTokens = methodPath.split(**"/"**);  *//****TODO If the lengths are different return false***  *//****TODO If there is a path variable {some id} ignore and continue to check if the path is the same***  }   **private** String findMethodPath(HttpServletRequest request, Method method) **throws** IllegalAccessException, InstantiationException {  *//****TODO Find the method path***  }   **private** List<Class> findAllControllers(String projectDirectory) **throws** ClassNotFoundException, IOException {  List<Class> controllerClasses = **new** ArrayList<>();  File directory = **new** File(projectDirectory);  File[] files = directory.listFiles();  **for** (File file : files) {  **if** (file.isFile()) {  *//****TODO Check if the class for that file is not null***  *//****TODO Check if the class has @Controller***  } **else if** (file.isDirectory()) {  *//****TODO Recursively check all directories for classes***  }  }   **return** controllerClasses;  }   **private** Class getClass(File file) **throws** ClassNotFoundException {  String absolutePath = file.getAbsolutePath();  *//****TODO Find the correct regex***  Class currentClass = **null**;  **if** (matcher.find()) {  *//****TODO Replace / with .***  **if** (!className.endsWith(**"DispatcherServlet"**)) {  currentClass = Class.*forName*(className);  }  }   **return** currentClass;  } } |

## Create Handler Action

Handler Action will be responsible for the executing the controller action.

|  |
| --- |
| **HandlerAction.java** |
| **public interface** HandlerAction {   String executeControllerAction(HttpServletRequest request, ControllerActionPair controllerActionPair) **throws** InvocationTargetException, IllegalAccessException, InstantiationException, NoSuchMethodException; } |

|  |
| --- |
| **HandlerActionImpl.java** |
| **public class** HandlerActionImpl **implements** HandlerAction {  @Override  **public** String executeControllerAction(HttpServletRequest request, ControllerActionPair controllerActionPair) **throws** InvocationTargetException, IllegalAccessException, InstantiationException, NoSuchMethodException {  *//****TODO Get the controller and it respective method to execute***  **for** (Parameter parameter : parameters) {  **if**(parameter.isAnnotationPresent(PathVariable.**class**)){  *//****TODO Set the path variable value***  }   **if**(parameter.isAnnotationPresent(RequestParam.**class**)){  *//****TODO Set the request parameter value***  }   **if**(parameter.getType().isAssignableFrom(Model.**class**)){  *//****TODO Pass the model values to the view***  }   }   *//****TODO Finally, Invoke the method***  }   **private** <T> T getPathVariableValue(Parameter parameter, PathVariable pathVariableAnnotation, ControllerActionPair controllerActionPair) {  *//****TODO Find path variable value***  }   **private** <T> T getParameterValue(Parameter parameter, RequestParam requestParamAnnotationClass, HttpServletRequest request) **throws** IllegalAccessException, InstantiationException {  *//****TODO Find request parameter value***  }   **private** <T> T convertArgument(Parameter parameter, String pathVariable){  Object object = **null**;  *//****TODO Find the correct regex can receive different types of parameters***  } } |

## Create Model

The Model will carry key-value pairs of data that should be passed to the view

|  |
| --- |
| **Model.java** |
| **public class** Model {   **private** HttpServletRequest **request**;   **private** Map<String, Object> **attributes**;   **public** Model(HttpServletRequest request) {  *//****TODO Initialize fields*** }   **public void** addAttribute(String key, Object value){  *//****TODO Add the parameters and then pass them to the view***  }   **public** Map<String, Object> getAttributes() {  **return attributes**;  } } |

## Test Your Framework

If you are done with the framework create a **new project** and add it as a **dependency**.

|  |
| --- |
| **BeerController.java** |
| @Controller **public class** BeerController {   @GetMapping(**"/beer"**)  **public** String getBeer(){  **return "beer"**;  }   @GetMapping(**"/beer/{id}"**)  **public** String getBeerId(@PathVariable(**"id"**) **int** id, Model model){  System.***out***.println(id);  model.addAttribute(**"id"**, id);  **return "beer"**;  }   @PostMapping(**"/beer"**)  **public** String postBeerBrand(@RequestParam(**"brand"**) String brand){  System.***out***.println(brand);  **return "redirect:/beer"**;  } } |

|  |
| --- |
| **beer.jsp** |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**>  <**head**>  <**title**>Beer</**title**>  </**head**>  <**body**>  Beer Stuff:  Id: **${**id**}** <**form method="post"**>  <**input type="text" placeholder="Enter a beer brand" name="brand"**>  <**input type="submit" value="Submit"**>  </**form**>  </**body**> </**html**> |