Table of Contents (TOC)

Preface

This document describes two applications, MachineMonitorManagement and AspectDataValidation.

MachineMonitorManagement implements Spring Boot, Spring Data JPA ,Thymeleaf and Bootstrap. AspectDataValidation implements additionally data input validation and aspect oriented programming

The source code for these two applications can be found by clicking the following link :

https://github.com/fullstack13579-demo-site

By the way, if you are interested in in my Java competence click the following link :

https://bitbucket.org/code345house/demosite

Here you can find Java game I made in 2012. It consists of over 1000 lines of code.

Spring Boot

I have studied intensively Spring MVC and Spring Boot. I work with Spring Boot in these applications. Spring Boot has several advantages compared to Spring MVC.

fast set up
reduction of boilerplate code
configuration is implemented with annotations, not with XML. This is good because now configurations are close to actual code.
Depency management is implemented with starters in POM file
Beans are automatically initialized, configured and wired.

Spring Data JPA

Object relation mapping (ORM) and database handling is implemented using Spring Data JPA with Hibernate.

Spring Data JPA is abstraction that reduces boilerplate code. Hibernate is used as JPA provider.

Thymeleaf

JSPs were earlier mainly used as presentation components in Java based web applications.

Because JSPs have limitations, several template engines have been developed (Velocity,FreeMarker, Mustache, Thymeleaf). I work with Thymeleaf in these applications.

Thymeleaf template engine works in web based and non-web based environments. It integrates well with Spring. Therefore, it is good choice for MVC based applications.

The main disadvantage with Thymeleaf is slower speed in comparison to Velocity and FreeMarker.

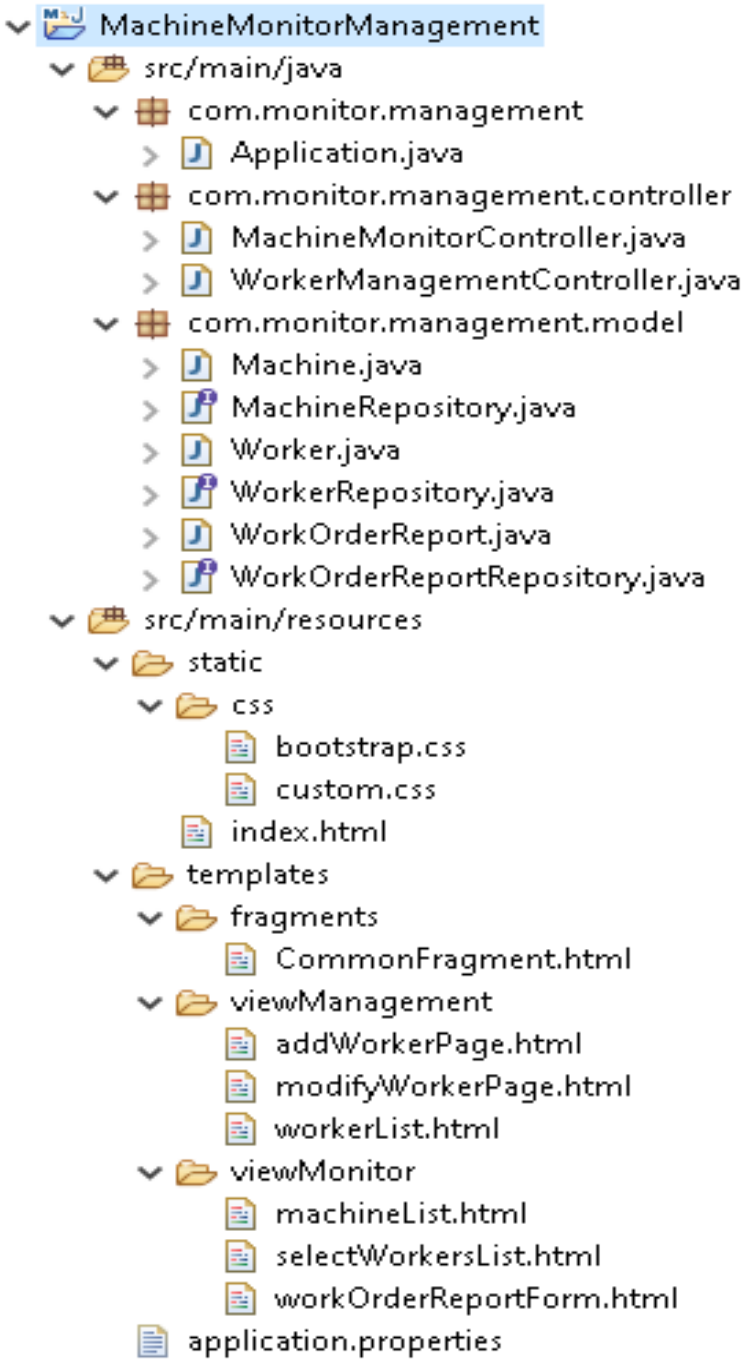
Bootstrap

Bootstrap is a CSS framework. It has wide array of components, but one of biggest use is responsive design.

Website scale now up and down according to the screen size of the device user is using.

MachineMonitorManagement

MachineMonitorManagement consists of two tasks, workforce management and machine monitoring.

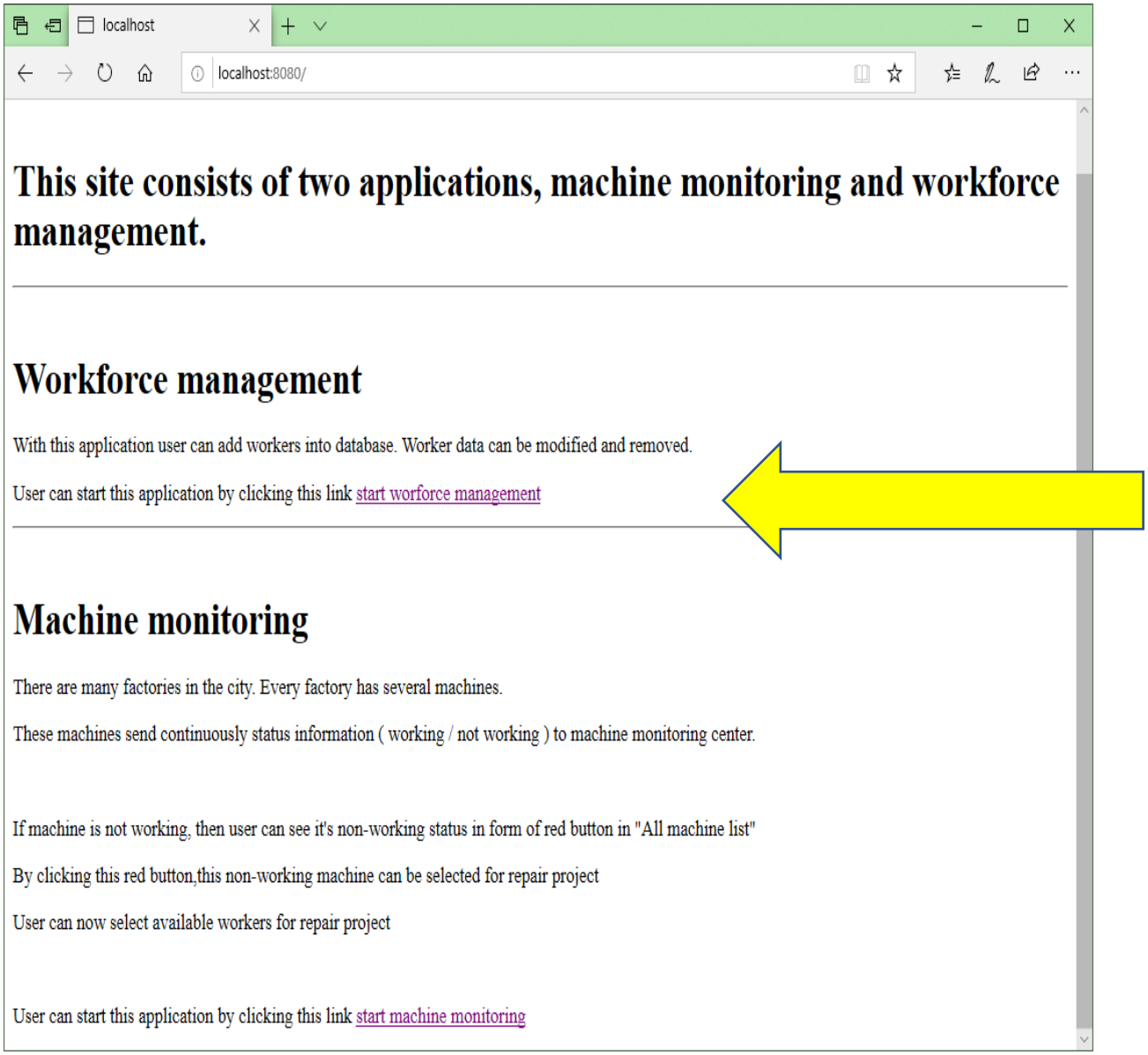


Workforce management

With this application user can add workers into database. User can remove and modify worker data. Program logic is implemented in WorkerManagementController.java.

WorkerManagementController is started by writing endpoint: localhost:8080 in browser. Index.html is now displayed on the screen.

Workforce management is then started by clicking text link : start workforce management

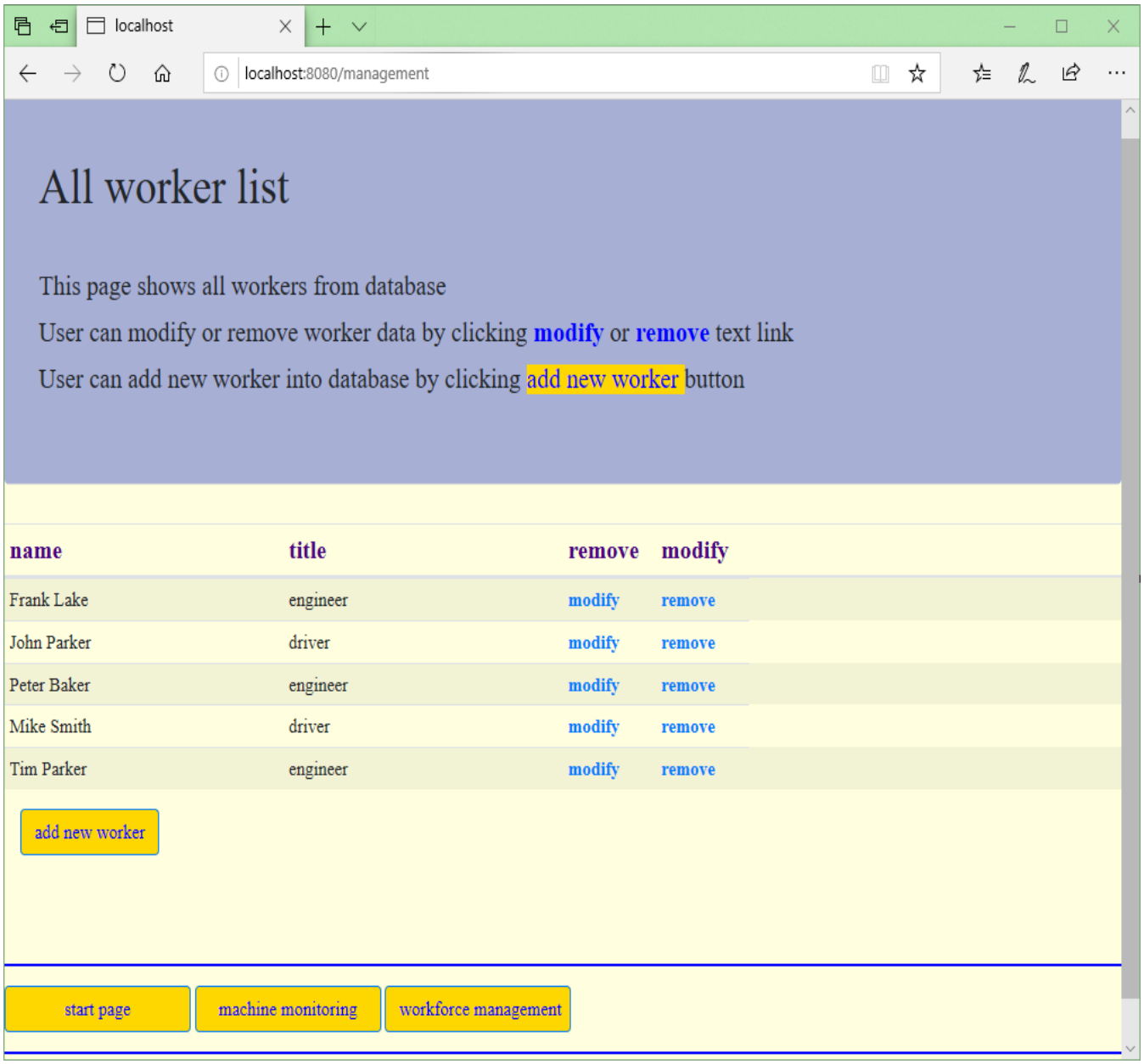


View component workerList.html is now displayed.

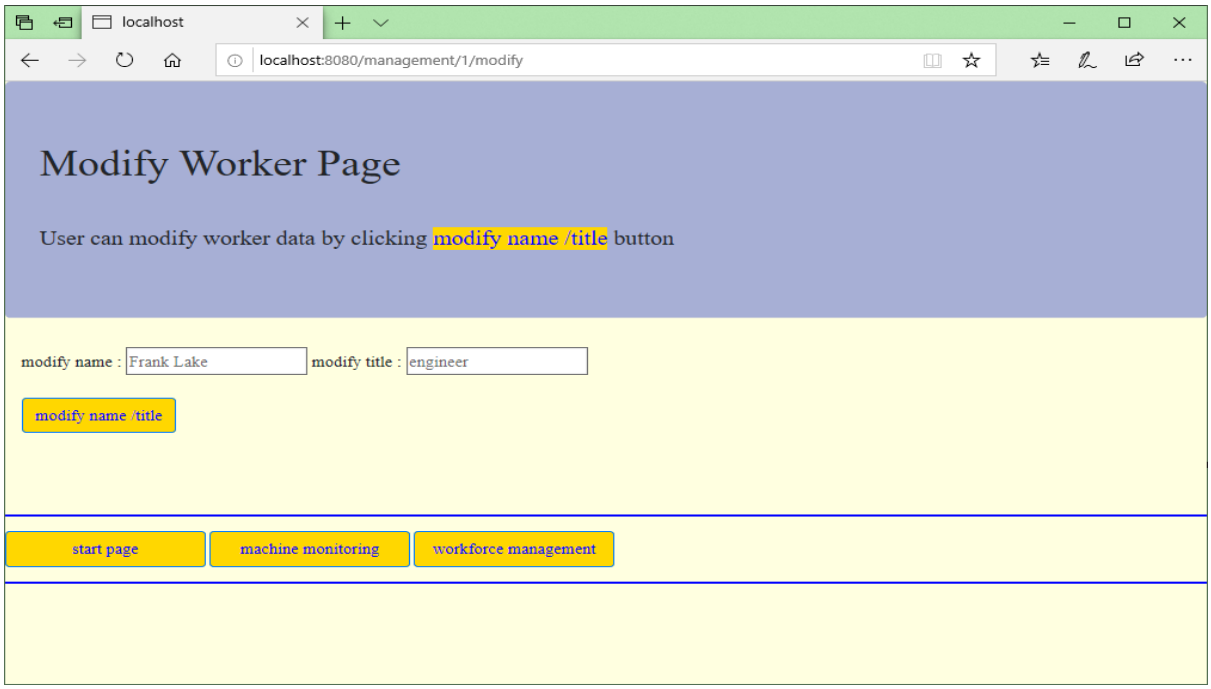
User can add new worker or modify/remove existing worker data.

Worker data is loaded into H2 database with help of run method from CommandLineRunner interface in Application.java file.

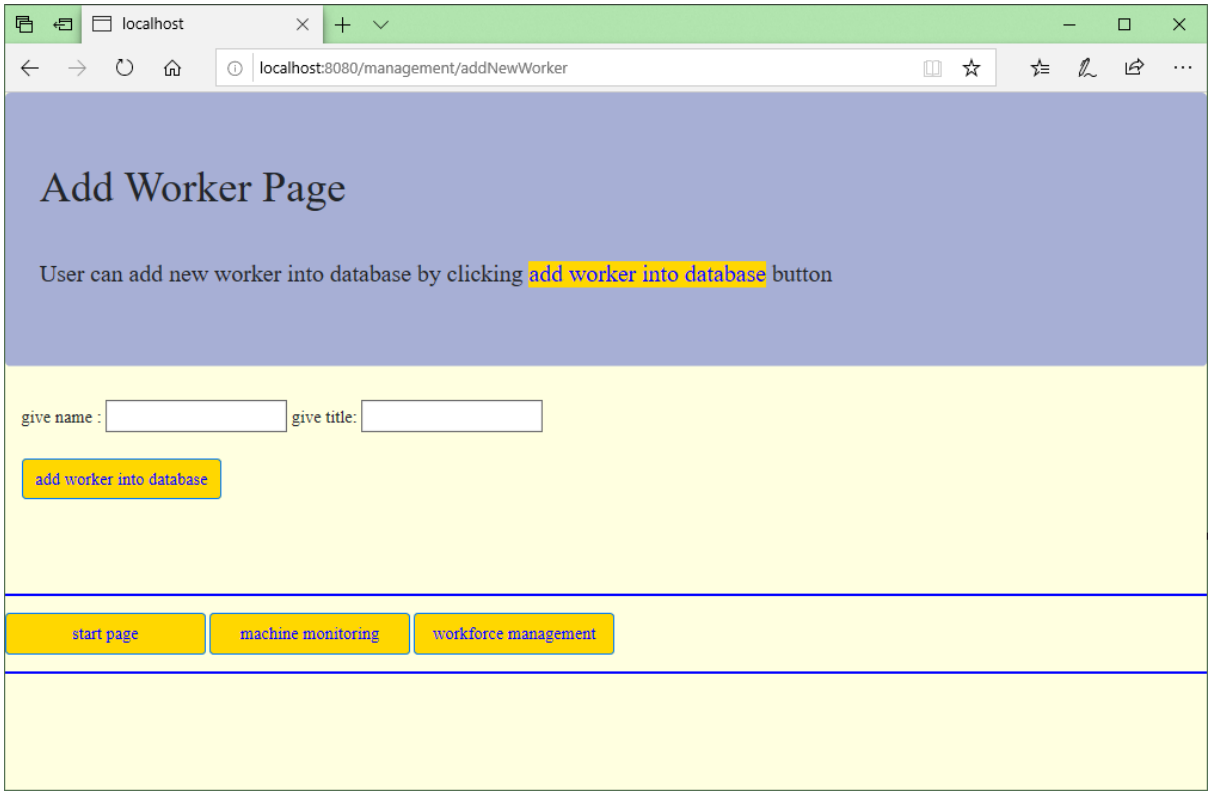
This run method is executed after application context is loaded, but before Spring Application run method completed. Now it is possible to check if beans/values exist or load data before application begins to run.



By clicking **modify** text link modifyWorkerPage.html is displayed.



by clicking add new worker button addWorkerPage.html is displayed



Machine monitoring

There are many factories in the city. Every factory has several machines. These machines send continuously status information on machine condition (working / not working) to machine monitoring centre.

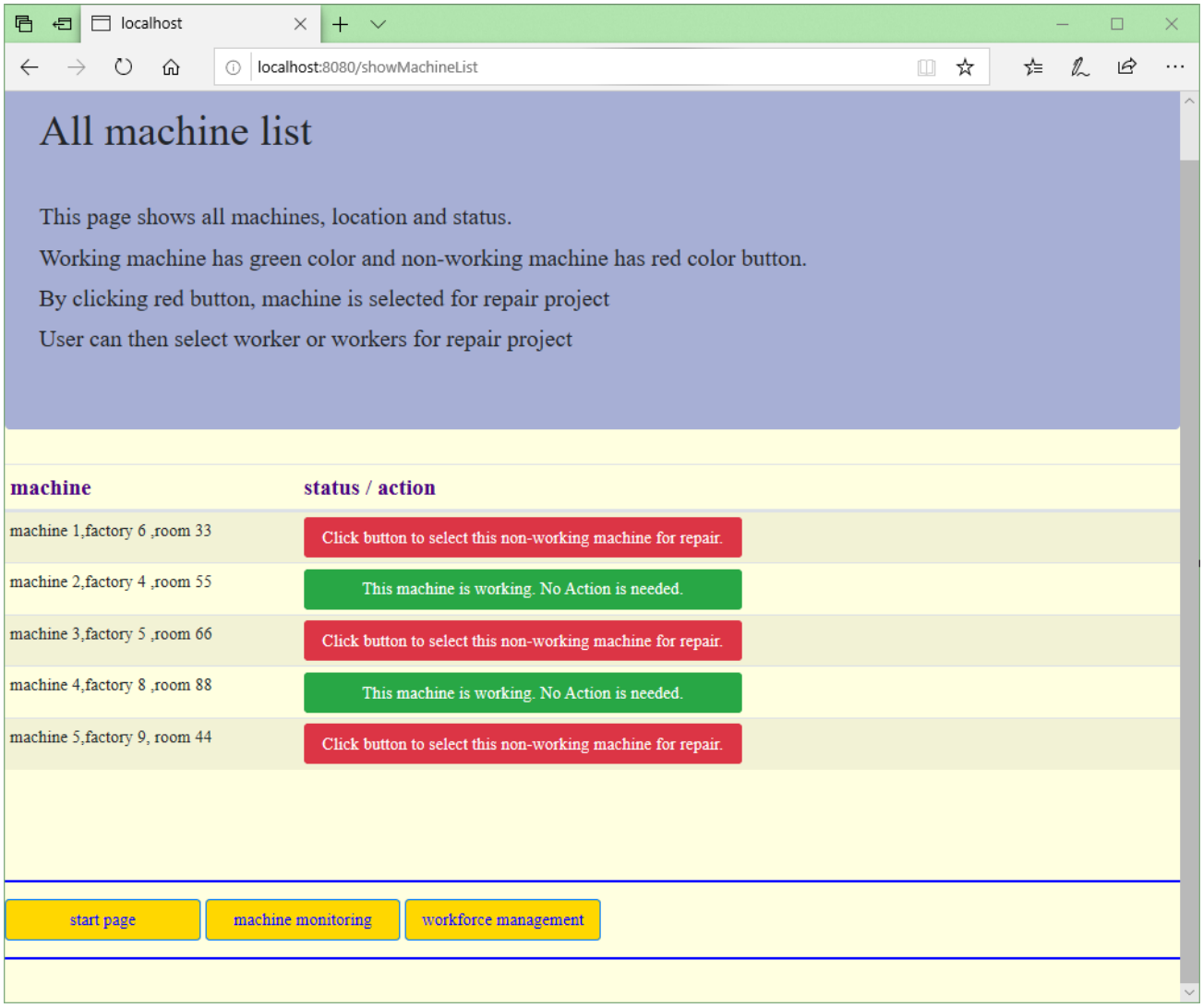
If machine is not working, then user in machine monitoring centre can see its non-working status in form of red button in machineList.html. By clicking this red button, this non-working machine can be selected for repair project.

User can now select available workers for repair project in selectWorkersList.html. WorkOrderReport is created and sent into database. WorkOrderReport is then displayed in workOrderReportForm.html.

Program logic is implemented in MachineMonitorController.java.

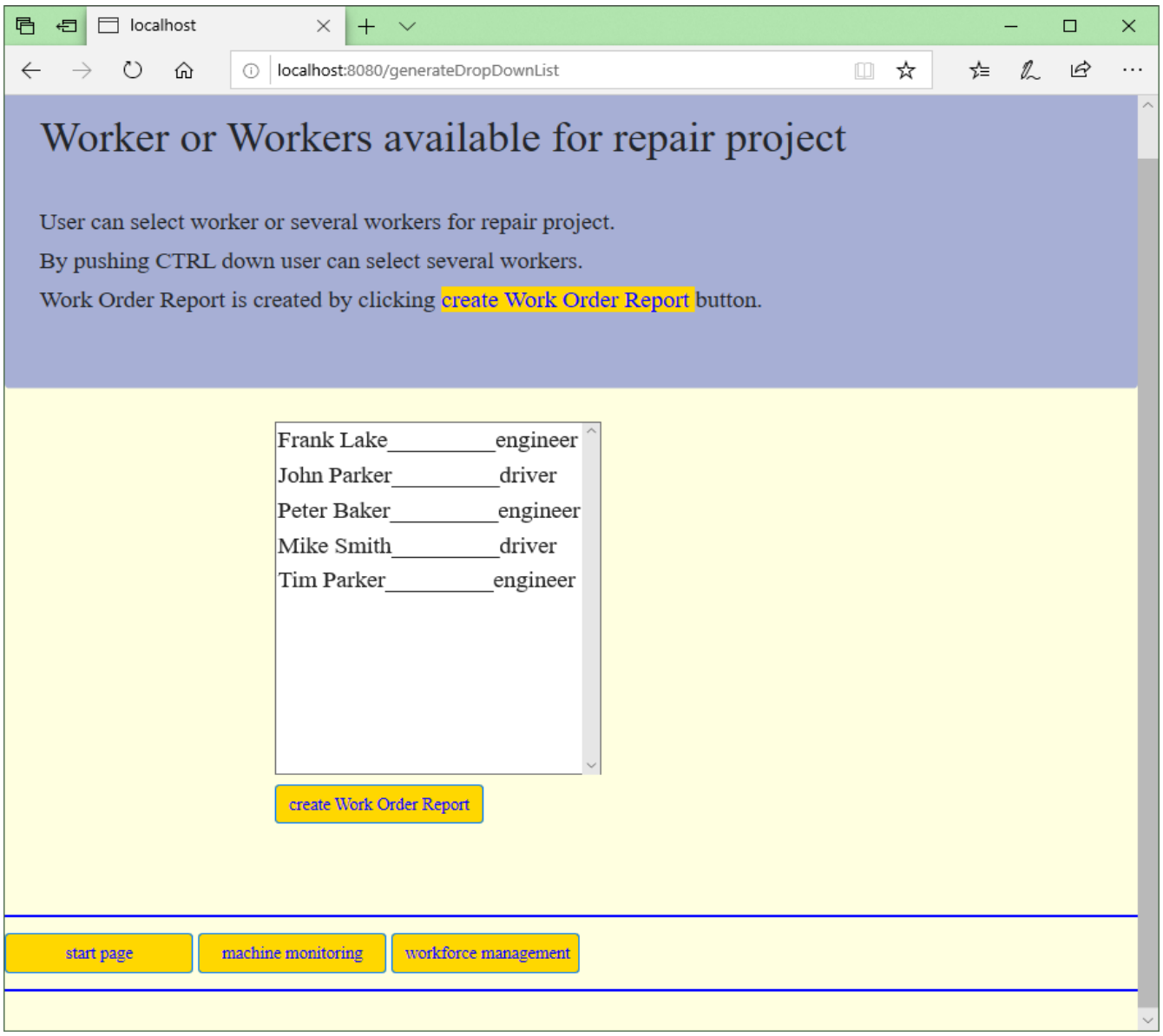
Machine monitoring is started by clicking text link: start machine monitoring in index.html.

machineList.html is now displayed



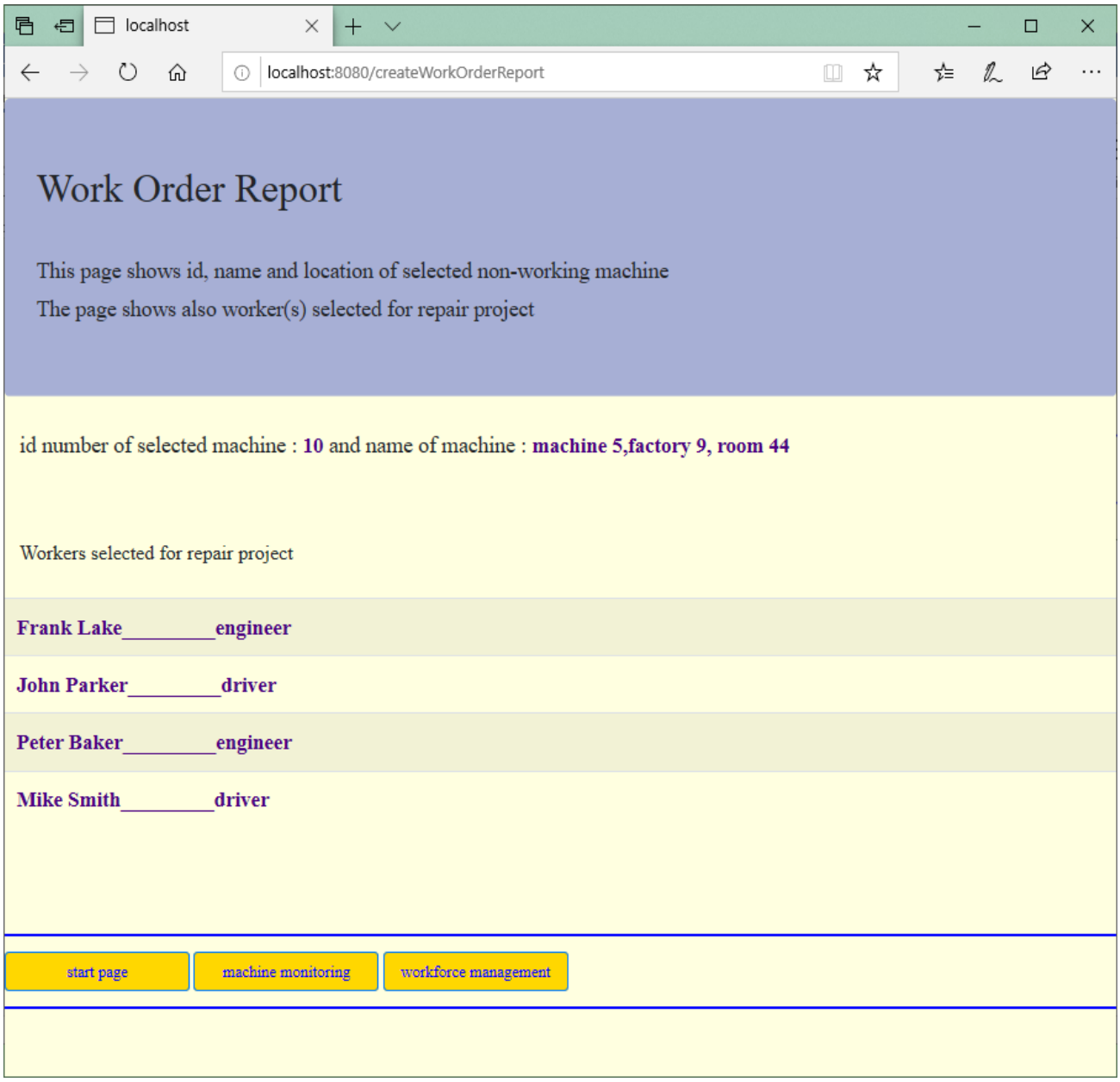
Red button is clicked and selectWorkersList.html is now displayed.

User can select available workers from drop down list. These workers come from model class Worker.java in Workforce application



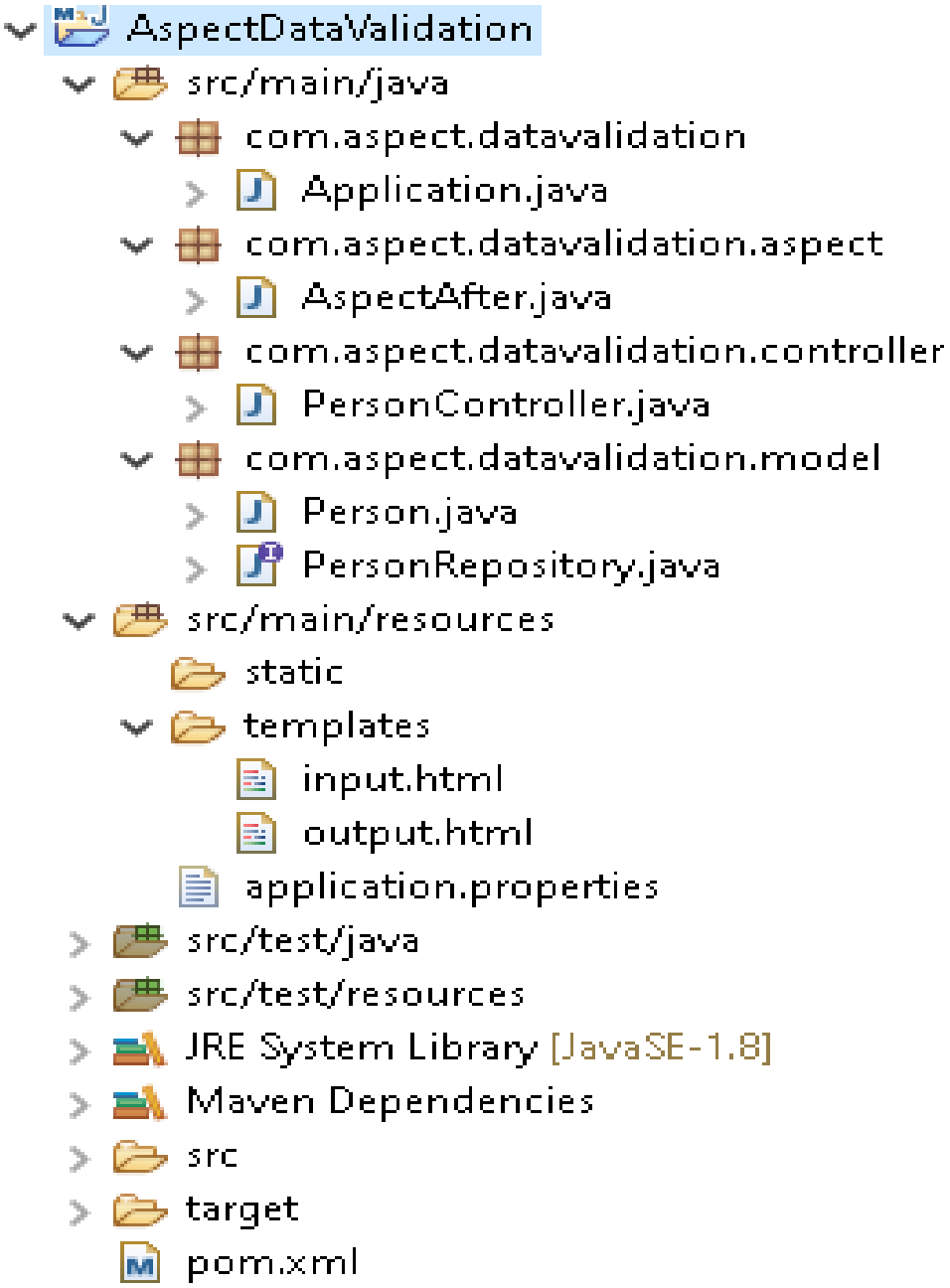
A new Work Order Report is created and sent into database when user has clicked create Work Order Report button.

workOrderReportForm.html is now displayed



AspectDataValidation

AspectDataValidation implements also data input validation and aspect oriented programming.



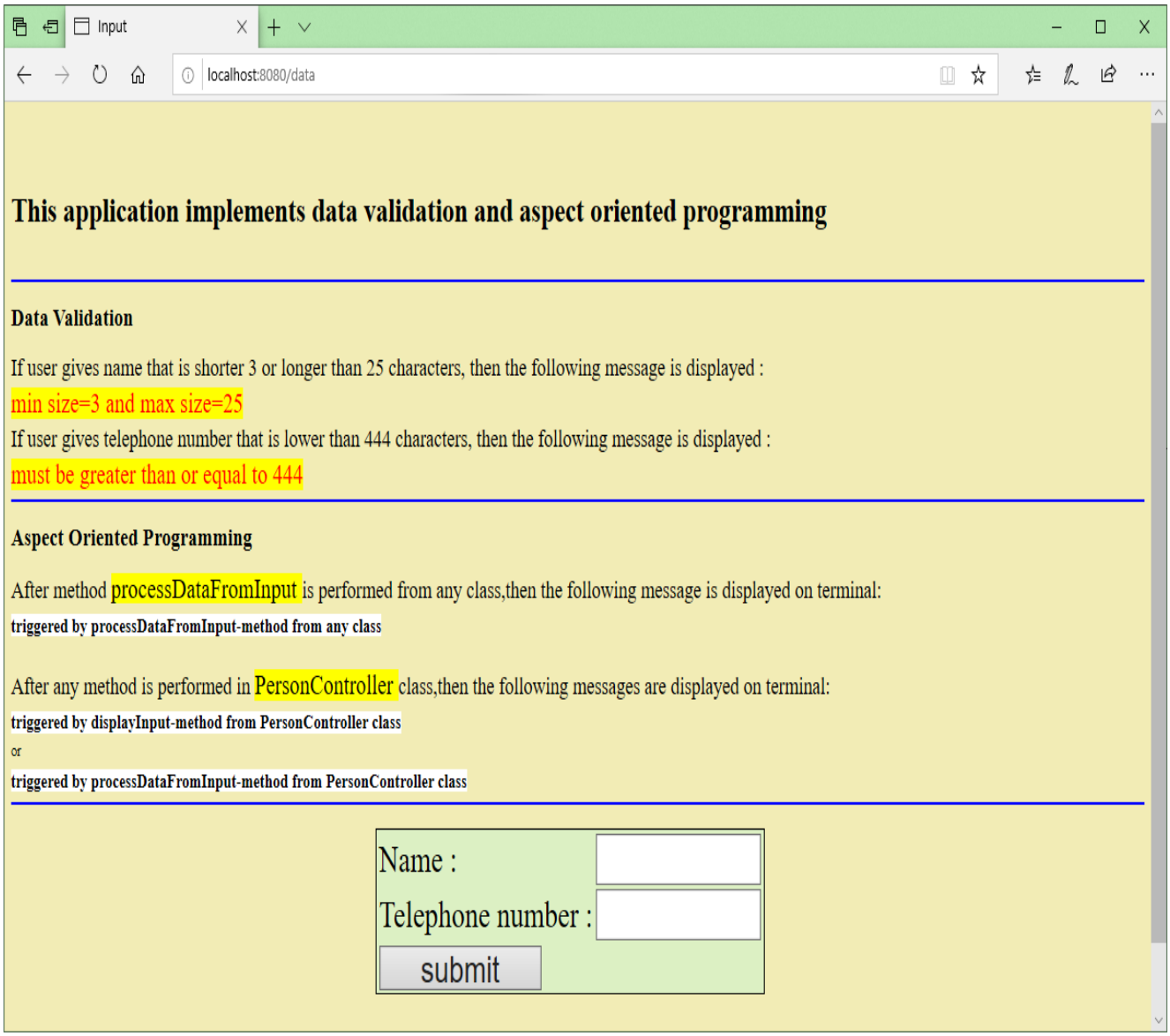
Data Validation

Data validation can be done in frontend (Javascript, jQuery, Angular…) or in backend .

In this application data validation is done with annotations in model class Person.java. If constraints defined by annotations are violated, operation will be bounced back to input.html file. When user has given valid input data, then data will be sent into database.

AspectDataValidation is started by writing endpoint: localhost:8080/data in browser.

input.html is now displayed.



Name : ab min size=3 and max size=25

Telephone number : 33 must be greater than or equal to 444

submit

User has given invalid data. Operation has been bounced back to input.html file and error messages are displayed.

Name : Frank Baker

Telephone number : 555

submit

User has given valid data. Data will be sent into database

output.html is displayed

Data Output

valid name : **Frank Baker** and telephone number: **555**

Aspect oriented programming (AOP)

Functions that span over many places in code are called cross-cutting concerns. These cross-cutting concerns are separated from business logic in aspect oriented programming (AOP).

We define cross-cutting concerns in one place. Now we can define where and how these cross-cutting concerns are applied without having to modify business logic classes.

Business logic classes contain only code of primary concern and secondary concern code have been moved to aspects.

AOP code in this application is implemented in AspectAfter.java.

Methods(displayInput, processDataFromInput) that trigger AOP code are implemented in PersonController.java.

The result from runtime operation is printed on terminal

…… output from terminal……

```
2019-09-16 13:50:19.063  INFO 9724 --- [           main] com.aspect.datavalidation.Application    :
Started Application in 25.445 seconds (JVM running for 27.3)
2019-09-16 13:50:28.095  INFO 9724 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       :
Initializing Spring DispatcherServlet 'dispatcherServlet'
2019-09-16 13:50:28.095  INFO 9724 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        :
Initializing Servlet 'dispatcherServlet'
2019-09-16 13:50:28.262  INFO 9724 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        :
Completed initialization in 167 ms

 triggered by displayInput-method from PersonController class

 triggered by processDataFromInput-method from PersonController class

 triggered by processDataFromInput-method from any class
```

……AspectAfter.java……

```
@Component
@Aspect
public class AspectAfter {

        @After(value="execution(*  com.aspect.datavalidation.controller.PersonController.processDataFromInput(..)  )  ")
        public void methodFromAnyClass(JoinPoint joinPoint) {

                System.out.println("\n triggered by "+joinPoint.getSignature().getName() + "-method from any class ");
        }


        @After(value="execution(*  com.aspect.datavalidation.controller.PersonController.*(..)  )  ")
        public void  anyMethodFromPersonControllerClass(JoinPoint joinPoint) {

                System.out.println("\n triggered by "+joinPoint.getSignature().getName() + "-method from PersonController class  );
        }
}
```

……

……PersonController.java……

```
@Controller
public class PersonController {

            @Autowired
            PersonRepository personRepository;

            @RequestMapping(value="/data",method=RequestMethod.GET)
            public String displayInput(Person person) {
                        return "input";
            }


            @RequestMapping(value="/data",method=RequestMethod.POST)
            public String processDataFromInput(Model model,@Valid Person person,BindingResult   bindingResult,@RequestParam String name,
                                                                                                                  @RequestParam  String telephoneNumber)
                {

                        if(bindingResult.hasErrors()) {
                           return "input";
                }

                        Person personObject= new Person();
                        personObject.setName(name);
                        personObject.setTelephoneNumber(telephoneNumber);
                        personRepository.save(personObject);

                        model.addAttribute("personObject",personObject);
                        return "output";

            }
}
```

……