

Table of Contents (TOC)

Preface

Spring Boot

Spring Data JPA

Thymeleaf

Bootstrap

MachineMonitorManagement

Workforce management

Machine monitoring

AspectDataValidation

Data Validation

Aspect oriented programming (AOP)

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 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.

Dependency 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**.

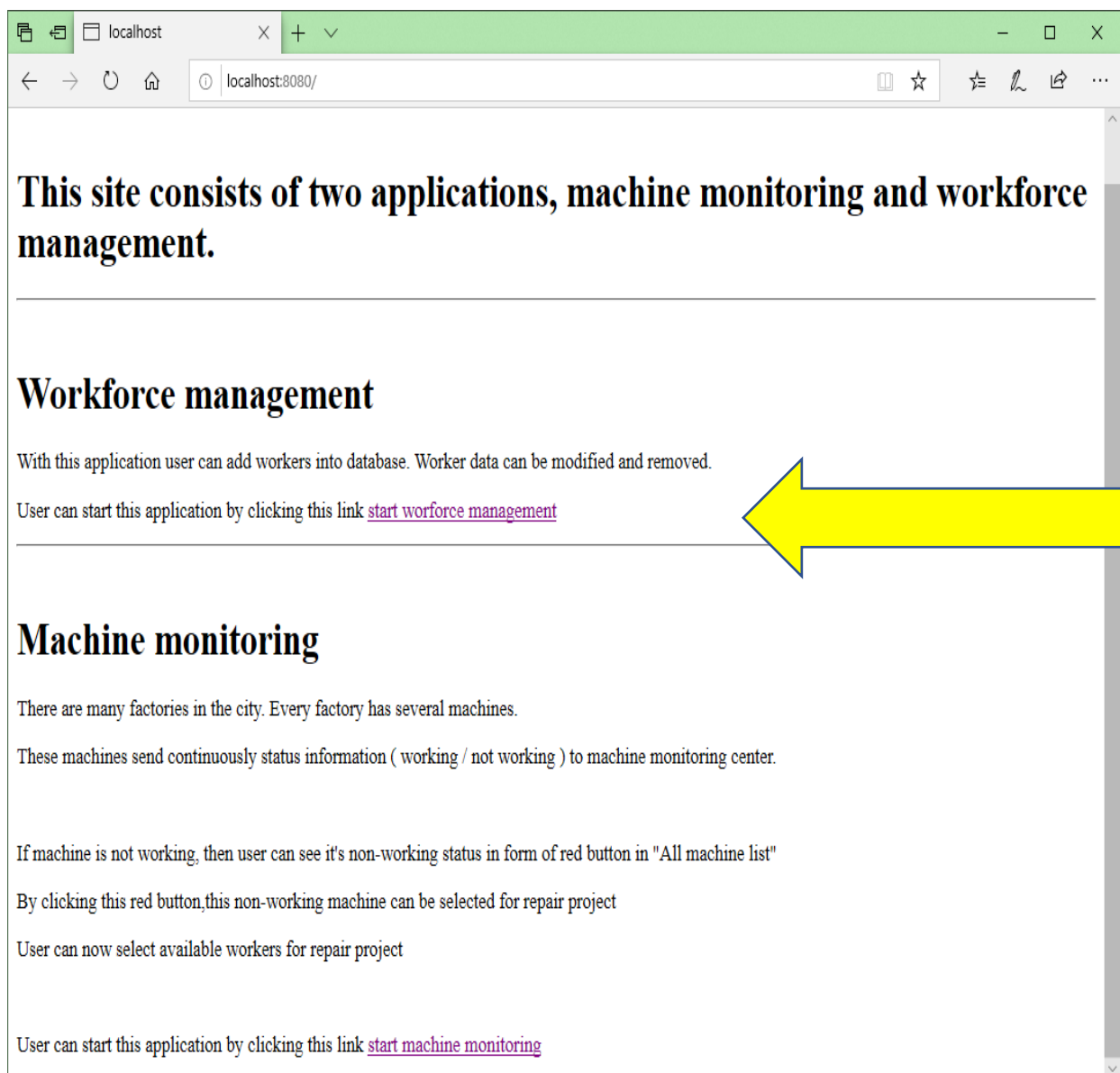
- ▼  MachineMonitorManagement
 - ▼  src/main/java
 - ▼  com.monitor.management
 - >  Application.java
 - ▼  com.monitor.management.controller
 - >  MachineMonitorController.java
 - >  WorkerManagementController.java
 - ▼  com.monitor.management.model
 - >  Machine.java
 - >  MachineRepository.java
 - >  Worker.java
 - >  WorkerRepository.java
 - >  WorkOrderReport.java
 - >  WorkOrderReportRepository.java
 - ▼  src/main/resources
 - ▼  static
 - ▼  css
 -  bootstrap.css
 -  custom.css
 -  index.html
 - ▼  templates
 - ▼  fragments
 -  CommonFragment.html
 - ▼  viewManagement
 -  addWorkerPage.html
 -  modifyWorkerPage.html
 -  workerList.html
 - ▼  viewMonitor
 -  machineList.html
 -  selectWorkersList.html
 -  workOrderReportForm.html
 -  application.properties

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.

All worker list

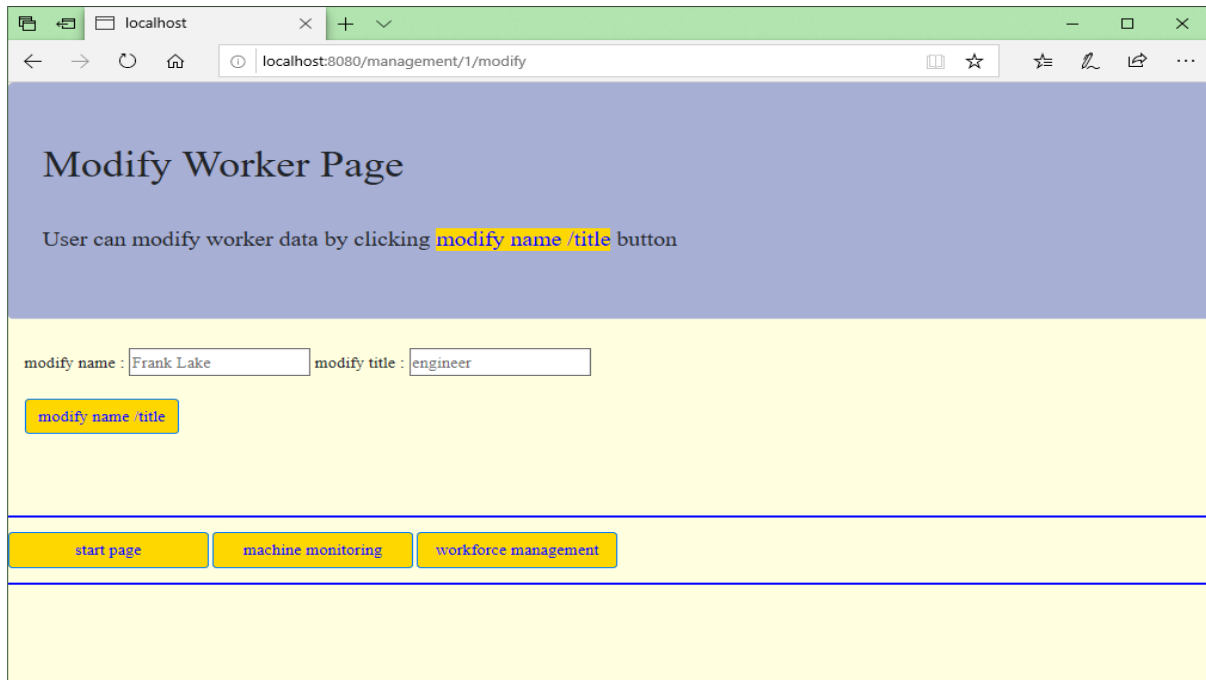
This page shows all workers from database
 User can modify or remove worker data by clicking [modify](#) or [remove](#) text link
 User can add new worker into database by clicking [add new worker](#) button

name	title	remove	modify
Frank Lake	engineer	modify	remove
John Parker	driver	modify	remove
Peter Baker	engineer	modify	remove
Mike Smith	driver	modify	remove
Tim Parker	engineer	modify	remove

[add new worker](#)

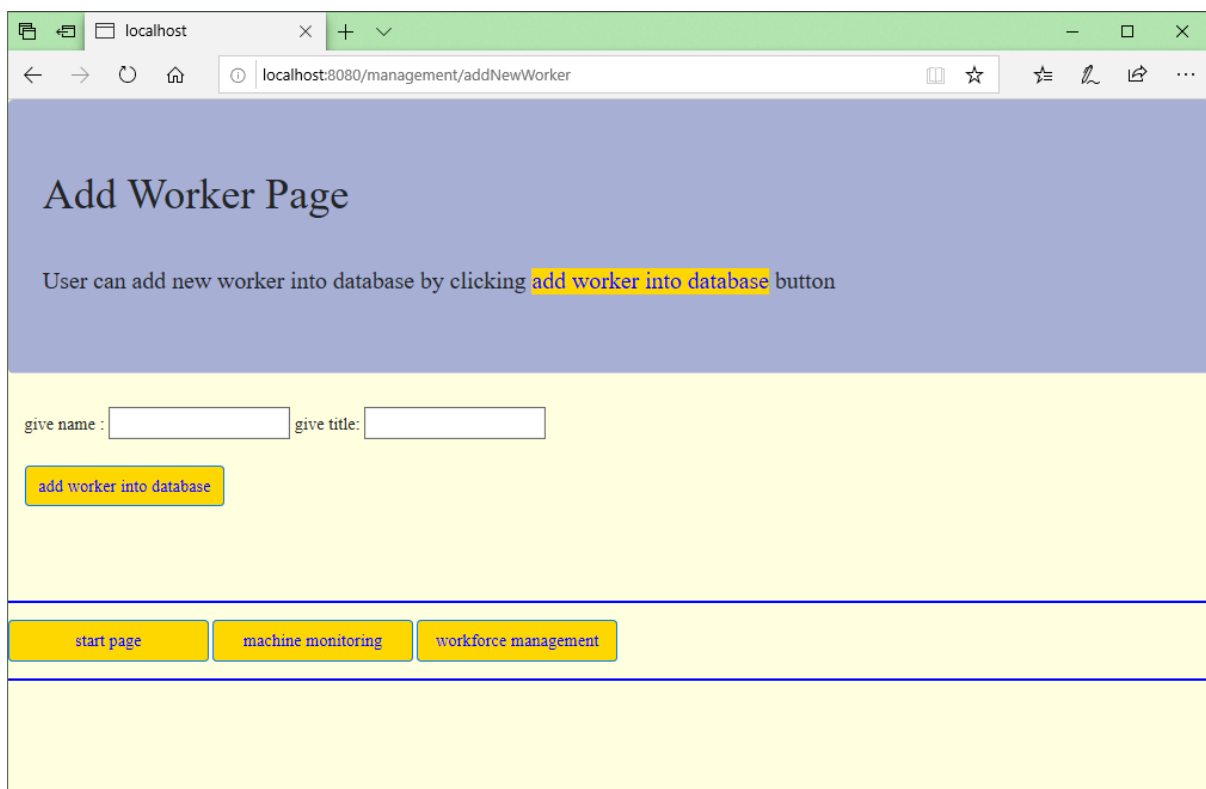
[start page](#)
[machine monitoring](#)
[workforce management](#)

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



A screenshot of a web browser window showing the 'Modify Worker Page'. The browser's address bar displays 'localhost:8080/management/1/modify'. The page has a light blue header with the title 'Modify Worker Page' and a subtitle 'User can modify worker data by clicking **modify name /title** button'. Below this, there are two input fields: 'modify name : Frank Lake' and 'modify title : engineer'. A yellow button labeled 'modify name /title' is positioned below the input fields. At the bottom of the page, there is a navigation bar with three yellow buttons: 'start page', 'machine monitoring', and 'workforce management'.

by clicking add new worker button **addWorkerPage.html** is displayed



A screenshot of a web browser window showing the 'Add Worker Page'. The browser's address bar displays 'localhost:8080/management/addNewWorker'. The page has a light blue header with the title 'Add Worker Page' and a subtitle 'User can add new worker into database by clicking **add worker into database** button'. Below this, there are two input fields: 'give name :' and 'give title:'. A yellow button labeled 'add worker into database' is positioned below the input fields. At the bottom of the page, there is a navigation bar with three yellow buttons: 'start page', 'machine monitoring', and 'workforce management'.

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

All machine list

This page shows all machines, location and status.
 Working machine has green color and non-working machine has red color button.
 By clicking red button, machine is selected for repair project
 User can then select worker or workers for repair project

machine	status / action
machine 1,factory 6 ,room 33	Click button to select this non-working machine for repair.
machine 2,factory 4 ,room 55	This machine is working. No Action is needed.
machine 3,factory 5 ,room 66	Click button to select this non-working machine for repair.
machine 4,factory 8 ,room 88	This machine is working. No Action is needed.
machine 5,factory 9, room 44	Click button to select this non-working machine for repair.

start page machine monitoring workforce management

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

The screenshot shows a web browser window with the address bar displaying `localhost:8080/generateDropDownList`. The page has a light blue header section with the title "Worker or Workers available for repair project". Below the title, there is instructional text: "User can select worker or several workers for repair project. By pushing CTRL down user can select several workers. Work Order Report is created by clicking `create Work Order Report` button." The main content area is light yellow and features a dropdown menu with the following list of workers and their roles:

Frank Lake	engineer
John Parker	driver
Peter Baker	engineer
Mike Smith	driver
Tim Parker	engineer

Below the dropdown menu is a yellow button labeled "create Work Order Report". At the bottom of the page, there is a navigation bar with three buttons: "start page", "machine monitoring", and "workforce management".

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

localhost

localhost:8080/createWorkOrderReport

Work Order Report

This page shows id, name and location of selected non-working machine
The page shows also worker(s) selected for repair project

id number of selected machine : 10 and name of machine : machine 5,factory 9, room 44

Workers selected for repair project

Frank Lake _____ engineer

John Parker _____ driver

























Peter Baker _____ engineer

Mike Smith _____ driver

start page machine monitoring workforce management

AspectDataValidation

AspectDataValidation implements also data input validation and aspect oriented programming.

- ▼  AspectDataValidation
 - ▼  src/main/java
 - ▼  com.aspect.datavalidation
 - >  Application.java
 - ▼  com.aspect.datavalidation.aspect
 - >  AspectAfter.java
 - ▼  com.aspect.datavalidation.controller
 - >  PersonController.java
 - ▼  com.aspect.datavalidation.model
 - >  Person.java
 - >  PersonRepository.java
 - ▼  src/main/resources
 -  static
 - ▼  templates
 -  input.html
 -  output.html
 -  application.properties
 - >  src/test/java
 - >  src/test/resources
 - >  JRE System Library [JavaSE-1.8]
 - >  Maven Dependencies
 - >  src
 - >  target
 - >  pom.xml

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.

This application implements data validation and aspect oriented programming

Data Validation

If user gives name that is shorter 3 or longer than 25 characters, then the following message is displayed :
min size=3 and max size=25

If user gives telephone number that is lower than 444 characters, then the following message is displayed :
must be greater than or equal to 444

Aspect Oriented Programming

After method `processDataFromInput` is performed from any class, then the following message is displayed on terminal:
 triggered by processDataFromInput-method from any class

After any method is performed in `PersonController` class, then the following messages are displayed on terminal:
 triggered by displayInput-method from PersonController class
 or
 triggered by processDataFromInput-method from PersonController class

Name :

Telephone number :

Name :	<input type="text" value="ab"/>	min size=3 and max size=25
Telephone number :	<input type="text" value="33"/>	must be greater than or equal to 444
<input type="button" value="submit"/>		

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

Name :	<input type="text" value="Frank Baker"/>
Telephone number :	<input type="text" value="555"/>
<input type="button" value="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";
    }
}
```
