Sample Primefaces JSF Web App Report

Name: Hao Li

Email: lih13800@gmail.com

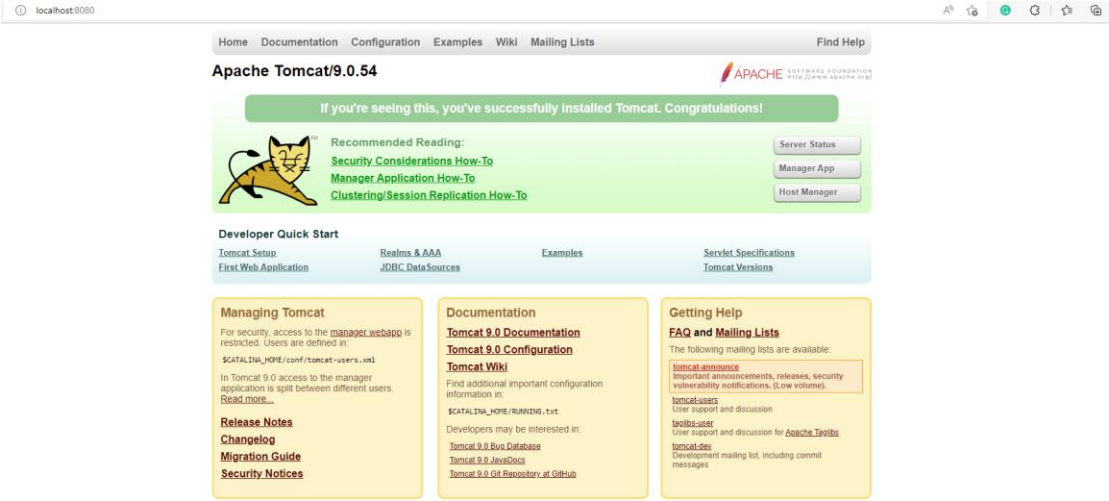
Submission Date: 15st May, 2022

URL Notation

In this project, after deployed to Tomcat, please use following URL:
http://localhost:8080/carProject/faces/index.xhtml

Environment Setup

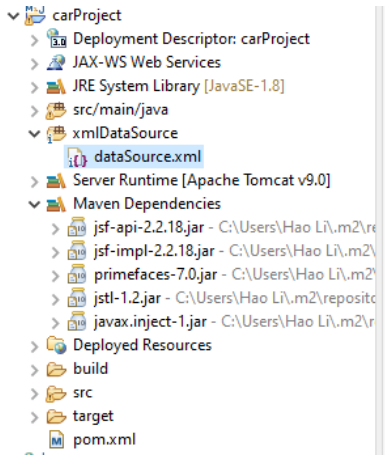
Apache Tomcat opened on the Eclipse side successfully.



Installed JSF and Primefaces through Maven in pom.xml file

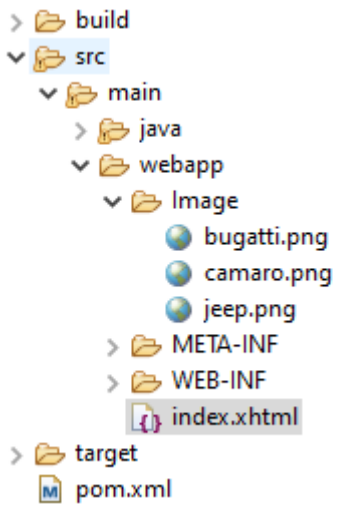
```
<dependencies>
  <!-- https://mvnrepository.com/artifact/com.sun.faces/jsf-api -->
  <dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.2.18</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/com.sun.faces/jsf-impl -->
  <dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-impl</artifactId>
    <version>2.2.18</version>
  </dependency>
  <dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>7.0</version>
  </dependency>
```

Setup source code in Eclipse's **resource folder**, named dataSource.xml.




```xml
<?xml version="1.0" encoding="UTF-8"?>
<cars>
    <car>
        <product_name>Jeep Wrangler</product_name>
        <drivetrain>AWD</drivetrain>
        <manufactured>03-01-2022</manufactured>
        <horsepower></horsepower>
        <engine_size>1995</engine_size>
        <comments></comments>
        <colour>Green</colour>
        <rating>5</rating>
        <image_name>jeep.png</image_name>
    </car>

    <car>
        <product_name>Chevrolet Camaro</product_name>
        <drivetrain>RWD</drivetrain>
        <manufactured>01-01-2021</manufactured>
        <horsepower>311</horsepower>
        <engine_size>3564</engine_size>
        <comments>Some comments go here</comments>
        <colour>Yellow</colour>
        <rating>5</rating>
        <image_name>camaro.png</image_name>
    </car>

    <car>
        <product_name>Bugatti Chiron</product_name>
        <drivetrain>AWD</drivetrain>
         <manufactured>06-01-2021</manufactured>
         <horsepower>1480</horsepower>
         <engine_size>7993</engine_size>
         <comments>Most Expensive</comments>
         <colour>Blue</colour>
         <rating>5</rating>
         <image_name>bugatti.png</image_name>
    </car>
</cars>
```

Stored images under the webapp folder, the name of images matched corresponding image_name XML element value


build
src
main
java
webapp
Image
bugatti.png
camaro.png
jeep.png
META-INF
WEB-INF
index.xhtml
target
pom.xml

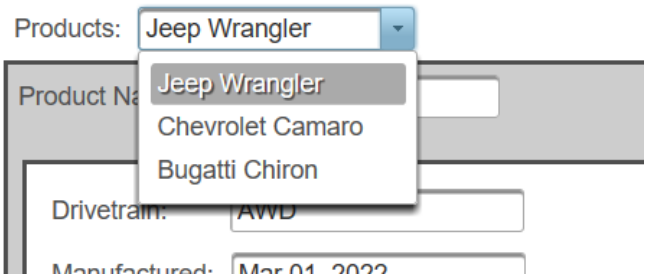
Functionality

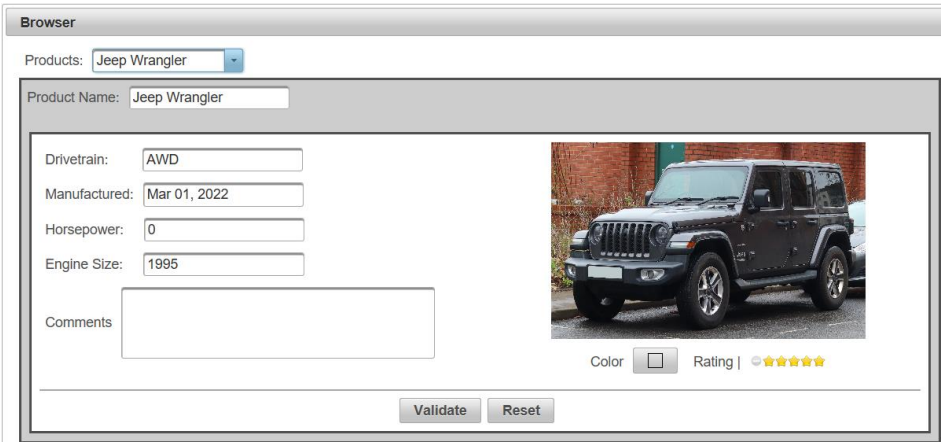
1. Read the data source through getClass().getClassLoader().getResource() method.
   This method helps us to find resource file's location correctly. In the beginning, Eclipse could find out the xml file from source folder according to the relative path. However, when the programming running on the Tomcat, the relative path failed. The class loader which implemented Java reflection can figure out this problem. Most importantly, it would not unable to find the source data even it is deployed in different environment in the future.

```
    private void unMarshalingExample() throws JAXBException, URISyntaxException {
        JAXBContext jaxbContext = JAXBContext.newInstance(Cars.class);
        Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();

        URL resource = getClass().getClassLoader().getResource("dataSource.xml");
//      System.out.println(resource.toURI().toString());

        Cars cars = (Cars) jaxbUnmarshaller.unmarshal(new File(resource.toURI()) );
//      Cars cars = (Cars) jaxbUnmarshaller.unmarshal(new File("xmlDataSource/dataSource.xml") )

        for(Car car : cars.getCars()) {
            System.out.println(car.getImage_name());
            System.out.println(car.getProduct_name());
            System.out.println(car.getManufactured());
        }
        setCars(cars.getCars());
    }
```

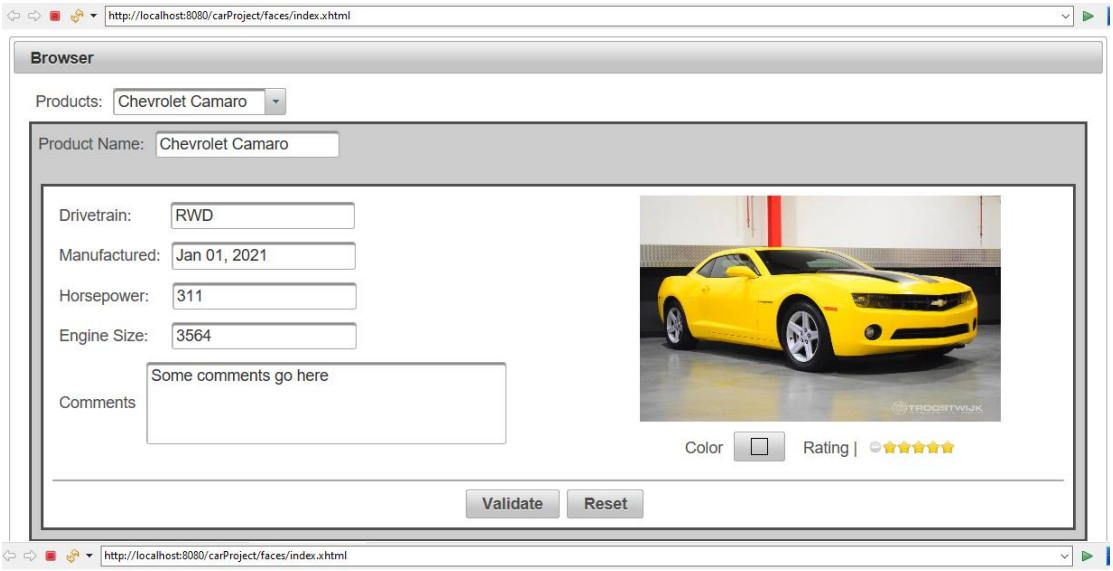
   Use jaxbUmarshaller to unmarshal the xml data source file.
   Each group of data is store in Car Object.

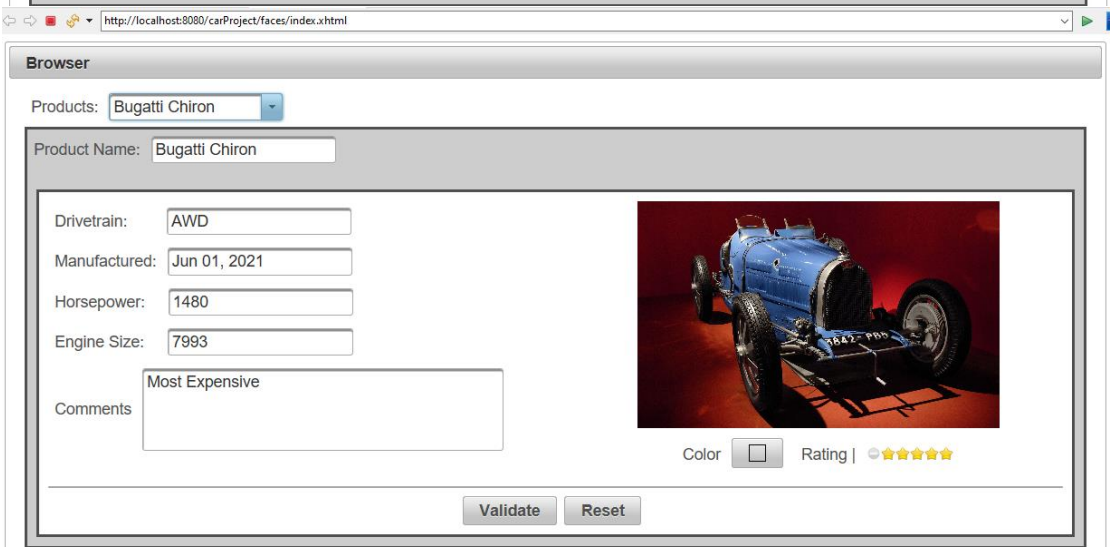
2. Created label 'Products' and a drop down with value from product_name XML element. Once the page load successfully, the data of first xml element loaded on the page.





3. When users change the drop down manual, it updated the panel with corresponding value successfully.

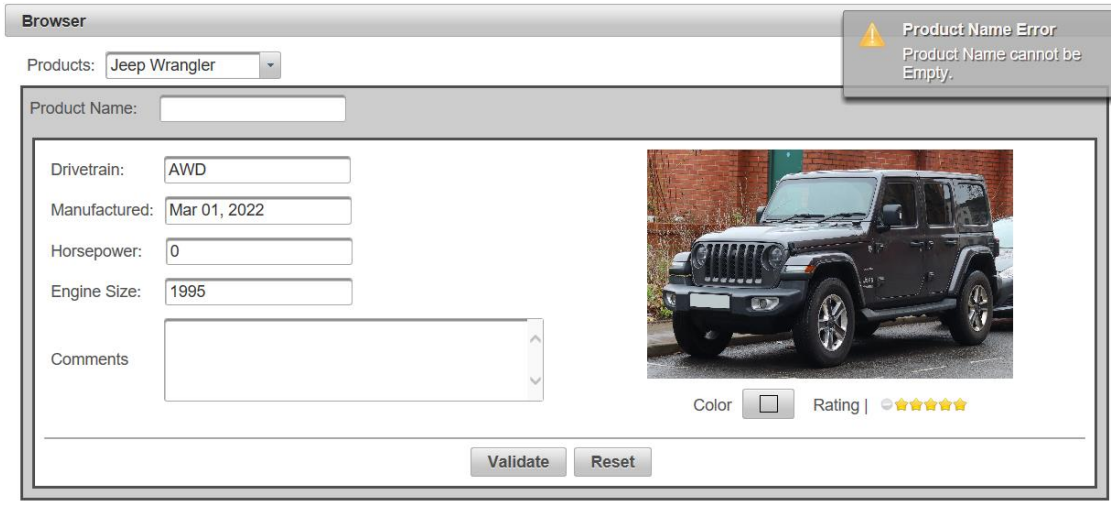


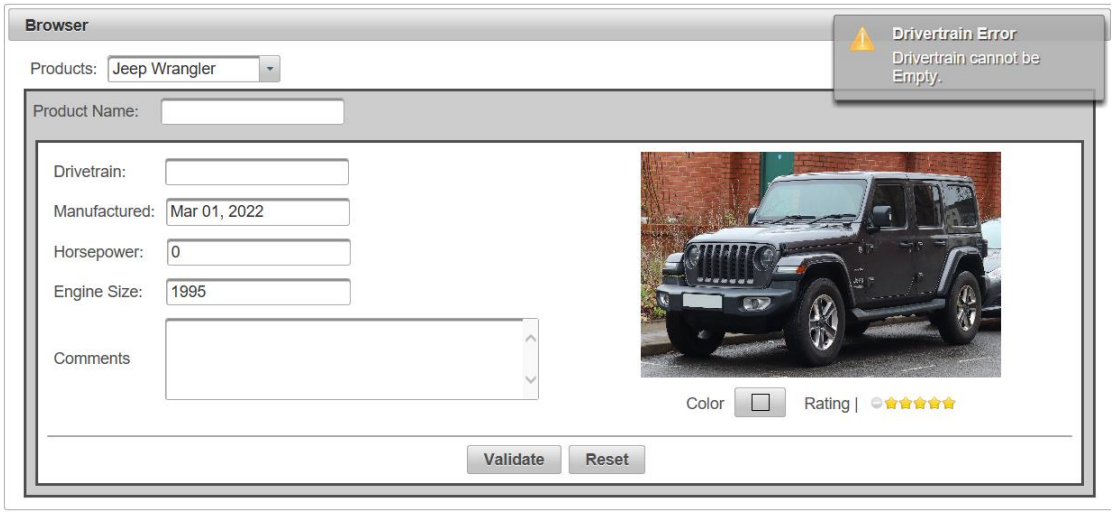


4. Implemented page level and Bean (model) validation to enforce these constraints

- "Product Name" and "Drivetrain" are text fields which must not be empty, and they can be up to 254 characters as maximum length. The error message would appear if the constraints were violated.

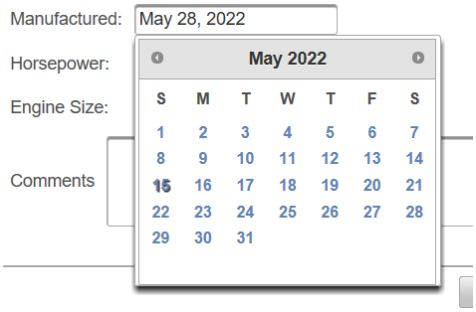
```
@NotNull(message = "Name cannot be null")
@Size(max = 254, message="No more than 254 characters")
private String selectedCar;

@NotNull(message = "Drivetrain cannot be null")
@Size(max = 254, message="No more than 254 characters")
private String selectedDrivetrain;
```





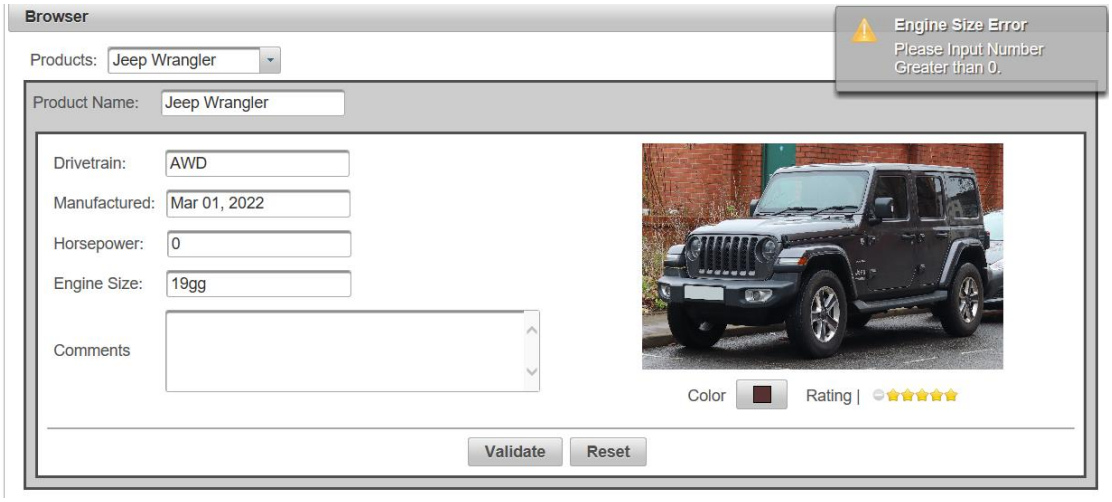
- "Manufactured" field was setup a date picker, and its formatted meets the requirement.

```
<div>
    <h:panelGrid columns="2" cellpadding="5" >
        <h:outputText value="Manufactured:" />
        <p:datePicker id="date" value="#{cpt.date}" pattern="MMM dd, yyyy" placeholder="#{cpt.selectedDate}">
        </p:datePicker>
    </h:panelGrid>
</div>
```



- Setup "number only" constraint for "Engine size" and "Horsepower" text filed. And the number should not less than 0. The error message would appear if any non-number character appeared.

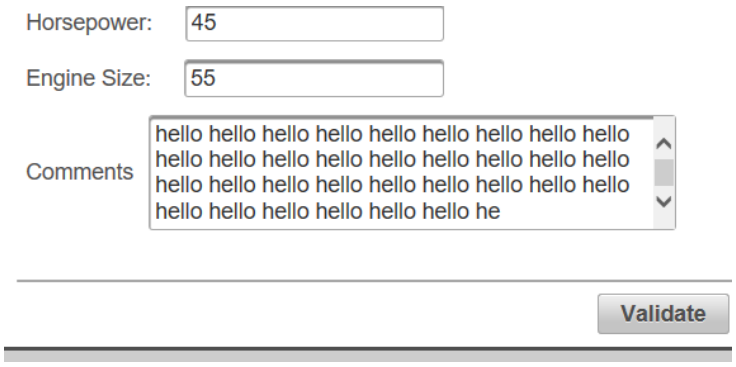
```
@Min(value = 0, message = "Horsepower should not be less than 0")
private Integer selectedPower;

@Min(value = 0, message = "Engine size should not be less than 0")
private Integer selectedEngine;
```



- Implemented the characters constraint on the multirow test area. Also, add on hard constraint on the page side. The text area would not receive any input when it achieved to its limitation.

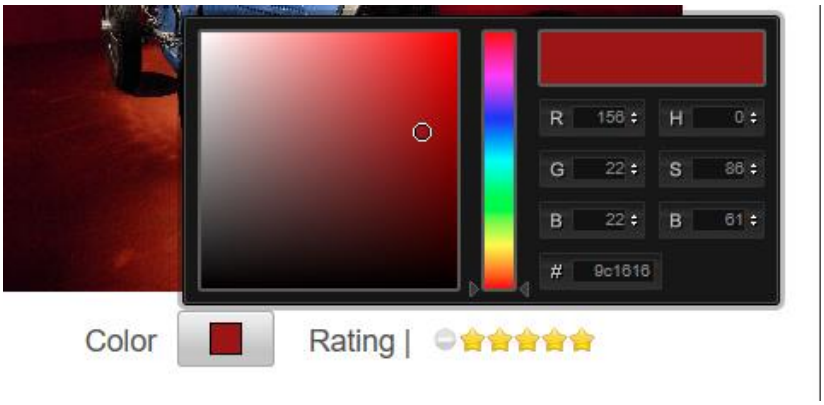
```
@Size(max = 200, message="No more than 200 characters")
private String selectedComment;
```

```
<div>
    <h:panelGrid columns="2" cellpadding="5" style="width: 116%">
        <h:outputText value="Comments" />
        <p:inputTextarea id="comment" value="#{cpt.selectedComment}" rows="4" cols="50" maxlength="200" autoResize="false" />

    </h:panelGrid>
</div>
```



5. Clicked the color block, it would open a colorpicker

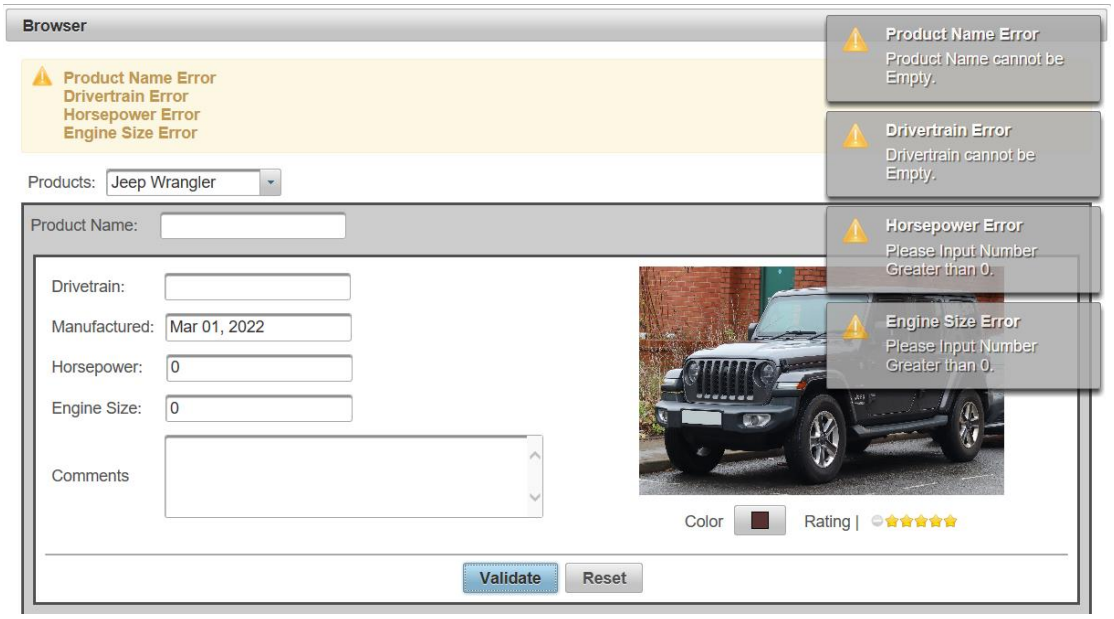
```
<h:outputText value="Color" />
<p:colorPicker value="#{cp.popupColor}" />
```



6. Rating component would use corresponding rating XML element value as its default value. Those lit stars could be canceled by mouse click. Also, click the little button on the left side of stars would extinguish all stars.

```
<h:outputText value="Rating |" />
<p:rating id="rate" value="#{cpt.selectedRating}" />
```

7. Validating all constraints by using “Validation” button and show error message on the top of the UI. The error message would disappear until the text filed match the constraint and press the “Validation” button again.



8. “Reset” button would reset those changed of current selected car.