Software architecture

Week 9 - JSP/JSF

Requirements

1. Netbeans 11
2. Java Development Kit 8
3. Payara Server

Sample project

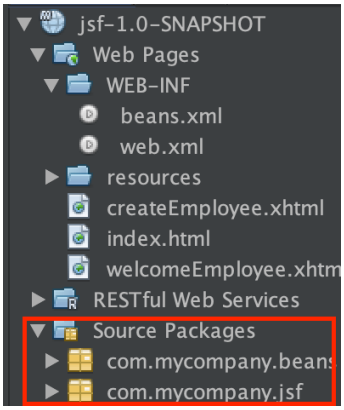
To introduce the principle of JSP and JSF, we will create a simple web project using these technologies. We will create our project using Netbeans and we will deploy it on a Payara server.

The goal of this exercise session is to make an overview of JSP and JSF technologies. We will explore Servlets, JSP and JSF.

Using JSF, we will create a simple page containing a form. We will use this form to register employees. Each employee should have a first name, last name and a position.

To create our project using Netbeans, we will follow the steps below:

1. Open Netbeans
2. Create a New Project (File > New Project > Java with Maven > Web Application)
3. Let's call it "SoAr_week9", click on *Finish*. Your project is ready!
4. Separate our source code into packages. In order to have a better understanding of our code, we will separate each part of our code into packages. For this project, we will use two different packages:
    - Beans (com.mycompany.beans)
    - Servlets (com.mycompany.servlets)



JavaServer Page

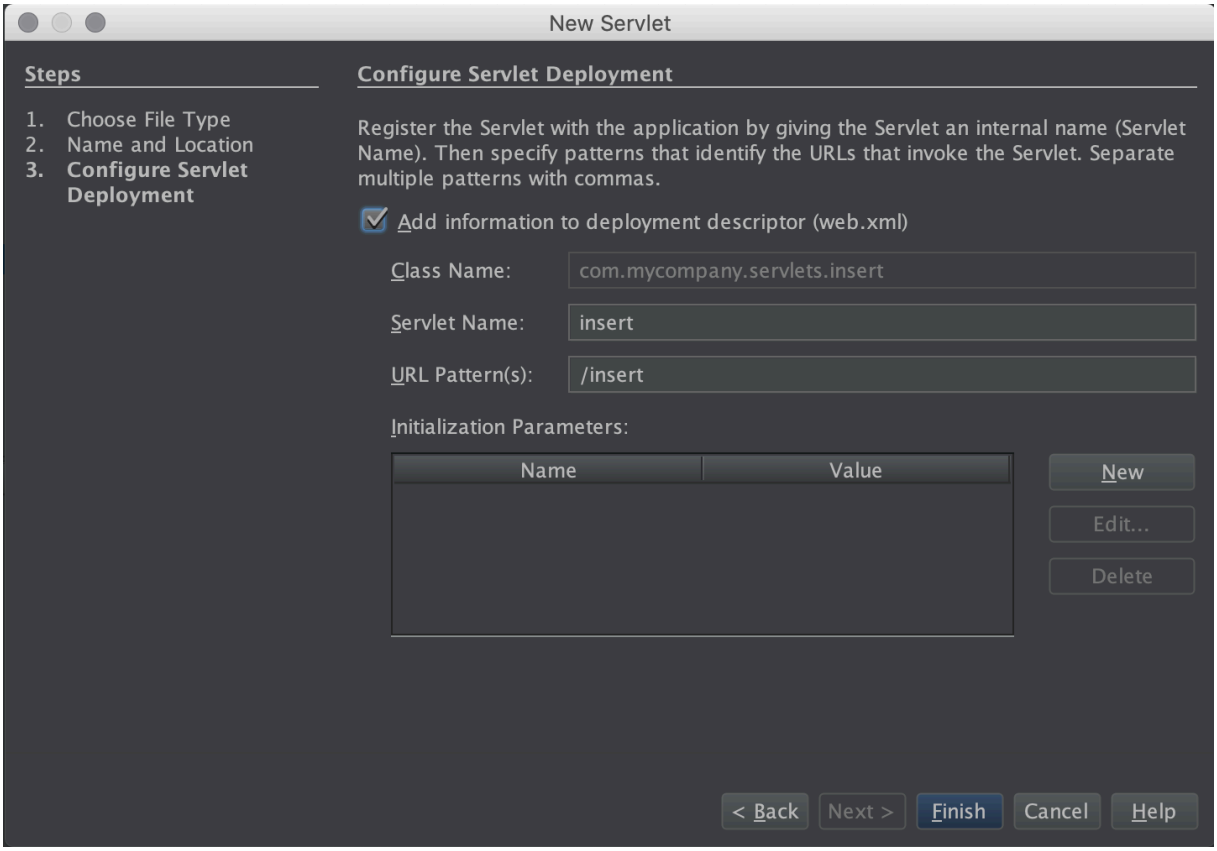
JavaServer Pages (JSP) is a collection of technologies that helps software developers create dynamically generated web pages based on HTML, XML, SOAP, or other document types. In this section, we will make an overview of Servlets, data transmission (using POST and GET) and JavaServer Pages.

1. Create the views (JSP): For this section on JSP, we will create two JSP files (A form containing two inputs and a confirmation page).

   We can see that the IDE automatically generate an index file. We will use it as a landing page. It should contains a link to the form. To create new pages, we have to right-click on our project > New > JSP. We will create two pages called *openAccount* and *accountDetails openAccount* will contain the following form:

   ```
   <form method="post" action="insert">
      Last Name   <input type="text" name="lastname" value="Simpson"><br>
      First Name  <input type="text" name="firstname" value="Marge"> <br>
        <input type="submit" name="action" value="Create a new Account">
     </form>
   ```

2. Create a servlet: Right click on the servlet's package (com.mycompany.servlets) > New > Servlet. In the dialog box, give a name to your servlet (`insert`), click on `Next`. On the next page, check the box `Add information to deployment descriptor`



Looking at the generated code for the new `insert` servlet, you can see that the IDE's servlet template employs a processRequest method which is called by both doGet and doPost methods. (You may need to expand the code fold by clicking the plus icon in the editor's left margin to view these methods.) Because this application differentiates between *doGet* and *doPost*, we will add code directly to these methods and remove the processRequest method altogether.
In the `doPost` method, we will get the parameters of the form. To do it, we have to invoke the method `.getParameter("parameter name")` of request. Then we will provide these parameters to *accountDetails.jsp* by using a `RequestDispatcher`.
Our code should look like this:

```
@Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        RequestDispatcher rd =
request.getRequestDispatcher("/accountDetails.jsp");
        String firstname = request.getParameter("firstname");
        String lastname = request.getParameter("lastname");

        request.setAttribute("firstname", firstname);
        request.setAttribute("lastname", lastname);

        rd.forward(request, response);
    }
```

The method `rd.forward(request, response)` will redirect us to `accountDetails.jsp` with the defined attributes (`firstname` and `lastname`).

1. Get the attributes: In `accountDetails.jsp`, we will show a simple message containing in the attributes mentionned above. To get these attributes, we can use `request.getParameter("_Attribute name_")` or an EL `${param.firstname}`.

Our JSP file should look like this:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Confirmation message</title>
    </head>
    <body>
        <h1>Welcome ${param.firstname} ${param.lastname}</h1>
    </body>
</html>
```

Welcome Marge Simpson

JavaServer Faces

JavaServer Faces (JSF) is a user interface (UI) framework for Java web applications. It is designed to significantly ease the burden of writing and maintaining applications that run on a Java application server and render their UIs back to a target client.
JSF is based on the Model-View-Controller (MVC) Pattern.

To build our project using JSF, we have to follow the steps below:

1. In our project, we have to create a bean containing the properties of our employees (First name, last name, position). To create a new bean, we have to right click on the bean package(com.mycompany.beans) and click on *New > Other > JavaServer Faces > JSF CDI Bean*. We will call it *EmployeeBean*

2. Once the bean is created, we have to add the annotations `@Named(value = "employeeBean")` from `javax.inject.Named` and `@RequestScoped` from `javax.enterprise.context.RequestScoped`, then we have to add the attributes of *Employee* and generate the Getters and Setters. Each employee should have the following attributes:
   ```
   private String firstName, lastname, position;
   ```
   To generate the getters and setters using Netbeans, you can Right-click on each attributes and click on *Insert code* > Getter and Setter > *Select the attributes* > Click on Generate `Named(value = "employeeBean")`: allows you to specify to the server that this bean is now a CDI managed bean.

A managed bean is a POJO (Plain Old Java Object) that can be used to store data, and is managed by the container (e.g., the Payara server) using the CDI (Context and Dependency Injection) framework.
CDI is preferred over plain JSF backing beans because CDI allows for Java EE wide dependency injection. In a future release of JSF the @Managedbean will be removed

from the JSF package. CDI is more powerful than the JSF managed bean because you are not limited to JSF only. You can inject every bean managed by CDI.
A POJO is essentially a Java class that contains a public, no argument constructor and conforms to the JavaBeans naming conventions for its properties.
Note: In JSF 2.3, managed bean annotations are deprecated; CDI is now the preferred approach.

6. Creating the view: For this project, we will create two Facelets (Facelets are just XHTML pages with JSF
   tags): *createEmployee* and *welcomeEmployee*. *createEmployee* will contains the form to create a new employee. *welcomeEmployee* will show us the attributes defined in our bean.

   Netbeans makes it easy for us to create facelets by allowing us to use pre-designed templates.

   To create a new Facelet based on a defined template, we have to Right click on *Web Pages*> New > Other > JavaServer Faces > Facelets Template Name your template and select a Layout style, then click on Finish. Create the two facelets: *createEmployee* and *welcomeEmployee*

   i. The facelet *createEmployee* will contain a simple form with three inputs (firstname, lastname, position). The file should look like this:

```
  <?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">

    <h:head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <h:outputStylesheet name="/resources/css/default.css"/>
        <h:outputStylesheet name="/resources/css/cssLayout.css"/>
        <title>Creating a new Employee</title>
    </h:head>

    <h:body>

        <div id="top" class="top">
            <ui:insert name="top"><h2>Creating a new Employee</h2></ui:insert>
        </div>

        <div id="content" class="center_content">
            <h:form>
                <h:outputLabel for="firstName">First name: </h:outputLabel>
                <h:inputText id="firstName" size="20"
value="#{employeeBean.firstName}"/><br />
                <h:outputLabel for="lastName">Last name: </h:outputLabel>
                <h:inputText id="lastName" size="20"
value="#{employeeBean.lastName}"/><br />
                <h:outputLabel for="Position">Position: </h:outputLabel>
                <h:inputText id="position" size="20"
value="#{employeeBean.position}"/><br />
```

```
                <h:commandButton id="submit" value="Submit"
action="welcomeEmployee" />

            </h:form>
        </div>

    </h:body>

</html>
```

The JSF runtime searches for a file named `welcomeEmployee`. It assumes the file extension is the same as the extension used by file from which the request originated (`createEmployee.xhtml`) and looks for the `welcomeEmployee` file in the same directory as the originating file (i.e., the webroot).

2. The Facelet `welcomeEmployee` will show the overwritten attributes of `EmployeeBean`. We can simply call these attributes using EL expressions `#{employeeBean.(Atribute)}`. Our `welcomeEmployee`'s file should look like this:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">

    <h:head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <h:outputStylesheet name="resources/css/default.css"/>
        <h:outputStylesheet name="resources/css/cssLayout.css"/>
        <title>Welcome Employee</title>
    </h:head>

    <h:body>
        <!-- Header -->
        <div id="top" class="top">
            <ui:insert name="top"><h1>Welcome #{employeeBean.firstName},
#{employeeBean.lastName}</h1></ui:insert>
        </div>
        <!-- Content -->
        <div id="content" class="center_content">
            <ui:insert name="content">Position of the new Employee
<b>#{employeeBean.position}</b></ui:insert>
        </div>

    </h:body>

</html>
```

Expression Language (EL) are simple expressions to dynamically access data from JavaBeans components. EL provides a way to use simple expressions to perform the following tasks:

- Dynamically read application data stored in JavaBeans components, various data structures, and implicit objects
- Dynamically write data, such as user input into forms, to JavaBeans components

- Invoke arbitrary static and public methods
- Dynamically perform arithmetic, boolean, and string operations.
- Dynamically construct collection objects and perform operations on collections

7. Run the project: Right click on `createEmployee.xhtml` > Run File

Common errors:

1. Unable to find resource ./css/default.css: To fix it, remove "./" on your resources path. The new *h* tag should look like this: `<h:outputStylesheet name="resources/css/default.css"/>`
2. `... Target Unreachable, identifier 'bean' resolved to null`: When evaluating the EL expression, JSF finds no setter method for the bean name with the identifier mentioned in the error message. Indeed, the rule for a JavaBean requires that the method be correctly named, and since we have changed its name, JSF considers that there is no setter method. It therefore logically warns us that an exception `javax.el.el.PropertyNotWritableException` is thrown, and that the name property is considered to be read-only.
3. `unable to find matching navigation case with from-view-id '/mypage.xhtml' for action 'connexion' with outcome 'connexion'`: The JSF controller - the FacesServlet - is unable to find a Facelet named `mypage.xhtml`, and JSF automatically displays the error, directly within your page! To fix it, you have to make sure that `mypage.xhtml` is available in the appropriate path.

Difference between JSP and JSF

JSP technology is part of the Java technology family. JSP pages are compiled into servlets and may call JavaBeans components (beans) or Enterprise JavaBeans components (enterprise beans) to perform processing on the server. As such, JSP technology is a key component in a highly scalable architecture for web-based applications.
http://www.oracle.com/technetwork/java/faq-137059.html

JavaServer Faces technology is a framework for building user interfaces for web applications. JavaServer Faces technology includes:

- A set of APIs for: representing UI components and managing their state, handling events and input validation, defining page navigation, and supporting internationalization and accessibility.
- A JavaServer Pages (JSP) custom tag library for expressing a JavaServer Faces interface within a JSP page.

https://jcp.org/en/introduction/faq

Resources

Official documentation of JSF tags

Exercises

1. Create a simple landing page using JSF. The page must contain a header, a content div and a footer
2. Add views containing forms: the purpose of our project is to create a new bank account.
3. Persist the newly created Bank Account in a H2 database (JPA - Week 7)
4. Handle transactions between Bank accounts (deposit and transfer).

Oracle VM VirtualBox

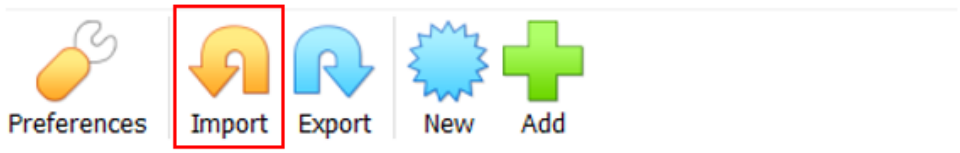
Download Oracle VM VirtualBox

1. Go to the following link: https://www.virtualbox.org/wiki/Downloads
2. Download Oracle VM VirtualBox (choose your OS):

How to use Oracle VM VirtualBox

Once you installed the application, all you need to do is to Import the OVA file (download the file here).

1. Click on **Import**

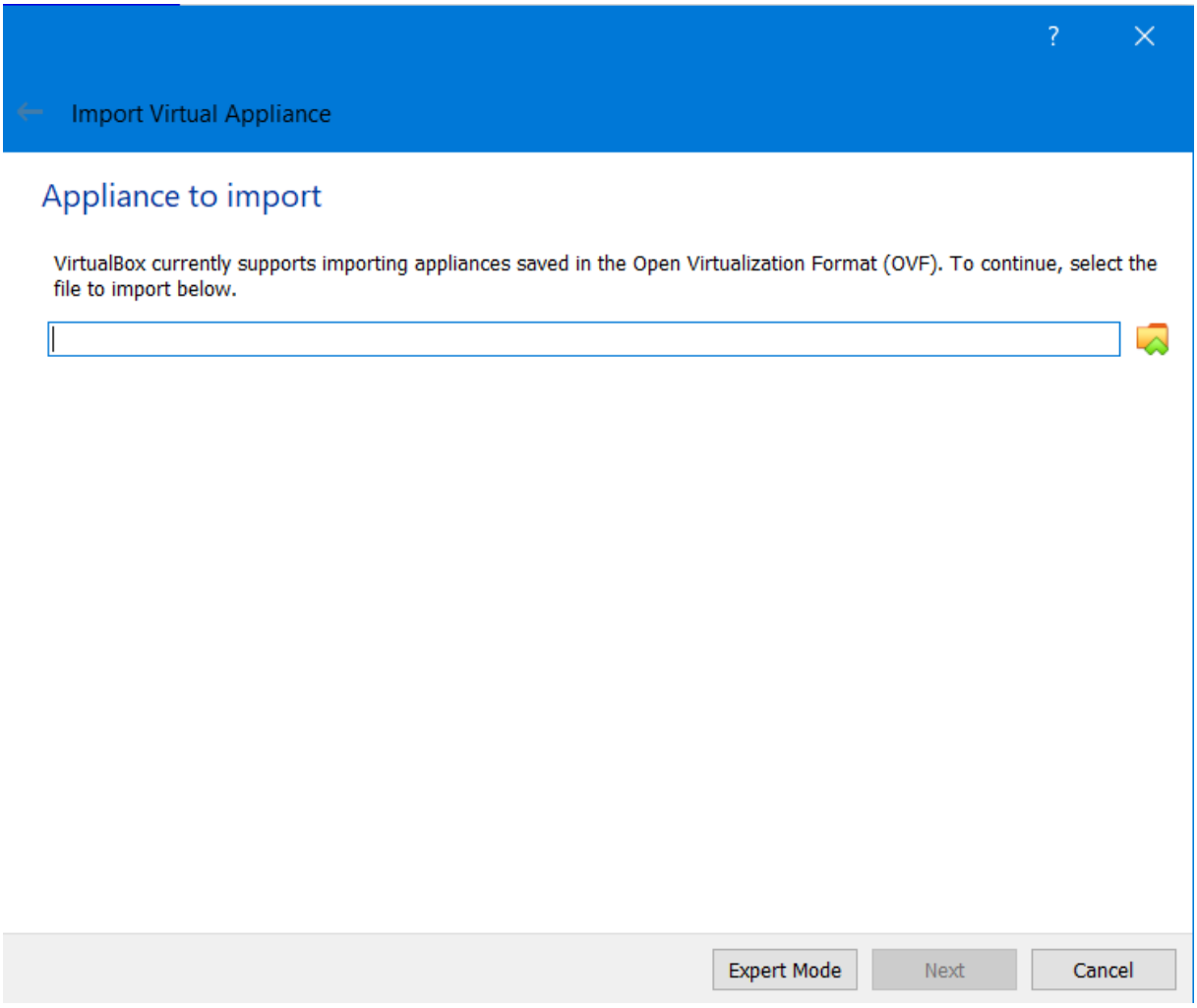


Welcome to VirtualBox!

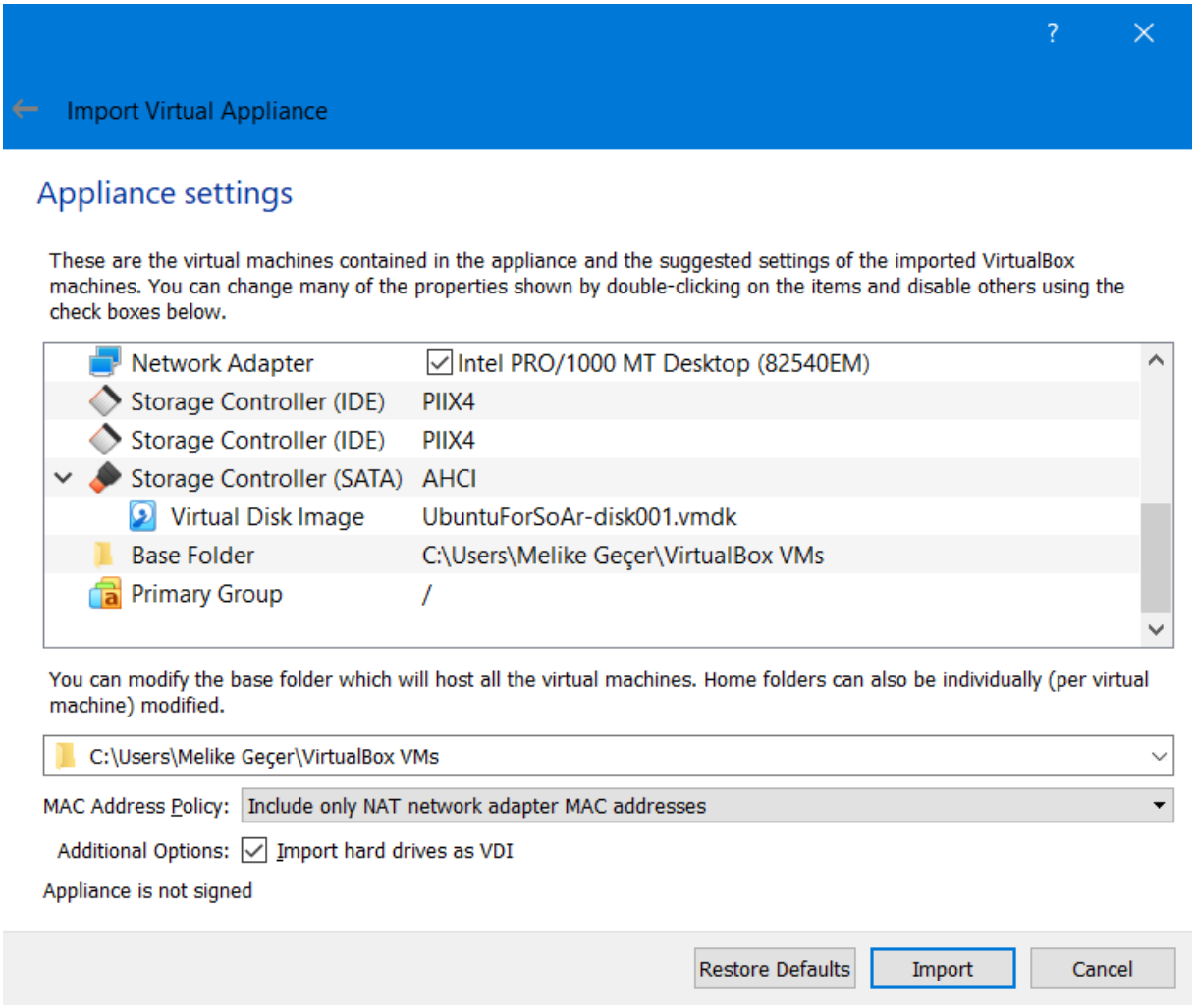
The left part of application window contains global tools and lists all virtual mach
selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for mor

2. Select the OVA file



3. Don't change anything and click on **Import**

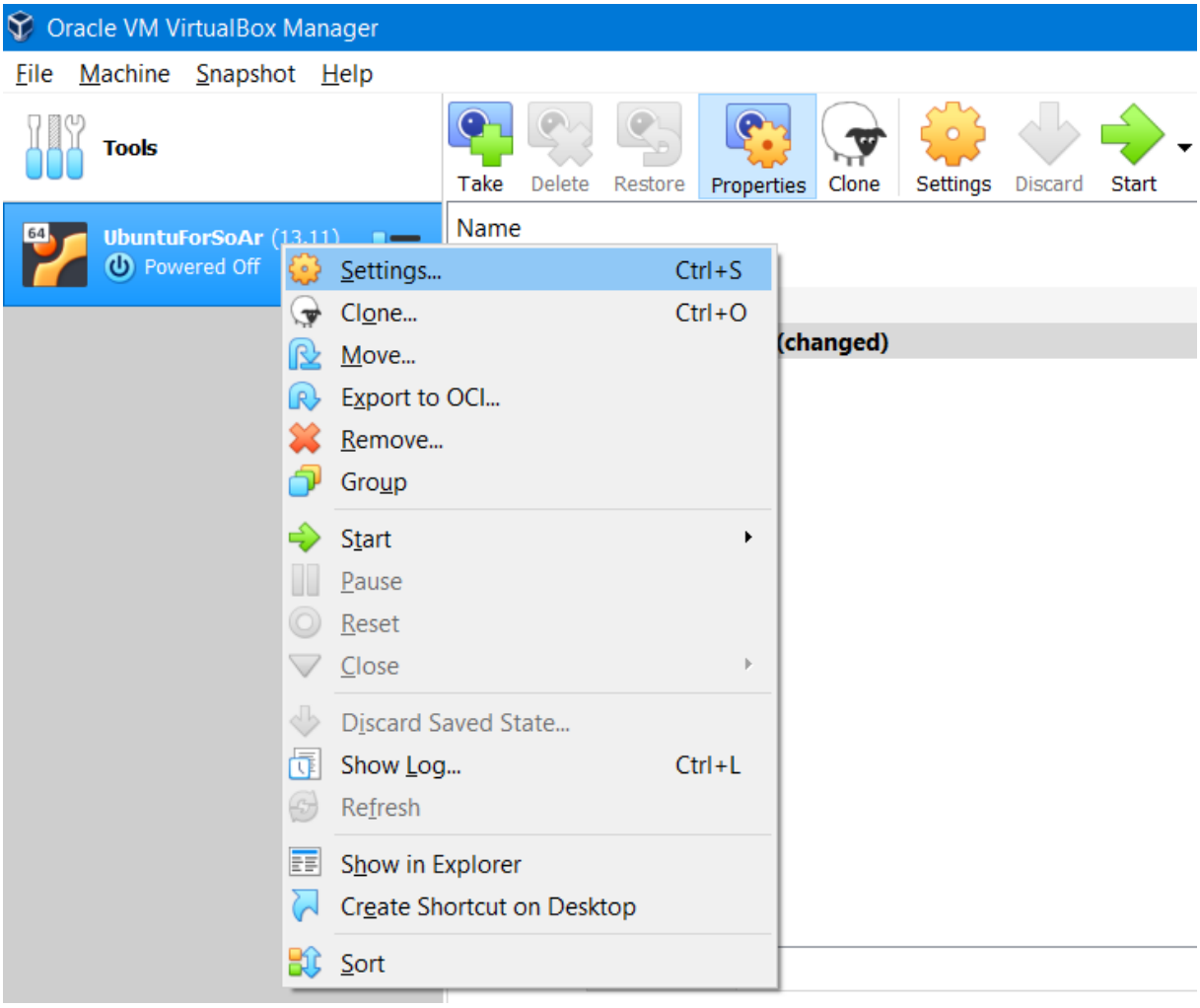


Important Information About the VM

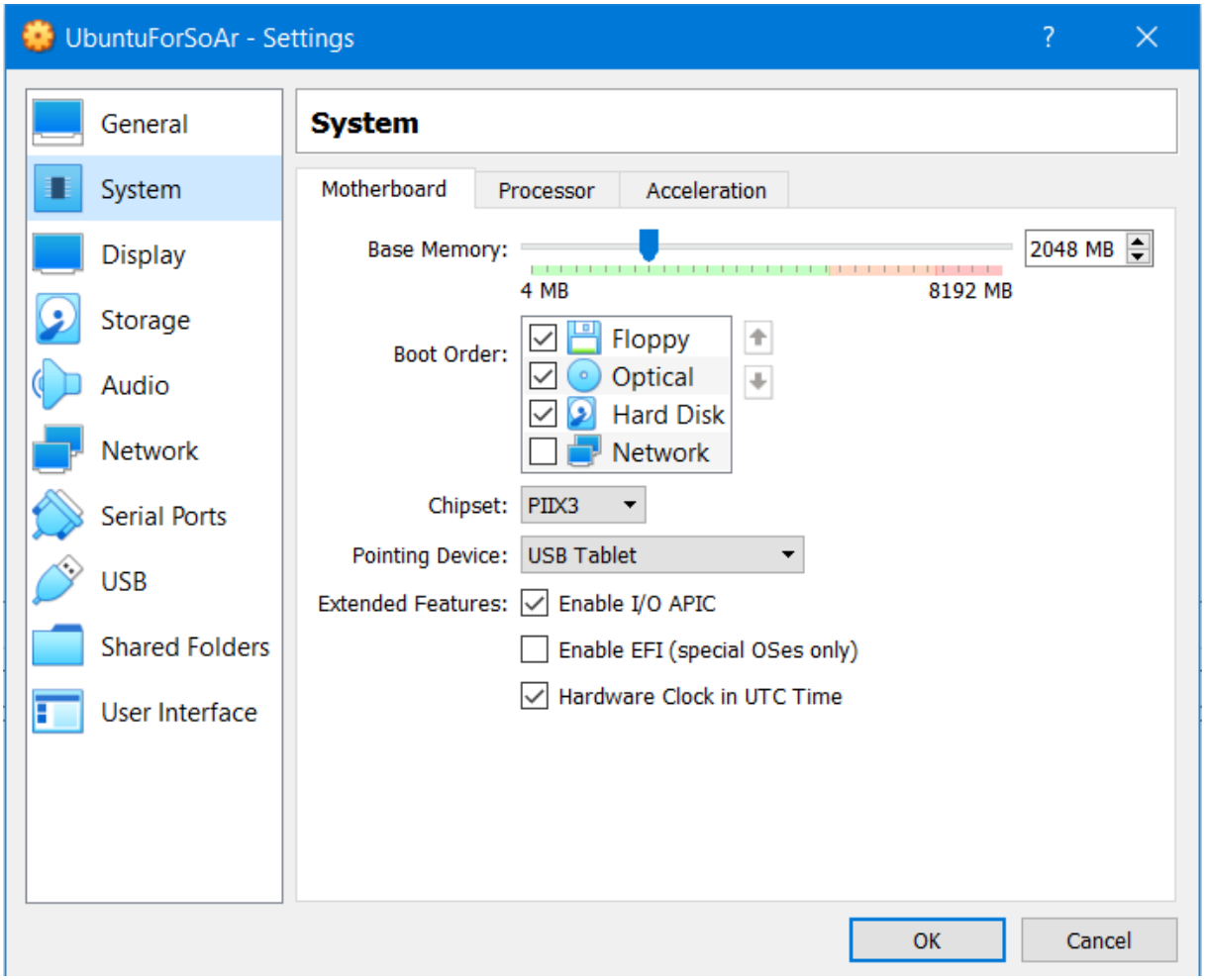
Username: soar Password: 1234

How to set RAM of the virtual machine

1. Click on **Settings**



2. Click on **Systems**



3. Set the RAM (Base Memory) as you wish. But make sure

- you have at least 4GB of RAM (Base Memory is at least 4096)
- the selector is always in green part
- the value must be at least 2048, which is also the default, and should be represented as 2 to the power X (you can set it to 2048, 4096, 8192, and so on).