

Proyecto BETS

Transformación JSF, Hibernate



**Nicolás Aguado, Xiomara G. Cáceres
y Eider Fernández**

21 de Diciembre del 2023
Ingeniería del Software II

Índice

Índice.....	1
URL del código fuente del proyecto.....	2
Configuración del Proyecto.....	2
Requisitos del Sistema.....	11
Implementación.....	12
Login y Registro.....	12
Contraseñas Hasheadas.....	12
Persistencia del Usuario.....	13
Apostar.....	13
A qué apostar.....	13
Cuestiones Monetarias.....	14
Query Question.....	14
Create Question.....	14
Experiencia con IntelliJ.....	15
Problemas encontrados.....	15
Tiempo invertido.....	16

URL del código fuente del proyecto

Para la entrega del proyecto se pide nada más el fichero .war correspondiente al código fuente para desplegar la aplicación. Ahora, creemos que de cara a facilitar una mejor integración del proyecto en el entorno de desarrollo IntelliJ es necesario aportar algunos archivos de configuración más. Es por esto que se proporciona el código fuente completo, alojado en un repositorio de github.

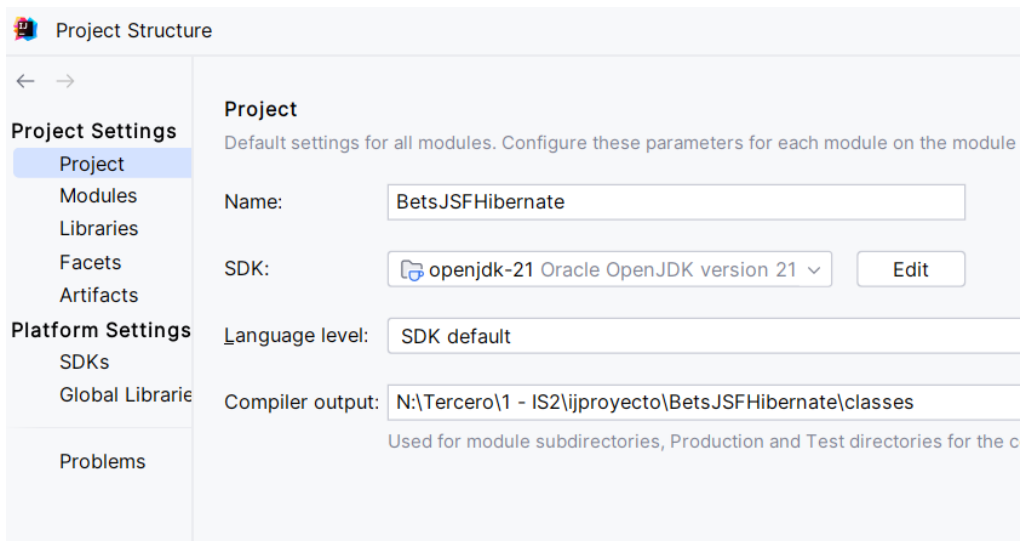
<https://github.com/nicoagr/BetsJSFHibernate>

(Las invitaciones a los repositorios de github expiran a los 15 días. Entonces, en caso de no tener acceso, escribir a naguado008@ikasle.ehu.eus ó a nico@nico.eus)

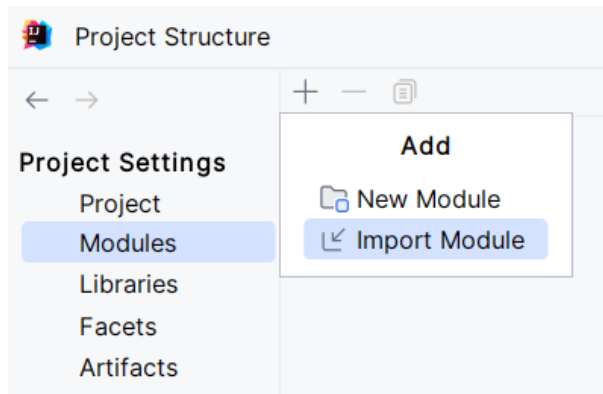
Configuración del Proyecto

Para desarrollar este proyecto, hemos decidido utilizar el entorno IntelliJ, ya que una vez que todo funcione correctamente, creemos que será la opción más cómoda y agradable para nosotros. Sin embargo, sabemos lo que eso conlleva a la hora de tener que elegir las versiones correctas para las librerías y los problemas que eso podría generarnos. La solución que hemos pensado para ahorrarnos este tedioso trabajo, es utilizar eclipse como herramienta para crear un proyecto que, siguiendo la configuración recomendada, nos asegure la compatibilidad de todas las librerías.

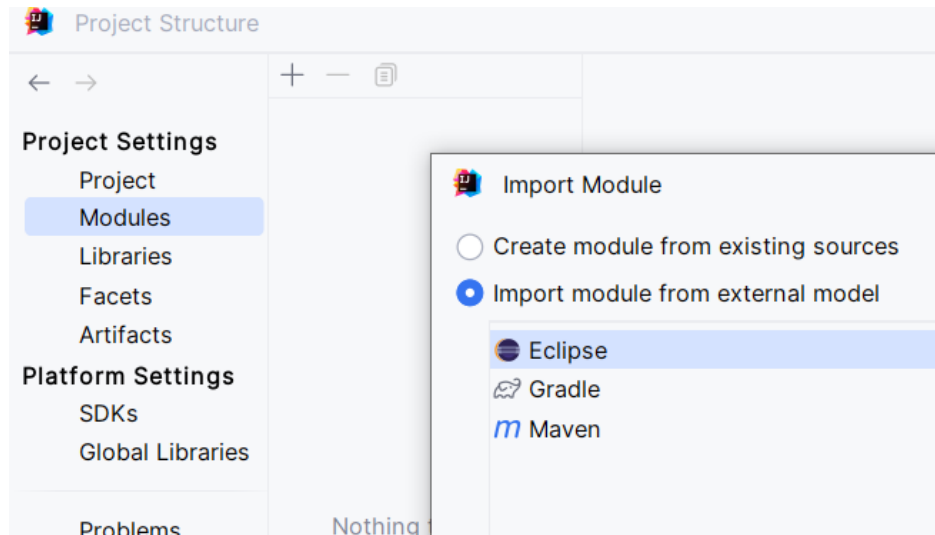
Para eso, lo primero que haremos será crear un proyecto vacío en eclipse, al que le añadiremos todas las librerías necesarias (javax faces, primefaces...). Una vez se ha creado correctamente el proyecto, llega la hora de importarlo a IntelliJ, y es aquí donde surge el primer problema. (Usaremos la última versión de IntelliJ, la IntelliJ 2023.3 Ultimate Edition) Éste no lo detecta como proyecto, sino como colección de archivos, así que tendremos que abrir la configuración del proyecto (File > Project Structure ó la tecla F4) y hacer una serie de cambios. En el primer apartado de “Project” tendremos que indicar la versión de java que queremos (funciona bien la actualmente última, la jdk-21), el lenguaje de nivel es el predeterminado de java y la carpeta de compilación será “classes”.



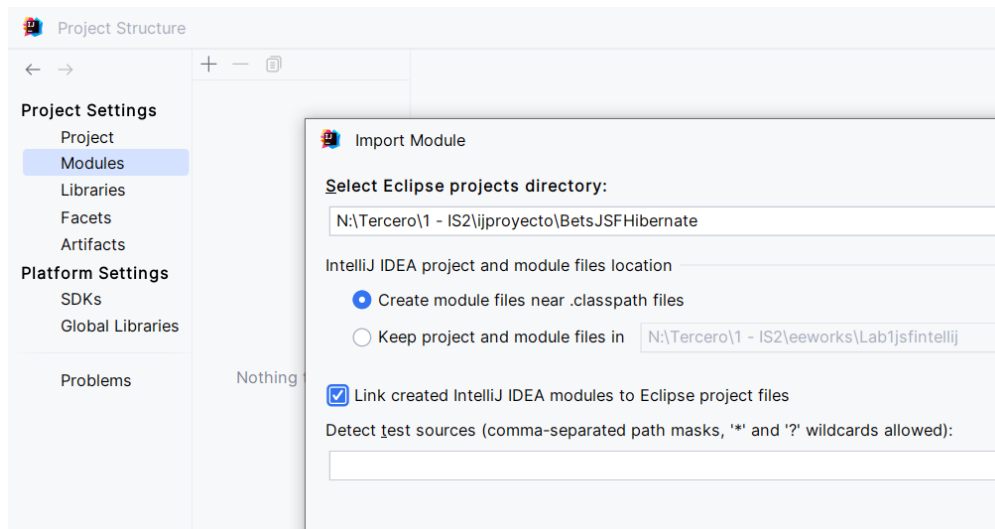
Entraremos al apartado de módulos y después de darle al icono de añadir (+) elegiremos la opción “Import Module”:

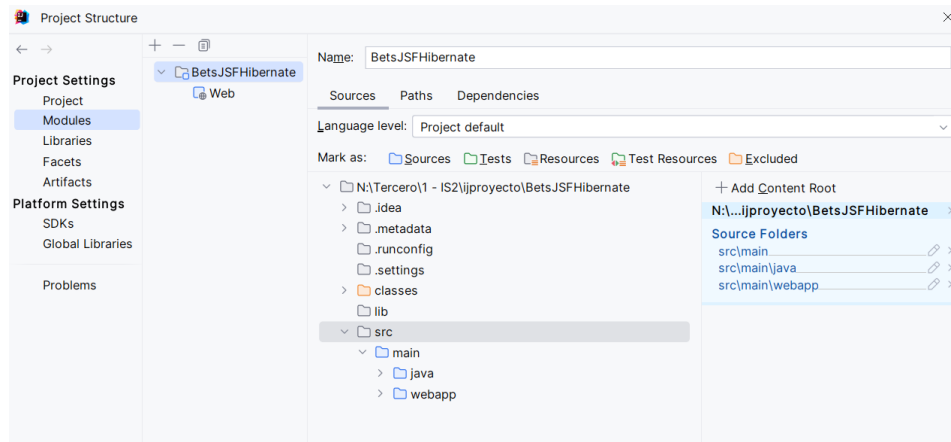


Y, después de seleccionar la carpeta del proyecto, se nos abrirá la siguiente ventana, en la que marcaremos la opción de importar un módulo desde una fuente externa (eclipse), tal y como aparece en la siguiente imagen:



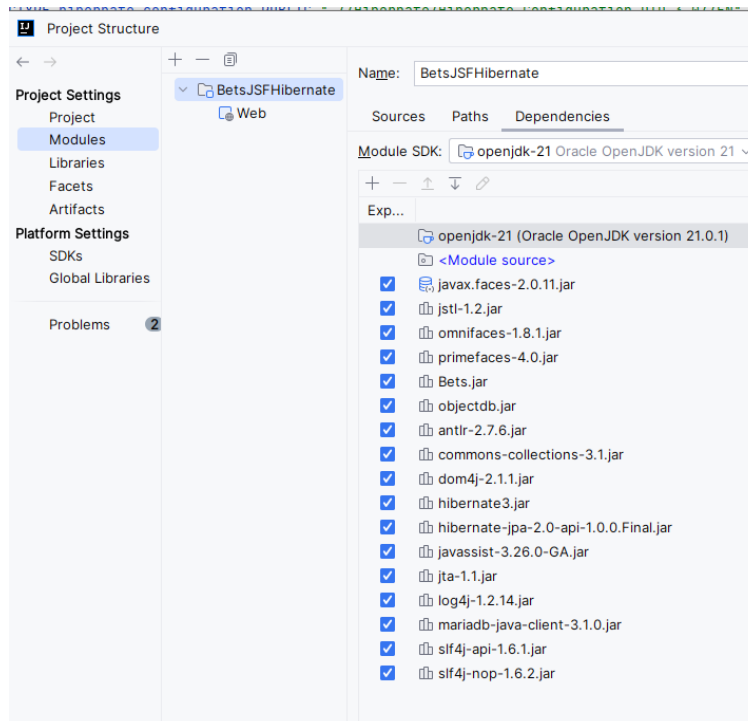
Importar un modulo desde eclipse y hemos tenido que decir que se enlacen los archivos de configuración de eclipse con los de intellij; es decir que el proyecto sea de eclipse e intellij a la vez, eso lo conseguimos dándole a la casilla “Link created IntelliJ IDEA modules to Eclipse project files” y “Create module files near .classpath files”. Además, dentro del módulo habrá que poner las carpeta “classes” como *excluded* y “src/main”, “src/main/java” y “src/main/webapp” como *carpetas raíz*.



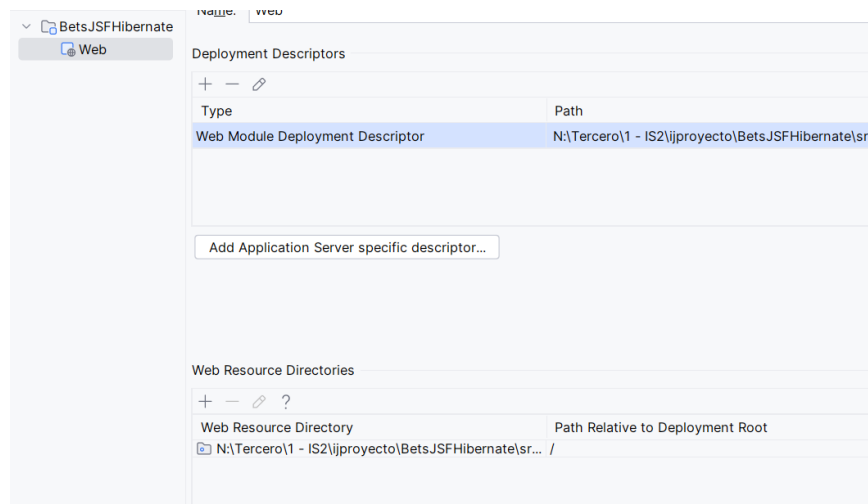


Además, en la pestaña “Paths” tendremos que indicar que queremos heredar la ruta del compilador,

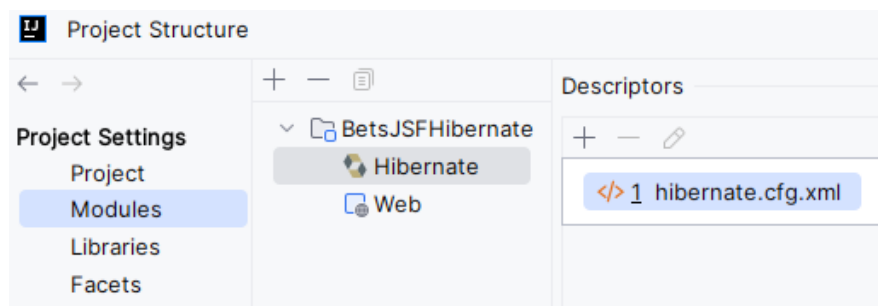
Y la pestaña de dependencias se tendrá que ver así, con todo marcado y con la SDK del módulo siendo la misma que se ha seleccionado antes (la del proyecto).



En caso de que no existiera el sub-módulo “Web”, clic derecho en BetsJSFHibernate > Añadir > Web. Ahí, habrá que poner como “Web Deployment Descriptor” el archivo web.xml y como “Web Resource Directory” el directorio webapp, para ambos casos, asegurarse de que estos directorios que elegimos sean los que se encuentran dentro de la carpeta src/main y no la carpeta web que IntelliJ genera directamente en la raíz del proyecto. En caso de que se genere esa última carpeta, habrá que eliminarla.

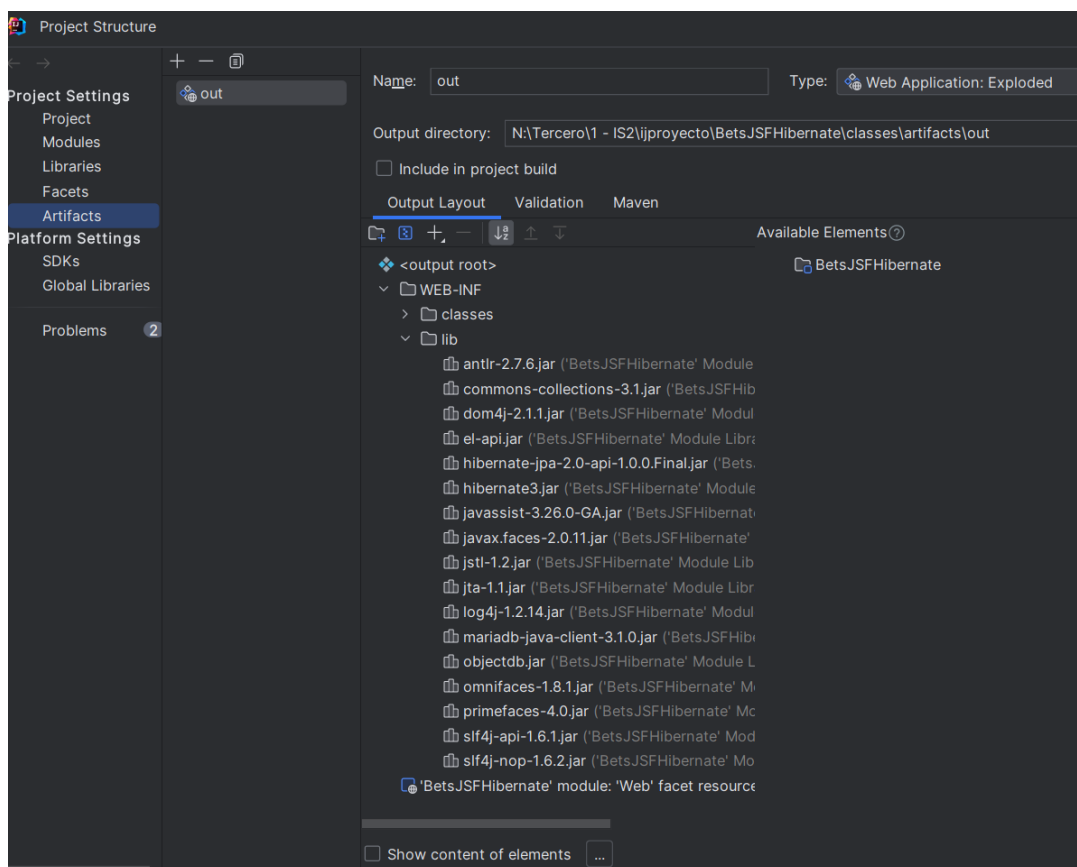
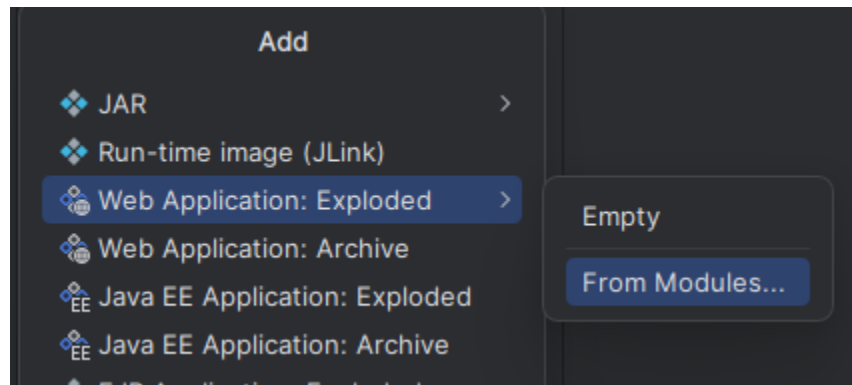


Además de añadir el descriptor de tipo web, también tendremos que añadir el descriptor de Hibernate. Clic derecho en BetsJSFHibernate > Añadir > Hibernate. Una vez aquí, tendremos que hacer clic en el “+” en la nueva ventana que se ha abierto y seleccionar el archivo hibernate.cfg.xml que IntelliJ ya ha detectado. OK y ya habremos acabado en la pestaña de “Modules”.



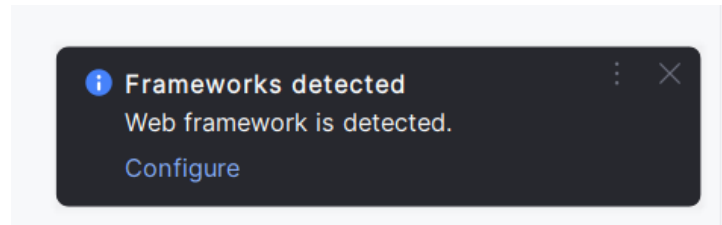
Ahora, pasando en la columna de la izquierda a la sección “artifacts”: crear nuevo artefacto web application: Archive > From modules, lo llamaremos “out”. Es muy

importante: añadir al build del proyecto y en el artefacto agregar al apartado “compilar” todo lo disponible, de tal manera que quede tal y como está en la foto.

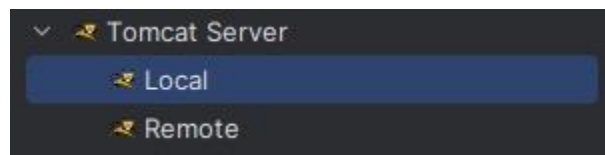


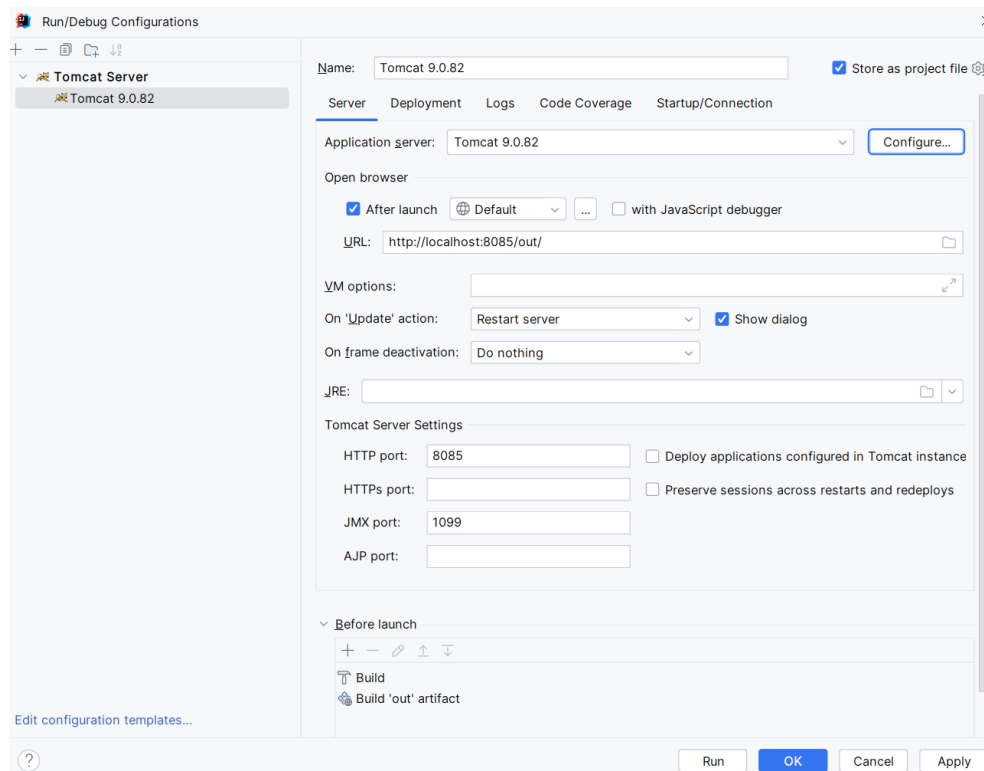
Si en algún momento sale la siguiente ventana, **NO** se debe pulsar en ‘configure’, ya que esto reorganizará el proyecto y romperá nuestra organización de carpetas. En caso de

clic accidental, volver a leer esta guía para reconfigurarlo de nuevo.

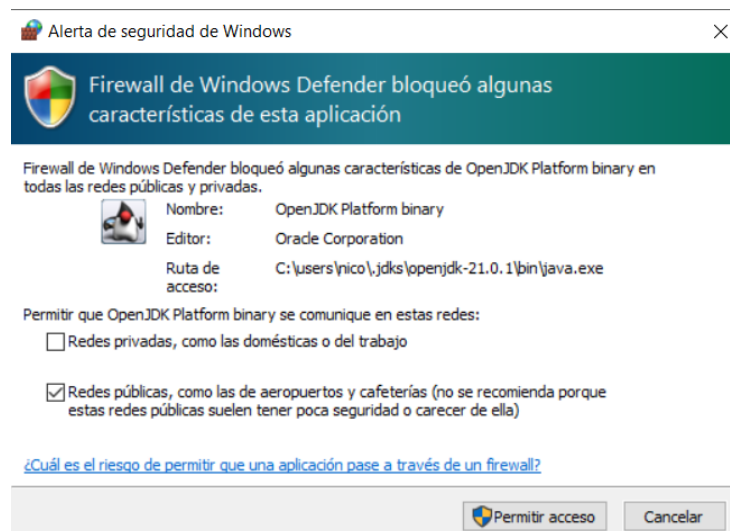


Para poder ejecutar el proyecto, habrá que configurar nuestra “Configuración de Lanzamiento”. Será un servidor Tomcat 9.0 (que tendremos que tener previamente descargado) y tendremos que indicar la ruta a nuestro artefacto que hemos compilado (/out/, porque el artefacto se llama out) y habremos de indicar también que se construye el artefacto.

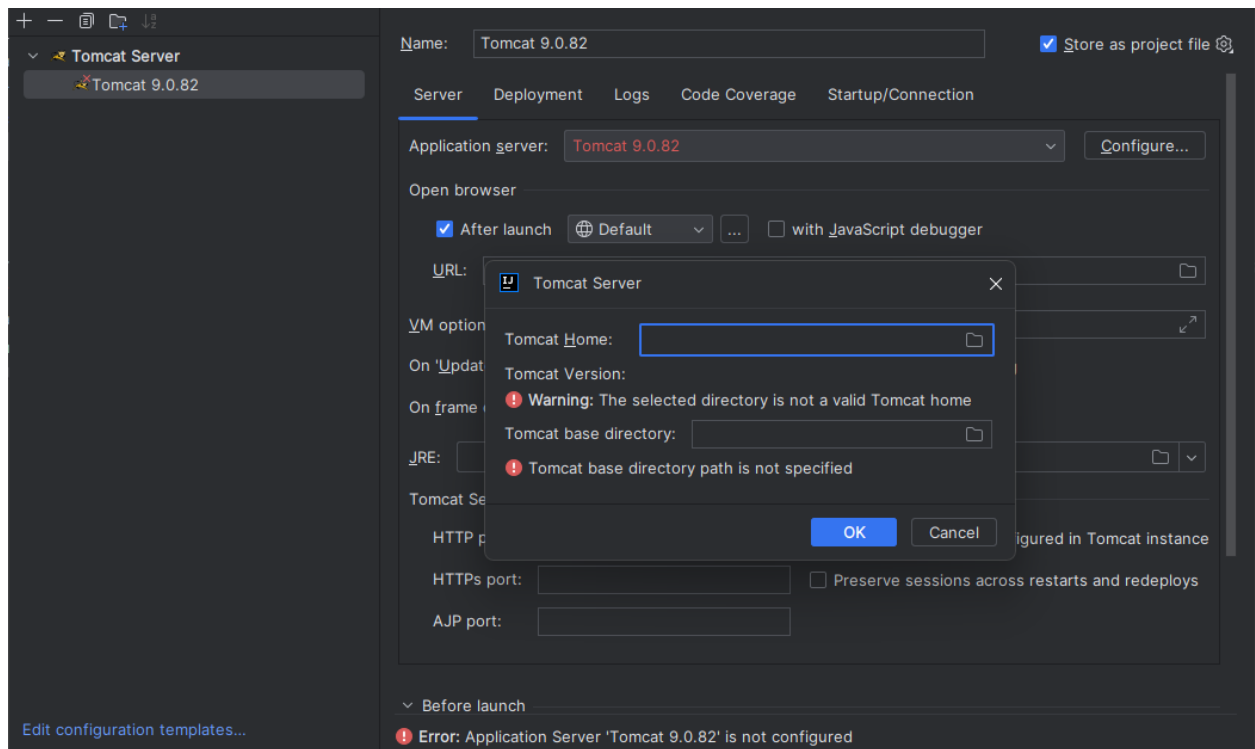




A la hora de lanzar la aplicación, si sale esta ventana entonces dar permisos de ejecución (para que el servidor de tomcat se pueda ejecutar en local) (será similar dependiendo de los sistemas operativos)

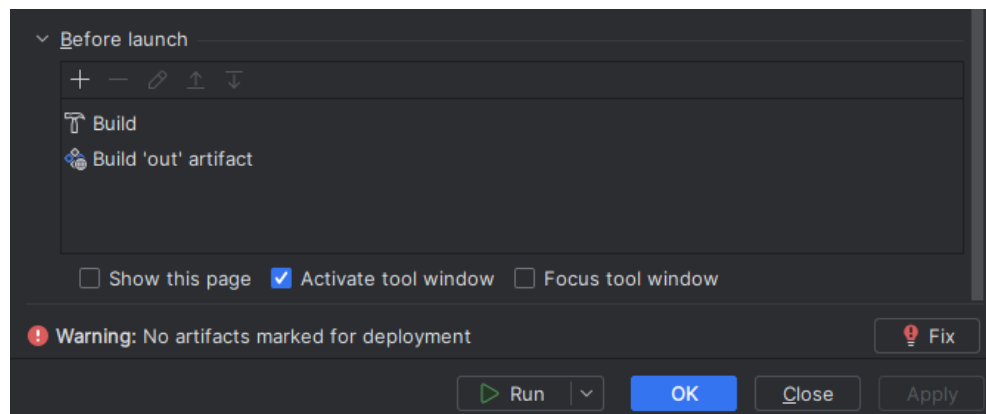


Pero, en caso de que aparezca lo siguiente:

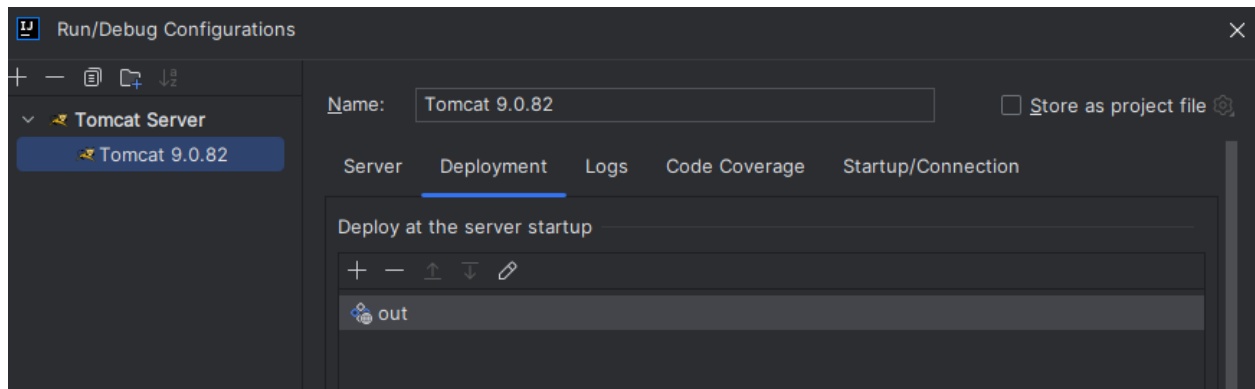


Habr  que establecer la ruta a la instalaci n tomcat en ambos campos. Estaremos usando [tomcat 9.0.82](https://tomcat.apache.org/tomcat-9.0.82/).

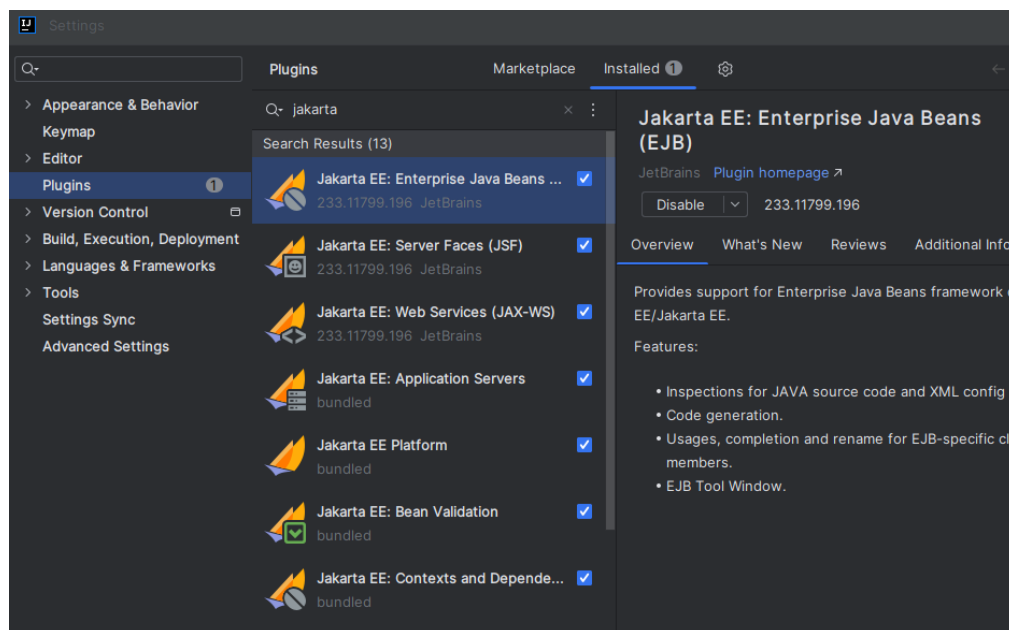
Tambi n, en caso de que aparezca el siguiente aviso:



simplemente le daremos al bot n fix y se a adir  lo siguiente a la pesta a “Deployment”:



Además, si queremos disfrutar al máximo del autocompletado que ofrece IntelliJ, necesitaremos su versión ultimate e instalar desde el menú Settings > Plugins todos los plugins relacionados con Jakarta EE.



Después de todo esto y para asegurarnos de que la aplicación funciona bien, haremos un reinicio de IntelliJ para que se apliquen bien todos los cambios y re-escanee todos los archivos.

Requisitos del Sistema

Para que la aplicación pueda funcionar, es necesario tener un servidor de MariaDB

(MySQL) (en concreto, el proyecto funciona bien con las versiones 10.11.5 ó 10.4.32) corriendo en segundo plano (localhost), en el puerto 3306 con usuario “root” y contraseña “admin”.

Además, será necesario que exista una base de datos llamada “bets” y que el usuario root tenga acceso a ella. Si no existiera, se puede crear abriendo una consola de mysql y ejecutando el comando “create database bets;”

```
# mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 22
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database bets;
Query OK, 1 row affected (0.002 sec)

MariaDB [(none)]> _
```

Además, como información útil al usuario, cabe destacar que la información ya pre-establecida se creará con fecha establecida a el día 15 del mes siguiente a cuando se esté ejecutando la aplicación.

Implementación

Login y Registro

En cuanto al desarrollo de la funcionalidad de “login” y “registro” se ha optado por crear una clase “User” que contenga toda la información relativa al usuario en sí. Los usuarios se guardarán automáticamente en la base de datos.

Se han insertado tres usuarios en la base de datos ya predefinidos para pruebas rápidas.

Usuario	Contraseña
nico	nico
xio	xio
eider	eider

Contraseñas Hasheadas

Pero claro, guardar contraseñas en una base de datos en texto plano es un fallo de seguridad de categoría extrema. Entonces, hemos optado por *cifrar* las contraseñas usando el algoritmo “BCrypt”. Una excelente implementación de este algoritmo en java es [jBCrypt](#), creada por Damien Miller. Entonces, gracias a esta librería, al registrar un usuario generaremos un “hash” de la contraseña:

```
String hashed = BCrypt.hashpw(passtxt, BCrypt.gensalt());
```

Y al hacer login un usuario comprobaremos la contraseña introducida contra ese hash:

```
if (BCrypt.checkpw(passtxt, hash)) {  
    return "ok";  
}
```

Persistencia del Usuario

Se ha hecho disponible para uso en cualquier parte de la aplicación al usuario que ha iniciado sesión en el bean encargado del login.

```
private User usuariologado;
```


Apostar

A la hora de implementar el caso de uso de las apuestas, se ha optado por extender una vista ya requerida por la aplicación. Entonces, aprovechando el “Consultar Preguntas”, una vez seleccionado un evento, aparecerá un botón de “Apostar”. Entonces, se le redirigirá al usuario a una página aparte.

A qué apostar

Se ha optado por hacer las preguntas sencillas: Esto es, que el resultado de una pregunta pueda ser o equipo 1 o equipo 2. De esta manera, se simplifica el proceso de apostar. Cada pregunta contendrá una “cuota si 1” o “cuota si 2” (Corresponderán al multiplicador que nos podremos encontrar en una casa de apuestas habitual)



¿AMBOS EQUIPOS MARCARÁN? 	
SI	2,05
NO	1,75

Pregunta real en casa de apuestas “Kirolbet” para un partido de la Real Sociedad.

Cuestiones Monetarias

Se aprovechará el atributo apuesta mínima de “Question” y se añadirá un atributo “Money” en el usuario. A todos los usuarios registrados se les regalarán 10 rupias indias para empezar a apostar. Se deja como ejercicio al lector el extender la aplicación para crear algún caso de uso que permita ganar dinero al usuario (por ejemplo, establecer resultado de pregunta).

Como moneda, se ha elegido usar la rupia india (₹) meramente por el aspecto estético del símbolo.

Query Question

Esta funcionalidad tiene su propia vista, en la que habrá un calendario (<p:calendario>) y dos tablas de datos (<p:dataTable>) inicialmente vacías. El usuario podrá seleccionar una fecha y, si hay apuestas para esa fecha, se mostrarán en la primera tabla de datos; posteriormente, se podrá pinchar sobre cada una de las apuestas para poder visualizar sus preguntas correspondientes en la segunda tabla.

En un principio se pensó en utilizar elementos tipo <h:messages> para hacer saber al usuario que no había eventos o preguntas para la fecha o evento seleccionado. Sin embargo, después de varios momentos de prueba y error (explicados en el apartado [problemas encontrados](#)) se decidió que esto se indicase dentro de la propia tabla en la que deberían mostrarse los datos, modificando el siguiente atributo:

```
<p:dataTable emptyMessage="No existen preguntas para el evento  
seleccionado.">
```

Se ve de la siguiente forma:

# Pregunta	Pregunta
No existen preguntas para el evento seleccionado.	

Además, reutilizamos las clases ‘Event’ y ‘Question’ del proyecto *Bets2021*, que guardan la información de los eventos y preguntas de los eventos, para poder así recuperar desde la base de datos estos eventos y preguntas y poder mostrarlos.

Create Question

Esta última funcionalidad también está realizada en una vista totalmente distinta.

Destacamos el calendario (<p:calendar>) con el que elegiremos la fecha de la cual obtendremos los eventos y serán mostrados en una *droplist* (<h:selectOneMenu>, <f:selectItems>). En caso de que no haya eventos en esa fecha aparecerá la lista vacía. El usuario podrá seleccionar uno de los eventos disponibles y seguir rellenando los siguientes campos, *Pregunta*, *Apuesta mínima*, *Cuota1* y *Cuota2*. Se comprobará el formato de los campos, *Integer*, *Float*, y que se hayan rellenado todos los campos. Además se comprobará que la fecha seleccionada sea igual o posterior a la actual. Si el usuario rellena todos los campos pero selecciona una fecha sin eventos, se le notificará de ello.

Cabe destacar que reutilizaremos las clases ‘Event’ y ‘Question’ del proyecto *Bets2021*, ya que tendremos que obtener la información de los eventos para recuperarlos desde la base de datos y crear una nueva pregunta con la información que te proporciona el usuario para insertarla en la base de datos.

Experiencia con IntelliJ

Así, en general, IntelliJ es un buen IDE. Pero tiene una curva de aprendizaje un poco alta y para poder conseguir que funcione todo igual que en Eclipse hay que pelearse mucho con las configuraciones. Con Eclipse se puede seguir un laboratorio guiado, mientras que con IntelliJ hay que *entender* realmente lo que estás haciendo, qué se compila, en qué orden, cuando, qué hay que incluir en el fichero de compilación, qué módulos usar, cómo guardar las librerías, etc...

Además, de vez en cuando, recibe una actualización y “detecta” que el proyecto que estás usando es de tipo web y crea su estructura de carpetas (distinta a la nuestra) y cambia de configuración. Entonces, le tienes que decir manualmente que no. Viene muy bien para frameworks, pero cuando te sales un poco de los esquemas, se puede volver un poco pesado.

Aún así, la experiencia ha sido positiva y todos los beneficios que ofrece han compensado: Debugging mucho más completo, asistencia al programar, copiado / pegado inteligentes, autocompletado perfecto, etc...

Es, como siempre, algo que valorar. Tiene sus pros y sus contras. Nosotros, en nuestra opinión, creemos que **sí** que merece la pena.

Problemas encontrados

Una de las complicaciones que nos han surgido a la hora de realizar el proyecto ha tenido que ver con ajax. Queríamos que al seleccionar una fecha en el calendario, en caso de que no hubiera eventos para esa fecha, se mostrase un mensaje (<h:message>). El problema que teníamos al querer llevar a cabo las acciones mediante ajax, el mensaje no se mostraba a pesar de que todo pareciese correcto, después de probar distintas soluciones, finalmente llegamos a la correcta: usar un botón en lugar de la tecnología ajax. De esta forma, aunque pueda ser más pesado para el usuario, para poder ver el resultado de elegir una fecha del calendario tendremos primero que pulsar el botón “Consultar” en lugar de visualizarlo directamente después de pulsar en la fecha deseada.

Otro de los problemas encontrados ha sido con la comunicación entre beans. Se ha intentado hacer uso de las anotaciones que ofrece jsf (@ManagedProperty), pero no ha resultado posible. Después de mucho probar con distintas versiones y librerías, con java enterprise y con módulos externos, no ha quedado más remedio que establecer algunos atributos como estáticos dentro de los dominios. Se ha considerado que es la solución más efectiva para el tiempo que era necesario invertir.

Asimismo, nos hemos encontrado con un problema a la hora de insertar la pregunta creada en la base de datos *bets* en la funcionalidad *Create Question*. En la clase *Question* el atributo *event* tenía en la etiqueta asignada *cascade=CascadeType.PERSIST*. Esto nos daba fallo ya que si el evento existía no se podría insertar a la BD la pregunta relacionada a ese evento. Para ello se debía cambiar a *cascade=CascadeType.MERGE* y ya no tendríamos problemas a la hora de insertarlos a la base de datos.

Tiempo invertido

Miembro del Grupo	Tiempo
Nicolás Aguado	~30 horas
Eider Fernández	~17 horas
Xiomara G. Cáceres	~14 horas