Introducción

El propósito de este proyecto, diseñado en una arquitectura de software de tres niveles **apuestas**será el desarrollo de un sistema de información que los gestione.

ahora Hasta que (0 iteración) hecho del proyecto fuente el código https://github.com/jononekin/Apuestas21 puedes encontrarlo en el enlace.

Este documento proporciona la información desarrollada hasta el momento (Iteración 0):

1. Descripción del sistema de información.
2. Modelo de dominio.
3. Modelo de casos de uso.
4. Diagrama de secuencia del caso de uso "Crear pregunta".
5. Código fuente e instalación de eclipse.
6. Estructura de la aplicación.
7. Instrucciones para iniciar la aplicación.
8. Código fuente.
9. La documentación mínima que se debe presentar en cada iteración.

1. Descripción del sistema de información

Queremos desarrollar una aplicación para gestionar apuestas deportivas.

El administrador del sistema creará eventos del sistema (por ejemplo, partido Real Madrid/ Barcelona), preguntas de eventos (por ejemplo, ¿quién ganará el partido? o ¿quién marcará el primer gol?) y predicciones para cada pregunta con sus ganancias. Por ejemplo, para la pregunta anterior se pueden crear 2 predicciones con beneficios diferentes:

Evento: Real Madrid vs. Barcelona
Pregunta: ¿Quién ganará? Regular 1-X-2
Apuesta mínima: 2 euros
Predicción: Victoria del Real
Madrid: 1,20 €

Evento: Real Madrid vs. Barcelona
Pregunta: ¿Quién ganará? Regular 1-X-2
Apuesta mínima: 2 euros
Predicción: Barcelona
Beneficio: 2€

Las características de un evento son su descripción y fecha. Cada pregunta tiene tres características: la descripción de la pregunta, el evento en el que se basa y la cantidad mínima de apuesta (por ejemplo: 2 euros). Finalmente, se describe un pronóstico sobre el resultado de una pregunta y la ganancia en euros por la apuesta en el boleto del euro.

Los usuarios registrados pueden apostar en una predicción si se cumplen las siguientes condiciones:

a) el evento en el que se basa la predicción aún no ha ocurrido, b) la cantidad de apuesta es mayor que la cantidad de apuesta mínima definida en esa pregunta y c) hay suficiente dinero en su cuenta para hacer esa apuesta.

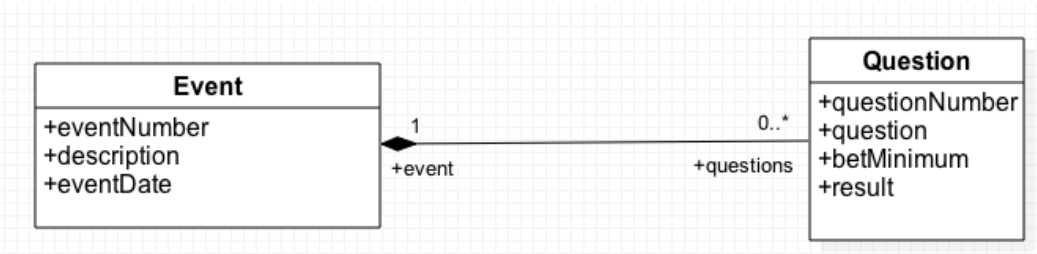
Un ejemplo de apuesta podría ser:

Evento: Real Madrid vs. Barcelona
Pregunta: ¿Quién ganará? Predicción
Regular 1-X-2: Real Madrid
Importe de la apuesta: 30 euros. (si acierta ganará 36 euros, de lo contrario lo perderá todo).

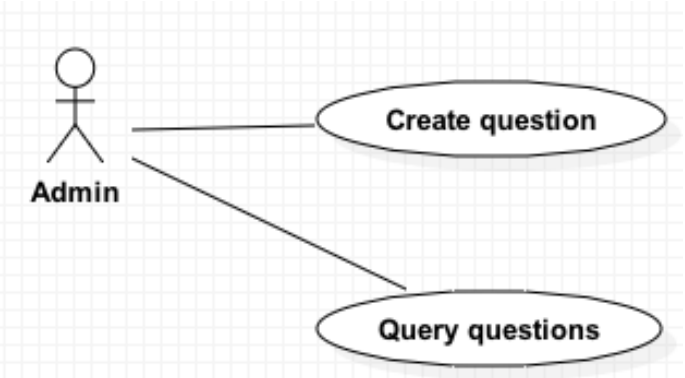
Cuando finaliza el evento, el administrador ingresa los resultados de las preguntas (por ejemplo, Real Madrid y Barcelona han empatado), y el sistema actualiza sus cuentas de acuerdo con las apuestas realizadas por los clientes. Si el resultado coincide con la predicción de la apuesta, la cuenta del usuario se incrementará en la cantidad de la apuesta multiplicada por la cantidad ganadora; de lo contrario, perderá toda la apuesta.

En cualquier momento, los usuarios deben poder consultar las apuestas realizadas, su estado (realizadas o no) y su ganancia (si finalizadas).

2. Modelo de dominio



Modelo de casos de uso



Flujos de eventos

Query question Flow of events

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
4. *Admin* selects an **event**
5. *System* displays the **questions** asssociated to the selected **event**

Alternative flow

1. There are no events on this date. Events cannot be shown.
2. There are no questions associated to the event. Questions cannot be shown.

Create question Flow of events

Basic Flow

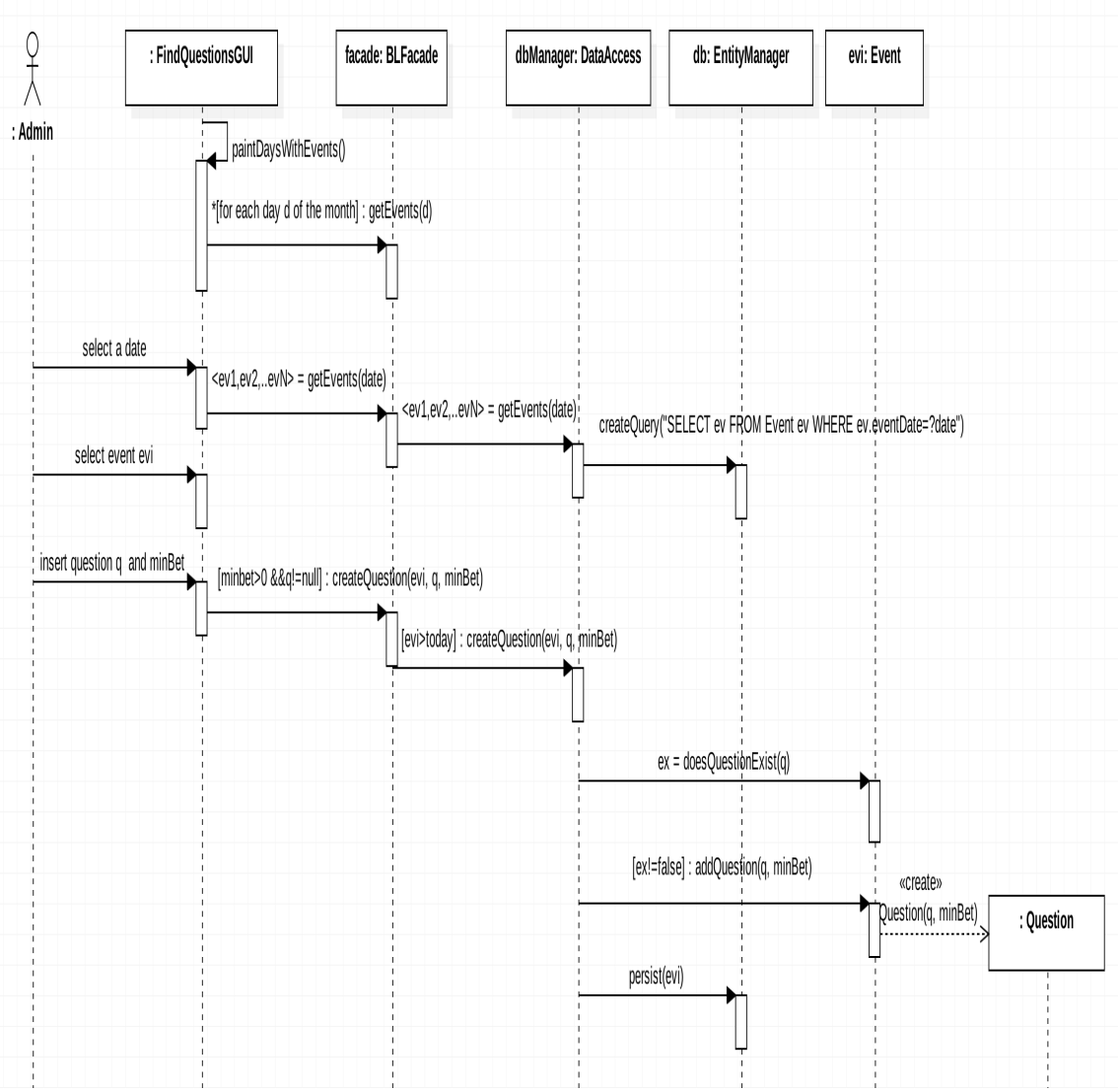
1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
4. *Admin* selects an **event**
5. *Admin* introduces a **question** and a minimum **betting price**
6. *System* adds the new **question** with a minimun **betting price** to the selected **event**

Alternative flow

1. There are no events on this date. Question cannot be added.
2. The question field is empty. Question cannot be added.
3. The minimum betting price is empty or it is not a number. Question cannot be added.
4. Event date has already finished (event day is before current day). Question cannot be added.

4. Diagrama de secuencia del caso de uso "Crear pregunta"

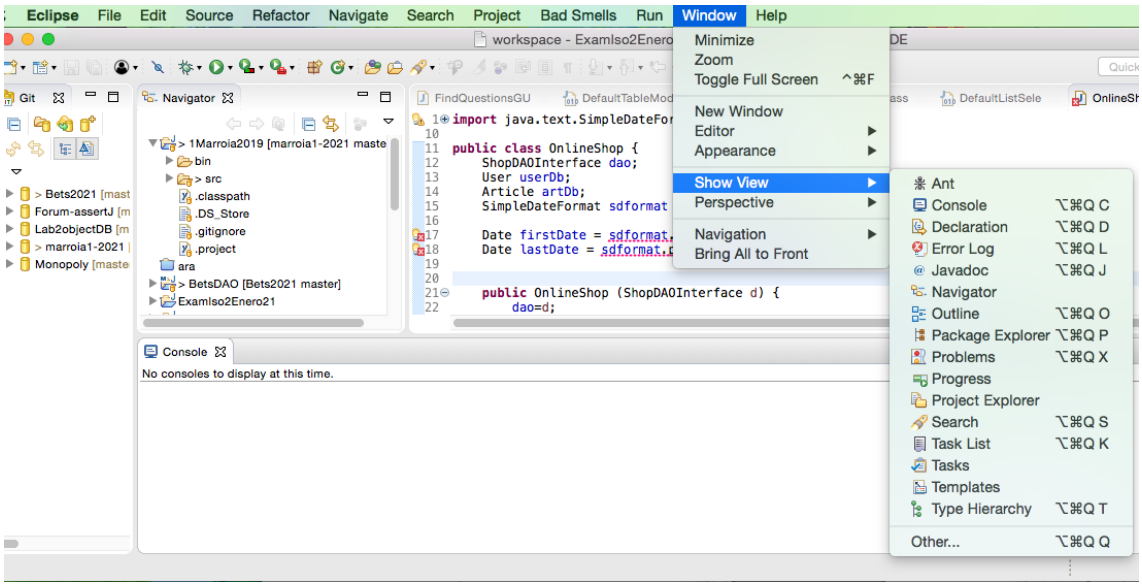
La aplicación está diseñada en una arquitectura de tres niveles. Se ha utilizado el lenguaje AWT/ SWING para definir las interfaces gráficas de usuario, se ha desarrollado el acceso a la lógica de negocio a través de Web Services, y para el nivel de persistencia se ha utilizado la base de datos objectdb. Para recoger todas las operaciones de la Lógica de Negocio, y siguiendo las recomendaciones del patrón GRASP, se ha definido una clase Fachada con todas las operaciones. A continuación se muestra un diagrama de secuencia de un caso de uso del sistema:


: Admin
: FindQuestionsGUI
facade: BLFacade
dbManager: DataAccess
db: EntityManager
evi: Event
paintDaysWithEvents()
*[for each day d of the month] : getEvents(d)
select a date
<ev1,ev2,..evN> = getEvents(date)
<ev1,ev2,..evN> = getEvents(date)
createQuery("SELECT ev FROM Event ev WHERE ev.eventDate=?date")
select event evi
insert question q and minBet
[minbet>0 &&q!=null] : createQuestion(evi, q, minBet)
[evi>today] : createQuestion(evi, q, minBet)
ex = doesQuestionExist(q)
[ex!=false] : addQuestion(q, minBet)
«create»
Question(q, minBet)
: Question
persist(evi)

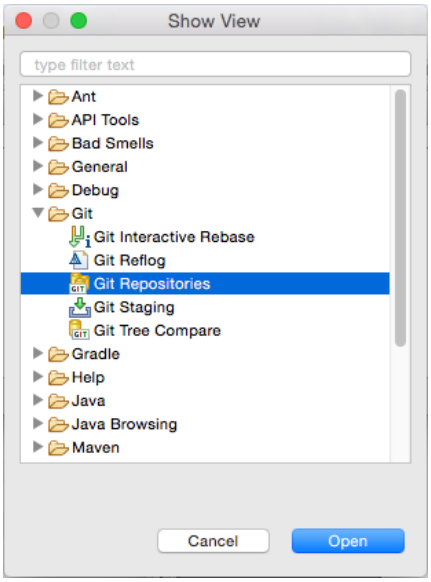
5. Código fuente e instalación de eclipse. Hecho hasta ahora (0 iteraciones). https://github.com/jononekin/Apuestas21 el código fuente del proyecto encontrarlo en el enlace.

Proyecto inicial código fuente https://github.com/jononekin/Apuestas21 hay
Primero ve a esta dirección, y está en la parte superior presionando el botón Fork
Siga los pasos requeridos para copiar el proyecto a su cuenta. Como resultado, tendrás una copia del proyecto en tu cuenta de Github.

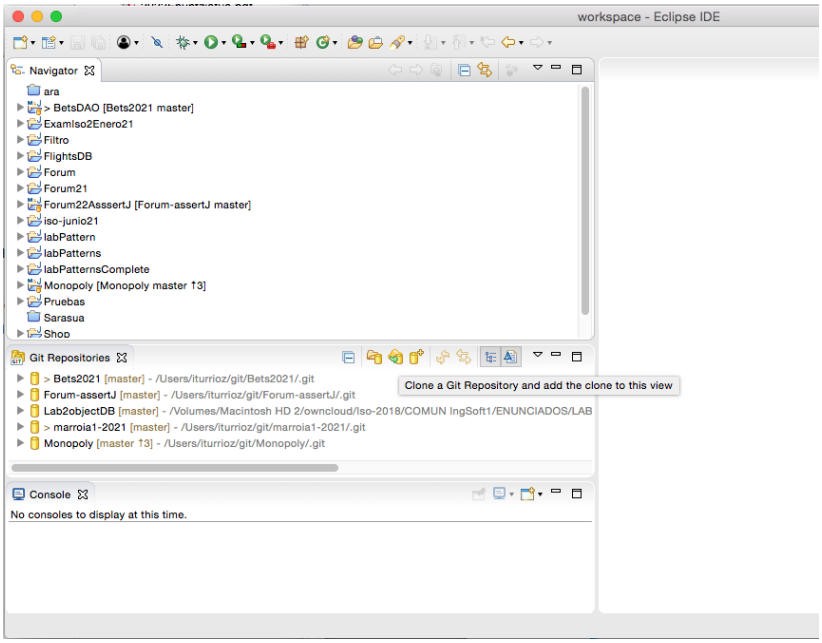
Para instalar el proyecto copiado en Eclipse, primero *Ventana->Mostrar vista-> Otro* elegir:



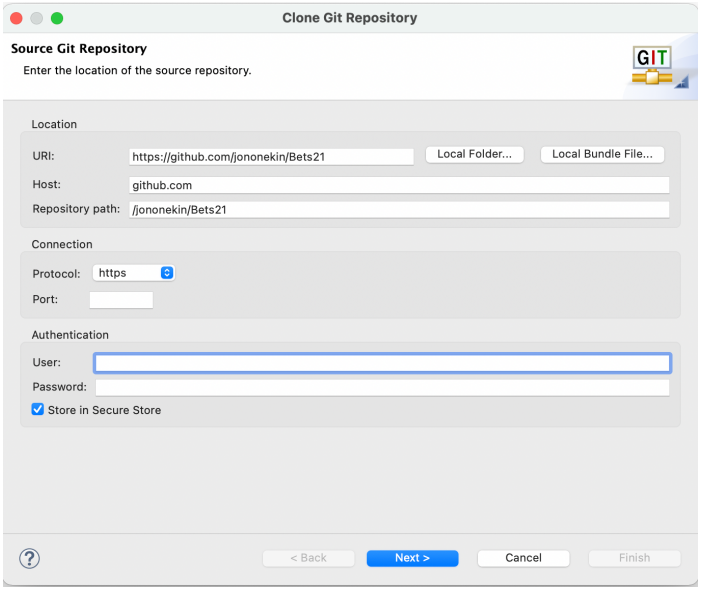
y abra la vista Git->Git Repositories como se muestra en la siguiente imagen:



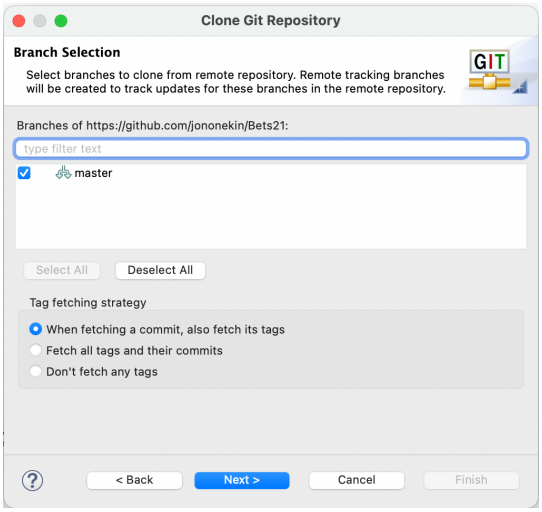
A continuación, de Eclipse**Repositorios Git**en la ventana**Clone un repositorio de Git y agregue el clon a esta vista**botón seleccionar



La URL del repositorio remoto que copió (es decir, la dirección de la copia que creó en su cuenta, es decir, http://github.com/$yourName$/Bets21),

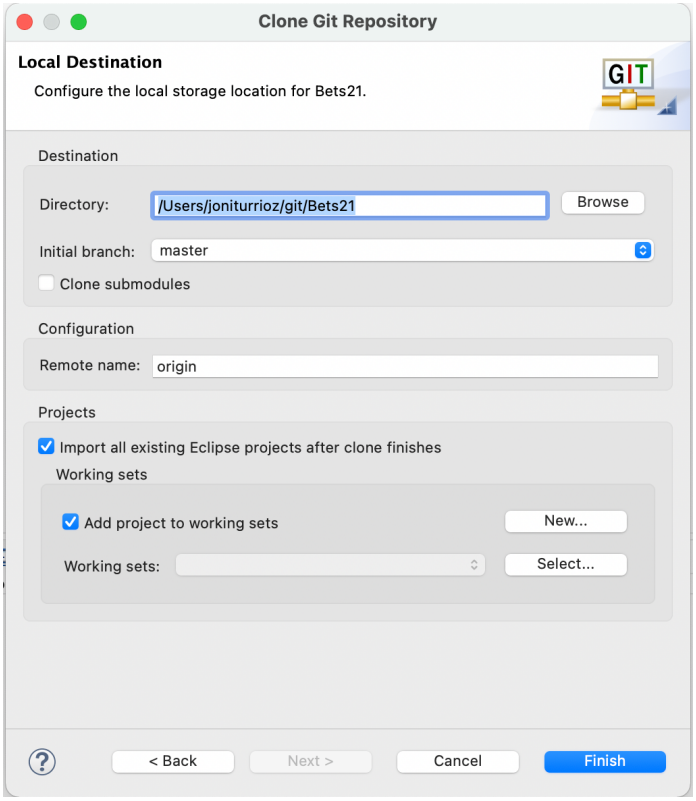


y**próximo**prensa Siguiente**Ventana de selección de rama**en la imagen**próximo**hazlo



Selección de sucursalesventana, qué parte del repositorio remoto es nuestro repositorio para indicar que queremos copiar y sincronizar localmente.

Eclipse necesita saber dónde dejar su repositorio local (la copia que se sincroniza con el repositorio remoto), esto**Destino local** en la ventana aparece**Es muy importante**"Seleccione Agregar proyecto a conjuntos de trabajo" antes de hacer clic en Finalizar.



Destino localventana, la dirección del repositorio remoto clonado para expresar

Esto hará que el proyecto aparezca entre sus proyectos de eclipse. Si desea obtener más información sobre Github (por ejemplo, cargar sus cambios en Eclipse a Github), puede consultar el Laboratorio 8 en Egel.

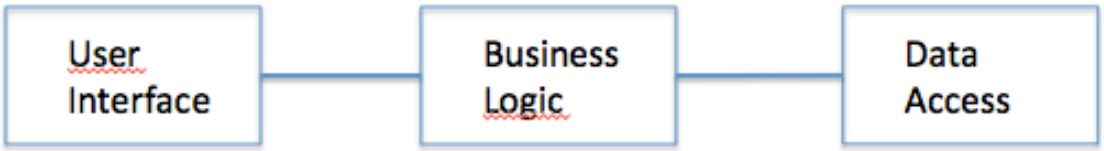
Habrá diferentes carpetas en el proyecto:

una. En la carpeta (src/main/java), estará el código fuente Java del proyecto.

b. La carpeta (src/main/resources) contendrá los recursos adicionales que necesita el proyecto, es decir, las bibliotecas JCalendar y objectDb para trabajar con una base de datos y un calendario, la base de datos (bets.temp) y el archivo de configuración del proyecto (config. xml).

6. Estructura de la aplicación.

La aplicación se divide en tres niveles lógicos: interfaz gráfica, lógica empresarial y acceso a datos.



La aplicación está organizada en 5 paquetes java:

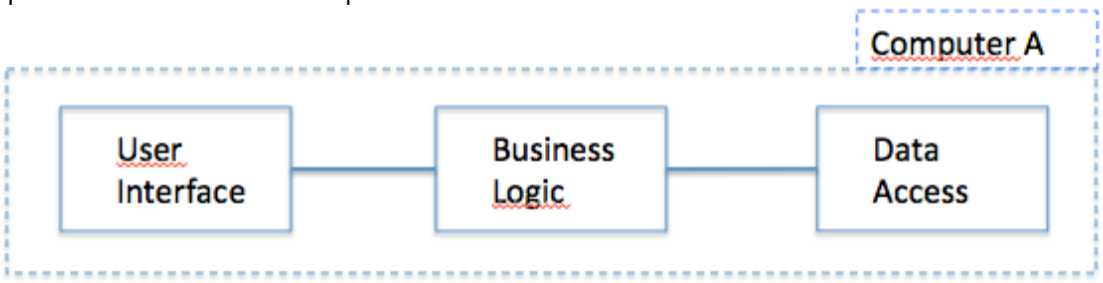
nosotros: clases de interfaz gráfica (nivel de interfaz de usuario). **lógica de negocios**: clases de lógica empresarial (nivel de lógica empresarial). **fechaAcceso**: clases de base de datos (nivel de acceso a datos). **dominio**: Clases definidas en el modelo de dominio. **excepción**: Clases que definen excepciones.

Cada nivel se puede ejecutar en la misma máquina o en una computadora diferente. Esta informacion*config.xml*se define en el archivo Veamos las opciones que existen.

7. Instrucciones para iniciar la aplicación.

Opción 1: Toda la aplicación en un nivel físico.

En esta opción, los niveles de Presentación, Lógica de negocios y Acceso a datos se implementan en la misma computadora.



Para hacer esto, el archivo config.xml debe configurarse de la siguiente manera:

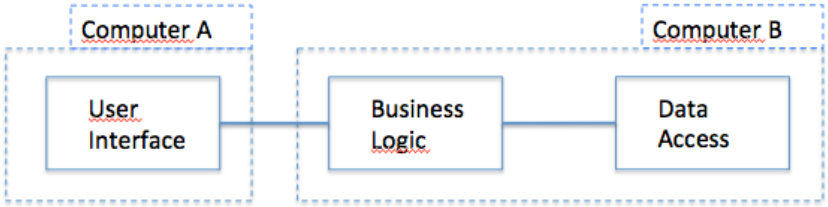
```
<?versión xml="1.0" codificación="UTF-8"?>
<config>
  <negocioLógica local="verdadero">
            <NodoLógicoNegocio>0.0.0.0</NodoLógicoNegocio>
            <PuertoLógicoNegocio>1099</PuertoLógicoNegocio>
            <NombreLógicoNegocio>Apuestas</
  NombreLógicoNegocio> </negocio lógico>
  <base de datos local="verdadero">
            <nodobase>0.0.0.0</nodobase>
            <nombreArchivoBD>bets.temp</
            nombreArchivoBD> . . . . . .
  </base de datos>
</config>
```

En esta situación, la lógica empresarial de las presentaciones (**<Lógica empresarial>**) se accede localmente (**local="verdadero"**) y base de datos de lógica de negocios (**<base de datos>**) al que también accede localmente (**local="verdadero"**) sabemos.

Ejercicio: Después de configurar el archivo config.xml*nosotros*en el paquete **Lanzador de aplicaciones**ejecutar la clase

Opción 2: Aplicación en ambos niveles físicos.

En esta opción, el nivel de presentación se implementa en una máquina y los niveles de lógica empresarial y acceso a datos se implementan en una computadora diferente.



Para iniciar la aplicación.

1. Defina la configuración de la aplicación modificando el archivo config.xml:

```
<?versión xml="1.0" codificación="UTF-8"?>
<config>
    <businessLogic local="falso">
      <NodoLógicoNegocio>0.0.0.0</NodoLógicoNegocio>
      <PuertoLógicoNegocio>1099</PuertoLógicoNegocio>
      <NombreLógicoNegocio>Apuestas</
     NombreLógicoNegocio> </negocio lógico>
    <base de datos local="verdadero">
      <nodobase>0.0.0.0</nodobase>
      <dbFilename>bets.temp</dbFilename>
    .....
    </base de datos>
</config>
```

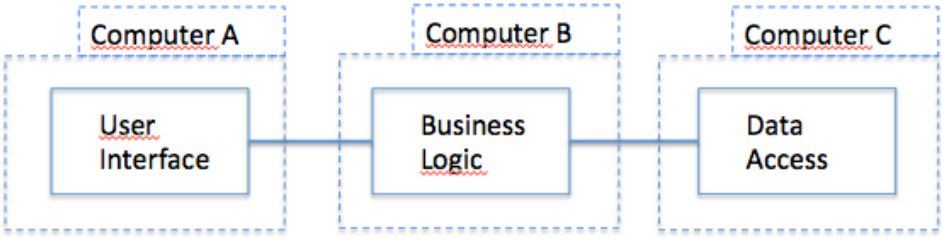
En esta configuración, detectamos que businessLogic no está en las instalaciones (**businessLogic local="falso"**). Como no es local, es necesario saber dónde se encuentra la lógica de negocios, es decir: a) En qué máquina se encuentra la lógica de negocios (**<NegocioNodoLógico>**), (0.0.0.0. indica la dirección del ordenador en el que estamos), qué puerto**(<BusinessLogicPort>)**y cual es el nombre del servicio (**<nombre de la lógica empresarial>**).

2. Ejecute la lógica de negocios. Para eso*lógica de negocios*en el paquete **BusinessLogicServer**ejecutar la clase El servidor de Business Logic se iniciará (como un servicio).

3. Ejecutar la presentación, que es la anterior*nosotros*del paquete**Lanzador de aplicaciones** ejecutar la clase y demostrar los casos de uso anteriores.

Ejercicio:¿Cómo ejecutaría la lógica comercial existente en otra computadora? Prueba entre dos ordenadores.

Opción 3: Aplicación en tres niveles físicos.



Esta opción implementa la capa de presentación en una computadora, la capa de lógica de negocios en otra computadora y la capa de acceso a datos en otra computadora.

Para iniciar la aplicación.

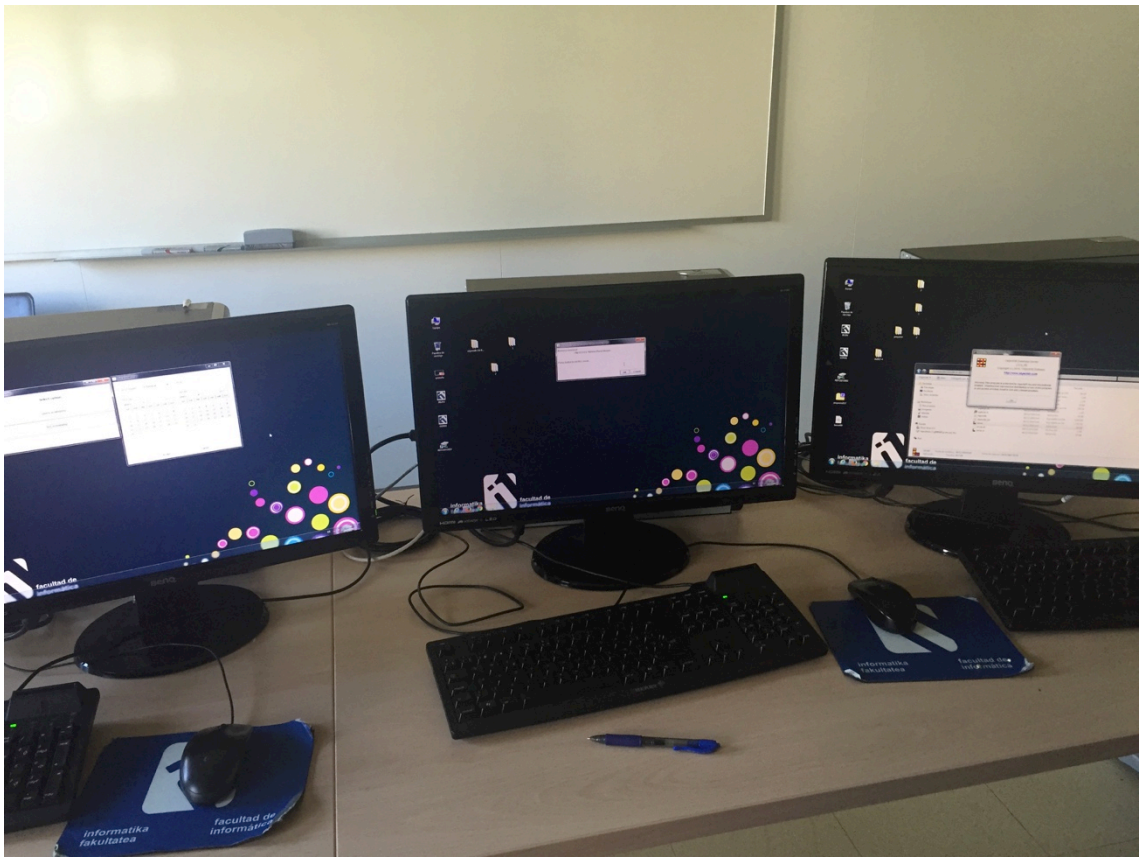
1. Defina la configuración de la aplicación modificando el archivo config.xml**<base de datos local="verdadero">**donde pone**<base de datos local="falso ">**poniendo

```
<?versión xml="1.0" codificación="UTF-8"?>
<config>
                <businessLogic local="falso">
                  <NodoLógicoNegocio>0.0.0.0</NodoLógicoNegocio>
                  <PuertoLógicoNegocio>1099</PuertoLógicoNegocio>
                  <NombreLógicoNegocio>Apuestas</
                 NombreLógicoNegocio> </LógicaNegocio>
                <base de datos local="falso">
                  <nodobase>0.0.0.0</nodobase>
                  <dbFilename>bets.temp</dbFilename>
                .....
                </base de datos>
</config>
```

2. Ejecute el servidor de la base de datos. Para eso,*fechaAcceso*en el paquete **ObjectdbManagerServerObjectdbManagerServer**ejecutar El servidor de la base de datos se iniciará (como un servicio).
3. Ejecute el servidor de lógica empresarial, anterior*lógica de negocios*en el paquete **BusinessLogicServer**ejecutando la clase.
4. Finalmente, ejecute la presentación, es decir*nosotros*del paquete **Lanzador de aplicaciones**el archivo

Ejercicio:

1. ¿Cómo implementaría la presentación y la lógica de negocios en una computadora y la base de datos en una máquina diferente? Prueba entre dos ordenadores.
2. ¿Cómo implementaría su aplicación en tres equipos diferentes? Eventualmente, debería aparecer la ejecución que se muestra en la siguiente imagen. En la computadora de la derecha, el servidor de datos se está ejecutando. En la computadora del medio, el servidor de lógica de negocios y, a la izquierda, la interfaz gráfica.


informatika fakultatea
facultad de informática

8. La documentación mínima que se debe presentar en cada iteración.

El proyecto se desarrollará durante la asignatura en 3 iteraciones. La documentación necesaria que se debe presentar para cada iteración es la siguiente:

1. Problemas encontrados. Cuáles son los problemas más importantes que ha tenido y cómo los ha abordado.
2. Modelo de casos de uso extendido.
3. Modelo de dominio extendido.
4. Diagramas de secuencia UML de casos de uso diseñados.
5. Clases de diseño
6. Código fuente de implementación en formato electrónico. No es necesario imprimir.