UNIVERSIDAD ORT
Uruguay

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel 902 15 05 Fax 908 13 70
www.ort.edu.uy

EVALUACIÓN	Examen	**GRUPOS**	**TODOS**	**FECHA**	**14/02/2023**
MATERIA	PROGRAMACIÓN 2				
CARRERA	AP/ATI				
CONDICIONES	**- Duración: 2 horas** **- Sin material** **-Recordar indicar nombre del docente del curso en primera hoja del parcial.** **- Consultas exclusivamente sobre la letra**				

Una empresa nos solicita un sistema para gestionar los mensajes que recibe. Esos mensajes pueden ser reclamos o sugerencias.

Los mensajes tienen un número identificatorio creciente (autonumérico), el autor del mensaje, una descripción, una fecha y un estado que puede ser uno de los siguientes: [Pendiente, Desestimado, Cerrado].

El autor del mensaje es uno de los funcionarios de la empresa. De esos funcionarios se conoce un id identificatorio numérico creciente (autonumérico), nombre, email y rol en el sistema, que será uno de los siguientes:
[Administrador, Encargado, Usuario].

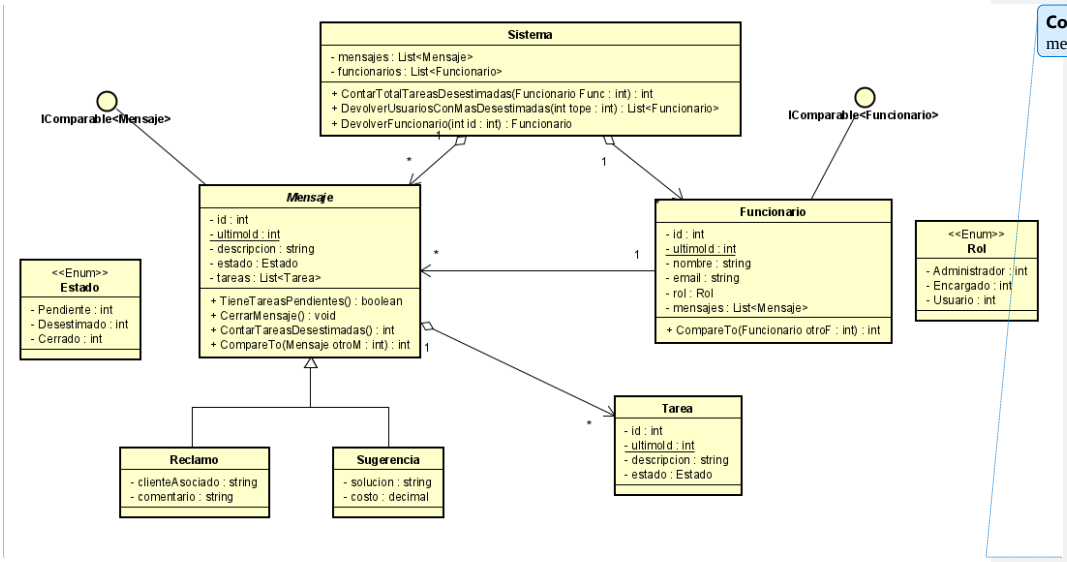
Como se mencionó, se manejan dos tipos de mensajes:

- Los reclamos tienen además de los datos comunes a todos los mensajes un texto con el nombre del cliente asociado al reclamo y el texto del reclamo.
- Las sugerencias incluyen también un texto con una propuesta de solución y su costo.

Los mensajes tienen tareas asociadas que las debe resolver el autor del mensaje antes de poder dar por cerrado el mismo. De las tareas se conoce un Id numérico creciente, una descripción, y un estado, que es el mismo que se usa en los mensajes. Una vez que todas las tareas están Cerradas, el mensaje se tiene que cerrar.

Se pide:

1. Diagrama de clases del dominio de la aplicación que incluya las clases con sus atributos, relaciones, y métodos para resolver los requerimientos abajo detallados. No incluir las properties, ni métodos que no estén directamente relacionados con los requerimientos solicitados. (30 puntos).



Comentado [MQ1]: Quedó al revés la asociación entre mensaje y funcionario.

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel 902 15 05 Fax 908 13 70
www.ort.edu.uy

2.

3. Implementar en C# las funcionalidades siguientes. En caso de que un método requiera utilizar otro, implementarlo.

a) Crear el método CerrarMensaje, e indicar en qué clase se ubica, de modo que dado un mensaje y analizando sus tareas este se pueda cerrar. Será posible cerrar el mensaje cuando todas sus tareas estén resueltas. No podrá cerrarse si queda alguna de sus tareas por resolver. (25 puntos)

En la clase Mensaje (Superclase de Reclamo y Sugerencia)

```
public bool TieneTareasPendientes()
  {
      foreach (Tarea t in tareas)
      {
          if (t.Estado == Tarea.TipoEstado.Pendiente)
          {
              return true;
          }
      }
      return false;
  }
  public void CerrarMensaje()
  {
      if (TieneTareasPendientes())
      {
          throw new Exception("El mensaje tiene tareas pendientes");
      }
      this.Estado = Tarea.TipoEstado.Cerrado;
  }
```

b) Dado un número entero, obtener los autores de los mensajes que tengan más que esa cantidad de tareas desestimadas. Los autores obtenidos se ordenarán ascendentemente por rol y dentro de un mismo rol se ordenarán descendentemente por nombre. Se controlará que en el retorno no aparezca dos veces el mismo autor. (30 puntos)

En la clase Mensaje

```
public int ContarTareasDesestimadas()
 {
     int contador = 0;
     foreach (Tarea t in tareas)
     {
         if (t.Estado == Tarea.TipoEstado.Desestimado)
         {
             contador++;
         }
     }
     return contador;

  }
```

En la clase Sistema

```
 int ContarTotalTareasDesestimadas(Funcionario func)
 {
     int conteo=0;
     foreach(Mensaje Mens in mensajes)
     {
         if (Mens.Autor.Equals(func))
         {
             conteo += Mens.ContarTareasDesestimadas();
         }
```

```
        }
        return conteo;
    }

    List<Funcionario> DevolverUsuariosConMasDesestimadas(int tope)
    {
        List<Funcionario> aux = new List<Funcionario>();
        foreach(Funcionario func in funcionarios)
        {
            if (tope <= ContarTotalTareasDesestimadas(func) )
            {
                aux.Add(func);
            }
        }
        aux.Sort();
        return aux;
    }
```

En la clase Funcionario

```
    public int CompareTo(Funcionario otroF)
    {
        if (this.rol != otroF.rol)
        {
            return (this.rol.CompareTo(otroF));
        }
        else
        {
            return (  -  this.nombre.CompareTo(otroF.nombre));
        }
    }
```

Comentado [MQ2]: Devuelve por el int subyacente. Hay que hacerle el toString

c) MVC:
Implementar lo que considere necesario a nivel de vista y el controlador que permita resolver lo siguiente:

a. Mostrar estado y descripción de todos los mensajes de un autor, ordenados por cantidad de tareas.
No se deberá implementar el código de dominio necesario para resolver el requerimiento, pero sí se incluirá en el diagrama de clases la firma del método (o métodos). (15 puntos)

En la clase Mensaje

```
    public int CompareTo(Mensaje otroM)
```

```
        {
            return (this.tareas.Count.CompareTo(otroM.tareas.Count));
        }
    }
```

En la clase Sistema

```
    List<Mensaje> MensajesDeAutor(Funcionario func)
    {
        List<Mensaje> aux = new List<Mensaje>();
        foreach (Mensaje Mens in mensajes)
        {
            if (Mens.Autor.Equals(func))
            {
                aux.Add(Mens);
            }
        }
        aux.Sort();
        return aux;
    }

    Funcionario DevolverFuncionario(int id)
    {
        foreach(Funcionario func in funcionarios)
        {
            if(func.Id == id )
            {
                return func;
            }
        }
        return null;
    }
```

Comentado [MQ3]: En clase Funcionario debería estar el override de Equals

En el controlador Funcionario

```
public class FuncionarioController : Controller
 {
     public IActionResult Index()
     {
         return View();
     }

     [HttpPost]

     public IActionResult Index(int id)
     {
         Sistema unS = new Dominio.Sistema();
         Funcionario func = unS.DevolverFuncionario(id);
         List < Mensaje > auxMens = unS.MensajesDeAutor(func);
         return View(auxMens);
     }
```

}

En las vistas

Vista con Get

```
@using Dominio;

<form method="post">
  <input type="text" name="id" placeholder="Ingrese Id del funcionario" />
  <button type="submit">Procesar</button>
</form>
```

Vista con Post

```
@using Dominio;
@model List<Mensaje>
<table class="table-striped">
<thead>
    <tr>
        <th>Estado</th>
        <th>Descripcion</th>
    </tr>
</thead>
@foreach (Mensaje unM in Model)
{
    <tr><td>@unM.Estado</td><td>@unM.Descripcion</td></tr>
}
</table>
```