Obligatorio
Individual

Objetivos:

El objetivo de este práctico es que el estudiante tenga una primera aproximación al *stack* de tecnologías .NET para el desarrollo de aplicaciones empresariales y que éste se familiarice con el ambiente de desarrollo que utilizará durante el laboratorio de la materia. Se espera que el alumno implemente de forma gradual un prototipo guiado, siguiendo buenas prácticas de arquitectura y diseño.

Descripción del problema:

El prototipo a desarrollar consiste en la administración de un sistema (*backoffice*) para la gestión de empleados de una empresa que emplea freelancers brindando servicios de trabajo remoto. Esta administración deberá contar con funcionalidades para listar la nómina de empleados de la empresa, agregar nuevos empleados, editar los datos de empleados existentes y eliminar empleados de la nómina que ya no forman parte de la empresa.

Los empleados pueden ser estar enfocados al desarrollo de produtos o la administracion de activos de la empresa. Los desarrolladores son empleados full time mientras que los administradores part-time. Los empleados que trabajan en modalidad part-time cobran su sueldo en base a un valor hora, mientras que los empleados que trabajan en modalidad full-time lo hacen en base a un salario fijo.

Arquitectura:

El archivo *ObligatorioIndividual.zip* contiene un archivo de solución y los proyectos de Visual Studio que son el esqueleto inicial del prototipo a implementar siguiendo una arquitectura empresarial simple en capas (acceso a datos, lógica de negocio, capa de servicios y presentación), con una capa compartida para las entidades, enumerados y componentes transversales a toda la aplicación (*CrossCutting Concerns* [1]) como se muestra en la figura 1. Puede profundizar en el diseño de aplicaciones en capas consultando en **[2]**.

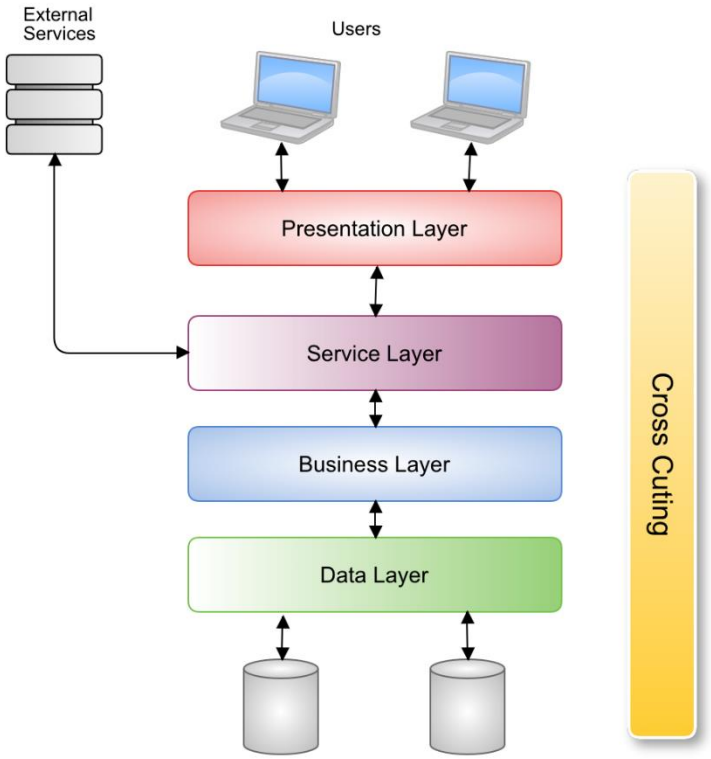


Figura 1- Arquitectura del prototipo

Ejercicio 0 (Configuración del ambiente)

a) Instale (si aún no lo ha hecho) los requerimientos de software:

I. Visual Studio 2017

II. SQL Server 2016 (la versión *express* es suficiente)

III. MongoDB 3.4.2 (para su configuración se recomienda seguir la lectura de **[3]**).

IV. Node.js latest stable version

b) Configure el sistema de versionado que utilizará durante el práctico y el laboratorio. Si bien no es obligatorio, se recomienda el uso de GIT con *Visual Studio Team Services* **[4]** o *GitLab* **[5]**

I. Configure el servidor, usuarios y permisos

II. Configure el cliente (integrado en *Visual Studio* o externo como *SourceTree*, linea de comandos etc.)

III. Cree un archivo de texto no vacío con su nombre y súbalo al servidor utilizando el/los comandos requeridos de acuerdo al sistema de versionado elegido (*commit*, *push*, *etc.*)

IV. Descargue los archivos subidos por el resto del grupo utilizando el/los comandos requeridos de acuerdo al sistema de versionado elegido (*update*, *pull*, *etc.*)

Ejercicio 1 (EntityFramework and impedance missmatch [6])

Se desea combinar el enfoque *Database First* de *EntityFramework* con un mapeo de jerarquía de entidades colapsando los atributos de las sub-entidades en la super-entidad (*Table Per Hierarchy*). Para ello, cree primero una base de datos según el esquema relacional *ER1* en *SQL Server* y luego utilice *EntityFramework* para generar el modelo orientado a objetos *MOO* como un elemento de tipo *ADO.NET Entity Data Model* bajo la carpeta Model del proyecto *DataAccessLayer*.

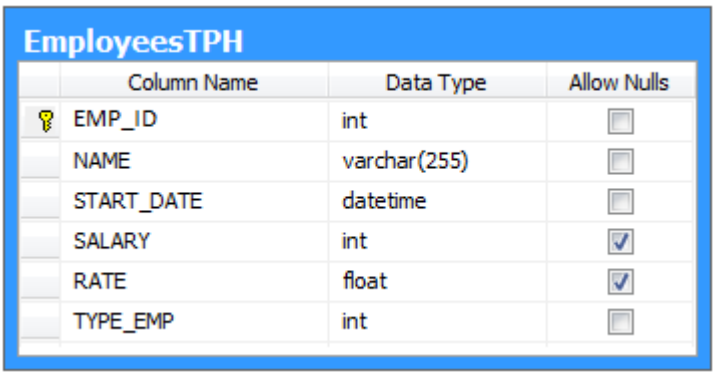


Figura 2 *– ER1: tabla de empleados utilizando enfoque Table Per Hierarchy (TPH)*

Observaciones:

- EMP_ID es clave primaria de tipo entero auto-incremental con valor inicial 1.

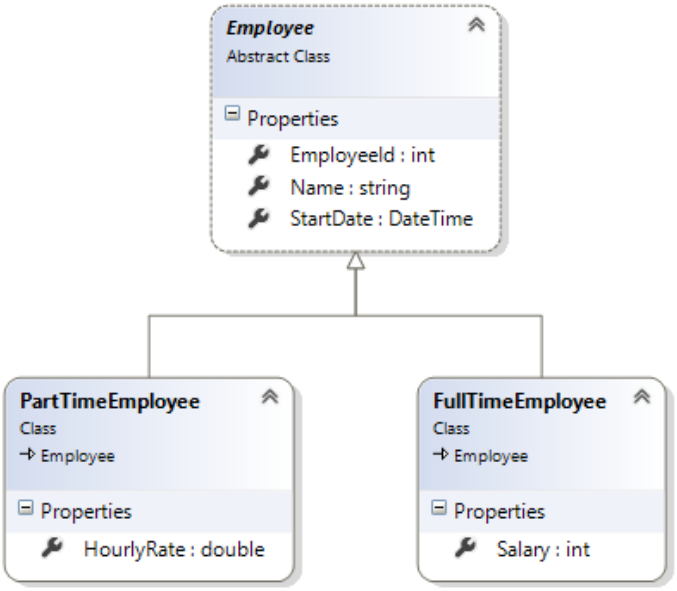


Figura 3 *– MOO: Diagrama de clases para las entidades del prototipo*

Observaciones:

- Los nombres de las columnas y de las propiedades son diferentes, por ejemplo, *HourlyRate* es diferente de *RATE*, *EmployeeId* es diferente de *EMP_ID*, etc
- Los nombres de las tablas y de las clases son diferentes, por ejemplo, *EmployeesTPH* es diferente de *Employee* (singular, sin TPH). Además de renombrar la clase, se aconseja renombrar el Entity Set Name a *Employee**s*** (plural, sin TPH)
- *Employee* es una clase abstracta
- *HourlyRate* y *Salary* NO son nulables en las clases *PartTimeEmployee* y *FullTimeEmployee* respectivamente
- Para este ejercicio se recomienda la lectura de **[7] y [8]**

Ejercicio 3 (capa de acceso a datos y LINQ)

El uso de interfaces y patrones de diseño es una buena práctica para desacoplar la capa de negocios de la tecnología de acceso a datos y las características del repositorio utilizado (base de datos relacional, NoSQL, archivos XML, etc.). Esto permite cambiar fácilmente el proveedor de acceso a datos de forma transparente para las capas superiores logrando así un bajo acoplamiento y mejorar los aspectos deseables de mantenibilidad, flexibilidad y testeabilidad (entre otros).

En nuestro prototipo se han definido cuatro implementaciones concretas para la interfaz de acceso a datos: una para la tecnología de EntityFramework (*DALEmployeesEF*), otra para acceso mediante SQL nativo (*DALEmployeesNativeSQL*), otra para MongoDB (*DALEmployeesMongo*) y finalmente un objeto simulado (*mock*) implementado mediante *LINQ To Object* [12]con fines de test para el ejercicio 8 (DALEmployeesMock) .

Observe además que para poder ocultar la tecnología específica de acceso a datos hacia las capas superiores, la interfaz IDALEmployees utiliza la clase *Shared.Entities.Employee* (independiente de la tecnología de acceso a datos) y no *DataAccessLayer.Model.Employee* (requiere la dependencia específica a EF).

a) Implemente los métodos de *DALEmployeesEF usando LINQ Query Expressions* [9, 11], *y/o Lambda Expressions* [10,11]

b) Implemente los métodos de *DALEmployeesMongo* usando el driver oficial *C# and .NET MongoDB Driver* ***[13]***

Ejercicio 4 (capa de negocio)

Implemente la capa de lógica de negocios. En particular interesa la operación *CalcPartTimeEmployeeSalary*, la cual permite calcular el salario a pagar a un empleado part-time de acuerdo a las horas que se indiquen que ha trabajado. En caso de que el empleado no exista o que no sea un empleado part-time la operación debe arrojar una excepción controlada específica hacia las capas superiores.
Para el resto de las operaciones, la capa de negocios debe resolver las operaciones invocando a la misma operación a la capa de acceso a datos (de la cual depende y posee una referencia explícita de su interfaz)

Ejercicio 5 (capa de servicios)

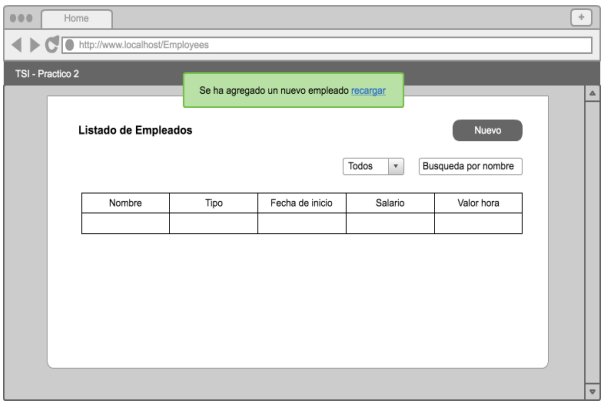
Dado que la compañía tiene varias sucursales en diferentes paises el sistema operará en la internet. Por ello, se quiere implementar una capa de servicios que haga uso de estándares de comunicación REST utilizando ademas seguridad en todos los endpoints debido a que la informacion que intercambiaran estos servicios es confidencial. Para garantizar la seguridad de los endpoints se utilizara el standard JWT (Json Web Token) [14].
Es necesario también que se habilite el Cross Domain, necesario para los ejercicios siguientes. [15]
Las operaciones a exponer por este servicio son:

1. ABM de empleados de la empresa.
2. Ingreso de horas trabajadas por un empleado en un rango de fechas.
3. Listado de empleados con los siguientes campos de interes: nombre, fecha de ingreso y tipo de empleado y salario generado en el mes. Filtros disponibles: identificador del empleado, nombre, fecha de ingreso y tipo de empleado.

Ejercicio 6 (capa de presentación)

Se desea implementar el listado de empleados mediante una aplicación angular [16, 17, 18].

i. Realizar los cambios necesarios en el archivo *employees.service.ts* que permitan obtener los datos de los empleados.
ii. Los cambios necesarios en el archivo *employees-list.component.ts* que permitan exponer el listado en su correspondiente html *employees-list.component*.html
iii. Implementar el filtrado de elementos de la lista con Angular (Opcional)



Ejercicio 7

Se desea ahora implementar el alta de empleados en el sistema. Implementar los cambios necesarios en el componente manage-employees.component.ts y el servicio employees.service.ts para realizar el alta de empleados. El alta exitosa de un empleado debe redirigir al listado.

Ejercicio 8

Se desea ahora realizar un sistema de notificaciones para indicar a los usuarios que estén consumiendo la aplicación sobre otros usuarios interesados en consumir ver los datos de los empleados.

Dicho de otra manera, se desea que cada vez que un nuevo usuario acceda al listado de empleados notifique a todos los usuarios que estén utilizando el sistema en tiempo real. Realizar los cambios necesarios en el WebApi y en la aplicación de Angular para lograrlo.
Se recomienda el uso de SignalR o websockets. [19, 20, 21]

Referencias:

1. *Crosscutting Concerns: https://msdn.microsoft.com/en-us/library/ee658105.aspx*
2. *Layered Application Guidelines: https://msdn.microsoft.com/en-us/library/ee658109.aspx*
3. *Setup the MongoDb environment: https://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/*
4. *Visual Studio Team Services: https://www.visualstudio.com/es/team-services/*
5. *Gitlab: https://gitlab.fing.edu.uy/*
6. *Entity Framework: https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx*
7. *EF Designer TPH Inheritance: https://msdn.microsoft.com/en-us/data/jj618292*
8. *Table Per Hierarchy Inheritance in Entity Framework: http://blogs.microsoft.co.il/gilf/2010/01/24/table-per-hierarchy-inheritance-in-entity-framework/*
9. *LINQ Query Expressions (C# Programming Guide): https://msdn.microsoft.com/en-us/library/bb397676(v=vs.140).aspx*
10.
11. *How to: Use Lambda Expressions in a Query (C# Programming Guide): https://msdn.microsoft.com/en-us/library/vstudio/bb397675.aspx*
12. *Query Syntax and Method Syntax in LINQ: https://msdn.microsoft.com/en-us/library/vstudio/bb397947.aspx*
13. *LINQ to Objects: https://msdn.microsoft.com/en-us/library/mt693052.aspx*
14. *C# and .NET MongoDB Driver: https://docs.mongodb.org/ecosystem/drivers/csharp/*
15. *http://blogs.quovantis.com/json-web-token-jwt-with-web-api/*
16. *Enabling Cross Origin in ASP.NET Web API: http://www.asp.net/web-api/overview/security/enabling-cross-origin-requests-in-web-api*
17. *Angular: https://angular.io/,*
18. *Using a package.json https://docs.npmjs.com/getting-started/installing-npm-packages-locally*
19. *Typescript: https://www.typescriptlang.org/*
20. *SignalR: http://signalr.net/*
21. *Real Time AngularJS with SignalR: https://blog.sstorie.com/integrating-angular2-and-signalr-part-1/*
22. *Websockets, Mozilla MDN: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API*