Prueba Técnica Parlem Telecom

Documento técnico

Martín Márquez Rivero

Enero 2023

Contenido

[Introducción 2](#_Toc125317820)

[Descripción del proyecto 2](#_Toc125317821)

[Arquitectura 2](#_Toc125317822)

[Front-end 2](#_Toc125317823)

[Back-end 3](#_Toc125317824)

[Autenticación 3](#_Toc125317825)

[Base de datos 3](#_Toc125317826)

[Capa de dominio 4](#_Toc125317827)

[Test Unitarios 4](#_Toc125317828)

[Diagrama MER 5](#_Toc125317829)

# Introducción

El presente documento describe la prueba técnica de Parlem Telecom. El documento pretende servir de guía para entender la solución propuesta, así como las posibles opciones de mejora.

# Descripción del proyecto

En dicha prueba se solicita realizar dos aplicaciones, un front-end y un back-end. El front-end tiene que ser capaz de mostrar una lista de clientes y sus respectivos productos contratados.

El back-end tiene que ser un api capaz de tener los datos y la lógica de negocio necesaria para mostrar los datos.

El proyecto ha sido desarrollado en Visual Studio Enterprise 2022 y SQL Server Express. Para el control de versiones se ha utilizado GIT. El código está alojado en GitHub.

# Arquitectura

A continuación, se describe la arquitectura de ambas partes.

### Front-end

Las soluciones han sido desarrolladas en .Net Core 3.1. El front-end ha sido desarrollado con Razor.

Para mostrar los datos en tablas se han utilizado librerías de [Datatables.net](https://datatables.net/). Esta librería facilita el renderizado de datos al proveer de un mínimo estilo CSS, un buscador y paginador.

Se ha desarrollado con la arquitectura MVC. Existe una capa de servicios que llama al back-end vía api.

Por simplificar la prueba, la web esta sin autenticar. No existe una página de login como tal ni gestión de permisos por roles.

Todos los datos necesarios para que la web funcione se han informado en el fichero appsettings.config. Se ha desarrollado una pequeña librería para acceder a la configuración llamada “parlem.common” que puede albergar funciones propias de la compañía desarrolladas por el equipo de desarrollo.

En el apartado de vistas, se ha desarrollado utilizando layouts. Existe un layout común el cual todas las paginas utilizan que contiene la cabecera y el pie de la página, así como scripts y estilos CSS comunes a toda la aplicación.

Como mejora se propone el desarrollo de una página de login completa, así como el registro, olvido de contraseña etc de los usuarios.

Se propone también que se tengan varios roles que permitirán securizar algunos apartados de la aplicación, siendo accesibles por diferentes roles.

### Back-end

El back-end es un servicio api. En esta aplicación esta toda la lógica de negocio necesaria para procesar los datos, así como el acceso a la base de datos y la autenticación.

Se utiliza Dependency injection de ASP.NET Core como patrón para la inyección de dependencias. Este patrón se utiliza tanto en el back-end como en el front-end. Para el registro de los servicios se hace mediante “AddScoped”. Con esto conseguimos que los servicios tengan un contexto nuevo para cada petición a los controladores. Para cada petición se inyecta la dependencia, siendo la memoria independiente entre peticiones.

Como mejora a nivel de arquitectura, se deberían separar los servicios en una capa adicional (similar a la capa de dominio) permitiendo desconectar los mismos del proyecto.

### Autenticación

El back-end utiliza la autenticación de Aspnet.core.identity mediante JSON Web Token (jwt). Existe un usuario dado de alta a modo de ejemplo que tiene los permisos necesarios para acceder a la aplicación:

USER 🡪 apiuser

PASS 🡪 Password@2023

El token de autenticación se puede obtener llamando a la url <http://localhost:61955/api/authenticate/login> con el user y pass enviado por json en el body. Por motivos de simplicidad, existe un token ya generado en el fichero de configuración del front-end con caducidad de un año.

### Base de datos

Se utiliza el patrón CodeFirst para la generación de la base de datos. La base de datos se programa completamente en clases entidad y se genera mediante migraciones ejecutadas con la consola de Visual Studio. Se provee de una copia de la base de datos para facilitar la ejecución del proyecto.

La base de datos contiene todas las tablas necesarias para la autenticación con ASP.NET Core Identity y las tablas mínimas para la prueba. En el apartado “Opciones de mejora” se describirán los cambios propuestos en la base de datos.

### Capa de dominio

En la capa de dominio llamada “parlem.domain” se encuentran todas las clases necesarias para el acceso a la base de datos.

En la carpeta “Abstract” esta la interfaz de Repository. Aquí se definen los métodos de interacción con el contexto de la base de datos.

En la carpeta “Concrete” se encuentra definido el contexto de la base de datos, así como la implementación de los métodos de interacción con el contexto de la base de datos.

En la carpeta “Entities” se encuentran las entidades de la aplicación. Estas clases serán las tablas de la base de datos. La carpeta “Models” contiene los datos que se van a intercambiar entre capas. Por simplicidad en muchos casos se está utilizando las propias entidades para el intercambio de información, pero es importante recalcar que siempre se tienen que definir modelos para estos propósitos.

### Test Unitarios

Se han desarrollado test unitarios con [XUNIT](https://xunit.net/) utilizando MOQ para simular las entidades de la aplicación. Por simplicidad solo se han incluido los test unitarios de la capa de dominio.

Se podrían incluir test unitarios de la capa de controllers del back-end así como test end-to-end en el front-end con [Playwright](https://playwright.dev/) por ejemplo.

# Diagrama MER

El diagrama MER (Modelo Entidad Relación) actual es el siguiente:

Diagrama

Descripción generada automáticamente

Es un diagrama muy sencillo y solo a modo de ejemplo para que funcione la aplicación propuesta.

Como mejora se propone que la relación entre los clientes y los productos contratados sea mediante una tabla relacionada “ClientesProductos” como se describe a continuación:

|  |  |  |
| --- | --- | --- |
| ClientesProductos | | |
| Id | IdCliente | IdProducto |

Se tendría que eliminar “IdCustomer” de la tabla de “Productos” y relacionarla con la tabla “ClientesProductos”

A nivel de maestros, se podría indicar tablas maestras para Tipos de documento, tipos de producto etc.