**Cambio Arquitectura CitenikBank**

Se decidió realizar un cambio de arquitectura al proyecto **CitenikBank** con el fin de dividir las responsabilidades y de esta manera quitar de los controladores la lógica de negocio relacionada con las diferentes peticiones. De esta forma el proyecto será dividido en 2 capas, una primer capa “app” en donde se encontrarán los controladores y las clases referentes a la configuración del proyecto NetCore 6.0, y una segunda capa “src” en donde encontraremos la lógica de negocio dividida en cada uno de los servicios en base al controlador que los necesite utilizar.

| Arquitectura CitenikBank BackEnd vieja | Arquitectura CitenikBank BackEnd nueva |
| --- | --- |
|  |  |
| Se estructura el proyecto teniendo en cuenta un solo nivel.  Los **Controladores** era el responsable de consultar a la Base de Datos la información requerida por parte de las diferentes API 's, conteniendo toda la lógica de negocio dentro de las diferentes peticiones.  A la hora de realizar la consulta de información, es necesaria la utilización de la clase **ApplicationDBContext,** quien será la encargada de conectarse directamente con ella.  Luego tendremos otras carpetas como **DTOs,** y **Entities, Validations** que se relacionan directamente con los modelos correspondientes de las diferentes API 's.  Por último, en **Utilities** encontraremos herramientas que serán utilizadas para diferentes cuestiones, como por ejemplo seguridad, o el mapeo de las Clases propias del lenguaje de programación con respeto a las tablas de la Base de Datos.  Al estar todas las carpetas contenidas dentro del mismo proyecto, no es necesaria la realización de ningún tipo de inyección de dependencias. | Se estructura el proyecto en base a dos capas para la implementación de servicios:  **-app**  **-src**  Dentro de la carpeta **app** encontraremos tanto los controladores como las clases de configuración para iniciar el proyecto como son StartUp y Program. Además, para que estas dos capas puedan comunicarse entre sí, es necesario una clase adicional para permitir la inyección de dependencias.  Por lo cual dentro de la carpeta **src** tendremos una carpeta **core** y un nuevo proyecto **Application**, éste último contendrá: Modelos, Servicios, Inyecciones.    En base a esto se tiene la carpeta de **Service** que contiene los servicios y la inyección de dichos servicios ( ej. ClientesService.cs y IClienteServices.cs)  En las carpetas: **Entities** migraremos las entidades, en **DTO** encontraremos los modelos de dichas entidades, ambas carpetas respetan la arquitectura anterior, pero se ubican dentro del proyecto Application.  La carpeta **Validations** no contiene ningún cambio, salvo que estará contenida dentro del proyecto Application.  Por último las clases ApplicationDbContext.cs, AutoMapperProfiles.cs y Jwt.cs quedarán incluidas en Application, eliminando la carpeta Utilities.  Por último se actualizan las dependencias en ambos proyectos, según las necesidades de cada clase. |