**Prueba para Grability-Rappy.**

**Presentada por: Diego Fernando Alvarez Andrade.**

**Desarrollador de Software Senior.**

**Teléfono: 3103121656**

**CODING CHALLENGE**

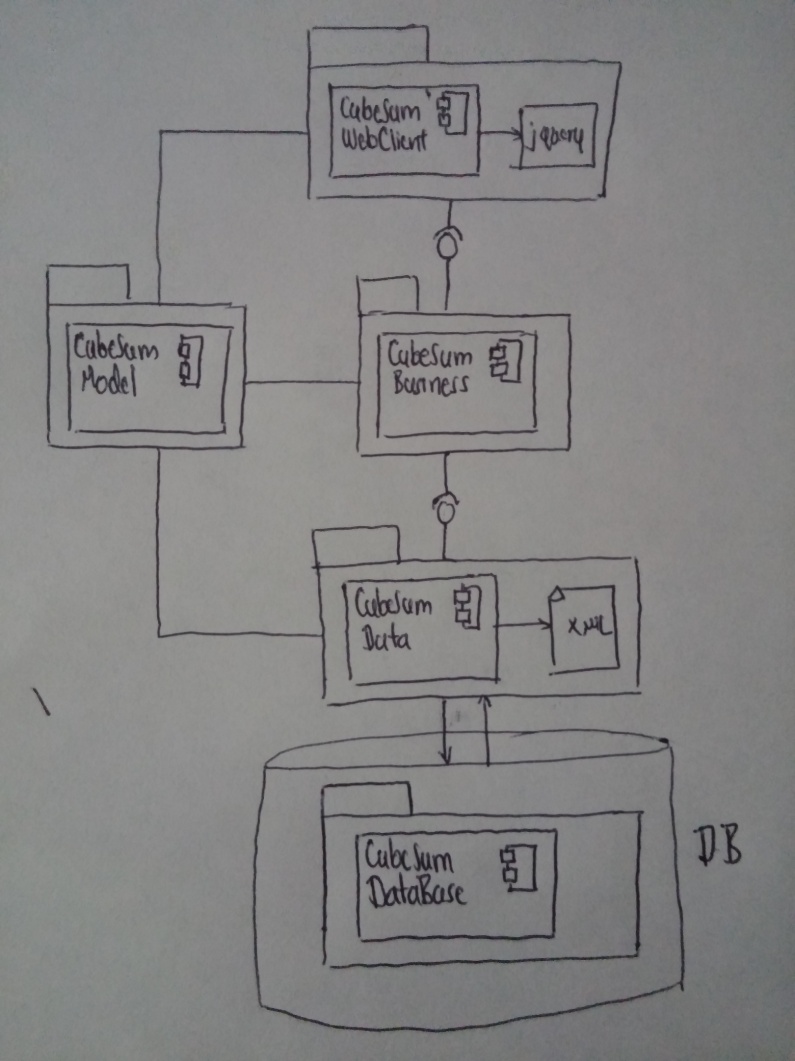
**Descripción**

Se entrega solución CubeSum en Visual Studio 2012 framework 4, el cuál contiene 6 proyectos. La capa de presentación es un proyecto WEB realizado con asp.net web forms. Las otras capas son librerías de clases.

**Funcionamiento**

Al cargar la página web abre un popup de ayuda que permite generar las expresiones. Las expresiones también pueden ser escritas en el área **Input**. La ayuda de expresiones también puede ser abierta desde el botón **Expression Helper**. Cuando se tiene la expresión en el área **Input**, se debe hacer click en el botón **Process** y en el área Output se mostrará el resultado. Si ocurre algún error, será mostrado en un texto de color rojo en la parte inferior de las áreas de texto.

**Componentes**

****

**Figura 1.**

**CubeSumDataBase:** representa la base de datos. Las clases que pertenecen a este proyecto existen únicamente para crear una base de datos dinámica cuando se cargue el proyecto. Esto se carga por medio de la clase Initializer.cs.

**CubeSumModel**: responsable de representar el modelo de datos.

Clases:

* Cube.cs: clase responsable de representar una matriz
* Cell.cs: clase responsable de cada celda de la matriz.
* Query.cs: clase que representa un query a la matriz. Su funcionalidad es almacenar un query.
* QueryValue.cs: lase que representa los parámetros de un query (texto y valor). Su funcionalidad es almacenar los valores de los queries
* Expression.cs: clase que representa una expresión completa. Su responsabilidad es el manejo de las operaciones de la expresión.

**CubeSumData**: capa de persistencia. Simulación de la base de datos en un archivo XML

Clases:

* IDataHelper.cs: interfaz que se expone. Implementa patrón fachada para esconder funcionalidad
* DataHelper.cs: clase responsable del acceso a los datos.
* DataXMLWrapper.cs: clase responsable de crear envoltorio del archivo XML que contiene las expresiones válidas. Implementa patrón Wrapper para envolver acceso a base de datos y patrón singleton para que solo sea instanciada una vez.
* DBConnection: responsable de crear una única conexión a la base de datos. Implementa patrón Singleton.`
* QueryDTO: responsable de transformar un objeto de tipo **QueValDB** (Representación de DB) de la en un objeto de tipo **Query** (Modelo)

**CubeSumBusiness**: capa de negocio. API para la implementación de negocio de sumatoria de matriz.

Clases:

* ICubeSumManager.cs: interfaz que se expone. (Implementa patrón fachada para esconder funcionalidad)
* CubeSumManager.cs: clase principal de la capa de negocio. Su responsabilidad es recibir una petición para ser procesada y entregar una respuesta.
* ExpressionValidator.cs: clase para validación de expresiones. Su responsabilidad es validar la expresión completa que se recibe.
* BQuery: responsable de obtener y guardar todo lo relacionado con Query en la base de datos.

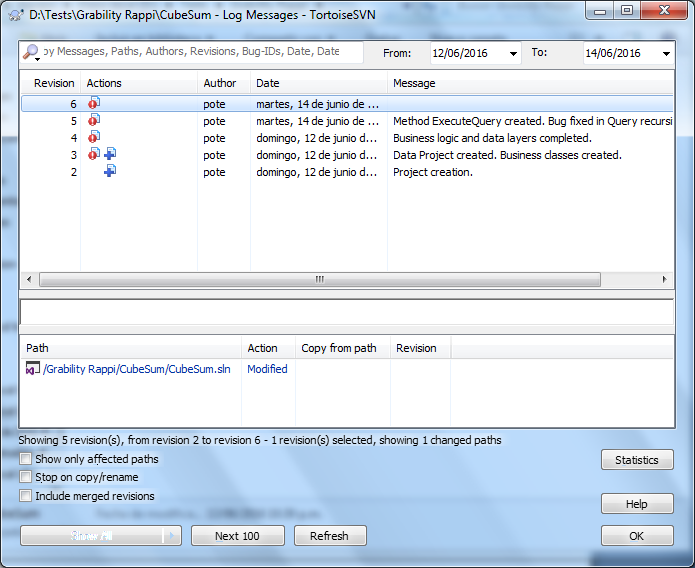
**UnitTest:**  proyecto de pruebas unitarias.

**WebClient**: aplicación web que cuenta básicamente con 2 aéreas de texto, una de entrada de datos y otra de salida. El archivo Web.config contiene los valores mínimos y máximos utilizados para realizar las validaciones, las cuales se hacen en la capa de negocio

Cuenta con una opción que ayuda a generar expresiones la cual está hecha en js y jquery.

**Adicionales**

|  |  |  |
| --- | --- | --- |
| **Descripción** | **OK** | **Comentarios** |
| Mecanismo web entrada y salida de datos | **X** | Aplicación Web |
| Uso SVN | **X** | Utilicé SVN pero en un momento se desconfiguró. (Figura 2) |
| Pruebas unitarias |  | Tuve problemas con la base de datos dinámica para realizar los test unitarios. |

****

**Figura 2.**

**CODE REFACTORING**

1. **Malas prácticas**

* Los nombres de las variables no son claros. La variable $id se usa para recuperar el id del servicio, debería llamarse algo así como servicioId.
* El id del conductor (driver\_id) debe guardarse en una variable al igual que el id del servicio (service\_id) al comienzo del método, ya que se utiliza varias veces más adelante.
* Existen comentarios que no dicen nada útil
* Hay código comentado que no debería estar
* El código tiene retornos en muchas partes, se debe evaluar todas las posibilidades, asignar los errores a una variables y al final retornar el error
* Se utiliza mucho un if después de otro, se debe utilizar else
* Los comentarios deben ir separados por un espacio después del "//"
* No se realiza manejo de excepciones

1. **Refactorización** (Se incluye archivo anexo CodigoPHP.php)

**public function post\_confirm() {**

**$driverId = Input::get('driver\_id');**

**$servicioId = Input::get('service\_id');**

**$servicio = Service::find($servicioId);**

**$error = '0';**

**if($servicio != NULL) {**

**if($servicio->status\_id == '6') {**

**$error = '2';**

**}**

**else if($servicio->driver\_id == NULL && $servicio->status\_id == '1') {**

**$servicio = Service::update($servicioId, array(**

**'driver\_id' => $driverId,**

**'status\_id' => '2'**

**));**

**Driver::update($driverId, array(**

**"available" => '0'**

**));**

**$driverTmp = Driver::find($driverId);**

**Service::update($servicioId, array(**

**'car\_id' => $driverTmp->car\_id**

**));**

**// Notifica al usuario**

**$pushMessage = 'Tu servicio ha sido confirmado!';**

**$servicio = Service::find($servicioId);**

**$pushMessage = Push::make();**

**if($servicio->user->uuid != '') {**

**if($servicio->user->type == '1') { // iPhone**

**$result = $push->ios($servicio->user->uuid, $pushMessage, 1, 'honk.wav', 'Open', array('serviceId' => servicio->id));**

**} else { // android**

**$result = $push->android2($servicio->user->uuid, $pushMessage, 1, 'default', 'Open', array('serviceId' => servicio->id));**

**}**

**}**

**} else {**

**$error = '1';**

**}**

**} else {**

**$error = '3';**

**}**

**return Response::json(array('error' => $error));**

**}**

**PREGUNTAS**

1. **Responsabilidad única**

Está relacionado con la cohesión. Básicamente consiste en que una clase no debe tener más de una responsabilidad, si tiene más de una se debe separar en varias clases. Siempre se busca que el código tenga responsabilidades definidas por cada clase y no que una clase tenga muchas responsabilidades lo que la convertiría en una clase gorda.

1. **Código limpio**

* Cualquier programador que no lo conozca puede darse una idea de lo que hace el código.
* Evita comentarios innecesarios, el código por si solo debe decir que está haciendo.
* No declara variables innecesarias
* Todas las variables son nombradas en el mismo formato.
* Las variables se utilizan lo más cerca posible a donde van a ser utilizadas.
* Los métodos no son muy largos.
* Cada clase tiene una única responsabilidad
* Debe tener alta cohesión, bajo acoplamiento