Presentado por: Eugenio Pájaro Almagro

CUBE SUMMARY

Para la solución se utilizó JavaScript, por ser un requerimiento que no requiere persistencia, por lo que solo se realiza al nivel del cliente.

1. Las capas de la aplicación

* **Las capa de Vista** :
  1. Matriz.
  2. CoreConsulta.
  3. Controlador\_Acciones.
  4. Prueba

* **Controlador**: No aplica por ser una aplicación que por la definición se aplica sólo a nivel del cliente.

* **Modelo**: No se aplica por ser una solución que hasta este nivel carece de persistencia.

2) La responsabilidad de cada clase:

**Matriz**: Se encarga de contener los bloques iniciales, además de contar con analizadores y modificadores de acceso, al igual de ofrecer métodos que permiten la suma de campos en un intervalo definido, el llenado de los bloques de acuerdo a el atributo de definición **N** y la validación de sus atributos no se utiliza patrón de variaciones protegidas por ser un lenguaje de scripting.

**CoreConsulta**: Se crea con el fin de establecer una relación de dependencia con la clase **Matriz** con el fin de mantener cohesión y el principio de patrón experto, se encarga de ejecutar las operaciones de consultas sobre el objeto matriz, la validación de parámetros, métodos analizadores y modificadores de **M y T.**

**Controlador\_acciones :** no es una clase pero simula un controlador encargado de la recepción de eventos de la vista y de acuerdo a estos se realizan las acciones de respuesta por parte de los objetos expertos.

**Prueba**: es la encargada de realizar instancia de un objeto de CoreConsulta, para realizar el un test a las funcionalidades, es ejecutado por el api Qunit.

CODE REFACTORING

1) Las malas prácticas de programación que en su criterio son evidenciadas en el código.

1. No valida la existencia del parámetro **driver\_id**.
2. No se valida la existencia del conductor.
3. Exceso de lógica en el controlador.
4. Los errores comentados.
5. No permite la reutilización.

2) Cómo su refactorización supera las malas prácticas de programación.

1. Se lleva la lógica a una clase compuesta, para este caso le coloque operación, con el fin de mantener una alta cohesión y un bajo acoplamiento.
2. Posibilidad de reutilizar la lógica en otros controladores.
3. Aplicar el principio de variaciones protegidas, posibilitando la escalabilidad en las operaciones sobre los modelos.

PREGUNTAS

1. ¿En qué consiste el principio de responsabilidad única?¿Cuál es su propósito?

Rt/ En que un objeto debe tener una única responsabilidad en otras palabras debe contener un único punto de variación.

Su principal propósito es el mantener un bajo acoplamiento y una alta cohesión, en la medida de que cada componente deba realizar una única tarea y delegue tareas complementarias a otros componentes, con en el fin mantener la escalabilidad en el desarrollo frente a los puntos de variación.

2. ¿Qué características tiene según tu opinión “buen” código o código limpio?

Rt/Las características presentes deben coherencia, simplicidad, consistencia, legibilidad y escalabilidad. El código debe ser mínimo tanto clases como métodos deberían ser concisos, mantener una responsabilidad única por componente, buena indentación y nombres acorde al contexto permitiendo una fácil lectura a cualquier desarrollador.