- Baldevenito Joaquin
- Loser Facundo
- Marcial Valentin

Informe de Refactorización de Código en Ruby

Refactorización de aplicación en Ruby utilizando controladores y pruebas moduladas.

Modificaciones Realizadas:

1. **Modularización de `server.rb` en controladores:**
   - Se dividió el archivo `server.rb` en dos controladores independientes:
     -`authentication_controller.rb`

     -`game_controller.rb`.

2. **Modularización de `test_spec.rb`:**
   - Se separaron las pruebas del archivo `test_spec.rb` en cuatro archivos:
     -`account_spec.rb`

     `-admin_spec.rb`

     -`auth_spec.rb`

     -`game_spec.rb`.

3. **Corrección de Code Smells detectados por RuboCop:**
   - Se realizaron cambios según las recomendaciones de RuboCop para mejorar la calidad del código.

Justificación de las Modificaciones

1. **Modularización de `server.rb` en Controladores**
   - **Problema Detectado:** La clase `server.rb` presentaba una sobrecarga de responsabilidades, un **code smell** conocido como **Large Class**, en el que una clase realiza demasiadas tareas.
   - **Solución Aplicada:** Dividimos `server.rb` en módulos independientes para manejar la autenticación y la lógica del juego por separado, dejando todos los endpoint en server.rb. Esto facilita la gestión y el mantenimiento del código.
   - **Fundamentación:** Según Martin Fowler en *Refactoring: Ruby Edition*, la **Extract Class** es una técnica recomendada para mejorar la cohesión y reducir la complejidad. La separación de responsabilidades ayuda a seguir el principio de **Single Responsibility**.

- Baldevenito Joaquin
- Loser Facundo
- Marcial Valentin

2. **Modularización de `test_spec.rb`**
    - **Problema Detectado:** `test_spec.rb` contenía múltiples pruebas para diferentes componentes, lo que dificultaba el seguimiento y mantenimiento.
    - **Solución Aplicada:** Dividimos las pruebas en archivos más pequeños y específicos, lo que facilita la comprensión y organización.
    - **Fundamentación:** La refactorización de pruebas mejora la capacidad de identificar fallos específicos y sigue el principio de **Separation of Concerns**. Esto también es compatible con las mejores prácticas de desarrollo orientado a pruebas.
3. **Corrección de Smells de RuboCop**
    - **Problemas Detectados:** RuboCop identificó varios code smells, como **Long Methods**,**Snake Case for Variables**,**No Trailing Whitespace**, entre otros.
    - **Acciones Tomadas:** Se refactorizaron métodos largos en otros más pequeños usando la técnica **Extract Method**, se estandarizaron los nombres de variables y métodos, y se arreglaron automaticamente los code smells de sintaxis.
    - **Fundamentación:** La refactorización según las reglas de RuboCop garantiza que el código sea más limpio y eficiente. Estas acciones mejoran la legibilidad y evitan errores futuros, lo que se alinea con el principio de **Clean Code** y las prácticas recomendadas en *Refactoring: Ruby Edition*.

Resultados de las Refactorizaciones

- **Mejoras en la Mantenibilidad:** La modularización ha simplificado el código, permitiendo una evolución más sencilla y segura de la base de código.
- **Facilidad de Pruebas y Depuración:** Las pruebas ahora están mejor organizadas, lo que facilita la identificación y corrección de errores.
- **Cumplimiento de Normas de Calidad:** El código es ahora más legible, coherente y conforme a las mejores prácticas de RuboCop.

Asi quedo el overview que nos dio el análisis de rubycritic subiendo mas de 30 puntos