Análisis de deuda técnica

Para llevar a cabo este análisis, se ejecutaron pruebas unitarias que ya venían incorporadas en el código. Además, se utilizaron herramientas de análisis estático, como linters, para detectar errores de sintaxis y otros problemas relacionados con los estándares de codificación. Las herramientas de análisis de código estático pueden detectar errores en el código sin ejecutar el programa. Estas herramientas analizan el código para detectar problemas como variables no utilizadas, convenciones de nomenclatura incorrectas, código duplicado, entre otros. También se realizó testing exploratorio a través del equipo de testing.

Durante el proceso, el equipo acordó establecer un límite para la cantidad de problemas que se podrían encontrar, dado que la cantidad de errores era considerable. Una vez alcanzado este límite, se detuvo la búsqueda de más problemas para respetar los tiempos establecidos.

Las herramientas utilizadas para hacer los análisis son las siguientes:

- **EsLint:** Es una herramienta para analizar de manera estática el código TypeScript en proyectos de Angular.
- **ReSharpe**r: Es una herramienta de análisis de código estático para C# que proporciona sugerencias de refactorización, análisis de código, navegación y búsqueda.

Clasificación de las issues:

Las issues las clasificamos haciendo uso de las labels de github. Las labels las categorizamos en función de:

- Tipo: el motivo por el cual fue creada la issue. Puede ser por un **bug**, ya sea por la ausencia o mal funcionamiento de una feature, o pidiendo un **refactor** de código.
- En caso de que sea un bug, la severidad del mismo:
  - **Crítica:** un defecto que obstaculice o bloquee completamente la prueba o el uso de un producto o función.
  - **Mayor:** una función principal que no cumpla con los requisitos y se comporte de manera diferente a lo esperado. Cuando funciona muy lejos de las expectativas o no está haciendo lo que debería estar haciendo.
  - **Menor:** función que no cumpla con sus requisitos y se comporte de manera diferente a lo esperado, pero su impacto es insignificante hasta cierto punto o no tiene un impacto importante en la aplicación.
  - **Leve:** defecto cosmético
- A qué rol del sistema le corresponde la issue: acomodador, administrador, artista, espectador, vendedor, cualquiera o ninguno
- La prioridad del issue:

- **Inmediata:** plazo máximo 24 hs.
- **Alta:** plazo máximo 48 hs.
- **Media:** plazo máximo un par de semanas.
- **Baja**: sin plazo.

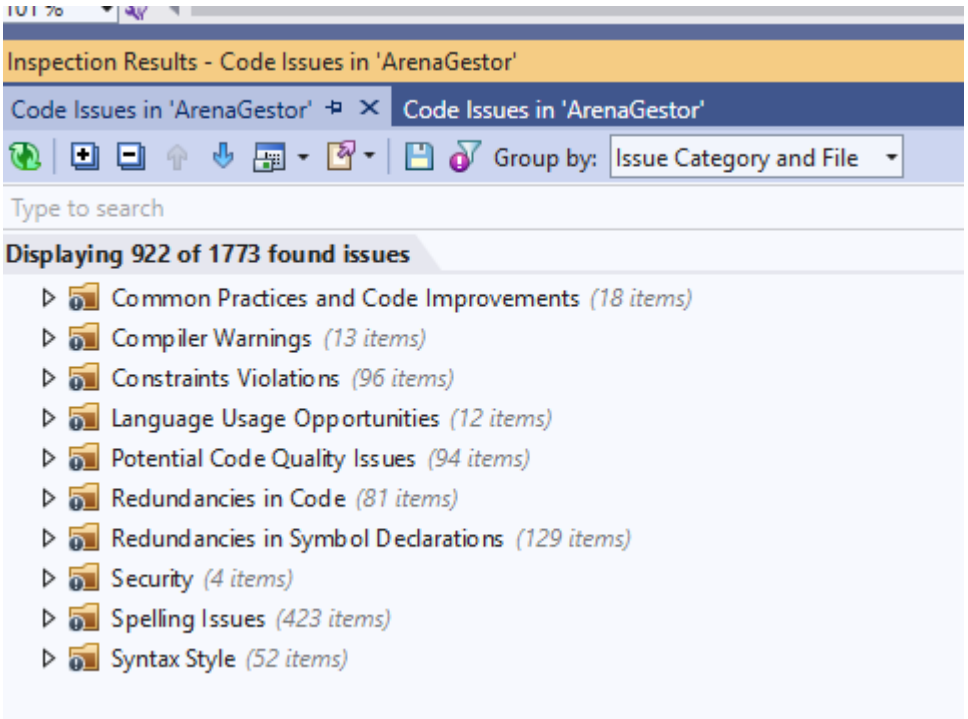
- El tipo de test mediante el que se encontró el issue (Test-type)

En github todas estas son labels independientes que fuimos colocando en los issues. La categoría que va a determinar sobre qué issues se van a trabajar es la prioridad, y en caso de que bajo el criterio de prioridad queden seleccionadas más de 2 issues se van a seleccionar en función de la severidad. El resto de labels son para darnos más información sobre el issue y facilitar su arreglo.

Resultados del análisis de código estático

Resharper: Testeo del código del backend realizado con C#

Utilizamos la herramienta ReSharper como analizador de código estático en busca de errores y mejoras de código.
En la imagen a continuación se muestran los diferentes bugs y mejoras categorizados en carpetas.



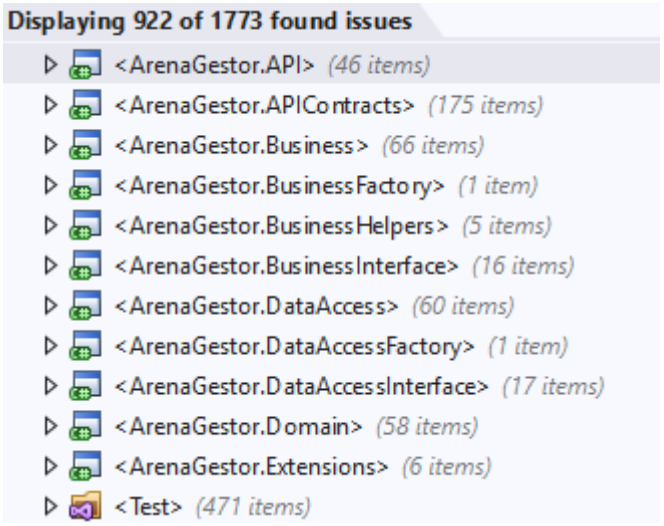
Se deja evidencia de los bugs y mejoras según cada carpeta que representa la categoría de issue y dentro de ellas los diferentes tipos de issue:.

1) **Common Practices and Code Improvements:**
    - Empty statement is redundant

- Field can be made readonly
- Replace with single call to Any()
- Replace with single call to First()
- Replace with single call to FirstOrDefault()

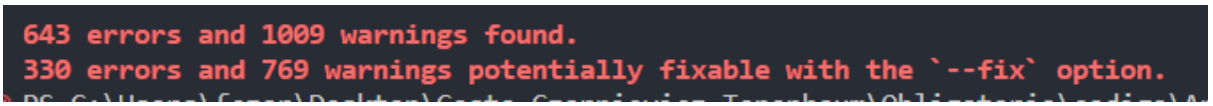
2) **Compiler warnings:**
   - Class overrides Object.Equals(object o) but not Object.GetHashCode()
   - Field is never used
   - Unassigned field
3) **Constraints Violations:**
   - Inconsistent Naming
   - Namespace does not correspond to file location
   - Possible ‘null’ assignment to non-nullable entity
4) **Language Usage Opportunities:**
   - ‘if’ statement can be written as ‘??==’ assignment
   - Merge null/pattern checks into complex pattern
5) **Potencial Code Quality Issues:**
   - IQueryable is possibly unintentionally used as IEnumerable
   - Local variable hides member
   - Non-accessed field
   - Possible ‘System.NullReferenceException’
   - Variable in local function hides variable from out scope
   - Virtual member call in constructor
6) **Redundancies in Code:**
   - Assignment is not used
   - Explicit delegate creation expression is redundant
   - Expression is always ‘null’
   - Expression is always ‘true’ or always ‘false’
   - Redundant ‘object.ToString()’ call
   - Redundant empty object or collection initializer
   - Redundant name qualifier
   - Redundant string interpolation
   - Redundant type arguments of method
   - Redundant using directive
7) **Redundancies in Symbol Declarations:**
   - Type member is never accessed via base type
   - Type member is never used
   - Underlying type of enums is ‘int’
   - Unused local variable
   - Unused parameter
8) **Security:**
   - NuGet package is vulnerable
9) **Spelling Issues:**
   - Typo in identifier
   - Typo in string literal
10) **Syntax Style:**
   - Use preferred style of ‘new’ expression when created type is evident

Vista de Issues por Paquete:



ESLint: Testeo del código del frontend realizado con angular y typescript

Al analizar el código del front de manera estática utilizando ESLint se generó un reporte de issues que queda adjuntado al repositorio. A grandes rasgos se dio la siguiente salida de errores y warnings:



Se definieron reglas específicas para ESLint para poder mantener un código limpio y con buenas prácticas.

Rules:

- "no-console": "warn" → Prohíbe el uso de la función console.log y otros métodos de console y muestra un warning si se viola dicha regla
- "no-unused-vars": "warn" → Detecta variables que no se están utilizando en el código y muestra un warning si se viola dicha regla
- "no-undef": "error" → Detecta variables que no han sido declaradas y muestra un error si se viola dicha regla
- "semi": "warn" → Asegura que las sentencias estén separadas por punto y coma y muestra un warning si se viola dicha regla
- "quotes": ["warn", "single"] → Asegura el uso de comillas simples para los strings y muestra un warning si se viola dicha regla
- "func-names": "error" → Establece que todas las funciones deben de tener un nombre y muestra un error si se viola dicha regla
- "no-underscore-dangle": "warn" → Establece que las variables privadas deben tener un guión bajo al comienzo del nombre y muestra un warning si se viola dicha regla
- "camelcase": "warn" → Establece que todas las variables deben estar en CamelCase y muestra un warning si se viola dicha regla