


# Check list dev test Backend

En el presente documento se mencionan y explican los pasos o el debido proceso para realizar los dev test en backend y cumplir con esta parte importante de los criterios de calidad del software

## Criterios de aceptación para la realización de los dev test Backend

Cuando se requiere agregar, modificar o eliminar código en un proyecto de backend es muy importante garantizar el bienestar del mismo, que después de estos cambios su compilación, empaquetación, construcción y despliegue se pueda ejecutar sin ningún problema, y no afecte a demás servicios o clientes que lo estén usando, el fin de este documento es mostrar los criterios que se deben tener en cuenta a la hora de hacer un dev-test, ya que si se cumplen todos y cada uno de ellos, la probabilidad de que después de aceptar un merge request o aprobar la subida de un código algo falle en su proceso de integración sea casi nula.

### Verificar la correcta compilación del código.

Si el proyecto no compila, mucho menos podrá ser empaquetado y desplegado, por esta razón, éste es uno de los criterios más importantes a la hora de revisar el código desarrollado, en algunas ocasiones se pueden evidenciar estos problemas echándole un simple vistazo al código, pero en la mayoría de casos no ocurre de esta manera, es por esto que es muy útil apoyarse de algunos comandos que nos ofrece [maven](#) , para verificar que se compile correctamente el proyecto.

```
mvn compile
```

Con el anterior comando maven se encargará de compilar el código escrito en archivos “.class” y este proceso mostrará si hay algún tipo de error o inconveniente, de ser así, aquel que tenga el rol de “dev-tester” tendrá la responsabilidad de hacersélo saber al desarrollador encargado de dicho código para que pueda ser corregido y nuevamente revisado.

### Revisar el despliegue y la integración de los MR en gitlab.

Anterior mente el ci/cd se realizaba con la herramienta de gitlab y jenkins, actualmente se está utilizando gitlab ci/cd de modo que para revisar el estado de los merge request se debe primeramente ingresar a gitlab, como desarrolladores tenemos en la home page de gitlab los proyectos a los cuales tenemos acceso y la simbología de gitlab es bastante intuitiva para entender el flujo de nuestro despliegue, a primera vista tendremos un resultado como el siguiente.

The screenshot shows the GitLab 'Projects' page. At the top, there's a navigation bar with 'Menu', a search bar, and user avatars. Below the navigation bar, the 'Projects' section is displayed. It includes tabs for 'Your projects' (30), 'Starred projects' (0), 'Explore projects', and 'Explore topics'. A 'Filter by name...' input field and a 'Name' dropdown are also present. The main content area shows a list of projects under the 'All' tab. Each project entry includes a status icon (a blue clock-like symbol), a star icon, a fork icon, and an 'Updated' timestamp. The projects listed are:

- RSI / KERNEL / Back Adjuntos (Developer) - Updated 1 hour ago
- RSI / DGTB / ADMON PAGOS / Back Admon Pagos (Developer) - Updated 1 hour ago
- RSI / DGTB / ADMON PERSONAL / Back Admon Personal (Developer) - Updated 1 hour ago
- RSI / VIE / Apoyo / Back Apoyo (Developer) - Updated 1 hour ago
- RSI / AUTHORIZATION / Back Authorization (Developer) - Updated 1 week ago
- RSI / TEMPLATE / Back Base Line (Maintainer) - Updated 55 minutes ago

El simbolo que encontraremos durante la ejecución (cuando aún no ha acabado) es un como un reloj de color azul que representa que el estado está corriendo.

Los simbolos una vez acaba el proceso estarán presentes en tres colores, verde, amarillo o rojo, el verde significa exitoso, el amarillo significa que aunque el despliegue fue exitoso hay warnings o advertencias, las cuales deben ser acatadas para mejorar los procesos y el rojo significa que el despliegue ha fallado.

Adicionalmente en cualquier momento podemos ver a detalle el estado de nuestro merge pulsando en el símbolo y aparecerá a detalle los estados por los que ha pasado el despliegue y en cual ha fallado o cuales han resultado exitosos, o cual se encuentra en ejecución.

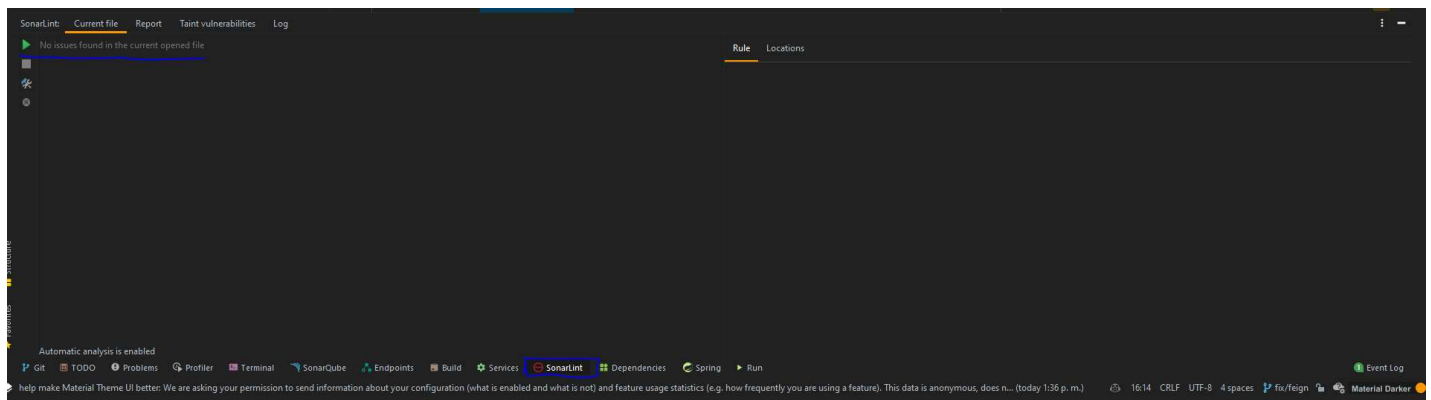
The screenshot shows the GitLab Merge Request page for the commit 5c0f36e3. The page header includes the repository path 'RSI > ... > Investigación > Back Investigación > Commits'. The commit details section shows the commit hash '5c0f36e3', the author 'Mario Andres Anaya Merchan', and the commit message 'fix: var added to properties'. Below the commit details, the merge request is titled 'Merge branch 'feature/security-rbac' into 'predev''. The pipeline status section shows that the pipeline #5985 passed with stages (indicated by green checkmarks) in 1 minute and 39 seconds. The 'Changes' tab is selected, showing the pipeline details. The pipeline table has columns for Status, Pipeline, Triggerer, and Stages. The pipeline #5985 is shown with a status of 'passed' and a triggerer of 'predev'.

Status	Pipeline	Triggerer	Stages
passed	Merge branch 'feature/security-rbac' into 'predev' #5985 predev -O- 5c0f36e3	Mario Andres Anaya Merchan	passed

## Revisar el reporte de sonarqube y la deuda técnica que pueda generar.

Otro paso importante que se debe tener en cuenta es la calidad del código, para esto sonarqube nos permite definir una serie de reglas que indican los estándares que debe cumplir nuestro código y que no sea solo subir código por subir sino subir código de calidad, sonarqube debe ser integrado a nuestro IDE ya sea intellij, eclipse o block de notas pero lo importante es que se revise la calidad del código que estemos subiendo y se cumpla con lo establecido y recomendado.

A nivel de intellij podemos descargar un plugin de sonarlint el cual nos permite integrar a nuestros proyectos el servicio de sonarqube, para esto debemos descargarlo y configurarlo, la configuración consiste en enlazar el proyecto con el servicio correspondiente que se encuentra en Bochica.uis.edu.co y de esta manera poder ejecutar los análisis sobre nuestros archivos, a continuación una vista de como se ve dicho plugin.



Lo importante en la fase de dev test es revisar que en los archivos modificados no haya errores de sonarqube, en lo posible que estén en cero, pero hay ciertos casos donde las reglas pueden ser omitidas porque es inalcanzable, pero en general se busca siempre cumplir al máximo con las recomendaciones, siempre se busca que sea lo mínimo para que de esta forma la deuda técnica ( que es esa deuda que en un momento debe ser saldada y consiste en refactorizar el código y aplicar los arreglos para cumplir con sonarqube en su totalidad ) sea mínima.

## Revisar test unitarios y “coverage” de los mismos.

Los test unitarios que aunque no se han implementado a fecha de hoy, junio de 2022, será algo indispensable en el futuro y que se debe revisar siempre a la hora de aprobar un merge request, los test unitarios son pruebas que se realizan a cada componente tomándolo como una unidad y que nos permiten validar el comportamiento de dicha unidad, a su vez se pueden trabajar junto con una metodología de desarrollo orientada a test. Si se está interesado en leer más sobre test puede hacerlo a continuación, [aquí](#).

La manera en la que se pueden ejecutar los test puede ser a nivel de **IDE**, en cada servicio de test y ejecutar cada clase, o podemos utilizar Maven para su ejecución, en el ciclo de vida de los proyectos **maven** tenemos el comando **mvn test** el cual ejecuta los test que llegue a encontrar en base a la configuración que tenga definida, para nuestro caso con la ayuda del plugin surefire se definió que todas las clases del árbol de paquete de test terminadas en Test serán ejecutadas con **maven** así que tenemos esta manera fácil para ver su comportamiento, en caso de todo ir bien quedaría lista la fase de test y en caso de ir mal se debe reportar al desarrollador.

Los test y en particular test unitarios en la capa de servicio también Deben ser analizados con **coverage**, el **coverage** nos dice que tanto se logró cubrir del servicio que estamos probando, de modo inicial, se establece una meta de al menos el **70%**, es decir al menos el 70% de las líneas del servicio, o del servicio en general deben ser probados, teniendo en cuenta que los **métodos** que se prueban son los métodos públicos . se sugiere usar mockito y todo lo demás relacionado a metodología y técnicas se encuentra [aquí](#).

## Tiempos de subida de un “merge request”.

Hay que mencionar que una vez es aprobado un merge request se dispara el proceso de integración continua “CI” gestionado por gitlab, desde ese momento hasta que ocurre el despliegue pueden pasar unos minutos, esto es importante tenerlo en cuenta, el “dev-tester” debe esperar hasta que este proceso termine para verificar si fue o no exitoso.

The screenshot shows a GitLab Merge Request (MR) for commit `1b8fe9fd` authored by Arley David Velasco Basto 2 hours ago. The merge request is titled "Merge branch 'feature/feign' into 'predev'" and includes the commit message "feat: resttemplate was replaced to feign". It references merge request `!283`. The parents of the merge are `7b9384f0` and `a0664504`, both pointing to the `predev` branch. Below this, it shows "1 merge request !283 feat: resttemplate was replaced to feign". At the bottom, a CI pipeline `#5991` is shown as passed with four green checkmarks, taking 2 minutes and 56 seconds.

Desde la aprobación del MR hasta completarse el proceso de CI pasaron 2 minutos y 56 segundos.

## Tiempos de aprobación de un “merge request”.

Es importante tener en cuenta los periodos, en los cuales se estén haciendo “dev-test”, ya que si se está en etapa de cierre de sprint, o de pruebas intensivas por parte del equipo de “QA”, es necesario que la aprobación de los “merge request” se haga fuera de horario laboral, por ejemplo, 12 de mediodía, antes de las 7 am, o después de las 5 pm.

También es importante estar atento a los diferentes mensajes que líderes del proyecto RSI emitan por los diferentes medios de comunicación, como por ejemplo, horas de mantenimiento, estabilización de proyectos, entre otros.

## Dependencias/código que no se debe subir.

Antes de aprobar y darle “merge” al código desarrollado por un compañero hay que tener mucho cuidado con los archivos en los que queda guardada la información de las dependencias del proyecto y los archivos que guardan

variables globales, éstos son `pom.xml` y `application.properties`.

Para el caso del `pom.xml`, este es un archivo que en la gran mayoría de casos no debería ser subido, a menos que se agregue una nueva dependencia, con la previa aprobación del líder de backend.

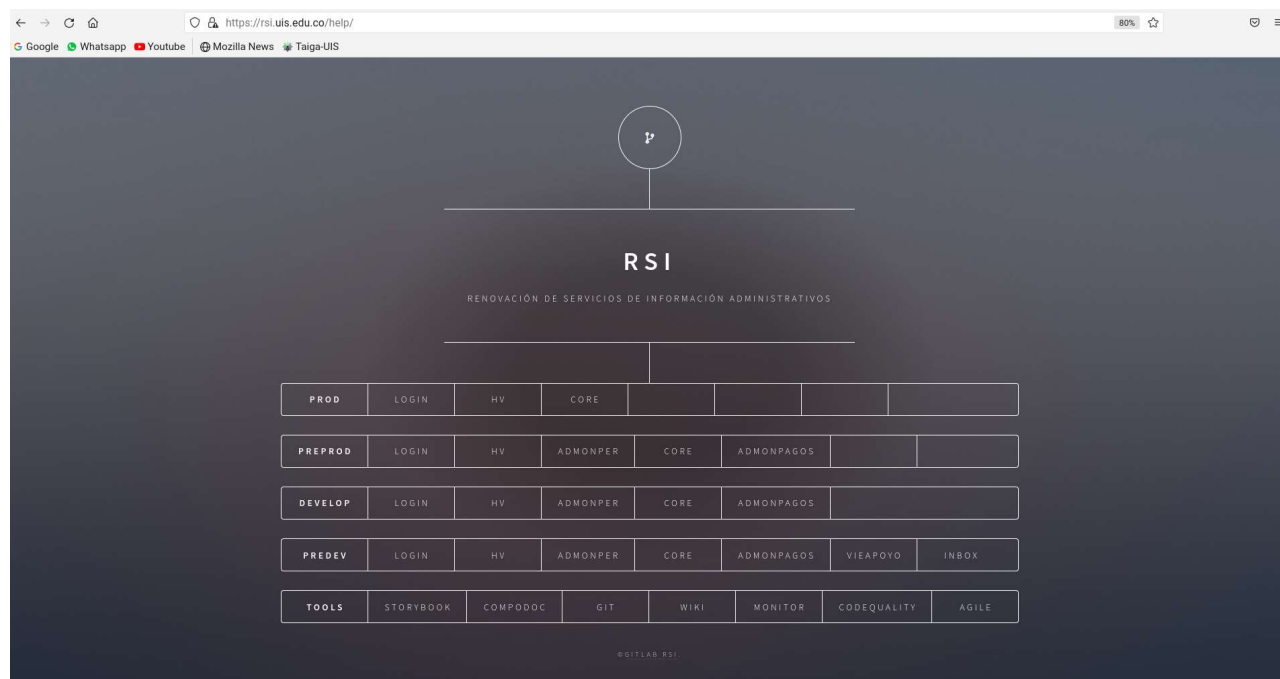
El archivo `application.properties` contiene la información de variables de entorno, usadas para despliegue, conexión a base de datos, configuración de servicios, entre otros. Este archivo tampoco es subido comunmente, a menos que se esté haciendo una configuración transversal al proyecto, y esto debe hacer con la aprobación del líder del equipo.

## Revisar mensajes que deban ir en los “messages.properties”.

Es común que como desarrolladores tengamos que usar mensajes para responder al cliente si hubo un error durante un procedimiento o método o para los mensajes de validación, los proyectos "back-end" están preparados para guardar estos mensajes en los archivos `messages_XX.properties`, El "dev-tester" deberá revisar que estos mensajes sean puestos allí y no dentro del código.

## Revisar que los proyectos “corran” en local, predev, develop, etc.

Es importante que una vez aprobado un "merge request" el proyecto en cuestión corra de manera adecuada, esto se puede ver, en las rutas de acceso disponibles donde se puede ver el despliegue de estos servicios, tanto "back-end" como "front-end".



Menú de visualización de los proyectos en los diferentes ambientes. (junio/2022)

## Correcta ejecución de la HU en los ambientes RSI de desarrollo.

Es responsabilidad del “dev-tester” velar que el código, no solamente esté estable y no tenga errores de compilación o de ejecución, sino que también cumpla con los criterios de aceptación mencionados en la historia de usuario respectiva, es por esto que la nomenclatura de las ramas debería tener el número de la historia que

está siendo desarrollada, esto le da la facilidad al “dev-tester” de buscar la historia que está siendo desarrollada, leerla, y verificar que lo que se desarrolló cumpla los criterios de aceptación.



Ejemplo nomenclatura de ramas:

`feature/[número de la historia]/description`

## Revisar que repositorios deben ser auditados por JaVers.

En algunos casos, las entidades deben ser auditadas, hay una herramienta llamada JaVers que permite hacer esto. Los proyectos de backend actualmente están preparados para esto, y en la capa repository, se debe verificar que entidades necesitan ser auditadas, esto según la historia que se está desarrollando y que por lo tanto su respectivo repositorio lleve la anotación `@JaversSpringDataAuditable`. *en caso de necesitarse una lógica distinta con EntityManager para la modificación, eliminación y la inserción de entidades se debe usar en la capa del servicio la anotación `@JaversAuditable` con esto los cambios serán escuchados igualmente pero a nivel de capa de acceso a datos para entidades y no directamente sobre los repositorios.*

## Que se debe hacer en aspectos de seguridad cuando un nuevo proyecto se genera a nivel de backends

La seguridad y autorización es un aspecto clave y debe ser analizado con cuidado previamente al desarrollo, es importante tener la idea clara de quien tendrá acceso a las funcionalidades, que roles y bajo que permisos se podrá acceder para restringir las acciones de nuestros servicios a personal estrictamente autorizado, para ello queda pendiente las tablas finales que deben ser añadidas y en las cuales se especificarán dichos accesos.

## Cuando deben llevar seguridad rbac nivel 2 (hasrole y acciones, preauthorized), que deben comunicarle a los analistas para su implementación (coming soon - pendiente reunión y definición de tablas sql)

Para la seguridad intermedia y avanzada basándonos en roles y permisos (RBAC) debemos controlar tanto los endpoints finales como los permisos y roles de usuarios que autorizan dichas endpoints, está pendiente la actualización y refactorización del schema authorization para las tablas finales.

## Documentación de endpoints, open-api (coming soon) → pendiente del desarrollo de dicha funcionalidad

La documentación de endpoints facilita conocer que estructura tienen nuestras apis, como deben ser invocados los métodos, que parámetros reciben, que tipos de datos, que rutas, que devuelven, que método http maneja etc, esto facilita la comprensión del desarrollo, sus test y el consumo del mismo.

## Manejo de adjuntos (Consumo de microservicios)

Por último cuando la funcionalidad que debemos probar tiene manejo de adjuntos es necesario y por seguridad realizar la prueba de dicha generación de adjuntos, para esto se recomienda solicitar la ruta y parámetros que ejecutan dicho método y realizar la ejecución con cualquier herramienta de prueba como **postman**, de esta manera garantizamos el buen funcionamiento de la historia y la calidad de la misma cuando el servicio que se consume invoca otros microservicios.

Hay otros casos donde no necesariamente se trabajan los adjuntos pero que también tienen invocación de microservicios como de generación de documentos (**reporteador**) de nuevo se sugiere realizar la prueba local para asegurarse que la rama lleva integrados todos los cambios necesarios para que dichos servicios funcionen correctamente.