

ANDREI GARCÍA CUADRA

100405803

MIGUEL HERNÁNDEZ CASSEL

100405956

PRINCIPIOS DESARROLLO SOFTWARE

MEMORIA I

## **PRINCIPIOS DESARROLLO DE SOFTWARE**

<b>Memoria 1.....</b>	<b>3</b>
1. Preconfiguración del proyecto .....	3
2. Normativa y colectividad de código.....	11
3. Backlog proyecto .....	14

# MEMORIA 1

## 1. PRECONFIGURACIÓN DEL PROYECTO

### 1.1. Creación de este documento (Memoria)

Este documento (en adelante, *Memoria*) contiene información relativa al ejercicio actual, manteniendo la preconfiguración (en ocasiones extendida) de proyectos anteriores.

Se ha creado utilizando el software *Pages*, ya que hemos estimado que un mejor diseño y formato en esta memoria la mantendrá más legible y entretenida que con un diseño burdo y común.

Para el trabajo compartido, hemos estimado utilizar *Pages for iCloud*, de modo que ambos podemos editar este documento en tiempo real sin interferir en nuestro trabajo. Una vez finalizada la memoria, se subirá al repositorio.

**1.1. Creación de este documento (Memoria)**

Este documento (en adelante, Memoria) contiene información relativa al ejercicio actual, manteniendo la preconfiguración (en ocasiones extendida) de proyectos anteriores.

Se ha creado utilizando el software Pages, ya que hemos estimado que un mejor diseño y formato en esta memoria la mantendrá más legible y entretenida que con un diseño burdo y común.

Para el trabajo compartido, hemos estimado utilizar Pages for iCloud, de modo que ambos podemos editar este documento en tiempo real sin interferir en nuestro trabajo. Una vez finalizada la memoria, se subirá al repositorio.

**1.2. Configuración ClickUp**

Hemos determinado utilizar la herramienta de gestión de proyectos gratuita ClickUp, la cual nos permite obtener un tablero

**1.3. Configuración y selección IDE**

**1.4. Configuración claves SSH y GitLab**

**1.5. Análisis, diagnóstico y solución a errores GitLab**

**1.6. Configuración de Maven**

Para configurar Maven, hemos determinado la implementación basada en IntelliJ, utilizando el menú contextual > Add Framework Support.

De esta manera se generó el documento pom.xml, sobre el que determinamos añadir tres perfiles: dev, test y prod.

El primero se utilizará para el desarrollo de la aplicación y estará activado por defecto. Test se utilizará para las pipelines de desarrollo sobre el proyecto y, una vez se validen todos los cambios del sprint para una determinada versión (siendo cada versión un ejercicio), se subirán en modo prod.

PÁGINA 2

MIGUEL Y ANDREI

PRÁCTICA 1

PÁGINA 3

Realizando la memoria en sus primeros pasos, utilizando la herramienta Pages.

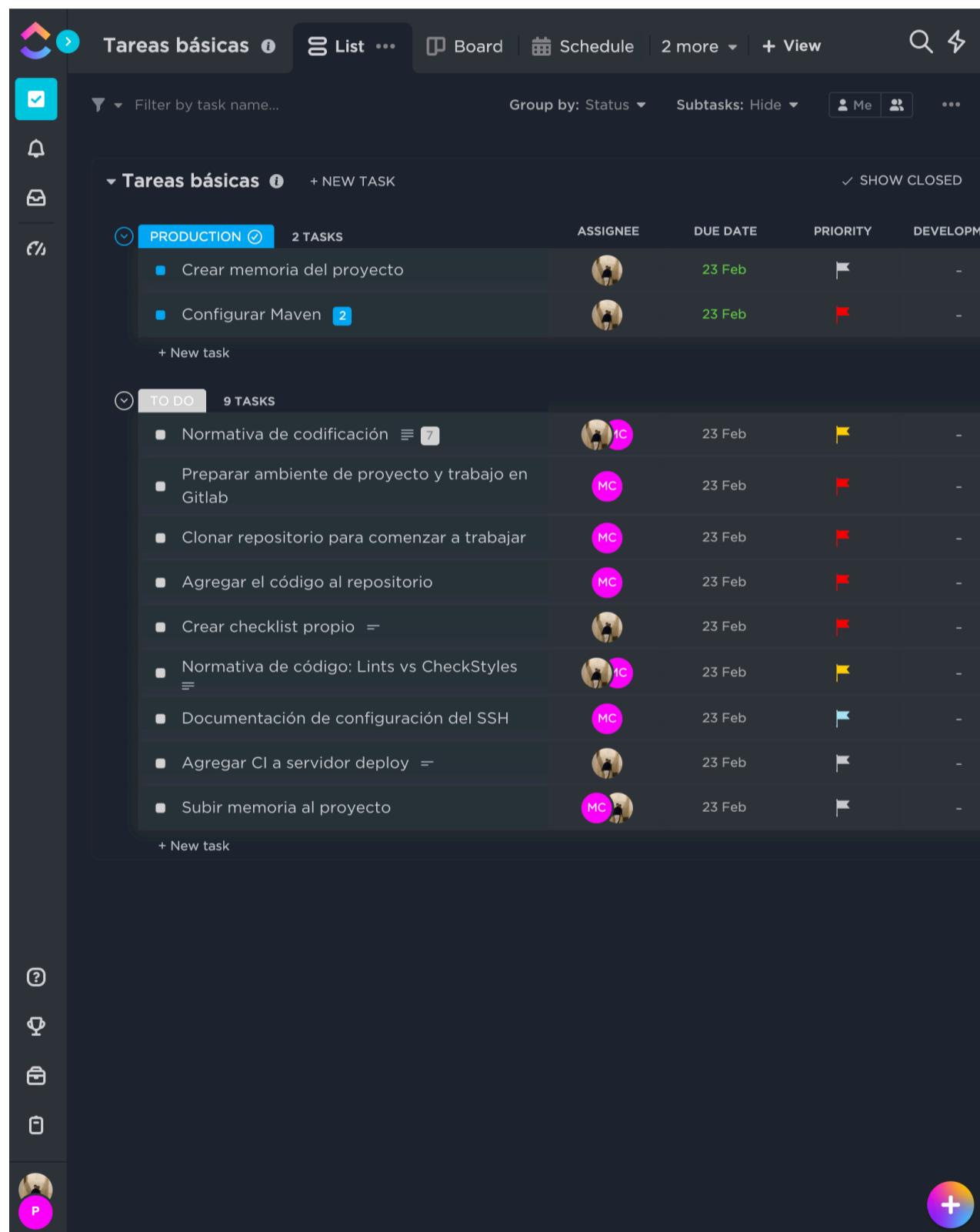
## 1.2. Configuración ClickUp

Hemos determinado utilizar la **herramienta de gestión de proyectos gratuita ClickUp**, la cual nos permite obtener un tablero con información relativa a cada tarea a realizar, de modo que, tras una **organización presencial previa**, podemos trabajar manteniendo un **flujo de trabajo constante y a demanda**, sin interferir en el trabajo ajeno y con una **comunicación constante**.

Hemos utilizado una metodología de trabajo basada en **Scrum-Kanban**, ya que mantenemos hilos de comunicación constante (Scrum) y un tablero con cada fase de la tarea (Kanban), pero **sin pertenecer nuestro trabajo puramente a ninguna de las metodologías**.

Se han organizado como tareas básicas, tareas extendidas y tareas supersaiyan, siendo respectivamente más fáciles y más difíciles, distribuidas en función de la carga y dificultad de la tarea.

**Podemos ceder acceso a los profesores si esto se solicita.**



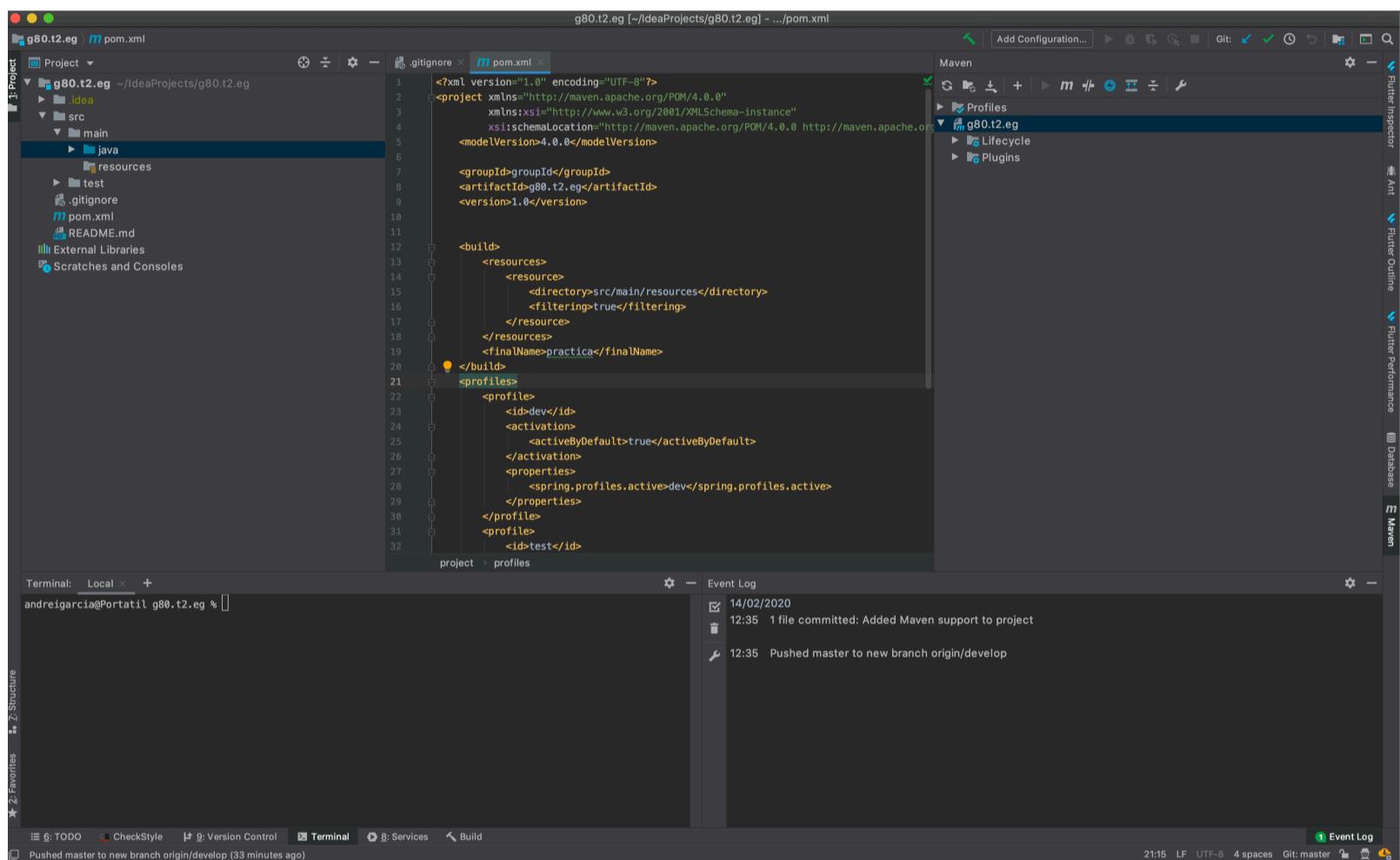
The screenshot shows the ClickUp application interface. At the top, there's a navigation bar with 'Tareas básicas' (Basic Tasks), 'List ...', 'Board', 'Schedule', '2 more ...', '+ View', and search/filter icons. Below the navigation, there are filters for 'Group by: Status' (Status), 'Subtasks: Hide' (Hide), and user selection ('Me'). A 'SHOW CLOSED' checkbox is also present. The main area displays two sections: 'PRODUCTION' (2 TASKS) and 'TO DO' (9 TASKS). The 'PRODUCTION' section contains tasks: 'Crear memoria del proyecto' and 'Configurar Maven'. The 'TO DO' section contains tasks: 'Normativa de codificación', 'Preparar ambiente de proyecto y trabajo en Gitlab', 'Clonar repositorio para comenzar a trabajar', 'Agregar el código al repositorio', 'Crear checklist propio', 'Normativa de código: Lints vs CheckStyles', 'Documentación de configuración del SSH', 'Agregar CI a servidor deploy', and 'Subir memoria al proyecto'. Each task has a status icon (green for PRODUCTION, grey for TO DO), an assignee icon, a due date (23 Feb), a priority level (high or low), and a development status indicator. A sidebar on the left contains icons for 'Tareas básicas', 'List', 'Board', 'Schedule', '2 more', '+ View', 'Help', 'Profile', and a user icon. A large '+' button is located at the bottom right.

*Utilización de la herramienta ClickUp, con las tareas de la primera fase organizadas*

## 1.3. Configuración y selección IDE

En este ámbito, se debatió sobre el uso de *Eclipse* entre los miembros del equipo, escogiendo finalmente como *IDE IntelliJ Idea* estandarizado sobre el proyecto.

Utilizamos dicha herramienta con una **licencia no-comercial** de uso únicamente estudiantil.



*Entorno de desarrollo IntelliJ Idea Ultimate, corriendo bajo la licencia no comercial estudiantil. El fichero en cuestión es pom.xml.*

## 1.4. Configuración claves SSH y GitLab

Las claves SSH se preconfiguraron utilizando la herramienta *ssh-keygen*, en este caso, ya que el ordenador de un compañero contaba aún con *Windows 7* (el cual ya **no tiene soporte** en muchos aspectos), **ambas claves se generaron desde un solo pc** corriendo el sistema operativo MacOS. Ambas claves fueron almacenadas en la ruta *~/.ssh/ida\_rsa* para la clave privada y *~/.ssh/ida\_rsa.pub* para la clave pública.

Esto demostró pues la buena compatibilidad entre sistemas operativos de este estándar de criptografía.

Tras su generación, se agregaron las claves a **GitLab** en cada respectiva cuenta.

*Configuración de las claves de acceso para poder utilizar git con las credenciales ssh.*

## **1.5. Análisis, diagnóstico y solución a errores GitLab**

Tras toda esta configuración, hayamos un error relacionado con la subida de código, pues al tener permisos developer no podíamos crear nuevos branches en el repositorio.

Esto acarreó serios retrasos en el proyecto, no obstante, **comunicamos el error** y ayudamos en la medida de lo posible a su solución, consiguiendo así que los developer pudieran crear nuevos branches.

No obstante, se originó otro error derivado del bloqueo de las ramas. Se volvió a alertar de dicho error, el cual fue subsanado de forma rápida.

Actualmente **podemos tanto subir código como crear nuevas branches**, para determinar nuestra estrategia de desarrollo.

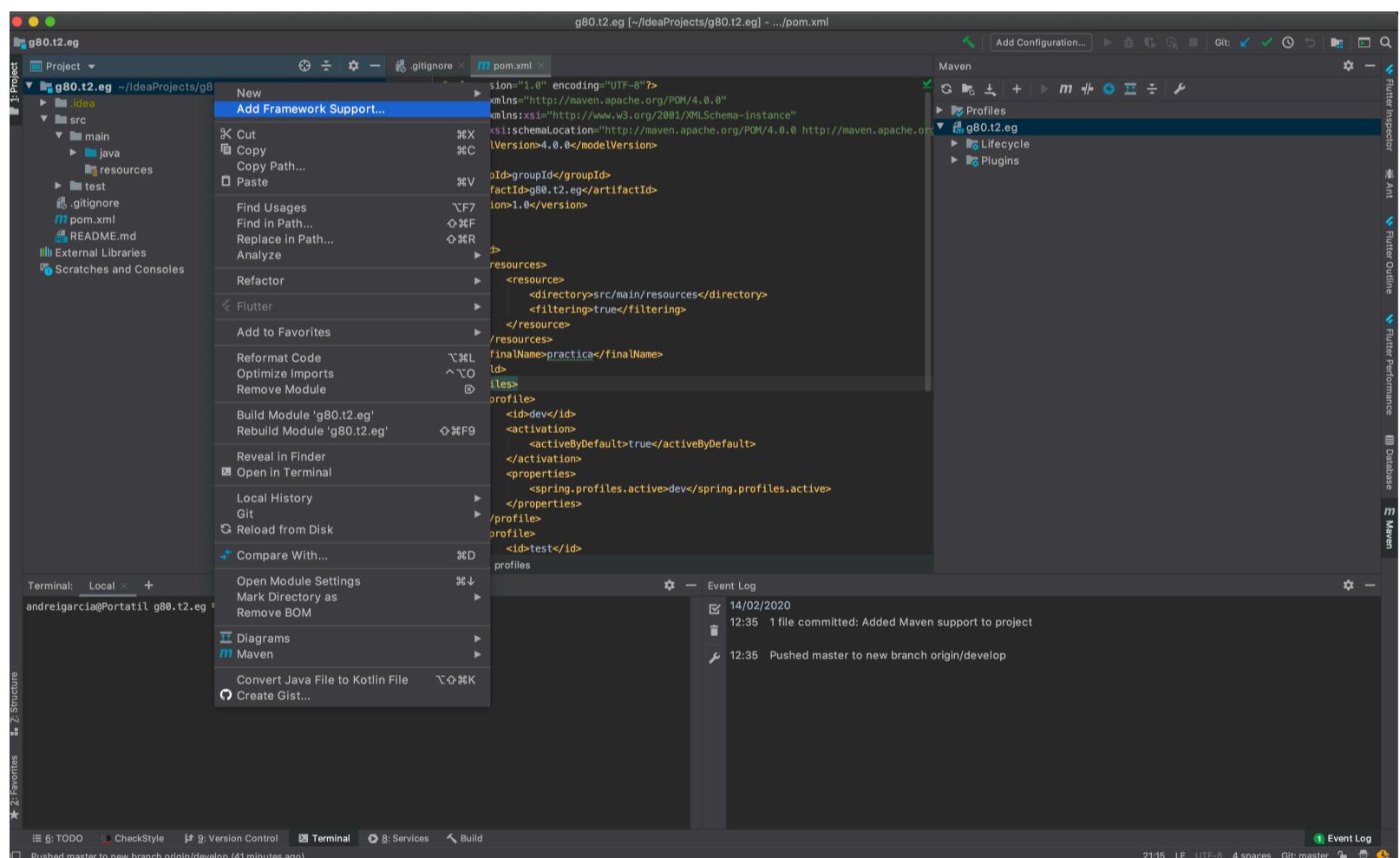
## 1.6. Configuración de Maven

Para configurar Maven, hemos determinado la implementación basada en IntelliJ, utilizando el menú contextual > *Add Framework Support*.

De esta manera se generó el documento pom.xml, sobre el que determinamos añadir tres perfiles: *dev*, *test* y *prod*.

El entorno *dev* se utilizará para el desarrollo de la aplicación y estará activado por defecto. *Test* se utilizará para las pipelines de desarrollo sobre el proyecto y, una vez se validen todos los cambios del sprint para una determinada versión (siendo cada versión un ejercicio), se subirán en modo *prod*.

Hemos utilizado la versión de Java 12.



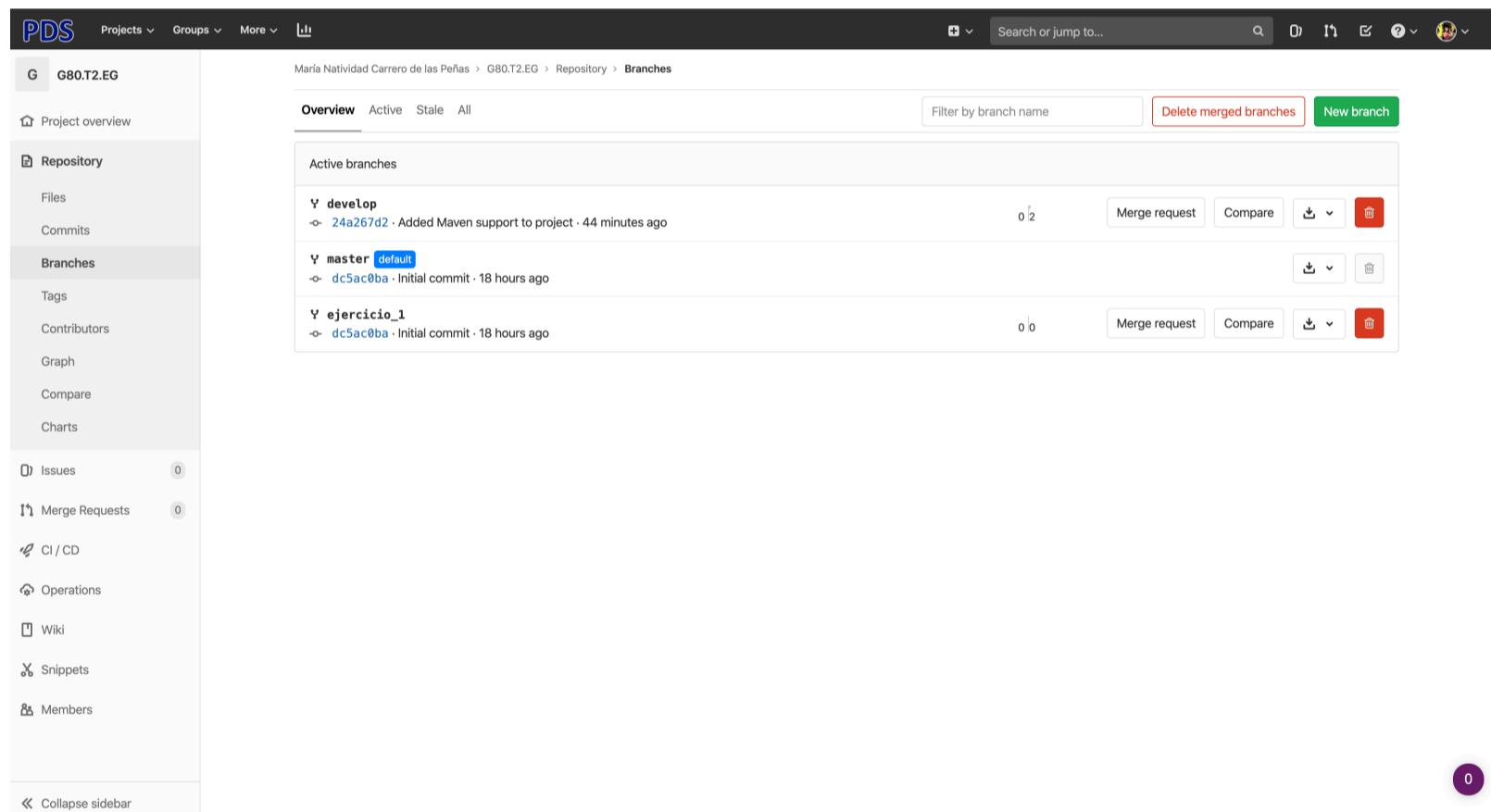
*Agregando el soporte para Maven sobre el proyecto, con la configuración de entornos en el fondo de la imagen, sobre la edición de texto.*

## 1.7. Branching strategy

Hemos estimado la utilización de la rama “*develop*” para la subida de **código intermedio**, la rama **master** como **rama principal de transgresión test/prod** y la creación de una nueva **rama** para cada **ejercicio**, de modo que guardaremos el código fuente de cada ejercicio para poder continuar sobre la última versión del código, manteniendo así el **código fuente de cada versión**.

La rama *develop* se ha marcado por defecto como la rama principal (en lugar de *master*).

Además, se han protegido las ramas *ejercicio\_2*, *ejercicio\_3*, y *ejercicio\_4*, para que sólo puedan editarse haciendo un merge automático de la rama *develop* en las mismas.



The screenshot shows a project management interface for a repository named G80.T2.EG. The sidebar on the left lists various project sections: Project overview, Repository (selected), Files, Commits, Branches (selected), Tags, Contributors, Graph, Compare, Charts, Issues (0), Merge Requests (0), CI / CD, Operations, Wiki, Snippets, and Members. The main content area displays the 'Branches' page, which includes a navigation bar with Overview, Active, Stale, All, Filter by branch name, Delete merged branches, and New branch buttons. Below this is a table titled 'Active branches' containing three rows:

Branch	Last Commit	Actions
develop	24a267d2 · Added Maven support to project · 44 minutes ago	Merge request Compare
master (default)	dc5ac0ba · Initial commit · 18 hours ago	
ejercicio_1	dc5ac0ba · Initial commit · 18 hours ago	Merge request Compare

*Creación de las ramas correspondientes para poder realizar la estrategia acordada.*

## 1.8. Otros conceptos a tener en cuenta

### Mirroring

Para facilitar nuestro futuro laboral y guardar el código, hemos creado el repositorio <https://github.com/d3sd1/uc3m-pds-eg> con nuestra cuenta de GitHub. Dicho **repositorio privado** está enlazado en modo “mirror” con el repositorio de **GitLab de la UC3M**. Así, en caso de que el servidor se caiga, seguiremos disponiendo del código (**alta disponibilidad**) para poder seguir trabajando. El mirror se ejecuta en cada actualización del repositorio (branch management, push, ...).

El propósito principal del mirroring, en este caso, es **mantener** todo nuestro **código** bajo nuestro **propio repositorio** para tener más código que enseñar de cara a entrevistas de trabajo y/o discusiones varias de experiencia.

Esto se ha logrado creando un token de acceso con *GitHub*, y añadiéndolo mediante el menú *Settings > Repositorio > Mirroring*.

Esto se ha realizado tanto para el proyecto final como para los ejercicios.

### Wiki

La documentación del repositorio (sección *Wiki*) contiene documentación técnica del repositorio.

### Snippets

La sección snippets contiene pequeñas piezas de código, *workarounds*, y funcionalidades básicas para reutilizar.

### Issues

Hemos gestionado incidencias y bugs mediante la sección *Issues*. Dentro de ella se pueden ver los bugs actuales y los resueltos.

## 2. NORMATIVA Y COLECTIVIDAD DE CÓDIGO

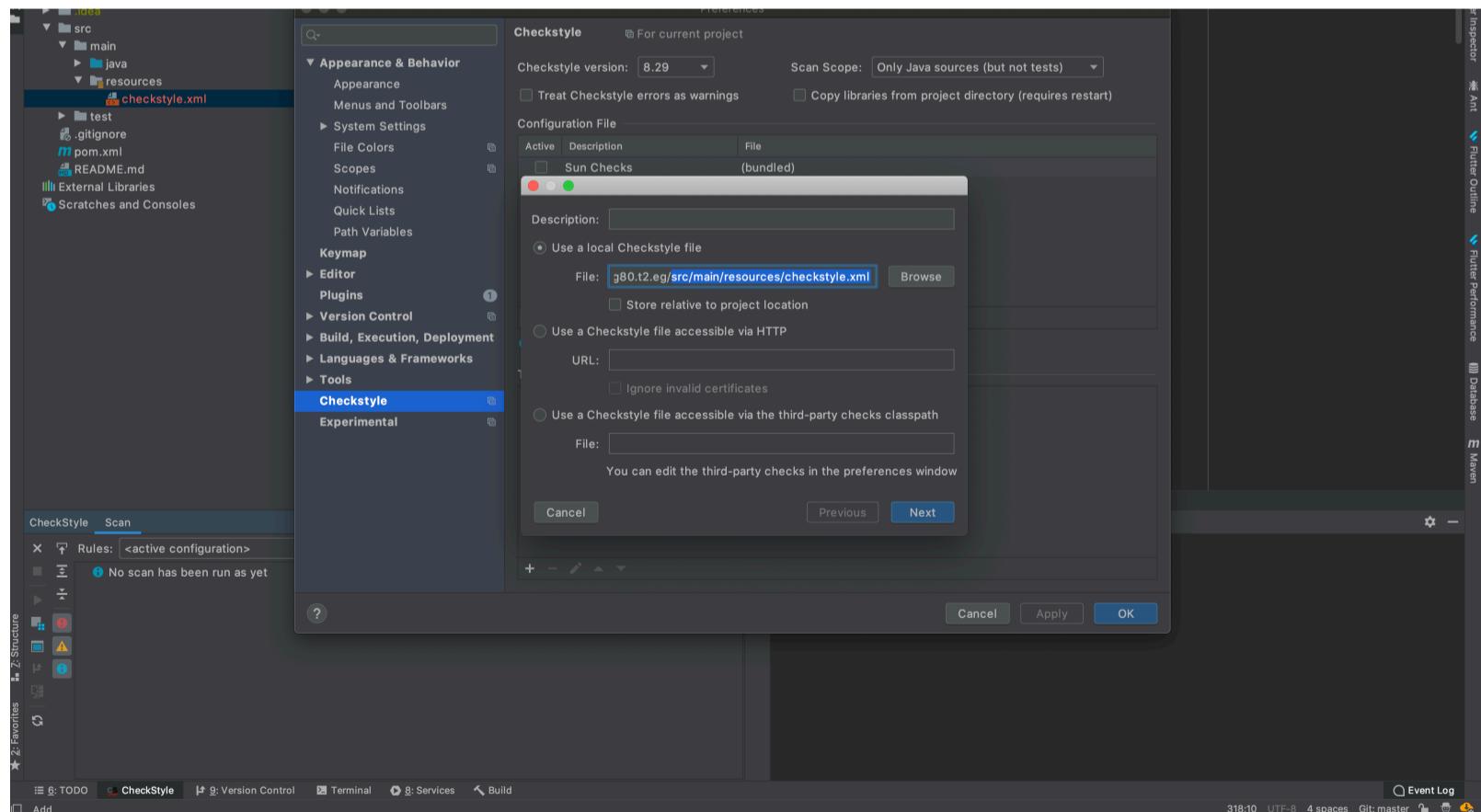
Determinamos el uso de *lint* como estrategia de revisión, estandarización y análisis de código.

Esta elección se debió a la **utilización inter-operativa**, para tener **preconfigurado** el entorno de desarrollo (ya que *lint* se integra con Maven) y no tener que configurar cada equipo de forma individual (como es el caso de *CheckStyle*).

No obstante, esta **tarea fue denegada**, y nos ceñimos al enunciado de la **práctica**, el cual **narra explícitamente el uso de *CheckStyle***.

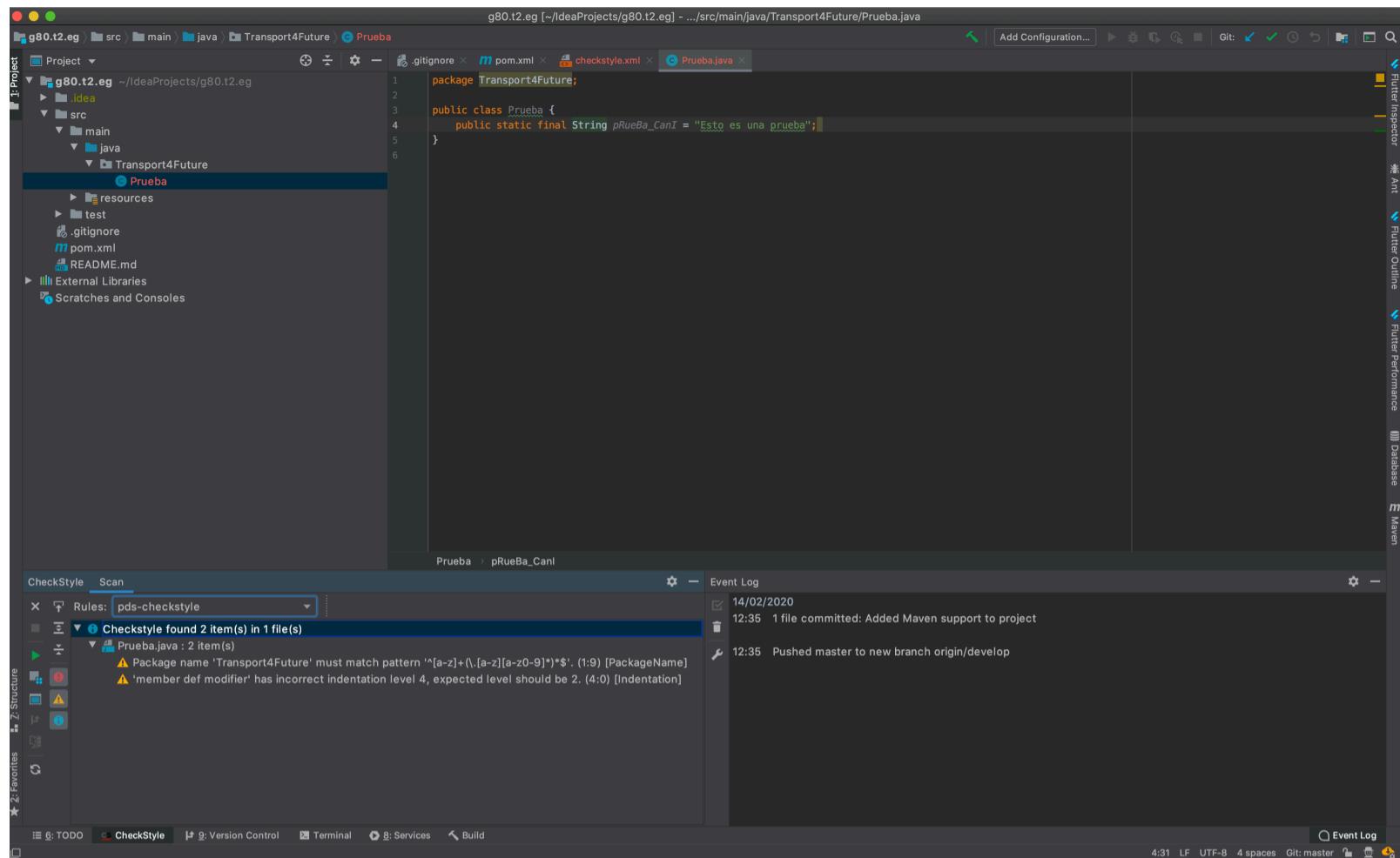
Hemos creado un *CheckStyle* propio, el cual puede apreciarse bajo la ruta *src/resources/checkstyle.xml* del repositorio. No podemos documentar todos los estándares, pero sí los importantes. Lo hemos integrado en el *IDE* utilizando el plugin *CheckStyle-Idea*, y agregando la ruta correspondiente al fichero.

Nos hemos **basado** en el *CheckStyle* de *Google* ya que determinamos que se nos adaptaba al completo, editando cualquier **sección** que consideramos y **adaptándola al proyecto**.



*Integración de CheckStyle propio en IntelliJ Idea, utilizando el plugin CheckStyle-Idea.*

# PRINCIPIOS DESARROLLO DE SOFTWARE



Prueba de uso de CheckStyle, el cual está funcionando correctamente y nos indica.

## 2.1. Estándar para la organización de archivos

- El nombre de archivo es sensible a mayúsculas y minúsculas.
- El nombre de archivo debe contener la extensión **.java**.
- El nombre de archivo no debe contener espacios ni caracteres especiales.
- Los paquetes deben ser escritos en minúsculas y no contener caracteres especiales.

## 2.2. Estándar para nomenclaturas y variables

- Las **variables** deben ser **CamelCase** capitalization.
- Las **constantes** deben ser letras mayúsculas y usar **\_** donde hubiese espacios.

## 2.3. Estándar para métodos y clases

- Los métodos que ejecuten una función determinada (no contamos setters, getters, **toString**, ...) deben estar documentados con **JDoc**.

## 2.4. Estándar para el tratamiento de excepciones

- Las **excepciones** deben ser **escaladas** y no usar try catch en exceso, es decir, lanzar excepciones desde los métodos.

## **2.5. Estándares extra**

- Cada archivo debe incluir la **licencia del proyecto** y no la personal.
- La **importación de paquetes no puede ser wildcard**. Debe ser una importación limpia del paquete en cuestión y no pueden ser importados estáticamente.

## **2.6. Estándar para el diseño y patronización de la aplicación**

- La **indentación es libre** y se utiliza la del IDE actual (ya que IntelliJ detecta la indentación actual y la establece como por defecto para todos los refactor).

## **2.7. Estándar para la revisión del código y actualización del repositorio**

- No hay **estándares** para esto, pero **sí integración continua**, que realiza esta tarea de forma automatizada.

### **3. BACKLOG PROYECTO**

#### **3.1. Obtención e importación del código fuente**

El código fuente proporcionado para el ejercicio se encuentra en la página de **Aula Global**. Este código se encuentra comprimido en un archivo .zip (*TokenManagement.zip*), el cual una vez descomprimido muestra dos carpetas conteniendo el código, una de ellas destinada para MAC-OS, y ambas con el código fuente ***TokenManagement***. Una vez ya tenemos el código, el siguiente paso es importarlo al IDE que utilizamos para los ejercicios, dentro de la carpeta SCR del repositorio de GitLab.

Para importar nuestro código al repositorio utilizamos un par de comandos, *ctrl + k* (empaqueta el código) y *ctrl + shift + k*. (Sube el código a la rama dentro del repositorio que sea maestra)

#### **3.2. Integración de Telegram**

Hemos integrado Telegram, la potente herramienta de comunicación social, en nuestro proyecto. De este modo, cada vez que se produce cualquier tipo de actualización sobre el repositorio, recibimos un mensaje en nuestros dispositivos para estar al tanto de todas las actualizaciones. Esto se ha conseguido añadiendo un *GitHook* con el puntero hacia el bot de *Gitlab* de *Integram* (gratuito y open source).

**Nota importante:** En principio íbamos a integrar *Slack*, pero al ser una herramienta de pago, nos decantamos por *Telegram*.

#### **3.3. Integración ReadTheDocs**

En una primera instancia, se propuso la integración de *ReadTheDocs* para el proyecto, pero esta idea fue abolida en pos de la documentación del propio repositorio (ubicada en la sección *Wiki*), debido a la complejidad inicial y curva de aprendizaje del lenguaje *Sphinx* (el cual utiliza *ReadTheDocs*). No obstante, esta herramienta fue integrada y testeada satisfactoriamente.

#### **3.4 Ética del proyecto**

Se ha seguido la ética otorgada en las primeras sesiones de clase, además de la no inclusión de software de terceros, a excepción de *integram*, un bot de telegram que permite la integración de la app de mensajería con GitLab.

### 3.5. Fichero README.md

Se ha modificado el fichero README.md para determinar las FAQ's básicas del proyecto y determinar a primera vista.

### 3.6. Uso de SourceTree

Hemos decidido por votación unánime no usar *SourceTree*, en pos de la utilización de *IntelliJ Idea Integrated Git System*.

### 3.7. Maven y pom.xml

Hemos configurado el fichero *pom.xml* para que contenga las siguientes restricciones:

1. Versión de *Java* se compila en su versión 12.
2. Nuestro **proyecto**, su *source* y sus dependencias usan la versión 12 de *Java*.
3. Hemos añadido que la **compilación** se realice sobre **Java 12** en su máquina virtual.

### 3.8. Modificaciones sobre el proyecto para que se adecúe al CheckStyle

Hemos realizado modificaciones sobre el proyecto para que no se produzca ningún *warning* por normativa de código.

### 3.9. Javadoc

Hemos integrado *Javadoc* en el proyecto, documentación ubicada en la carpeta *javadocs* del proyecto. Dicha documentación contiene información relativa a toda la introspección del proyecto, y se regenera automáticamente en cada subida de parche.