PLAN DE GESTIÓN DE LA CONFIGURACIÓN

PAWTEL

Buscador y Comparador de Hoteles para Mascotas

Andrés Martínez Reviriego
Claudio Cortés Carrasco
Daniel Flores De Francisco
David González Martínez
Fernando Castelló Sánchez
Francisco Miguel Jiménez Morales
Javier García Sebastián
Javier Ruiz Garrido
Jorge Gómez de Tovar
Luis Mellado Díaz
Manuel Castillejo Vela
Rafael Castillo Cebolla
Sergio Trenado González
Yesica Garate Fuentes

Fecha: 19/02/2025

Grupo: G11

Entregable: Devising a Project

ÍNDICE

1. HISTORIAL DE VERSIONES

NOMBRE DEL PROYECTO:	Pawtel
CÓDIGO DEL PROYECTO:	G11
DIRECTOR DEL PROYECTO:	Luis Mellado Díaz Daniel Flores de Francisco
FECHA DE ELABORACIÓN:	19/02/2025

HISTORIAL DE VERSIONES			
FECHA	**N° DE VERSIÓN**	**DESCRIPCIÓN**	**ELABORADO POR**
17/02/2025	v 1.0	Versión inicial del documento.	Rafael Castillo Cebolla
19/02/2025	v 1.1	Corrección política de ramas.	Luis Mellado Díaz
19/02/2025	v 1.2	Eliminación de frecuencia de *commits* y *merges*. Aclaración de *tests* en *pull requests*. Cambio de nomenclaturas de carpetas.	Rafael Castillo Cebolla

2. RESUMEN EJECUTIVO

El Plan de gestión de la configuración tiene como objetivo definir los aspectos clave de la gestión de la configuración del proyecto de Pawtel, describiendo la forma en la que los elementos del proyecto se registran y actualizan para asegurar la integridad y consistencia del proyecto. Se abordan temas como las políticas de *issues* y ramas, los formatos de los documentos, los roles y permisos de modificación y las tareas de automatización.

3. DESARROLLO

3.1. GESTIÓN DEL EQUIPO

COMUNICACIÓN DEL EQUIPO

- Plataformas principales: WhatsApp, GitHub, Microsoft Teams y Discord.

- Reuniones: por defecto, virtualmente mediante Google Meet o en su defecto Microsoft Teams o Discord. Extraordinariamente, presenciales.

- Comunicación con los profesores: Correo electrónico vía Microsoft Outlook.

Todos los medios digitales cumplen plenamente las normativas europeas de interacción digital y protección de datos.

ROLES DEL EQUIPO

Dirección

ROL	DERECHOS Y RESPONSABILIDADES
Director (*Project Manager*)	- Supervisar globalmente la ejecución del proyecto con el fin de lograr los objetivos y la calidad propuesta. - Gestionar el repositorio. - Establecer y monitorear el cronograma del proyecto, asegurando el cumplimiento de los plazos establecidos. - Facilitar la comunicación entre los diferentes equipos, promoviendo una colaboración eficiente y un desarrollo efectivo. - Priorizar tareas y asignar recursos de manera estratégica para maximizar la productividad del equipo. - Planificar y dirigir reuniones de seguimiento. - Actuar como punto de contacto principal entre el equipo de desarrollo y las partes interesadas externas (clientes, patrocinadores, etc.).

Frontend

ROL	DERECHOS Y RESPONSABILIDADES
Líder Frontend, UI/UX	- Liderar y supervisar el desarrollo del frontend de la aplicación software. - Diseñar, desarrollar e implementar funcionalidades del frontend de la aplicación software.

Desarrollador Frontend	- Diseñar, desarrollar e implementar funcionalidades del frontend de la aplicación software. - Implementar pruebas para verificar el correcto funcionamiento del frontend de la aplicación software.

Full stack

ROL	DERECHOS Y RESPONSABILIDADES
Líder Full Stack	- Liderar y superivsar el desarrollo de la aplicación software de forma global, enfocándose en la conexión entre backend y frontend.
Desarrollador Full Stack	- Diseñar, desarrollar e implementar funcionalidades de la aplicación software, enfocándose en la conexión entre backend y frontend. - Implementar pruebas para verificar el correcto funcionamiento de la aplicación software, enfocándose en la conexión entre backend y frontend.

Backend

ROL	DERECHOS Y RESPONSABILIDADES
Líder Backend	- Liderar y supervisar el desarrollo del backend de la aplicación software. - Diseñar, desarrollar e implementar funcionalidades del backend de la aplicación software.
Desarrollador Backend	- Diseñar, desarrollar e implementar funcionalidades del backend de la aplicación software. - Implementar pruebas para verificar el correcto funcionamiento del backend de la aplicación software.

DevOps

ROL	DERECHOS Y RESPONSABILIDADES
DevOps	- Gestionar y mantener el despliegue de la aplicación software, incluyendo automatización, integración continua (CI) y despliegue continuo (CD). - Desarrollar y mantener la página web.

Gestión

ROL	DERECHOS Y RESPONSABILIDADES
Analista	- Analizar los requisitos del proyecto.
QA y Prueba	- Verificar el correcto cumplimiento de los requisitos. - Diseñar y documentar pruebas para verificar el correcto funcionamiento de la aplicación software.
Publicidad y Marketing Documentación	- Desarrollar, implementar y monitorizar estrategias de marketing y comunicación para publicitar el proyecto y el producto. - Crear y mantener documentación técnica, funcional y de usuario de carácter global del proyecto.

TOMA DE DECISIONES

En caso de desacuerdo, los directores del proyecto (*Project Manager*) se reservan el derecho último de decisión, referente a cualquier ámbito del proyecto. Cualquier desacuerdo a cualquier nivel puede ser elevado a los directores.

Cada miembro tendrá potestad de decisión única y exclusivamente bajo los derechos y responsabilidades definidos en sus roles pertinentes.

En caso de desacuerdo, los jefes de equipo tendrán autoridad por encima del resto de integrantes del equipo. En caso de desacuerdo total, elevar a dirección.

Para decisiones de suma importancia del proyecto, se considerará la opinión de los miembros del proyecto implicados en la decisión, y. de forma general, ésta se someterá a votación mayoritaria.

La asignación de tareas será responsabilidad, en un principio, de los directores del proyecto. Sin embargo, los miembros del equipo pueden realizar solicitudes.

Cualquier opinión justificada de cualquier miembro del proyecto, competente o no de las decisiones en cuestión, es siempre bienvenida.

RESOLUCIÓN DE CONFLICTOS Y SANCIONES

El equipo busca fomentar el buen clima del que parte. No es la inteligencia lo que valoramos, sino la dedicación, el esfuerzo y el interés. Buscamos gente comprometida al proyecto, que no tema fallar, sino que quiera hacer las cosas bien, estando dispuesta a aceptar sus errores, aprender de ellos y seguir trabajando hasta lograr sus objetivos.

Si bien siempre se optará primero por la motivación y el apoyo entre los compañeros, en los casos en los que miembros demuestren un muy pobre rendimiento y un incumplimiento de sus acordadas responsabilidades, se responsabilizará al individuo de su culpa y se aplicará un sistema de penalizaciones. Del mismo modo, se recompensará a los miembros que cumplan de manera destacada con sus compromisos. Todas las incidencias serán registradas en el documento Registro de incidencias.

Para más detalle, consultar el documento de Compromiso de participación, disponible en el repositorio.

3.2. GESTIÓN DE ARTEFACTOS

ALOJAMIENTO DE ARTEFACTOS

Se usará un único repositorio de GitHub para almacenar todos los artefactos de código, de este modo se facilitará la integración continua.

La documentación quedará alojada en su totalidad en el canal de Microsoft Teams del equipo.

PROCESO DE INTEGRACIÓN CONTINUA Y DESPLIEGUE CONTINUO

Para llevar a cabo un sistema de integración continua (CI) y despliegue continuo (CD), nos basamos en los siguientes conceptos:

1. El código se mantiene en un único repositorio remoto. Los repositorios locales se extraen de él.
2. Los *builds* se automatizarán con herramientas de Git.
3. Los *commits* se realizarán con frecuencia para asegurar la integración continua.
4. Todos los *commits* realizados en el repositorio remoto lanzan un trabajo de CI.
5. Las pruebas serán realizadas en una copia del entorno de producción.
6. Los cambios en la rama main remota dispararán un trabajo de CD para desplegar automáticamente la nueva versión.

GESTIÓN DE ACTIVIDADES

Se usarán *issues* de GitHub para representar las actividades a realizar por el equipo y un Project de GitHub para gestionarlas.

POLÍTICA DE RAMAS

El equipo ha decidido descartar GitFlow y adoptar un enfoque alternativo. Por norma general, se creará una rama para cada actividad de código a realizar. Salvo excepciones, las ramas seguirán los siguientes patrones en función del tipo de *issue*.

Si son tareas (*task*):

task/<nombre_de_tarea>/<número_issue>

Si son funcionalidades (*features*):

feature/<nombre_de_feature> /<número_issue>

Si son incidencias (*incidence*):

incidence/<nombre_de_incidencia> /<número_issue>

No se permiten caracteres especiales de ASCII como tildes o dos puntos.

He aquí un ejemplo de una *issue*:

feature/login_and_register/33

Entre las excepciones, se incluye la rama main, que funcionará como la rama principal destinada al despliegue y la rama build, que se empleará para realizar cambios específicos en la configuración del entorno.

FRECUENCIA DE COMMIT Y MERGE

No se establece ninguna frecuencia fija de realización de *commit* o *merge* en el repositorio.

CIERRE DE ACTIVIDADES: MERGE DE PULL REQUESTS Y BORRADO DE RAMAS

El encargado de una actividad de desarrollo de código y los tests pertinentes creará sus propias *pull requests* con la rama correspondiente. Él mismo, o en su defecto, un director, asignará como revisor de la *pull request* al miembro que haya sido designado como su revisor al comienzo del sprint. Se le pondrá las etiquetas pertinentes y se conectará con su *issue*. El revisor revisará cualitativamente el contenido de la *pull request* y el cumplimiento con los requisitos. Cabe recalcar que las actividades de desarrollo siempre incluirán la implementación o actualización de las pruebas pertinentes para confirmar que el nuevo código funciona. Las *pull requests*, por tanto, deberán incluir dichos tests.

Si el revisor aprueba la *pull request*, éste mismo se encargará en el momento de hacer *merge*. Si no la aprueba, el revisor dejará abierta la *pull request*, y escribirá sus comentarios en ella. Notificará al encargado de la tarea para que revise los errores. Una vez supuestamente arreglados, los cambios aparecerán en la misma *pull request*, y se volverá a empezar este ciclo.

Una vez se le ha hecho *merge* a una tarea y se ha cerrado, el encargado de la tarea para la cual se ha creado la rama se encargará de borrar la rama.

GESTIÓN DE ACTIVIDADES

El proyecto utilizará Conventional Commits, una especificación que tiene como objetivo hacer que los mensajes de *commit* sean legibles tanto para humanos como para máquinas. La información correspondiente a este apartado se ha extraído de la página oficial. Se escribirá en inglés. Por tanto, los *commits* deben seguir la siguiente estructura:

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Explicación:

- **<type>** se refiere a uno de los varios tipos preestablecidos de operaciones que se pueden realizar en el proyecto. Este atributo es obligatorio. Los tipos determinados y su uso son:

- **feat**: nueva funcionalidad.
- **fix**: corrección de errores.
- **docs**: cambios solo en la documentación.
- **build**: cambios de construcción del repositorio o la aplicación.
- **style**: cambios que no afectan el significado del código (espacios en blanco, formato, punto y coma faltante, etc.).
- **refactor**: cambio de código que no corrige un error ni agrega una funcionalidad.
- **perf**: cambio de código que mejora el rendimiento.
- **test**: cambios en el banco de pruebas.
- **chore**: cambios en el proceso de construcción o en herramientas auxiliares y bibliotecas, como la generación de documentación.
- **revert**: reversión de uno o más *commits*.
- **misc**: cambios que no cumplen con los requisitos anteriores.

- <scope> es el area afectado por el cambio del *commit*. Este atributo es opcional. Por ejemplo, 'api', 'lang', u 'owner'.

- <subject> contiene el mensaje del *commit*, o en otras palabras, la descripción corta del cambio realizado. Incluirá cuando haga falta la palabra clave 'Closes' y el ID de la *issue*, para poder cerrarla automáticamente al aceptar la *pull request* correspondiente. Se usará minúscula al comienzo, y no se usará punto al final. Empezaremos por un verbo en imperativo, a poder ser. Este atributo es obligatorio.

- <body> contiene una explicación más detallada de la motivación del cambio y/o cómo este contrasta con el código anterior. Este atributo es opcional.

- <footer> contiene cualquier información extra, como cambios importantes en la API o referencias a problemas de GitHub o *commits* revertidos. Incluirá el ID de la *issue* (si procede), para poder relacionar el *commit* con ella. Este atributo es obligatorio.

Ejemplo:

```
feat(user-profile): add prophile picture in backend. Closes #123

It adds a new feature that allows users to upload prophile pictures by themselves.
This commit includes the backend implementation. The frontend one will be made soon.

issue: #123
author: David Villa <dv7@example.com>
reviewer: Fabián Ruiz <fr8@example.com>
```

Se hará uso de un *hook* para controlar que no se realiza un *commit* con mensaje erróneo.

MILESTONES DEL PROYECTO

Las fechas marcadas del desarrollo del proyecto serán representadas en GitHub como *milestones*. Las actividades a realizar se vincularán a una determinada *milestone* y deben estar acabadas. Cada sprint contará con una *milestone* que representará su fecha de finalización.

TÍTULOS DE PULL REQUEST

Por motivos de claridad y unificación, el equipo ha definido unas directrices para el nombramiento de las *pull requests*. No es un formato fijo, sino que se basa en los siguientes principios:

- No se usará el nombre por defecto que provee GitHub, el cual *parsea* el nombre de la rama de origen.
- No se usará el formato de Conventional commits.
- Será una descripción breve y sin adornos del objetivo de la *pull request*. Se ha de asemejar en parte al título de la *issue* a la que corresponde, si procede.

VERSIONADO DEL PRODUCTO

El proyecto seguirá las directrices de versionado conocidas como versionado semántico. La información correspondiente a este apartado se ha extraído de la página oficial. Por lo tanto, el versionado funcionará de la siguiente manera:

- Versión mayor: Este número se incrementa cuando se realizan cambios significativos e incompatibles en la API del proyecto. Indica que la nueva versión puede no ser compatible con las versiones anteriores y requiere una cuidadosa consideración durante las actualizaciones. (Ejemplo: 2.2.2 -> 3.0.0)

- Versión menor: Incrementar este número significa la adición de nuevas funcionalidades al proyecto de manera compatible con versiones anteriores. Los usuarios pueden esperar que las características existentes se mantengan intactas, lo que garantiza transiciones suaves entre versiones. (Ejemplo: 2.2.2 -> 2.3.0)

- Parche: Los incrementos en la versión de parche están reservados para correcciones de errores que son compatibles con versiones anteriores. Estos cambios abordan problemas sin introducir nuevas funcionalidades o romper la funcionalidad existente, proporcionando a los usuarios una experiencia fluida durante las actualizaciones. (Ejemplo: 2.2.2 -> 2.2.3)

3.3. GESTIÓN DE ISSUES

ASIGNACIÓN DE ACTIVIDADES

Al comienzo de cada sprint, los directores definirán las actividades a realizar y las asignarán a los miembros del equipo. Contarán con la ayuda de los analistas, jefes de equipo y cualquier miembro necesario. Se intentará equilibrar el tiempo y esfuerzo de trabajo, a la vez que aprovechar las habilidades e intereses de cada miembro.

TIPOS DE ISSUES

Una *issue* representará una necesidad o solicitud de cambio para el sistema. Consideraremos los siguientes tipos:

- Funcionalidad (Feature): nueva funcionalidad para el proyecto.

- Tarea (Task): actividades adicionales para el proyecto que no sean funcionalidades.

- Incidencias (Incidence): creadas por el equipo o por usuarios para reportar un problema con el funcionamiento o una solicitud de cambio.

A las *issues* generadas automáticamente por bots no se les aplicarán estas políticas.

ROLES EN ISSUES

Cada *issue* será asignada a un único miembro del equipo para su ejecución. Si se necesitara llevarla a cabo por una persona más, se crearía una *issue* adicional, repartiendo el trabajo.

PRIORIDAD DE ISSUES

- Alta: tareas críticas para el progreso del proyecto. Estas tareas afectan directamente el avance, ya que otras tareas dependen exclusivamente de su finalización.

- Media: tareas que tienen una doble función: dependen de otras tareas para empezar o completarse y, al mismo tiempo, otras tareas dependen de ellas. Son importantes, pero no bloquean el progreso de manera inmediata.

- Baja: tareas independientes que no tienen ninguna otra tarea que dependa de ellas. Su ejecución no afecta directamente el avance del proyecto.

ESTADOS DE ISSUES

Una *issue* de tarea pasará por cuatro estados. En GitHub Projects serán representados mediante columnas.

ESTADO	DESCRIPCIÓN
TODO	Para tareas sin empezar.
In Progress	Para tareas en las que se está trabajando.
Pending review	Para tareas supuestamente acabadas sin errores de integración, pero que aún necesitan revisión.
Done	Para tareas aprobadas.

ETIQUETAS DE ISSUES

Se usarán las siguientes etiquetas para definir un cambio:

- **Tipo de issue:** feature, task e incidence.
- **Tipo de cambio:** build, database, deployment, documentation, feat, fix, hotfix, meeting, refactor, release, style y test.
- **Área del sistema:** backend, frontend, full stack.

NOMENCLATURA DE ISSUES

Nomenclatura de funcionalidades

El título de las funcionalidades seguirá el siguiente patrón:

```
Feature: <Nombre de la Feature>
```

El cuerpo seguirá la siguiente estructura:

```
## <Titulo de la feature>

### Descripción:
<Explica la nueva funcionalidad propuesta>

**Criterios de aceptación:**
<
- Debe permitir...
- El usuario podrá...>

**Subtareas:**
<
- Crear estructura de base de datos
- Implementar API
- Diseñar interfaz>

**Dificultad:** <Fácil | Media | Difícil >
**Prioridad:** <Baja | Media | Alta>
**Tiempo estimado:** <X horas/días>
```

Nomenclatura de tareas

El título de las tareas seguirá el siguiente patrón:

```
Tarea: <Nombre de la Tarea>
```

El cuerpo seguirá la siguiente estructura:

```
## <Título de la Tarea>

**Descripción:**
<Descripción del trabajo a realizar>

**Prioridad:** <Baja | Media | Alta>
**Tiempo estimado:** <X horas/días>
```

Nomenclatura de incidencias

El título de las incidencias seguirá el siguiente patrón:

```
Incidencia: <Nombre de la incidencia>
```

El cuerpo seguirá la siguiente estructura:

```
## <Título de la incidencia>

### Descripción:
<Explica el error encontrado>

### Pasos para reproducir:
<
1. Ir a...
2. Hacer clic en...
3. Se produce el error...>

### Resultado esperado:
<Qué debería ocurrir en su lugar>

### Resultado actual:
<Qué ocurre realmente>

### Posible solución:
<Si se conoce, explica cómo solucionarlo>

###  Impacto:
<Cómo afecta el bug al sistema o a los usuarios>

**Prioridad:** <Baja | Media | Alta>
**Tiempo estimado:** <X horas/días>
```

3.4. GESTIÓN DE DOCUMENTACIÓN

La documentación constituye una excepción a varias de las normas aquí descritas para el resto de los artefactos del proyecto.

NOMENCLATURA DE CARPETAS DE DOCUMENTACIÓN

Las carpetas respectivas a la documentación en Microsoft Teams se escribirán según el formato *snake case*, es decir:

- Las diferentes partes del nombre serán separadas mediante un guion, '-'.
- Se escribirá siempre en minúsculas, salvo por nombres propios u otra necesidad.
- Las palabras o números de cada parte se separarán mediante un guión bajo, '_'.

Ejemplo de una carpeta simple:

```
organización_semanal
```

Ejemplo de una carpeta con excepción de mayúsculas:

```
PMBOK
```

NOMENCLATURA DE DOCUMENTOS

Con tal de mantener una consistente legibilidad en el proyecto, se ha determinado un formato estricto para el nombramiento de documentos.

Por norma general, se usará el siguiente patrón:

- Las diferentes partes del nombre serán separadas mediante un guion, '-'.
- En casos en los que la fecha sea necesaria, ésta se colocará al principio por norma general.
- Cada parte empezará por mayúsculas y continuará en minúsculas, salvo por nombres propios, siglas u otra necesidad.
- Las palabras o números de cada parte se separarán mediante un guion bajo, '_'.

En resumen, la estructura es la siguiente:

```
[Fecha]-<Parte_1_nombre>-[...]-<Parte_n_nombre>.<extensión>
```

Ejemplo de un archivo simple:

```
Plan_de_gestión_de_la_configuración.pdf
```

Ejemplo de un archivo con fecha:

```
17_02_2025-Reunión_inicial.md
```

Ejemplo de un archivo con nombre por partes:

```
Semana_1-Informe_de_tiempo-Resumen.pdf
```

En caso de crear un archivo nuevo de una serie ya existente, se mantendrá la estructura del nombre de los archivos previos, suponiendo que los anteriores cumplían con la nomenclatura vigente.

FORMATO Y ORGANIZACIÓN DE DOCUMENTOS

Cualquier documento que cuente con plantilla deberá seguirla. En caso de crear un archivo nuevo de una serie ya existente, se mantendrá el formato de los archivos previos.

Las actividades de documentación por normal general no incluirán la creación de ramas y por consecuencia de *pull requests*. En su lugar, tras supuestamente completar la realización de un documento, éste será subido en formato Word editable por su autor a su correspondiente carpeta en el equipo de Microsoft Teams. El mismo autor avisará a su supervisor asignado y se realizará un proceso de revisión y mejora equivalente al de GitHub para el resto de las actividades. Se confía que la edición en Microsoft Teams sea más dinámica. Al término de un sprint, los directores se encargarán de convertir los documentos a formato PDF y subirlo al repositorio.

4. APROBACIÓN

Nombre	Cargo	Firma	Fecha
Pablo Trinidad	Patrocinador		
Daniel Flores de Francisco	Director del Proyecto		
Luis Mellado Díaz	Director del Proyecto		19/02/2024