Guía de Creación de Ramas y Draft PR en GitHub

> **Propósito de esta guía:** Esta guía describe los pasos necesarios para crear y gestionar ramas en el proyecto, partiendo de la rama `develop`, y cómo abrir y gestionar un `Draft PR` en GitHub para revisión y visibilidad de la tarea desde el inicio. Este flujo de trabajo sigue el modelo de **Git Flow**, que ayuda a mantener el código organizado y el proceso de desarrollo estructurado.

¿Qué es Git Flow?

Git Flow es una estrategia de ramificación que organiza el desarrollo de software mediante un flujo de trabajo basado en varias ramas principales y ramas de trabajo específicas para cada tarea o característica. Al seguir este modelo, garantizamos que:

- **La rama** `master` siempre contenga una versión estable y lista para producción.
- **La rama** `develop` actúe como una rama de integración para desarrollos en curso, donde se prueba el código antes de fusionarse en `master`.
- **Las ramas de características** (`feature`) y **correcciones** (`bugfix`) permiten a los desarrolladores trabajar en tareas o solucionar errores de forma independiente, y luego integrarlas de manera controlada en `develop` mediante Pull Requests.

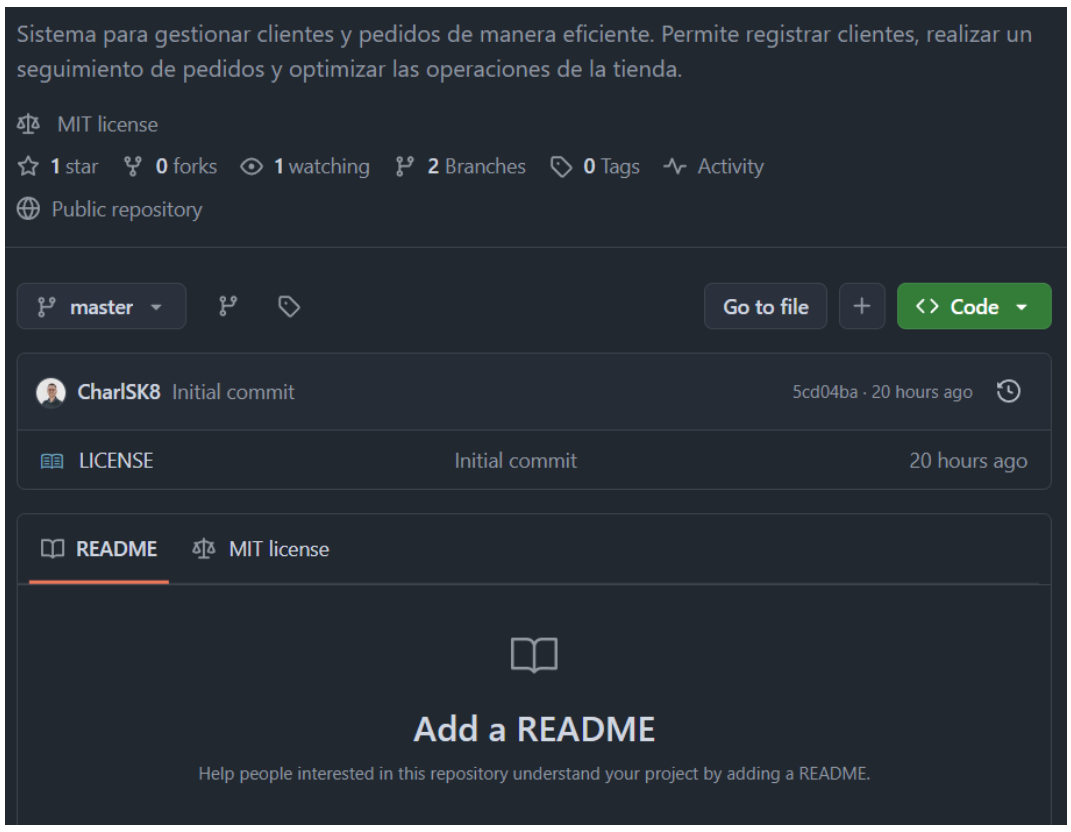
Este flujo garantiza que cada cambio esté documentado y revisado antes de ser fusionado en `develop`, lo que hace que el código en `master` se mantenga estable y listo para producción. A continuación, explicamos cómo implementar Git Flow en este proyecto de manera fácil.

Flujo de Trabajo en GitHub

Para este proyecto, trabajaremos con dos ramas principales en GitHub, además de las ramas que se crearán para cada tarea.

1. Rama `master`

- Contiene el código **estable y probado** del proyecto.
- Solo se fusionarán en esta rama las características y correcciones que hayan sido completamente revisadas y aprobadas en la rama `develop`.
- Representa la **versión de producción** del proyecto.



Vista de la Rama Master en GitHub

2. Rama `develop`

- Es la rama de **desarrollo principal** donde se integrarán las nuevas características y correcciones de errores antes de enviarlas a la rama `master`.
- Todas las tareas y cambios deben hacerse a partir de esta rama, creando una nueva rama en local con una nomenclatura específica y configurando un `Draft PR` en GitHub para que esté visible desde el inicio de la tarea.

1. **Rama `master`:**
   - Contendrá el código estable y probado del proyecto.
   - Solo se fusionarán en esta rama las características y correcciones que hayan sido completamente revisadas y aprobadas en el Develop.
   - Es la versión de producción del proyecto.

   Esta es lo que vemos cuando entramos al repositorio y vemos la rama `master` .

2. **Rama `develop`:**
   - Es la rama de desarrollo principal donde se integrarán las nuevas características y correcciones de errores antes de ser enviadas a la rama `master`.
   - Todos los desarrollos (tareas) y cambios deben hacerse a partir de esta rama creando una rama en tu local usando una nomenclatura y se fusionarán aquí mediante `Pull Requests` en el momento que se la tarea pase de l `In Review` a `Done`.

Vista de la rama develop

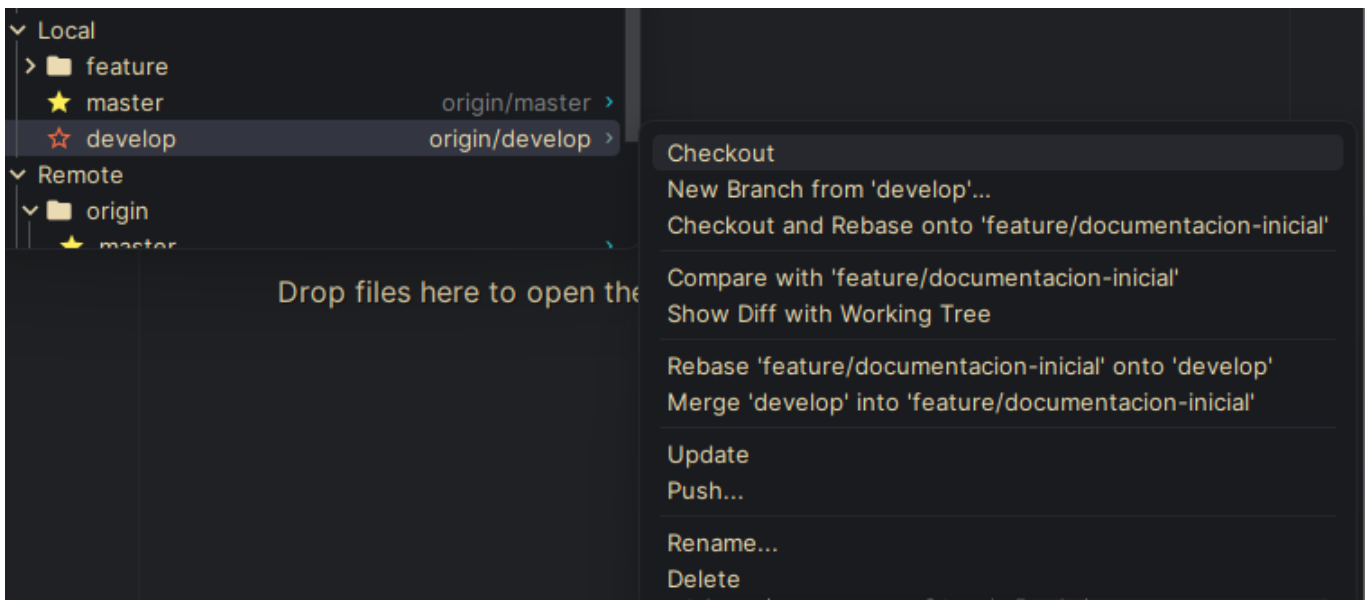
Guía de Creación de Ramas y Draft PR en GitHub (usando IntelliJ)

👩‍💻 Para trabajar en una nueva tarea o funcionalidad, seguiremos este flujo de trabajo.

Siempre tienes que crear la nueva rama desde la rama develop para no tener conflictos.

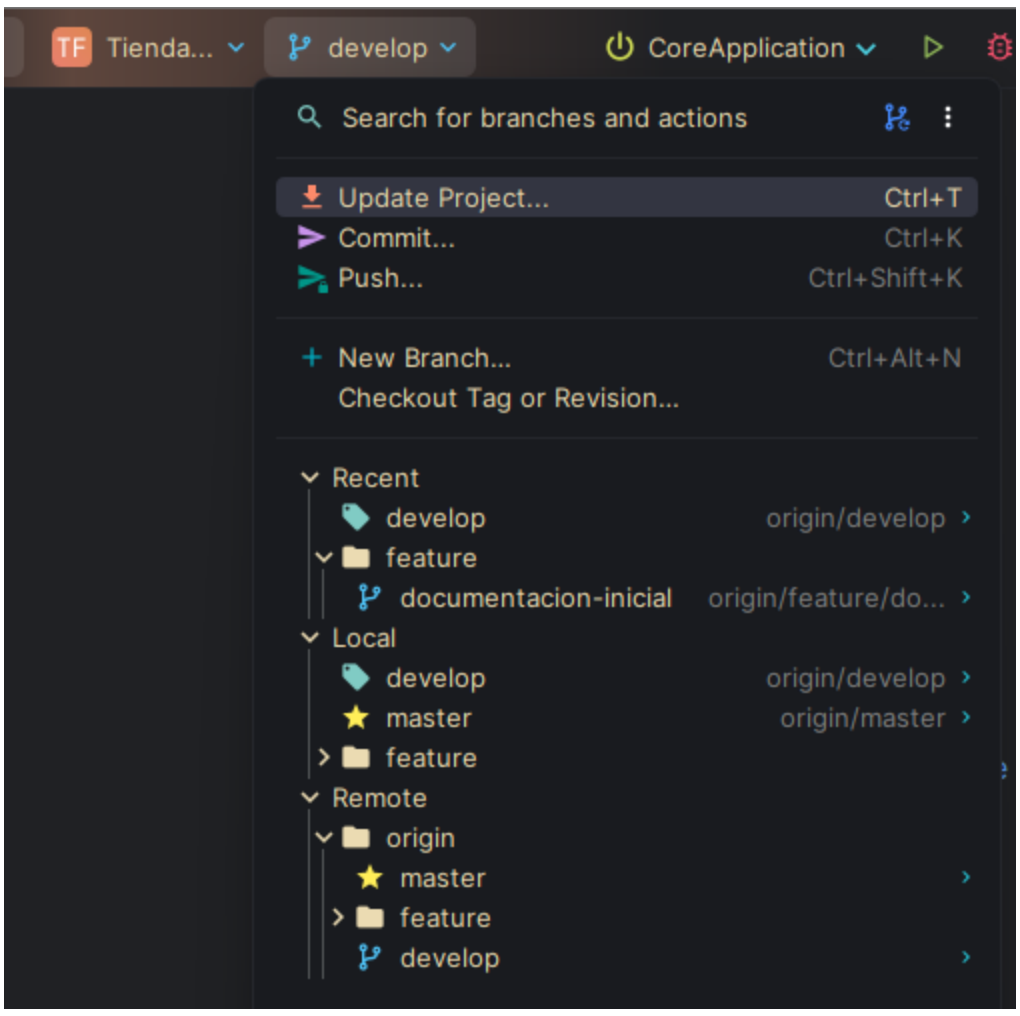
1. Asegurarse de Estar en la Rama `develop`

1. En **IntelliJ**, verifica que estás en la rama `develop` (puedes ver la rama activa en la esquina superiror derecha o en la esquina inferior derecha de la interfaz).
2. Si no estás en `develop`, haz clic en el nombre de la rama y selecciona **develop** y seleciona la opción de **checkuout** .



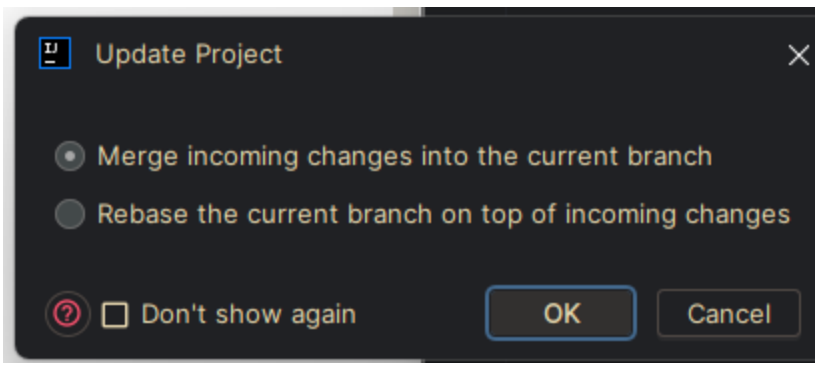
Vista dede el una rama cualquiera para moverte a la rama develop

3. Actualiza la rama `develop` con los últimos cambios del repositorio remoto:
   - Haz clic en `develop` y selecciona **Update Project**.



Vista desde tu git local y también la Remota

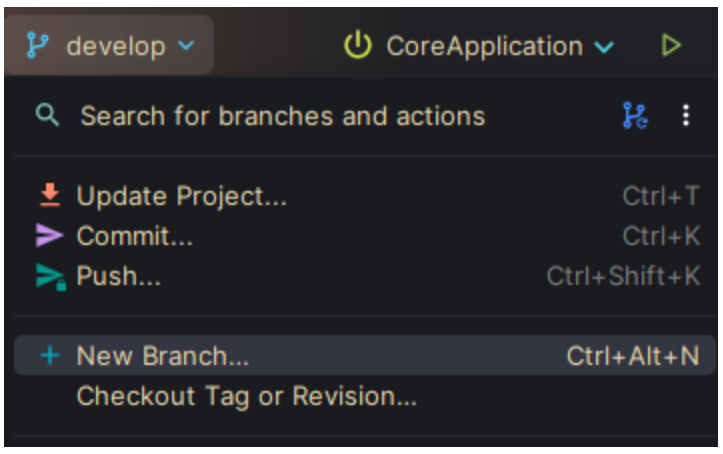
- Me saldra una pequeña ventada dicendo si estoy seguro/a y le damos `ok`.



Vista de ventana Update project, esta bien que te salga esto siempre que actalices la rama.

2. Crear una Nueva Rama para la Tarea

1. Haz clic en el nombre de la rama `develop` en la esquina inferior derecha o esquina superior derecha y selecciona **New Branch**.



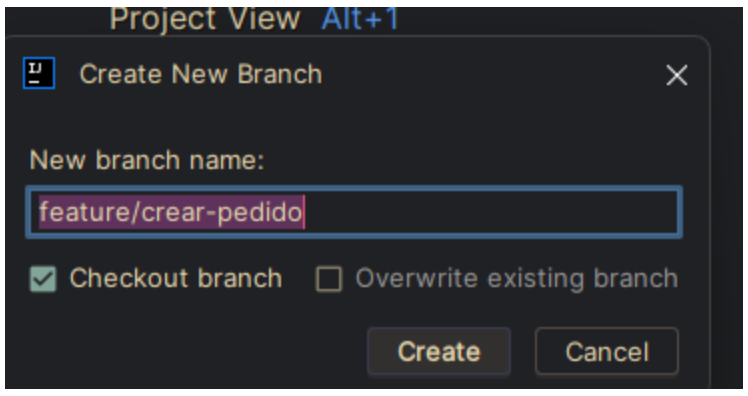
Vista de opciones de la rama develop. Si quieres usar los shortcuts adelante

2. En la ventana emergente, ingresa el nombre de la rama siguiendo la nomenclatura acordada (`feature/nombre-tarea` para nuevas tareas que es nuestro caso por ahora o `bugfix/nombre-tarea` para cuando esten en la rama develop o master).

> 📌 En este caso, vamos a hacer un ejemplo real para nuestro proyecto. Tengo asignada la tarea de Pedidos, pero concretamente voy a trabajar en una tarea específica dentro de la tarea global asignada. De todo el flujo de trabajo que implica el CRUD de Pedidos, voy a enfocarme solo en la creación de la rama para la funcionalidad de "Crear pedido", que es lo más común al abordar una tarea.

3. Haz clic en **Create** para crear la nueva rama de trabajo.
   - Aplicaremos la nomenclatura acordada, `feature/nombre-de-tu-tarea-especifica.`

> 📌 En este caso, la rama se llamará `feature/crear-pedido`.
>
> En esta rama, desarrollaré todo el flujo de trabajo para la funcionalidad de crear un pedido, incluyendo **Model → Controller → Service → Repository**, además del **DTO**, enfocándome solo en la parte de backend.

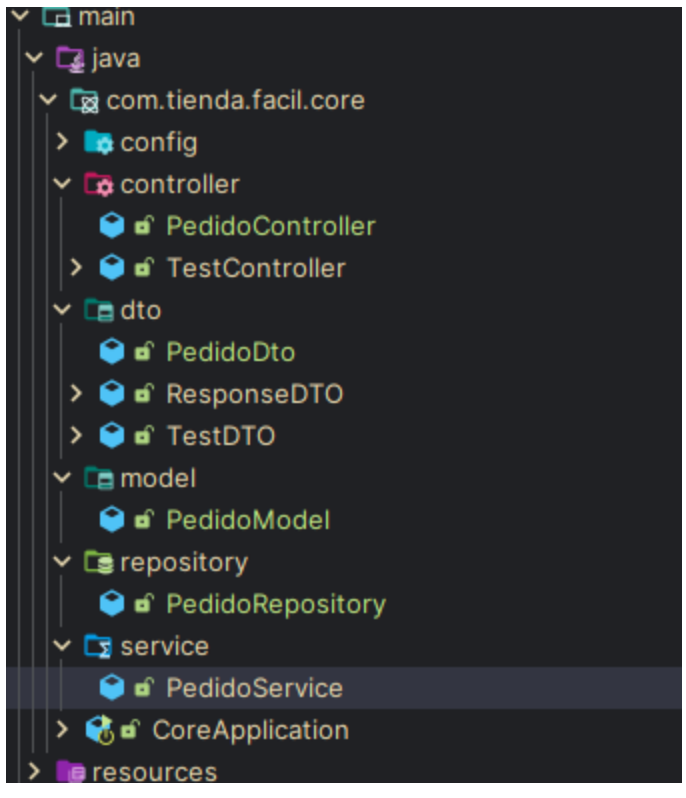


Automaticamente hará `checkout` a la rama creada. En otras palabras esta haciendo `git checkout -b feature/crear-pedido` .

3. Realizar el Primer Commit Mínimo

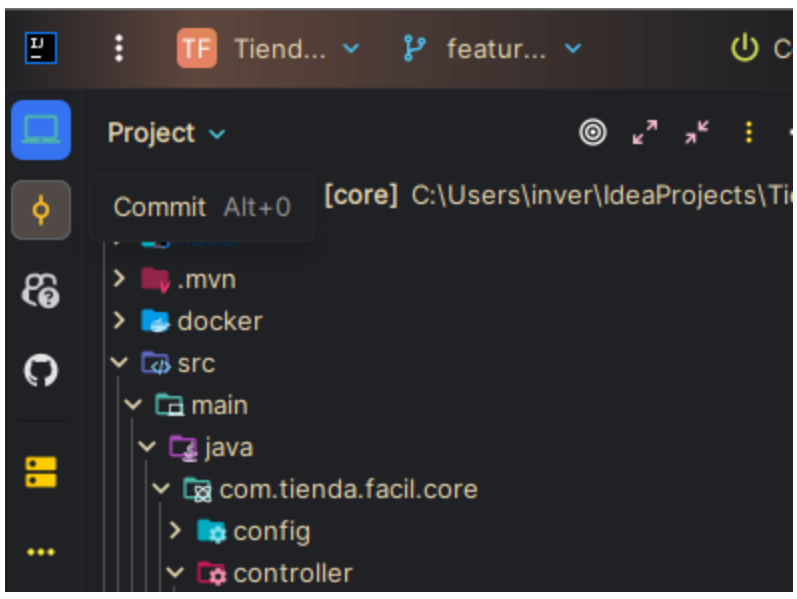
> 📌 Este commit inicial es necesario para tener un punto de control básico en la rama y así poder configurar correctamente el Pull Request (PR), lo cual se explicará más adelante. Este primer commit permite establecer la rama en el repositorio y facilita el seguimiento de los cambios futuros.

1. Crea los archivos iniciales necesarios para la tarea (puede ser un archivo de estructura básica para que Git rastree la rama).
   a. Si seguimos con el ejemplo voy a crear lo siguente:

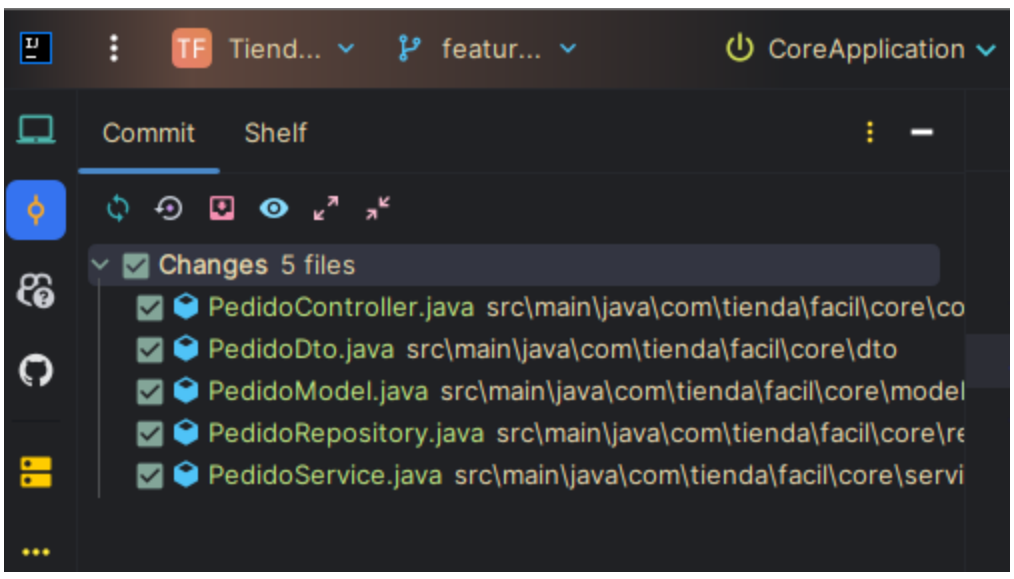


Lo que esta de color verde en mi caso los los archivos necesarios para el flujo necesario del CRUD, concretamente el de crear pedido.

2. Realiza un commit con estos cambios mínimos:
    - Tenemos que ir al gestor de commits, o donde nos muestra todo lo que esta modificado, borrado, archivos sin rastear, creados.



- Debemos selecionas los archivos que queremos hacer comits



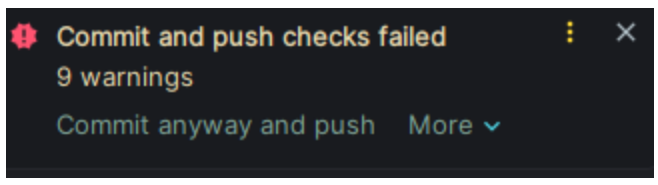
- Escribe un mensaje descriptivo siguiendo la nomenclatura de commits acordada.
    - **Commits descriptivos** para que el historial de cambios sea fácil de entender. En nuestro caso, es **obligatorio** usar una palabra clave al inicio de cada commit para identificar su propósito.

```
feat: indica que el commit implementa una nueva feature.

fix : indica la resolución de un fallo/bug/issue.

docs: indica cambios o nueva documentación.

refactor: indica que se ha refactorizado código, sin cambios en la funcionalidad.

remove : indica que se ha eliminado algo del proyecto.

test : indica cambios o nuevas implementaciones de tests.
```

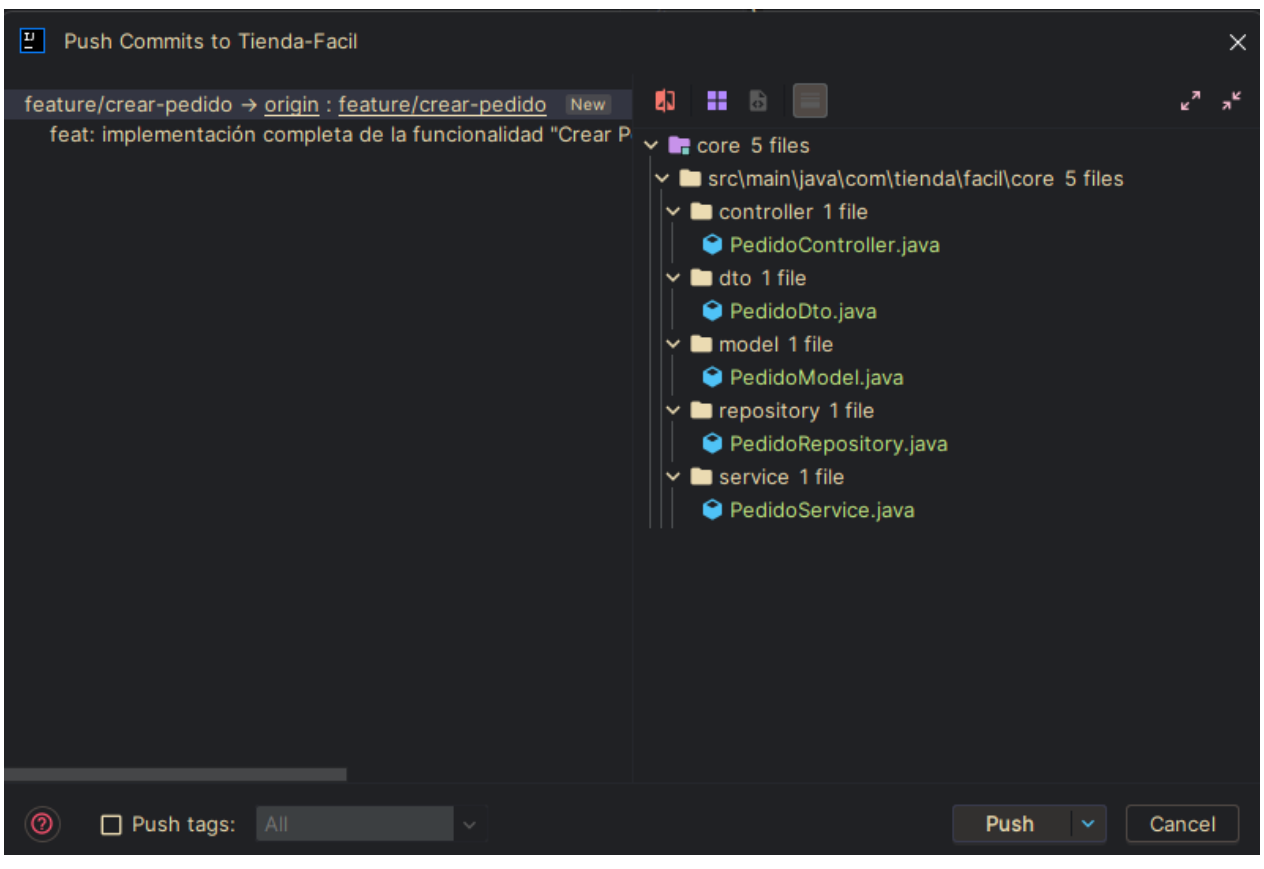
- Haz clic en **Commit**.
    - Siguiendo nuestro ejemplo practico seria lo siguiente:

En este caso hariamos `Commit and Push...` demanera que nos saldria 3 cosas.

- La primera saltara el plugin `SolarLint` avisando de que hay errores, pero por ahora le damos a `continuar` ya que nos dice que no se estan usando (es normal no hay nada solo queremos tener control de la PR).
- La segunda salte pop-up pero como es el primer commit de la funcionalidad, priorizamos tener una base de trabajo en el repositorio. Para futuros commits, es ideal que revisemos y resolvamos estas advertencias antes de hacer el push, manteniendo el código limpio y optimizado desde el inicio a ser posible no necesario por ahora.



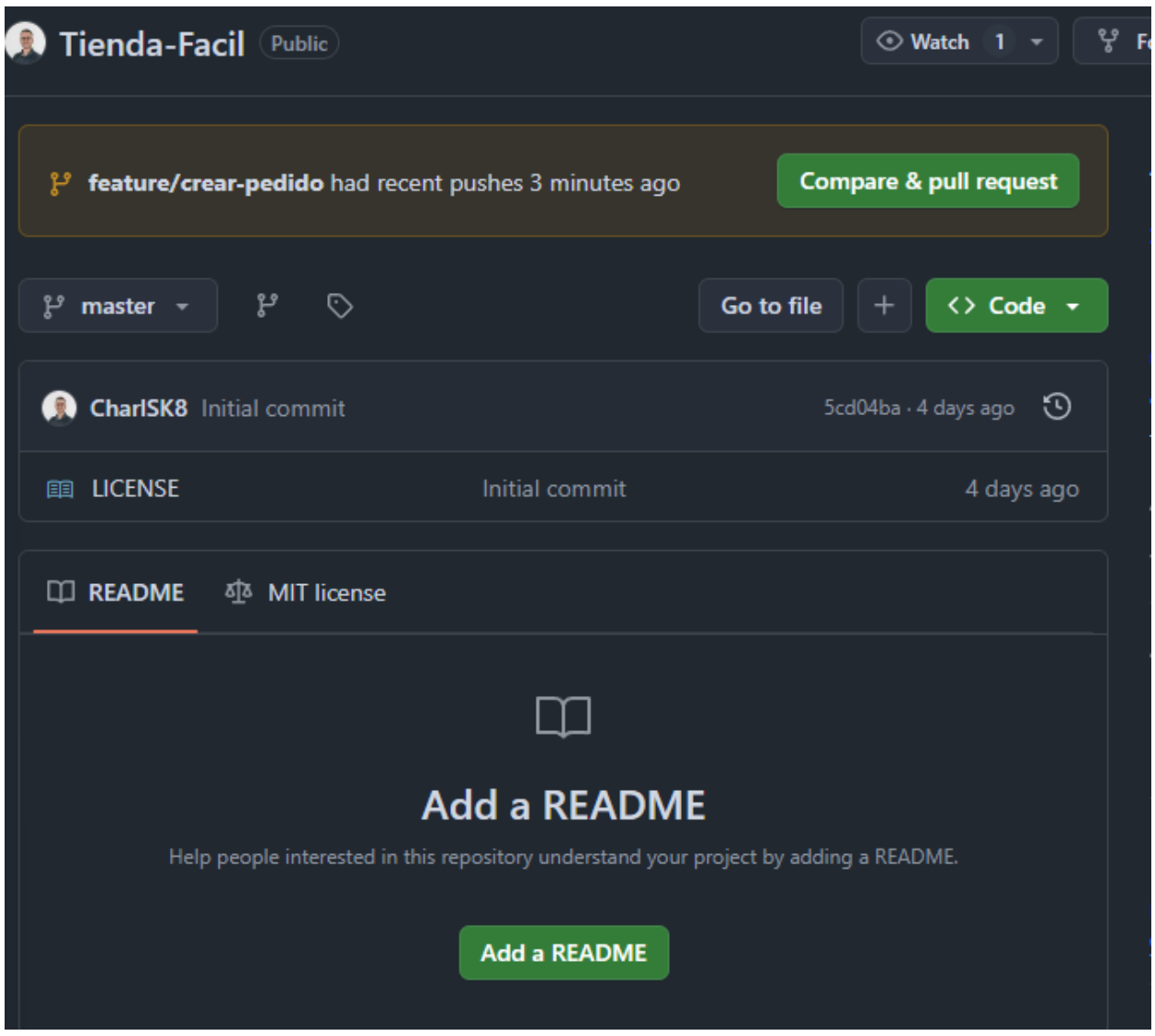
- Ahora no sale el contenido que vamos a enviar con nuestra rama y generando una recomendacion de `Crear Pull Request` esta parte lo continuaremos en la pagina del proyecto de GH.



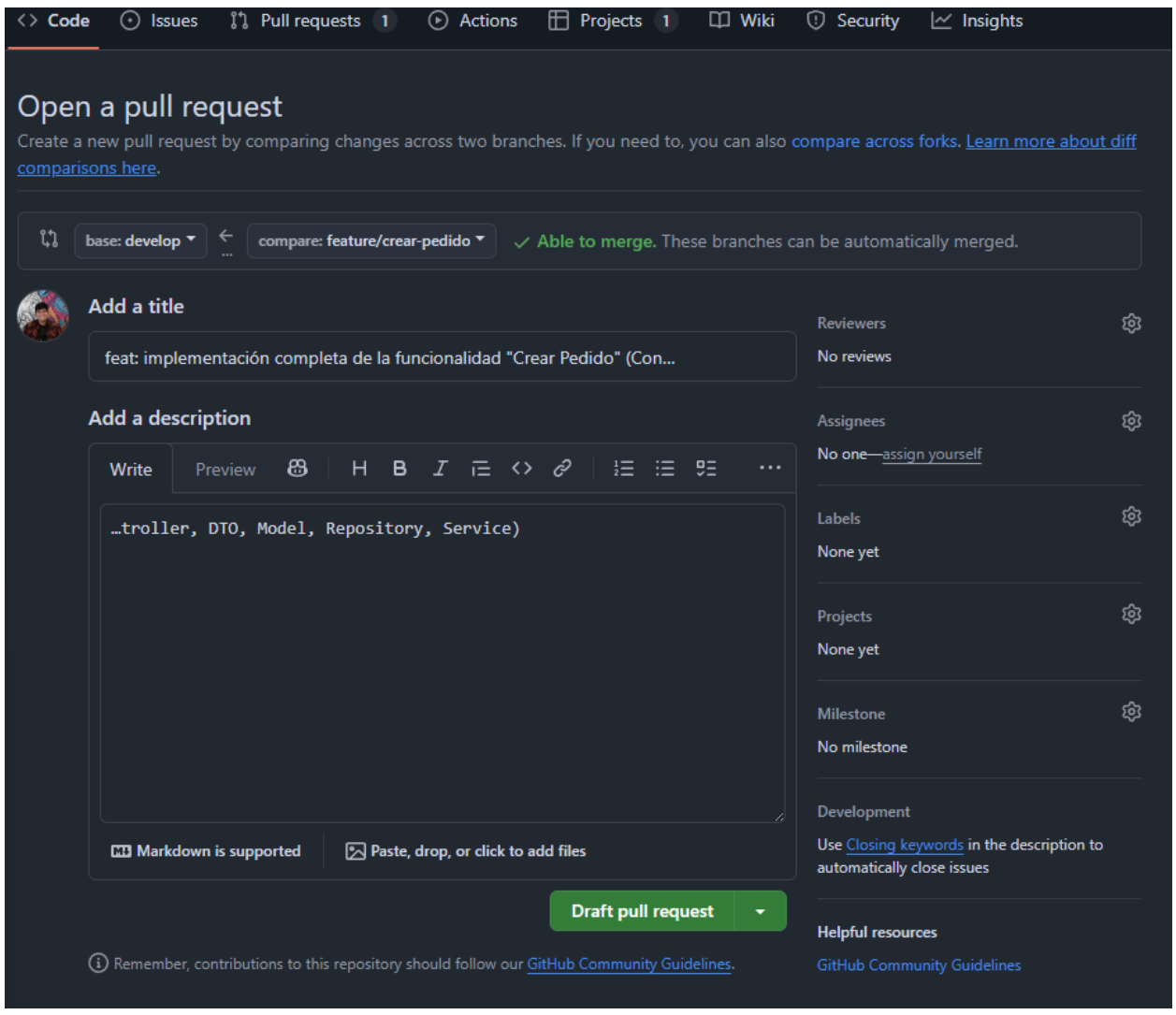
4. Configurar el Draft PR en GitHub

1. Configura el Draft PR en GitHub:
    - Abre GitHub en tu navegador y navega al repositorio.

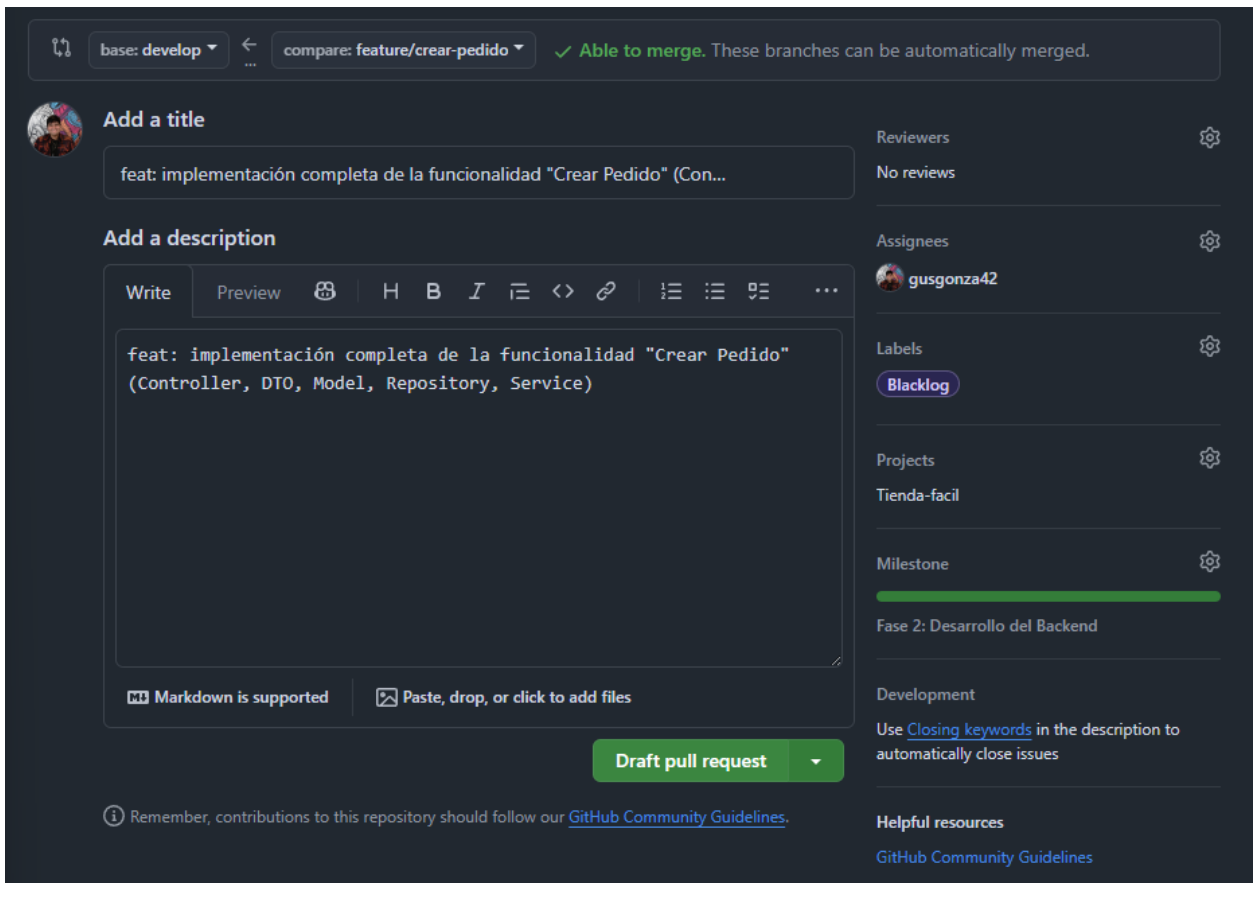
- Veras un notificacion de que hay una notificación donde deberemos hacer el `Compare & pull request.`



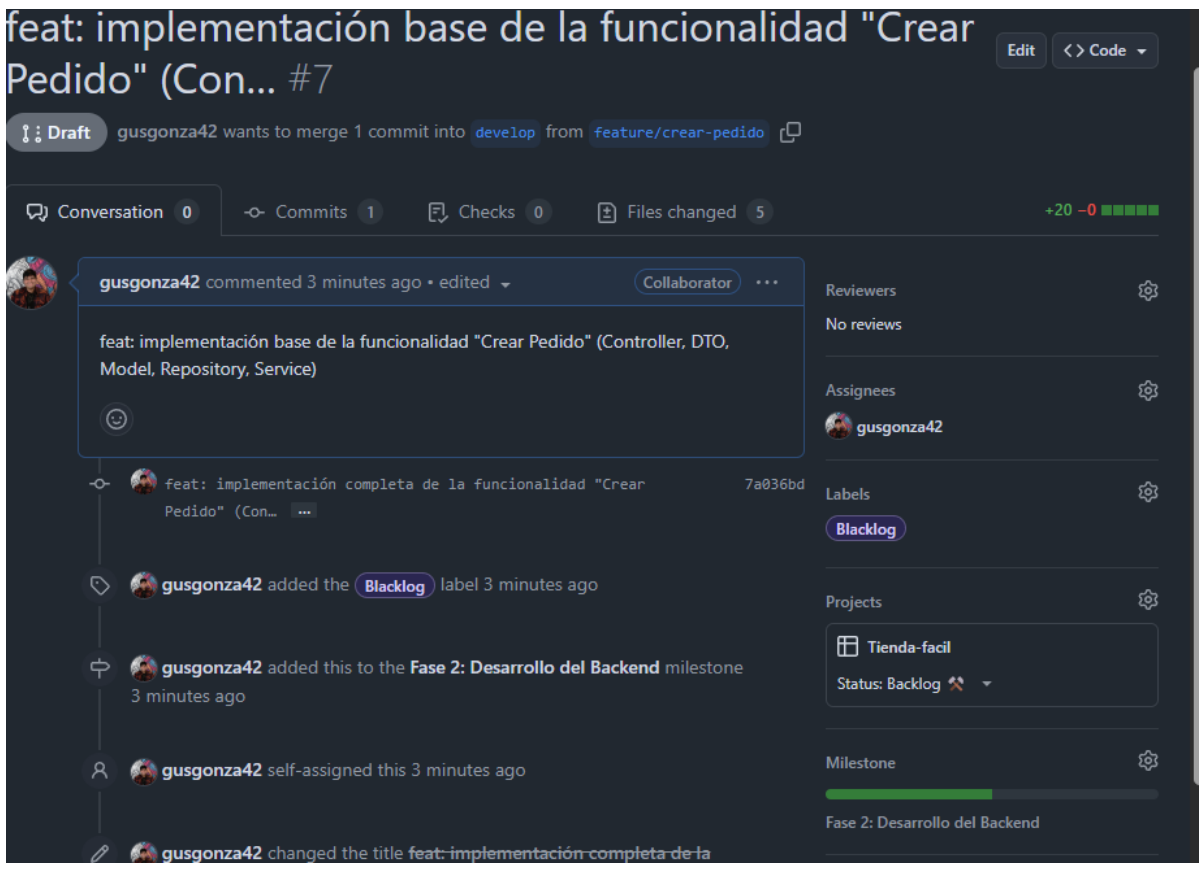
- Deberemos escoger que ramas tenemos que comparar. En nuestro caso siempre va ser cambiar de `base:master` a `base: develop` y a tu izquierda comprar con el nombre de la rama que creaste. En mi caso será `compare:/crear-pedido`. Automaticamente nos saldra esta pantalla.



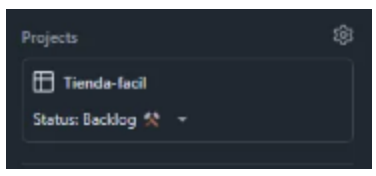
- Configura el Draft PR: asígnatelo, añade las etiquetas (`labels`) correspondientes, asigna el proyecto, y completa los detalles necesarios para el seguimiento de la tarea. Al crear no debes asignar la opción de `Reviewers` eso lo harás una vez termines esta tarea y deberas asignarlos a `Charlinson Perez`.



- En este debes de agregar una descripción, podrias pegar el mismo commit que pusiste en el intelliJ. Y asignas al ser la primera vez:
  - `Reviewers` : lo dejas sin signar
  - `labels`: **backlog/ In progress/ In review**
    - tienes que tu mismo controlar estas tres fases. Al ser la primera vez, lo comun es empezar el `backlog` o `In progress` si vas a empezar directamente.
  - `Project`: eliges Tienda-facil
  - `Milestone`: **fase 1 / fase 2 / fase 3 / fase 3**
    - va enfuncion de la tarea asignada que tienes
  - Ahora lo guardas eligiendo `Draft pull request`
- En la misma página se te mostrara tu Pull Request de tu rama en modo draft
  - Falta agregar en la sección de Projects: `status: no status` tiene que coincidir con lo que tienen en `labels` de arriba.



  - Esta sección debe de coincidir con `Labels`



> 📌 Recuerda que tiene que coincider con las tareas de las tarjetas de TRELLO, para tenerlo al dia. Cualquier duda, pides ayuda si alguna de las guías falta información que te ayude a entenderlo mejor.

5. Desarrollar la Tarea en tu Rama

1. Realiza commits frecuentes desde IntelliJ a medida que avances con la tarea, siguiendo la nomenclatura mencionada.
2. Estos cambios se actualizarán automáticamente en el Draft PR en GitHub, permitiendo que el equipo vea el progreso

6. Marcar el Draft PR como Listo para Revisión

1. Cuando hayas finalizado la tarea, ve al Draft PR en GitHub.
2. Actualiza tanto la sección de `Labels` como de `Projects` pasen de `In process` → `In review`.
3. Y selecciona **Mark as ready for review** para cambiar el estado del PR a revisión.
4. Asigna el PR a la persona encargada de la revisión (`Charlinson Perez`)y solicita oficialmente la revisión del código.

7. Revisión y Fusión en `develop`

1. Una vez que el PR sea aprobado, el responsable se encargara de proceder con la fusión en `develop` directamente en GitHub.
2. ¡Y puedes ya tendrias tu primera aportación al proyecto, sigue asi!

Labels

Las **labels** (etiquetas) ayudan a identificar rápidamente el estado de cada Issue o Pull Request. Utilizamos las mismas etiquetas que en Trello:

- **BACKLOG**: La tarea está pendiente de hacer.
- **IN PROGRESS**: La tarea está en desarrollo activo.
- **IN REVIEW**: La tarea está completada pero necesita revisión.
- **DONE**: La tarea ha sido revisada y aprobada.

Uso de Milestones para Organizar las Fases

Las **milestones** representan cada una de las fases del proyecto. Cada Issue debe estar asignado a una milestone correspondiente según su fase. Esto permitirá ver el progreso de cada fase en GitHub.

1. **Fase 1: Planificación y Configuración**
2. **Fase 2: Desarrollo del Backend**
3. **Fase 3: Desarrollo del Frontend**
4. **Fase 4: Documentación y Entregables del Proyecto**

Buenas Prácticas

- **Commits claros:** Escribe mensajes de commit que describan el cambio realizado.
- **Revisión mutua:** Fomenta que los PRs sean revisados por otro miembro del equipo.
- **Organización en GitHub:** Asegúrate de mantener las etiquetas y milestones actualizadas para facilitar el seguimiento del proyecto.