IES Abastos – DAW 7K
Proyecto Integrador
Grupo:
Brandon Acapa
Rocio Garcia
Diego Ramirez
Oscar Pereira

VOLUNTAPP

WORKFLOW

Organizacion:

El proyecto esta organizado en GitHub, dentro del repositorio se puede encontrar una carpeta para documentación interna y otra para el proyecto en si.

Nuestro repositorio es:
https://github.com/odps/VoluntApp

Como herramienta para la gestion de proyecto usaremos Trello, este se encuentra en:
trello.com/VoluntApp (ctrl + click para abrir enlace)

FLUJO DE TRABAJO EN GIT

Para el flujo de trabajo usaremos la metodologia **GitFlow**

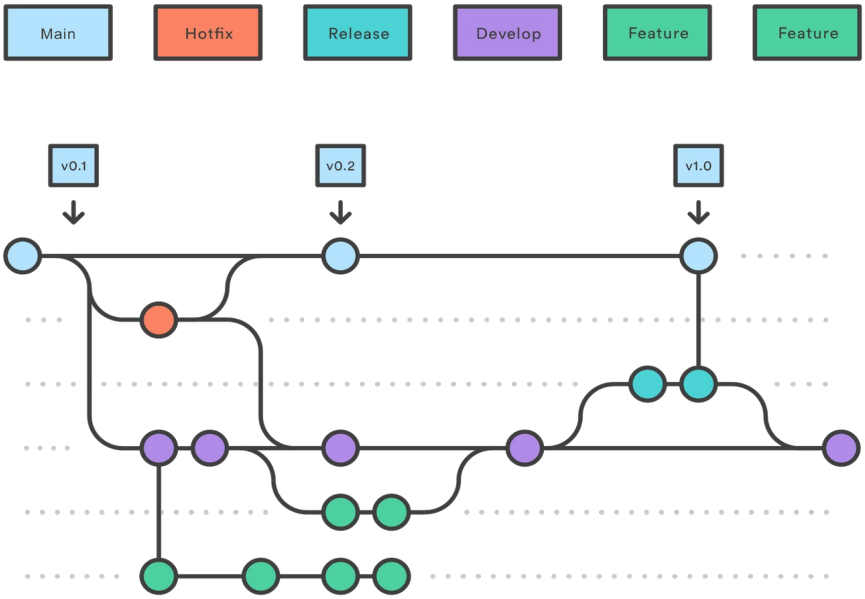
Main:
Exclusivamente para versiones en producción y completamente estables.
Solo se hace merge desde develop tras revisiones exhaustivas

Develop:
Recibe merges desde ramas feature, bugfix.
Es la base del trabajo en curso.

Feature:
Nombres descriptivos, prefijo feat_[nombre_rama] (ej: feat_loginView).
El alcance debe estar limitado a una tarea o funcionalidad específica.

Bugfix:
Para corregir errores críticos de producción. Se basan en main y luego se mergean

tanto en main como en develop, se deben nombrar siguiendo de la siguiente forma: bugfix_[descripción].



COMO PROCEDER A LA HORA DE TRABAJAR

Se comprueba la actividad asignada antes de empezar.

Antes de empezar a trabajar se debe hacer GIT PULL en todas las ramas para trabajar con informacion actualizada.

Verificamos que nos encontremos en la rama DEVELOP y NO en main.

Si es una funcionalidad nueva, se debe crear una rama para esta con nombre feat_[nombre_rama], si ya existe simplemente se continua trabajando sobre ella.

A medida que se va avanzando el proyecto, **recordar hacer commits regularmente**, especialmente al terminar funcionalidades o aspectos significativos de la tarea en curso.

Al hacer commit, los mensajes deben ser descriptivos claros y estructurados: [TIPO]: [Descripcion] (ej: feat: añadir autenticación, fix: corregir validación de formulario).

Antes de hacer PUSH se debe hacer PULL para evitar conflictos con compañeros que puedan estar trabajando al mismo tiempo o hayan hecho algún cambio reciente.

No se debe hacer merge a main hasta no haber hecho un review con el resto de

compañeros.

FLUJO DE TRABAJO AL TERMINAR

Disponemos de un archivo MarkDown llamado CHANGELOG.md, este archivo permite mantener un registro de las novedades y cambios introducidos en nuestro proyecto y que se vean reflejados en GitHub.

A la hora de agregar informacion a este fichero se realiza de siguiendo el siguiente esquema:

[fecha] - [funcionalidad] - [nombre integrantes]
New
Anadida funcionalidad {...}
Modified
Se ha cambiado la logica/estructura de {...}
Fixed
Corregido error en {...}
Deleted
Se elimina del codigo {...}

Estandarización de Tareas

A la hora de repartir el trabajo se debe crear un **"Definition of Done" (DoD)** para cada tarea. Esto asegura que el trabajo esté completo antes de cerrarlo.

Ejemplo de DoD:
El código pasa los linters y pruebas.
Hay documentación o comentarios relevantes.
Ha sido revisado por otro miembro del equipo.

Las funcionalidades grandes se dividen en tareas más pequeñas para facilitar el seguimiento.

Ejemplo:
Crear una vista de login:
-Diseñar el formulario.
-Validar inputs.
-Conectar con la API.

FLUJO DE TRABAJO DENTRO DEL EDITOR

Usaremos VisualStudio como IDE en nuestro proyecto.

El codigo tiene que estar correctamente indentado, usaremos **Prettier** y **ESLint** para homogenizar el estilo.

A la hora de declarar variables y funciones se deben respetar las siguientes normas:

Variables:

Usaremos camel case para los nombres de variables (ej: estoEsUnaFuncion).

Nombres descriptivos y bien escritos.

Contexto claro, evitar usar variables globales siempre que sea posible, si se puede declarar solo en el ambito en el cual se hara uso, hacerlo de esta manera.

Uso de let/const:
- **const:** Para valores que no cambian.
- **let:** Para variables que puedan ser reasignadas.

Funciones:

Evitar que las funciones esten sobrecargadas, si se requiere de alguna funcionalidad mas compleja, dividirlo en funciones auxiliares.

Separar funciones reutilizables en módulos independientes.

Las funciones deben estar declaradas en el documento antes del codigo que hacen uso de ellas.

Comentarios:

Deben incluir información breve pero específica, por ejemplo:

```
/**
 * Función que valida el correo electrónico del usuario.
 * @param {string} email - Correo electrónico a validar.
 * @returns {boolean} - Retorna true si el correo es válido.
 */
function validarEmail(email) {
  // lógica de validación
}
```

Control de Errores:

Usar bloques try/catch donde sea necesario.

Definir mensajes de error informativos.