Universidad Tecnológica de Aguascalientes

PROYECTO PRIMER PARCIAL

Gestión del proceso del desarrollo de software

Profesor:

Froylan Alonso Perez Alanis

Alumno:

Jesus Adrian Ibarra Tiscareño 220478

Documento Técnico - Caso TechNova Solutions

1. Introducción al caso de estudio

TechNova Solutions es una empresa que desarrolla una plataforma web para la gestión de inventarios. El equipo de desarrollo está distribuido en tres ciudades, lo que requiere una estrategia clara de colaboración, automatización y control de versiones.

2. Parámetros de configuración de herramientas utilizadas

Comunicación

- **Slack**
  - ✞ Función: Permite la comunicación en tiempo real entre los equipos distribuidos, con la posibilidad de crear canales temáticos.
  - ✞ Configuración: Se crean canales como #general para anuncios, #development para discusiones técnicas. Se integra con GitHub y Jira para notificaciones automáticas.
  - ✞ Justificación: Se elige Slack por su amplia adopción, integraciones con las demás herramientas y facilidad de uso en entornos distribuidos. Microsoft Teams también es una opción, pero Slack tiene una mayor integración con herramientas de desarrollo.

Gestión de tareas

- **Jira**
  - ✞ Función: Gestión de tareas, sprints y seguimiento de proyectos ágiles.
  - ✞ Configuración: Se configura un proyecto de tipo Scrum, con épicas, historias de usuario y tareas. Se definen flujos de trabajo

personalizados para los estados de las tareas (To Do, In Progress, In Review, Done). Se integra con GitHub para vincular commits y pull requests con las tareas.

- ✞ Justificación: Jira es una herramienta robusta para gestión de proyectos ágiles, con amplias capacidades de personalización y reporting.

Repositorio

- **GitHub**
  - ✞ Función: Almacenamiento del código fuente, control de versiones y colaboración mediante pull requests.
  - ✞ Configuración: Se crea un repositorio con ramas protegidas (main y sprint-*). Se configuran reglas para pull requests (requieren revisión y que las pruebas pasen). Se utiliza el archivo .gitignore para excluir archivos temporales.
  - ✞ Justificación: GitHub fue seleccionado por su integración nativa con GitHub Actions y su ecosistema completo para desarrollo colaborativo.

CI/CD

- **GitHub Actions**
  - ✞ Función: Automatización de la integración y despliegue continuo.
  - ✞ Configuración: Se crea un archivo .github/workflows/main.yml que define el pipeline. Este incluye steps para checkout, instalación de dependencias, ejecución de pruebas y despliegue en staging/producción bajo condiciones.
  - ✞ Justificación: Se elige por su integración directa con GitHub, facilidad de configuración y por ser gratuito para repositorios públicos.

Pruebas

- **Postman**
  - ✞ Función: Pruebas de API REST.
  - ✞ Configuración: Se crean colecciones de Postman para cada API y se pueden ejecutar desde la línea de comandos con Newman en el pipeline.
  - ✞ Justificación: Postman es una herramienta ampliamente utilizada para pruebas de API, con una interfaz gráfica y opciones de automatización.
- **Cypress**
  - ✞ Función: Pruebas end-to-end (E2E) de la interfaz de usuario.
  - ✞ Configuración: Los tests se escriben en JavaScript y se ubican en cypress/integration. Se ejecutan en el pipeline en un entorno headless.
  - ✞ Justificación: Cypress es moderno, fácil de usar y permite escribir pruebas E2E robustas.

Monitoreo

- **Prometheus**
  - ✞ Función: Recolección de métricas de la aplicación y infraestructura.
  - ✞ Configuración: Se configura un servidor Prometheus para scrapear métricas de los servicios. Se define un prometheus.yml con los targets.
  - ✞ Justificación: Prometheus es el estándar de facto para monitoreo en entornos cloud nativos.
- **Grafana**
  - ✞ Función: Visualización de métricas a través de dashboards.

- Configuración: Se configura Grafana para conectarse a Prometheus como fuente de datos y se crean dashboards para monitorear el rendimiento y la salud de la aplicación.
- Justificación: Grafana es la herramienta más popular para visualización de métricas y se integra perfectamente con Prometheus.

3. Plan de pruebas

Sprint 1: Autenticación y Módulo Básico

Entregables:

- Sistema de login y autenticación
- Estructura base de la aplicación
- Configuración inicial de base de datos

Pruebas a aplicar:

- **Unitarias:** Lógica de autenticación, validación de credenciales, encriptación
- **Integración:** Conexión auth-API-base de datos, flujo completo de login
- **Funcionales:** Flujo completo de autenticación desde UI
- **Regresión:** Validación que cambios no rompan funcionalidad base
- **Aceptación:** Validación con PO que cumple criterios de aceptación

Justificación:

Las pruebas unitarias aseguran la lógica crítica de seguridad. Las de integración validan que los componentes trabajen juntos. Las funcionales verifican la experiencia de usuario completa. Las de regresión mantienen estabilidad, y las de aceptación validan cumplimiento de requisitos business.

Sprint 2: Gestión de Productos

Entregables:

- CRUD completo de productos
- Categorización de inventario
- Búsqueda y filtros básicos

Pruebas a aplicar:

- **Unitarias:** Cálculos de stock, validación de campos, lógica de negocio
- **Integración:** Productos-API-base de datos, relaciones categorías-productos
- **Funcionales:** Flujos completos de crear/editar/eliminar productos
- **Regresión:** Validación que nuevo módulo no afecta autenticación

Justificación:

Las pruebas unitarias protegen la lógica de negocio crítica de inventario. Las de integración aseguran consistencia de datos. Las funcionales validan usabilidad. Las de regresión previenen efectos colaterales, y aceptación aseguran valor business.

Sprint 3: Reportes y Dashboard

Entregables:

- Dashboard de métricas de inventario
- Reportes de stock bajo y movimientos
- Exportación de datos

Pruebas a aplicar:

- **Unitarias:** Cálculos de métricas, generación de reportes
- **Integración:** Dashboard con módulos existentes, exportación de datos
- **Funcionales:** Navegación completa del dashboard, generación de reportes
- **Regresión:** Validación completa de todos los módulos integrados
- **Aceptación:** PO valida que reportes proveen insights accionables

Justificación:

Las pruebas unitarias aseguran precisión en cálculos de reportes. Las de

integración validan que el dashboard refleje datos reales. Las funcionales verifican usabilidad de reporting. Las de regresión aseguran estabilidad de sistema completo, y aceptación validan valor para toma de decisiones.

4. Casos de prueba

Prueba Manual

Nombre de la prueba: Validación de Login con Credenciales Válidas y No Válidas

Objetivo:
Verificar que el sistema de autenticación maneje correctamente tanto credenciales válidas como inválidas, mostrando los mensajes y comportamientos esperados en cada caso.

Entradas y condiciones iniciales:

- Usuario de prueba registrado: usuario = "admin_technova", contraseña = "SecurePass123!"
- Navegador web abierto en la URL de la aplicación
- Formulario de login cargado y visible
- Credenciales inválidas: usuario = "usuario_invalido", contraseña = "wrongpassword"

Resultado esperado:

- Caso válido: Redirección al dashboard principal, mensaje de bienvenida "Bienvenido admin_technova"
- Caso inválido: Mensaje de error "Credenciales inválidas. Por favor, intente nuevamente.", formulario mantiene campos vacíos

Herramienta utilizada: Navegador web (Chrome) - Ejecución manual

Ubicación en el repositorio: /tests/manual/login-validation.md

Pruebas automatizadas

1. Postman: Pruebas de API REST para Productos

Nombre de la prueba: Create Product API Test

Objetivo:
Verificar que la API REST para crear productos funcione correctamente y retorne los códigos de estado apropiados.

Entradas y condiciones iniciales:

- Método: POST
- URL: https://api.technova.com/v1/products
- Headers: Content-Type: application/json, Authorization: Bearer {token}
- Body:

```
{
   "name": "Monitor LG 24\"",
   "category": "TECH",
   "stock": 25,
   "price": 299.99
}
```

Resultado esperado:

- Status Code: 201 Created
- Response Body incluya el producto creado con ID generado
- Header Location con URL del nuevo recurso

Herramienta utilizada: Postman + Newman (para CI/CD)

Ubicación en el repositorio: /tests/api/product-api-tests.json

```
// Test script en Postman
pm.test("Status code is 201", function () {
   pm.response.to.have.status(201);
});
```

```
pm.test("Response has product ID", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.id).to.not.be.null;
});
```

---

2. **Cypress: Pruebas de Interfaz de Usuario para Agregar Productos**

Nombre de la prueba: Agregar producto mediante formulario web

Objetivo:
Verificar que un usuario pueda agregar correctamente un producto mediante la interfaz web completa.

Entradas y condiciones iniciales:

- Usuario autenticado en la aplicación
- Navegación a la página "Gestión de Inventario"
- Clic en botón "Agregar Producto"
- Datos de prueba:
    - ✞ Nombre: "Teclado Mecánico RGB"
    - ✞ Categoría: "Periféricos"
    - ✞ Stock: 50
    - ✞ Precio: 89.99

Resultado esperado:

- Formulario se envía correctamente
- Mensaje de confirmación "Producto agregado exitosamente"
- Producto aparece en la lista de inventario
- Stock mostrado = 50

Herramienta utilizada: Cypress

Ubicación en el repositorio: /tests/e2e/add-product.spec.js

```
javascript
describe('Agregar Producto', () => {
  it('debe agregar un nuevo producto al inventario', () => {
```

```
    cy.login('admin_technova', 'SecurePass123!');
    cy.visit('/inventory');

    cy.get('[data-testid="add-product-btn"]').click();
    cy.get('#product-name').type('Teclado Mecánico RGB');
    cy.get('#product-category').select('Periféricos');
    cy.get('#product-stock').type('50');
    cy.get('#product-price').type('89.99');
    cy.get('[data-testid="submit-product"]').click();

    cy.contains('Producto agregado exitosamente').should('be.visible');
    cy.contains('Teclado Mecánico RGB').should('exist');
    cy.contains('50').should('exist');
  });
});
```

3. PyTest: Validación de Cálculos de Stock

Nombre de la prueba: test_calculo_stock_promedio

Objetivo:
Verificar que los cálculos de métricas de inventario (promedio de stock) sean correctos.

Entradas y condiciones iniciales:

- Lista de productos con diferentes niveles de stock:
  - ✞ Producto A: stock = 10
  - ✞ Producto B: stock = 25
  - ✞ Producto C: stock = 40
  - ✞ Producto D: stock = 5

Resultado esperado:
Stock promedio calculado = 20.0

Herramienta utilizada: PyTest

Ubicación en el repositorio: /tests/unit/test_inventory_calculations.py

python

```python
import pytest
from inventory_calculator import InventoryCalculator

def test_calculo_stock_promedio():
    # Arrange
    productos = [
        {'nombre': 'Producto A', 'stock': 10},
        {'nombre': 'Producto B', 'stock': 25},
        {'nombre': 'Producto C', 'stock': 40},
        {'nombre': 'Producto D', 'stock': 5}
    ]
    calculator = InventoryCalculator()

    # Act
    promedio = calculator.calcular_stock_promedio(productos)

    # Assert
    assert promedio == 20.0, f"Se esperaba 20.0 pero se obtuvo {promedio}"
```

5. Flujo de trabajo para el control de versiones

Cómo Organizamos Nuestro Código con Git

- Nuestras Ramas Principales

Rama Principal (main)

Qué es: La rama más importante donde está el código estable

Reglas:

- Nadie puede modificar directamente esta rama
- Todos los cambios deben pasar por revisión
- Siempre debe funcionar perfectamente

Ramas por Sprint

Nombres: sprint-1, sprint-2, sprint-3

Para qué sirven: Juntar todo el trabajo de 2 semanas

Ejemplo: En sprint-1 metemos el login y menú principal

Ramas por Funcionalidad

Nombres: feature/login, feature/inventario

Para qué sirven: Desarrollar una cosa específica sin molestar a los demás

Ejemplo: feature/login es solo para trabajar en el login

Cómo Trabajamos Día a Día

Pasos para agregar una funcionalidad:

- Creo mi rama desde el sprint actual:
  git checkout -b feature/mi-funcionalidad sprint-2
- Trabajo en mi código y hago commits:
  git add .
  git commit -m "Agrego validación del email"
- Subo mis cambios:
  git push origin feature/mi-funcionalidad
- Creo un Pull Request para que revisen mi trabajo

- Qué es un Pull Request (PR)

Es como: Pedir permiso para agregar mi trabajo al proyecto

Lo que tiene que pasar:

- Dos compañeros revisan mi código
- Todas las pruebas pasan automáticamente
- No rompe lo que ya funciona

Solo cuando todo está bien se puede unir al proyecto

Etiquetas para Versiones

Para qué sirven: Marcar versiones importantes

v1.0.0 - Primera versión que usan los clientes

v1.1.0 - Agregamos reportes nuevos

v1.1.1 - Arreglamos un error pequeño

Proceso Automático de Pruebas

1. **Trigger Automático**

- Cada vez que haces git push o creas un Pull Request
- GitHub detecta el cambio automáticamente
- Se activa el sistema de pruebas sin necesidad de hacer nada manual

2**. Configuración del Flujo**

- Tenemos un archivo .github/workflows/pruebas.yml
- Este archivo contiene las instrucciones de qué pruebas ejecutar
- Define: qué pruebas correr, en qué orden, y con qué herramientas

3. **Ejecución de Pruebas**

- GitHub crea un ambiente limpio (como una computadora virtual)
- Descarga tu código recién subido
- Ejecuta secuencialmente:
    - ✞ Pruebas unitarias
    - ✞ Pruebas de integración
    - ✞ Pruebas de APIs

- ✞ Pruebas de interfaz

4. Resultados Inmediatos

- Si todas las pruebas pasan: Estado VERDE
- Si alguna prueba falla: Estado ROJO
- Los resultados se muestran directamente en GitHub
- Recibes notificaciones del resultado

5. Bloqueo de Merge

- Si las pruebas fallan, NO puedes unir tu código al proyecto principal
- Debes arreglar los errores y volver a subir
- Solo con todas las pruebas pasando puedes hacer merge

Estrategia de despliegue

1. Pipeline CI/CD

Pasos del Pipeline:

Checkout

- Descargar el código fuente desde GitHub
- Obtener la versión más reciente del repositorio

Instalación

- Instalar Java, Node.js y todas las dependencias
- Configurar el ambiente de desarrollo

Pruebas

- Ejecutar pruebas unitarias automáticas
- Ejecutar pruebas de integración
- Ejecutar pruebas de API y interfaz
- Verificar calidad de código

Empaquetado

- Compilar y empaquetar la aplicación

- Crear contenedores Docker listos para usar
- Generar archivos ejecutables

Despliegue

- Desplegar automáticamente en ambiente de staging
- Desplegar en producción después de aprobación manual

Entornos:

Staging

- URL: https://staging-technova.herokuapp.com
- Propósito: Pruebas finales antes de producción

Producción

- URL: https://technova-solutions.com
- Propósito: Ambiente real para usuarios finales

Condiciones para Desplegar:

- Todas las pruebas automáticas deben pasar
- Aprobación manual del Product Owner
- Pruebas manuales exitosas en staging
- Backup completo de base de datos de producción

2. Herramientas Complementarias

Docker

- Para empaquetar la aplicación en contenedores
- Garantiza que funcione igual en todos los ambientes

Terraform

- Para crear y gestionar infraestructura como código
- Automatiza la creación de servidores y redes

Prometheus y Grafana

- Para monitoreo después del despliegue
- Miden rendimiento y detectan problemas en tiempo real

6. Repositorio configurado para recibir el código fuente

El repositorio en GitHub incluye:

- Estructura de carpetas por módulo
- Archivo README con instrucciones
- Carpeta de pruebas automatizadas
- Archivo .github/workflows/main.yml para CI/CD
- Configuración de ramas protegidas

7. Rubrica

Criterio	Descripción	Puntos Máximos
1. Análisis del caso	El estudiante comprende el contexto de TechNova Solutions, identifica los desafíos del equipo distribuido y plantea objetivos claros.	10
2. Selección de herramientas	Se seleccionan herramientas adecuadas para cada fase del ciclo DevOps.	10
3. Parámetros de configuración	Se documenta cómo se configuran las herramientas (Slack, Jira, GitHub, CI/CD, etc.) con ejemplos específicos.	10
4. Plan de pruebas	Se presenta un plan de pruebas completo, vinculado a los entregables por sprint.	10
5. Casos de prueba	Se incluyen al menos 1 prueba manual y 3 automatizadas, bien redactadas y justificadas.	10
6. Flujo de control de versiones	Se describe el uso de Git con ramas, pull requests, etiquetas y CI.	10
7. Estrategia de despliegue	Se explica el pipeline CI/CD, condiciones de despliegue, entornos y herramientas complementarias.	10
8. Configuración del repositorio	Se presenta una estructura clara del repositorio, con carpetas, archivos clave y documentación.	10
9. Justificación técnica	Se justifican las decisiones tomadas en cada sección con base en el contexto del caso.	10

10. Presentación del documento	El documento está bien estructurado, redactado profesionalmente y cumple con los requisitos formales.	10