Reporte solución Prueba Técnica Coordinadora

Pedro Rafael Tinoco García

Parte 1: Consultas SQL sobre Modelos de Datos (Departamentos y Empleados)

Se tienen dos tablas: Departamentos y Empleados

Con los modelos de datos de Departamentos y Empleados previamente descritos, realiza las siguientes consultas SQL.

1. Determine el valor total que recibirán los empleados del departamento 3000 en su próximo pago, que incluye un reconocimiento adicional único de $500.000. El pago regular se compone de salario y comisión. La información debe estar ordenada alfabéticamente por empleado.

```
Query: SELECT nomEmp, salEmp + comisionE as "Pago regular", 500000 as "Reconocimiento
adicional unico",
       salEmp + comisionE + 500000 as "Valor Total"
          FROM Empleados
          WHERE codDepto = '3000'
          ORDER BY nomEmp;
```

---

2. Identifique los departamentos que tienen más de 3 empleados y el número de empleados en cada uno. Ordene los resultados de mayor a menor cantidad de empleados.

```
Query: SELECT d.nombre, COUNT(e.id) AS numero_empleados FROM departamentos d
       JOIN empleados e ON d.id = e.departamento_id
          GROUP BY d.nombre HAVING COUNT(e.id) > 3
          ORDER BY numero_empleados DESC;
```

---

3. Verifique si existen empleados que no estén asignados a algún departamento existente.

```
Query: SELECT nomEmp
       FROM empleados
          WHERE codDepto IS NULL;
```

---

4. Encuentre el departamento con la nómina total más alta, considerando salario y comisión.

```
Query: SELECT codDepto, salEmp + comisionE as "Nomina total"
       FROM departamentos
          ORDER BY codDepto DESC;
```

---

5. Identifique los tres directores con más personas a cargo.

```
Query: SELECT *
          FROM empleados
          WHERE cargoE = 'Director'
          LIMIT 3;
```

---

6. Determine el salario promedio general de la empresa, así como el departamento con el salario promedio más alto y el valor correspondiente.

```
Query: SELECT codDepto, avg(salEmp)
       FROM departamentos, empleados;
```

****
****

Parte 2: Pruebas Funcionales

Escenarios de Pruebas

Diseñe cinco escenarios de prueba manual que incluyan la creación, validación y revisión de errores en la solicitud de recogida, con especial énfasis en campos críticos y en la detección de duplicados.

Set de pruebas de creación, validación y revisión de errores Solicitud Recogida:

Plataforma / Aplicativo:	**Envíos**
Módulo:	Solicitud de Recogida
Descripción Breve:	Yo como un usuario requiero un módulo de solicitud de recogidas para facilitar el proceso de cargue y validación de una recogida de un cliente y tener la información precisa para llevar su paquete al destino final.
Tipo de Prueba:	Funcional – Manual – Automática
Ambiente de ejecución:	apiv2-test.coordinadora.com/recogidas/cm-solicitud-recogidas-ms/solicitud-recogida
Estado:	En ejecución

Código Caso prueba	Objetivo	Acción del paso	Resultado esperado	Numero de Iteraciones
CP001_Validar Campos Vacíos	Validar que los campos "direccion", "fechaRecogida", "nombreEntrega", "apellidosEntrega", "celularEntrega", "emailUsuario", "descripcionTipoVia", "aplicativo" no deben estar vacíos	1. abrir Postman 2. Lanzar servicio con los campos del payload mínimo vital con los campos "direccion", "fechaRecogida", "nombreEntrega", "apellidosEntrega", "celularEntrega", "emailUsuario",	1. El código de respuesta retornado debe ser 400 (BAD REQUEST) 2. Debe incluirse el mensaje de error "Los valores de entrada no son correctos." 3. Deben estar presente los campos en la respuesta JSON: "isError": true, "message": "Los valores de entrada no son correctos.", "code": "BAD_MESSAGE",	5 (Uno por cada campo del payload)

		"descripcionTipoVia", "aplicativo" estando sus campos vacíos 3. Clic en "Send" 4. Verificar mensaje	**"cause": "\"direccion\" is not allowed to be empty",** "timestap": "2024-11-20T18:02:47.806Z", "statusCode": 400, "id": "873ef84418d51a2984612aae6edeb7ea4bbdcf8e"	

CP002_Validar Recogida Exitosa	Validar el mensaje respuesta de cuando un payload con los datos correctos cree una recogida exitosa	1. Abrir Postman
		2. Lanzar servicio con los datos correctos del payload
		3. Clic en "Send"
		4 verificar mensaje de respuesta
			1. El código de respuesta retornado debe ser 200 (OK)
			2. Se deben incluir los campos en la respuesta como sigue:
			{
			"isError": false,
			"data": {
			"id_recogida": {
			"idRecogida": "26785582",
			"error": false,
			"message": "Solicitud recogida programada exitosamente"
			}
			},
			"timestamp": "2024-11-20T18:16:41.348Z",
			"id": "d58e14a834cf4028a1bdd41f31894f8ed4ca31a4"
			}
				1 (para creación única)

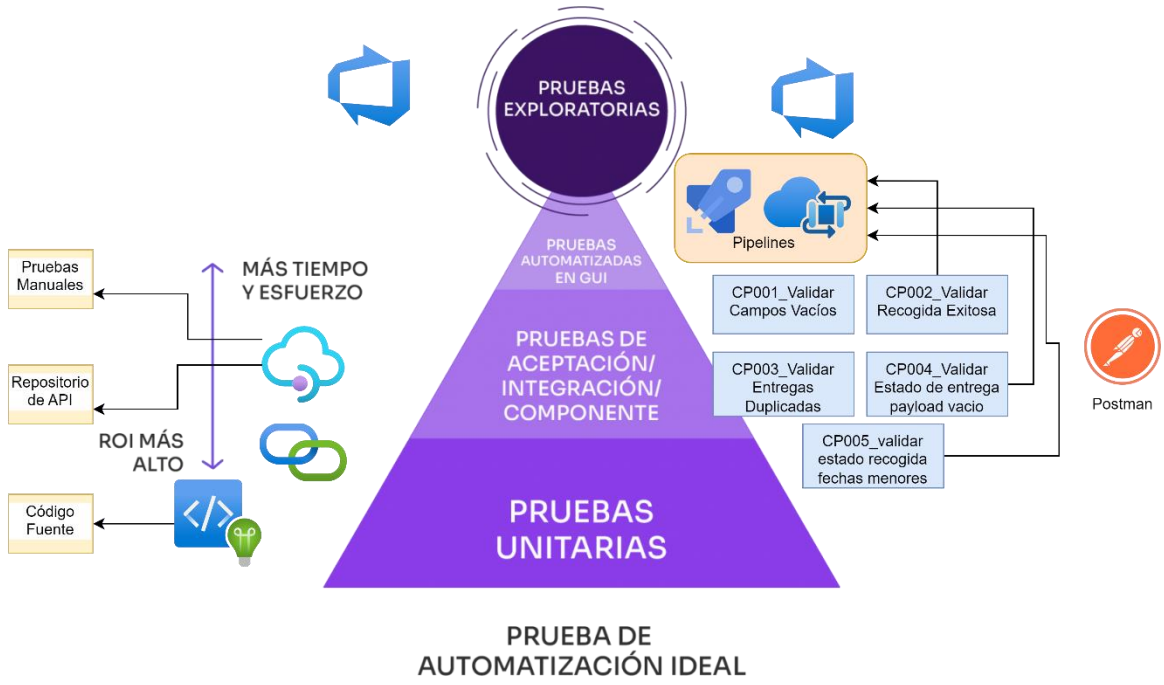
CP003_Validar Entregas Duplicadas	Determinar que cuando se vuelva a lanzar una petición con la misma fecha de una ya creada, exista la validación de que existe una entrega con la misma dirección y fecha de recogida.	1. Abrir Postman
		2. Lanzar servicio con los datos correctos del payload determinados en el caso anterior
		3. Clic en "Send"
		4 verificar mensaje de respuesta
			1. El código de respuesta retornado debe ser 200 (OK)
			2. Se deben incluir los campos en la respuesta en el siguiente formato:
			{
			"isError": true,
			"data": {
			"message": "Error, Ya existe una recogida programada para hoy, id: 26785582",
			"idRecogidaAnterior": "26785582",
			"recogida_anterior": true
			},
			"timestamp": "2024-11-20T18:20:30.419Z",
			"id": "2fd613276222ccc83ed9f325b4d133e2e11d7023"
			}
				1 (Para creación inicial)
				1 (Para comprobar existencia de duplicado)
				Total: 2 Iteraciones

CP004_Validar Estado de entrega payload vacio	Determinar que al lanzar una petición al servicio sin un payload válido, genere un mensaje de error	1. Abrir Postman
		2. Lanzar servicio sin payload
		3. Clic en "send"
		4. Verificar mensaje de respuesta
			1. El código de respuesta retornado debe ser 500 (INTERNAL SERVER ERROR)
			2. Se deben mostrar los campos de respuesta en el siguiente formato:
			{
			"isError": true,
			"message": "body should be object",
			"code": "UNKNOWN_ERROR",
			"cause": "Default translator error",
			"timestap": "2024-11-20T18:23:55.185Z",
			"statusCode": 500,
			"id": "9040c162b078c41405c6209413eef102f3430232"
			}
				1

CP005_validar estado recogida fechas menores	Determinar que al lanzar una petición con una fecha de recogida menor a la actual,	1. Abrir Postman 2. Lanzar servicio con una fecha menor a la	1. El código de respuesta retornado debe ser 200 (OK) 2. Se debe mostrar la respuesta como la del siguiente formato, mostrando el mensaje de fecha:	1

	genere un mensaje de validación	fecha actual (ej 2023-11-28)
		3. Clic en “Send”
		4. Verificar mensaje de respuesta
			{
			"isError": true,
			"data": {
			"message": "El campo fecha: 28-11-2023, no debe ser menor a la fecha actual.",
			"idRecogidaAnterior": "28-11-2023, no debe ser menor a la fecha actual.",
			"recogida_anterior": true
			},
			"timestamp": "2024-11-20T18:31:36.897Z",
			"id": "9ece956cc54d597175e913dc4152a7d53d7ebb1a"
			}	

1. Defina una estrategia de automatización para estos escenarios, organizando las validaciones en módulos: validaciones de formato, de campos obligatorios y flujo con excepciones.



Parte 5 preguntas teóricas

1. Explique las diferencias entre pruebas unitarias, de integración y de extremo a extremo (E2E).

R/ Las pruebas Unitarias corresponden a dichas pruebas que se realizan sobre el código fuente del sistema y son ejecutadas por los desarrolladores,

Las pruebas de integración corresponden a dichas pruebas cuyo proceso son realizadas y ejecutadas entre los diferentes componentes del sistema y la
relación existente entre cada uno de ellos.
Con respecto a las pruebas de extremo a extremo, tambien llamadas "End 2 End", se caracterizan por la ejecución de pruebas las cuales pretenden validar un flujo de
trabajo desde su inicio hasta su final, evaluando los resultados del flujo como tal a través de los diferentes módulos que lo componen.
---

2. ¿Qué es un framework de automatización de pruebas? Mencione ejemplos que haya utilizado.

R/ Un framework de automatización de pruebas es una colección de herramientas y directrices que resultan útiles a la hora de diseñar, crear y ejecutar casos de prueba.
Estas directrices incluyen normas de codificación, prácticas y procesos para el manejo de datos de prueba y de repositorios, entre otros, las cuales son de vital importancia
para las pruebas automatizadas.
Un ejemplo de framework de automatización es Selenium. Es un framework de automatización de pruebas de código abierto para las pruebas sobre aplicaciones web.
Este framework es un conjunto de herramientas de pruebas basadas en el framework de Java, el cual puede ser integrado a un IDE que soporte este lenguaje o cualquier otro como Python para la escritura de sus casos de prueba.
---

3. Defina el concepto de "Pirámide de Pruebas" y su importancia.

R/ la piramide de pruebas establece un nivel de rigurosidad al momento de ejecutar las pruebas teniendo en cuenta la necesidad, por ejemplo deben ser más solidas las pruebas unitarias,
ya que de ellas depende el éxito del sistema, por lo cual se localiza en la base de la piramide y a medida que se vaya ascendiendo las pruebas tenderán a ser menos rigurosas. En cada nivel
se presentan las herramientas adecuadas para su ejecución.
---

4. ¿Qué es la sincronización en pruebas automatizadas y por qué es esencial?

R/ la sincronización de eventos y la gestión del tiempo de espera son retos comunes que pueden provocar fallos en los scripts de prueba. Estos problemas ocurren cuando se intentan realizar operaciones con elementos web que no están presentes en la página o no están en un estado adecuado. Para abordar estos retos y mejorar la estabilidad y la precisión de los scripts de prueba, es importante: Sincronizar los eventos, Gestionar el tiempo de espera.
---

5. ¿Cómo integraría pruebas automatizadas en un proceso de integración continua (CI)?

Un proceso de integración continua típico incluye los siguientes pasos:

Revise el código fuente de la rama principal.
Ejecute pruebas unitarias automatizadas.
Ejecute la secuencia de compilación y decida si desea aceptar o rechazar la compilación.
Implemente en el entorno de prueba/control de calidad.
Implemente en el entorno de producción/en vivo.

Una alternativa para estas acciones es la creación de pipelines en entornos de DevOps. Un ejemplo claro es lo utilizado en Azure DevOps.

---

6. Discuta las diferencias entre pruebas de carga y de estrés, proporcionando ejemplos.

Las pruebas de carga y las pruebas de estrés son dos tipos de pruebas de rendimiento que se utilizan para evaluar el funcionamiento de las aplicaciones de software bajo diferentes condiciones:

Pruebas de carga

Simulan el tráfico normal y determinan si la aplicación puede manejar un volumen de tráfico esperado. Ejemplo, la evaluación del rendimiento de un formulario de inicio de sesión de un modulo de contabilidad en una hora normal

Pruebas de estrés

Simulan el tráfico extremo y determinan los límites superiores de la capacidad del sistema. Un ejemplo puede ser la misma evaluación hacia el módulo de inventario, pero en horas pico críticas para la empresa, es decir al momento de cargar, descargar articulos nuevos y haya carga demandante de procesos al mismo tiempo en horas pico.

---

7. ¿Cómo gestionaría la automatización de pruebas para una aplicación web con componentes dinámicos?

Se deben tener en cuenta los localizadores de los objetos, manejar los tiempos de espera de forma adecuada, ya que habrá componentes que no se despliegan en cierto tiempo.

Lo más importante es tener en cuenta el identificador del componente estático sobre el cual se puede ubicar, para que el autómata pueda retener el control sobre el cuándo sea llamado.

---

8. Explique la importancia de gestionar dependencias en la automatización de pruebas y cómo hacerlo en frameworks como Maven o Gradle.

La importancia de gestionar dependencias radica en que no se encuentran descargadas en el sistema en el momento de utilizarse, sino que son accedidas desde un repositorio externo, lo cual redunda en generar portabilidad al momento de recrear el proyecto en uno o mas dispositivos. al llamar a dichas dependencias se accede a los jars necesarios para que el proyecto pueda trabajarse. Un ejemplo es creando un archivo llamado "Build.gradle". Al crear este archivo, se colocan las dependencias necesarias para acceder a las librerias adecuadas.

Ejemplo de build.gradle:

```
plugins {
    id 'java'
    id "net.serenity-bdd.serenity-gradle-plugin" version "3.5.0"
}
repositories {
    mavenLocal()
    mavenCentral()
}
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa:2.7.3'
    implementation 'org.springdoc:springdoc-openapi-ui:1.6.4'
    implementation 'org.springframework.boot:spring-boot-starter-web:2.7.0'
    runtimeOnly 'com.h2database:h2:2.1.214'
    testImplementation 'org.springframework.boot:spring-boot-starter-test:2.7.0'
    testImplementation 'net.serenity-bdd:serenity-cucumber6:2.6.0'
    testImplementation 'net.serenity-bdd:serenity-screenplay:2.6.0'
    testImplementation 'net.serenity-bdd:serenity-screenplay-rest:2.6.0'
    testImplementation 'net.serenity-bdd:serenity-rest-assured:2.6.0'
```

```
    testImplementation 'junit:junit:4.13.1'
    testImplementation 'io.rest-assured:rest-assured:4.3.3'
}
test {
    testLogging.showStandardStreams = true
    systemProperties System.getProperties()
}
test.finalizedBy(aggregate)
serenity {
    reports = ["single-page-html"]
}

gradle.startParameter.continueOnFailure = true
```
