

Prueba técnica Qa automation
Juan David Coronado Dussan
Automatizador Senior

- COMPONENTE PRÁCTICO (Funcional)

Creación de la tabla Students

CREATE TABLE Students (

```
    student_id INT,  
    name VARCHAR(100),  
    birthdate DATE,  
    gender CHAR(1),  
    email VARCHAR(100),  
    phone VARCHAR(20)
```

);

```
INSERT INTO Students (student_id, name, birthdate, gender, email, phone) VALUES  
(9000185, 'PEDRO GOMEZ', '1995-10-30', 'M', 'PEDRO.GOMEZ@GMAIL.COM', '1185432478'),  
(9000186, 'JUAN HERNANDEZ', '1995-01-25', 'M', 'Juan.Hernandez@gmail.com', '1185432479'),  
(9000187, 'MARIA GONZALEZ', '1998-08-01', 'F', 'Maria.Gonzalez@hotmail.com', '1185432480'),  
(9000188, 'PABLO SANCHEZ', '1997-03-09', 'M', 'pablo.sanchez@gmail.com', '1185432481'),  
(9000189, 'ELENA CASTRO', '2000-06-15', 'F', 'Elena.Castro@gmail.com', '1185432482'),  
(9000190, 'MATIAS PEREZ', '1998-09-06', 'M', 'Matias.Perez@gmail.com', '1185432483'),  
(9000191, 'SEBASTIAN VARGAS', '2002-08-25', 'M', 'Sebastian.Vargas@outlook.com',  
'1185432484'),  
(9000192, 'PEDRO HERNANDEZ', '1998-05-19', 'M', 'Pedro.Hernandez92@gmail.com',  
'1185432485'),  
(9000193, 'CAROLINA LUNA', '1997-07-21', 'F', 'Carolina.Luna@yahoo.com', '1185432486'),  
(9000194, 'MONICA FIGUEROA', '1980-06-18', 'F', 'Monica.Figueroa@hotmail.com', '1185432487');  
SELECT * FROM Students;
```

The screenshot shows the Paiza.io online MySQL IDE interface. The top bar includes the Paiza.io logo, navigation links (Nuevo código, Códigos recientes, Web Dev(PaizaCloud)), and user options (Spanish, Registrarse, Iniciar sesión). The main editor area displays the SQL code from the previous block, with line numbers 1 through 23. A status bar at the bottom of the editor indicates 'Success' and '0, Post'. Below the editor, a terminal window shows the output of the SQL execution, displaying the table structure and the inserted data rows. The output is as follows:

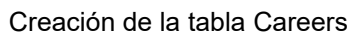
student_id	name	birthdate	gender	email	phone
9000185	PEDRO GOMEZ	1995-10-30	M	PEDRO.GOMEZ@GMAIL.COM	1185432478
9000186	JUAN HERNANDEZ	1995-01-25	M	Juan.Hernandez@gmail.com	1185432479
9000187	MARIA GONZALEZ	1998-08-01	F	Maria.Gonzalez@hotmail.com	1185432480
9000188	PABLO SANCHEZ	1997-03-09	M	pablo.sanchez@gmail.com	1185432481
9000189	ELENA CASTRO	2000-06-15	F	Elena.Castro@gmail.com	1185432482
9000190	MATIAS PEREZ	1998-09-06	M	Matias.Perez@gmail.com	1185432483
9000191	SEBASTIAN VARGAS	2002-08-25	M	Sebastian.Vargas@outlook.com	1185432484
9000192	PEDRO HERNANDEZ	1998-05-19	M	Pedro.Hernandez92@gmail.com	1185432485
9000193	CAROLINA LUNA	1997-07-21	F	Carolina.Luna@yahoo.com	1185432486
9000194	MONICA FIGUEROA	1980-06-18	F	Monica.Figueroa@hotmail.com	1185432487

creación de la tabla Career_Student

CREATE TABLE Career_Student (

```
    career_id INT,  
    student_id INT,  
    start_date DATE,
```

```
INSERT INTO Career_Student (career_id, student_id, start_date, estimated_end_date, end_date,
extended, calification) VALUES
(10025, 9000185, '2009-01-01', '2010-01-01', '2013-01-01', NULL, 85),
(10023, 9000186, '2018-01-01', '2021-01-01', NULL, NULL, 92),
(10026, 9000187, '2014-01-01', '2016-01-01', NULL, NULL, 58),
(10027, 9000188, '2017-01-01', '2020-01-01', NULL, NULL, 67),
(10028, 9000189, '2016-01-01', '2018-01-01', '2018-01-01', NULL, 91),
(10025, 9000190, '2019-01-01', '2022-01-01', NULL, NULL, 89),
(10026, 9000191, '2020-01-01', '2023-01-01', NULL, NULL, 75),
(10029, 9000192, '2021-01-01', '2024-01-01', NULL, NULL, 74),
(10025, 9000193, '2022-01-01', '2025-01-01', NULL, NULL, 91),
(10029, 9000185, '2008-01-01', '2013-01-01', '2013-01-01', NULL, 68);
SELECT * FROM Career_Student;
```



```
INSERT INTO Careers (career_id, name, description, subjects, cost) VALUES
(10025, 'INGENIERIA', 'INGENIERIA', 90, 25000.00),
(10026, 'ADMINISTRACION', 'ADMINISTRACION', 72, 18000.00),
```

```
(10027, 'CONTABILIDAD', 'CONTABILIDAD', 60, 15000.00),
(10028, 'DERECHO', 'DERECHO', 85, 23000.00),
(10029, 'INFORMATICA', 'INFORMATICA', 78, 19500.00),
(10030, 'COMUNICACION', 'COMUNICACION', 65, 17800.00);
SELECT * FROM Careers;
```

```

10025 INGENIERIA INGENIERIA 90 25000.00
10026 ADMINISTRACION ADMINISTRACION 72 18000.00
10027 CONTABILIDAD CONTABILIDAD 60 15000.00
10028 DERECHO DERECHO 85 23000.00
10029 INFORMATICA INFORMATICA 78 19500.00
10030 COMUNICACION COMUNICACION 65 17800.00

```

1. Redactar las siguientes consultas en SQL
 - a) Seleccione el total de estudiantes Masculinos Registrados

```
SELECT COUNT(*) AS total_masculinos FROM Students WHERE gender = 'M';
```

```
total_masculinos
6
```

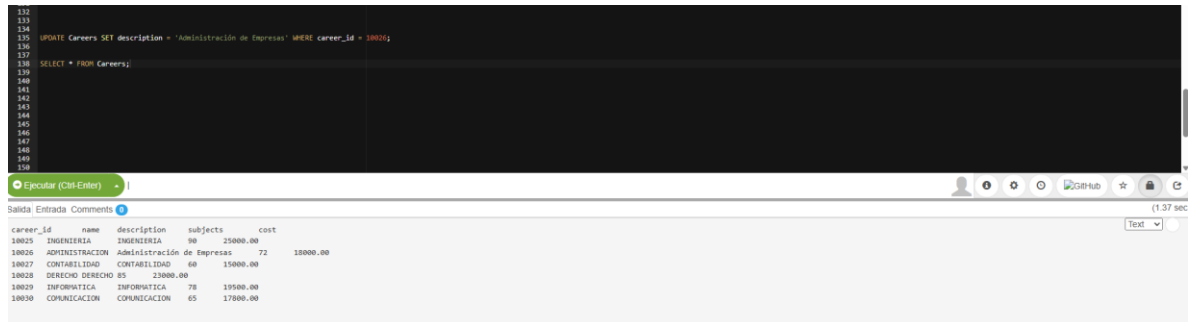
- b) Elimine los estudiantes nacidos antes 2000

```
DELETE FROM Students WHERE birthdate < '2000-01-01';
```

```
DELETE FROM Students WHERE birthdate < '2000-01-01';
```

- c) Actualice la descripción de la carrera con id '10026' de "Administración" a "Administración de Empresas"

```
UPDATE Careers SET description = 'Administración de Empresas' WHERE career_id = 10026;  
SELECT * FROM Careers;
```

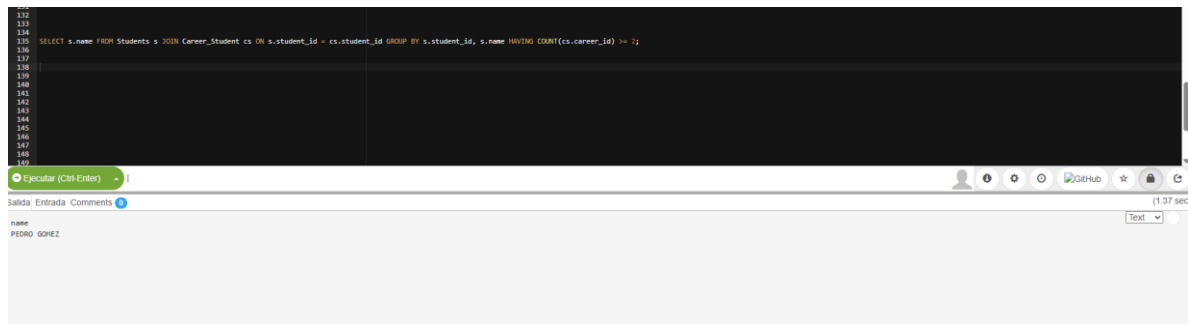


The screenshot shows a SQL IDE with a dark theme. The top pane contains the SQL code: `UPDATE Careers SET description = 'Administración de Empresas' WHERE career_id = 10026;` followed by `SELECT * FROM Careers;`. The bottom pane shows the output of the SELECT statement as a table with 5 columns: `career_id`, `name`, `description`, `subjects`, and `cost`. The table contains 6 rows of data.

career_id	name	description	subjects	cost
10025	INGENIERIA	INGENIERIA	90	210000.00
10026	ADMINISTRACION	Administración de Empresas	72	180000.00
10027	CONTABILIDAD	CONTABILIDAD	60	150000.00
10028	DERECHO DERECHO	DERECHO	85	230000.00
10029	INFORMATICA	INFORMATICA	78	195000.00
10030	COMUNICACION	COMUNICACION	65	170000.00

- d) Muestre los nombres de los estudiantes que estén cursando dos o más materias

```
SELECT s.name FROM Students s JOIN Career_Student cs ON s.student_id = cs.student_id  
GROUP BY s.student_id, s.name HAVING COUNT(cs.career_id) >= 2;
```

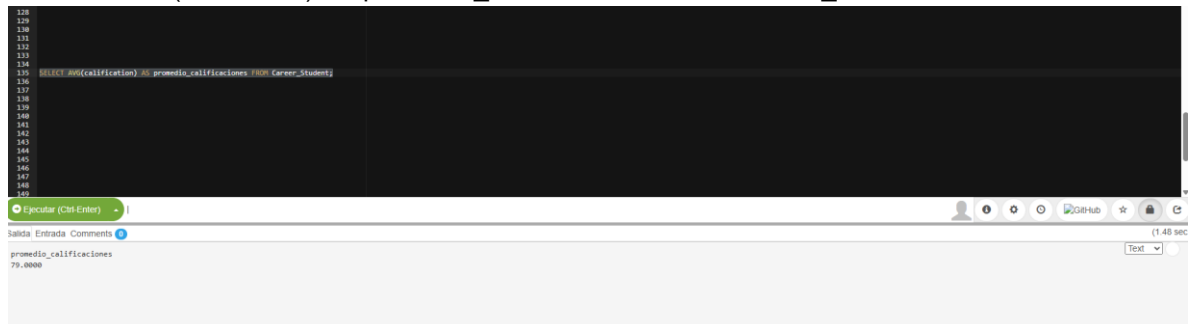


The screenshot shows a SQL IDE with a dark theme. The top pane contains the SQL code: `SELECT s.name FROM Students s JOIN Career_Student cs ON s.student_id = cs.student_id GROUP BY s.student_id, s.name HAVING COUNT(cs.career_id) >= 2;`. The bottom pane shows the output of the query, which is a single row with the name `PEDRO GOMEZ`.

name
PEDRO GOMEZ

- e) Calificación promedio de todos los estudiantes que estén cursando carreras

```
SELECT AVG(calification) AS promedio_calificaciones FROM Career_Student;
```



The screenshot shows a SQL IDE with a dark theme. The top pane contains the SQL code: `SELECT AVG(calification) AS promedio_calificaciones FROM Career_Student;`. The bottom pane shows the output of the query, which is a single row with the value `79.0000` under the alias `promedio_calificaciones`.

promedio_calificaciones
79.0000

- f) Muestra el nombre, carrera id, correos de los estudiantes que tienen una calificación mayor 70

```
SELECT s.name, cs.career_id, s.email FROM Students s JOIN Career_Student cs ON  
s.student_id = cs.student_id WHERE cs.calification > 70;
```

```

120
121
122
123
124
125
126
127
128
129
130
131
132
133
134 SELECT s.name, cs.career_id, s.email FROM Students s JOIN Career_Student cs ON s.student_id = cs.student_id HAVING COUNT(cs.career_id) >= 2
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149

```

name	career_id	email
PEDRO GOMEZ	10025	Pedro.Gomez@gmail.com
JUAN HERNANDEZ	10023	Juan.Hernandez@gmail.com
ELENA CASTRO	10028	Elena.Castro@gmail.com
MATIAS PEREZ	10025	Matias.Perez@gmail.com
SEBASTIAN VARGAS	10026	Sebastian.Vargas@outlook.com
PEDRO HERNANDEZ	10029	Pedro.Hernandez92@gmail.com
CAROLINA LUNA	10025	Carolina.Luna@yahoo.com

g) Cuántos estudiantes que estén cursando 2 o más carreras
 SELECT COUNT(*) AS estudiantes_multicarrera FROM (SELECT student_id FROM Career_Student GROUP BY student_id HAVING COUNT(career_id) >= 2) AS subquery;

```

127
128
129
130
131
132
133
134 SELECT COUNT(*) AS estudiantes_multicarrera FROM ( SELECT student_id FROM Career_Student GROUP BY student_id HAVING COUNT(career_id) >= 2 ) AS subquery
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149

```

estudiantes_multicarrera
1

2. De la siguiente imagen, reporte al menos un bug que incluya todos los datos que considere relevantes para que un desarrollador pueda replicarlo y solucionarlo

ID del Bug: BUG-PAGO-001

Título: Duplicación del método de pago "RappiCuenta" en la pantalla de selección

Fecha: 2025-10-06

Reportado por: QA Tester

Descripción:

En la pantalla de selección de métodos de pago, el método "RappiCuenta" aparece listado dos veces: una como "RappiCuenta o débito" y otra como "RappiCuenta", ambos con la misma descripción: "¡Pago fácil e inmediato!". Esta duplicación puede generar confusión en el usuario respecto a cuál opción debe seleccionar y sugiere un problema de diseño o lógica de renderizado.

Pasos para reproducir:

1. Abrir la app de Rappi.
2. Iniciar un pago.
3. Proceder a la pantalla de selección de método de pago.
4. Observar las opciones listadas.

Resultado esperado:

Cada método de pago debería aparecer una sola vez, evitando duplicados innecesarios o confusos.

Resultado actual:

Se presentan dos opciones visualmente similares:

- "RappiCuenta o débito"
- "RappiCuenta"

Ambas con descripciones idénticas y sin aclarar sus diferencias, si las hay.

Entorno:

- Dispositivo: Android / IOS
- Versión de la app: 1.7.3
- Pantalla: Método de pago

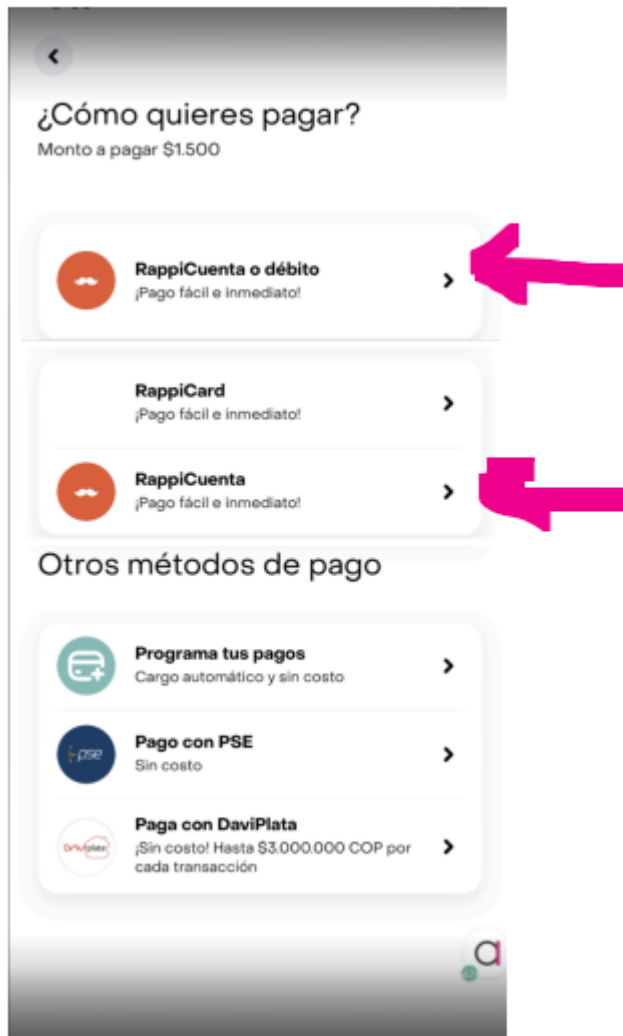
Severidad: Media

- Puede causar confusión en el usuario, pero no bloquea el flujo de pago.

Prioridad: Alta

- Es un tema de experiencia de usuario que puede impactar negativamente la percepción del sistema y generar errores en la selección del método de pago.

Imagen del bug:



- COMPONENTE TEÓRICO (Automation)

Basado en su experiencia, responda las siguientes preguntas:

1. ¿Cuáles son las principales diferencias entre automatizar en Android vs iOS con Appium?

ASPECTO	IOS	ANDROID
Requisitos del Sistema Operativo	Solo funciona en macOS (por requerimiento de Xcode).	Puede ejecutarse en Windows, macOS o Linux.
Instalación del entorno	Necesitas Xcode, Xcode Command Line Tools, y un perfil de desarrollador.	Necesitas Android SDK, Java JDK, emuladores, y configurar variables de entorno.
Dispositivos y Emuladores	Emuladores más limitados; requieren simuladores Xcode y perfiles de confianza.	Fácil de ejecutar en emuladores sin restricciones.
Interacción con el sistema	Permite más acciones (ej: manipular notificaciones, cambiar red)	Más limitado por seguridad del sistema
Firma de Aplicaciones	Puedes usar APKs sin firmar o firmarlos tú mismo.	Las apps deben estar firmadas con certificados válidos y, para pruebas en dispositivos reales, deben estar en el perfil del equipo de desarrollo.

2. ¿Cómo diseñaría un framework de automatización para mobile multiplataforma que sea mantenible y escalable?

Se propone el uso de Java 11+, Appium como motor de automatización y un framework de pruebas que puede ser:

- Serenity + Screenplay + Cucumber
- TestNG/Junit

Principios de diseño (SOLID)

El framework aplica los principios SOLID para garantizar mantenibilidad y extensibilidad:

- S – Single Responsibility: cada clase cumple una única responsabilidad (ejemplo: LoginPage solo modela la UI de login).
- O– Open/Closed: la arquitectura permite agregar nuevas plataformas o flujos sin modificar el código existente (ej. agregar un IOSCapabilitiesProvider).
- L – Liskov Substitution: las interfaces y abstracciones permiten intercambiar implementaciones (ej. Android/iOS) sin afectar a los consumidores.
- I – Interface Segregation: se definen interfaces pequeñas y específicas, evitando dependencias innecesarias.
- D – Dependency Inversion: las pruebas dependen de abstracciones, no de implementaciones concretas

Arquitectura propuesta (por capas)

1. Tests:
 - Define los escenarios de usuario (Gherkin o TestNG).
 - Agrupa por tags (@smoke, @android, etc.).
2. Acciones (Tasks/Helpers):
 - Encapsula pasos reutilizables (ej. login, scroll).
3. Pantallas (Pages/Screens):
 - Modela la UI con localizadores multiplataforma (@AndroidFindBy, @iOSXCUITFindBy).
4. Plataforma/Driver:
 - Configura el AppiumDriver por sistema operativo.

- Soporta ejecución local o en device farms.
- 5. Utils:
 - Funciones comunes: esperas, gestos, logs, datos.
- 6. Config:
 - Archivos .properties/.yaml para entorno, plataforma y usuario.
- 7. Reportes:
 - Serenity (detallado), Allure/Extent (ligeros) o ExtentReports (versátil, personalizable para TestNG/JUnit).

Escalabilidad y ejecución

- Ejecución en paralelo con TestNG o Serenity.
- Dependencias de Maven/Gradle.
- CI/CD con Azure Devops, Jenkins o GitLab CI para ejecución automática.
- Integración con device farms para ampliar cobertura de dispositivos y versiones.

Buenas prácticas

- Centralizar localizadores y configuración.
- Evitar Thread.sleep(), usar esperas explícitas.
- Mantener independencia entre pruebas.
- Definir nomenclatura estándar para clases y escenarios.
- Manejo de naming

3. ¿Qué criterios usa para seleccionar qué casos se deben automatizar y cuáles no?

Criterios para automatizar:

- Alta frecuencia de ejecución.
- Casos críticos para el negocio.
- Repetitivos y estables en el tiempo.
- Funcionalidades ya estabilizadas.
- Costos y beneficios.

Criterios para NO automatizar:

- Flujos que cambian con frecuencia.
- Procesos muy esporádicos o de bajo volumen.
- Casos que requieren interacción física (ej: huella dactilar).
- Pruebas exploratorias o de usabilidad.

4. ¿Qué estrategias aplicaría para integrar pruebas automáticas en un pipeline CI/CD?

- Cual herramienta usar para la creación del pipeline
- Diseña un pipeline con etapas claras (Build, ejecución de prueba, sonar, escaneo de vulnerabilidades)
- Ejecución por rama
- integración con granjas de dispositivos remotas
- Paralelización y optimización del tiempo de ejecución
- Resultados automáticos
- Notificaciones
- Integrar con los desarrollos

5. Mencione las ventajas y desventajas de trabajar con dispositivos físicos vs. dispositivos en remoto / granja de dispositivos cuando ejecuta una suite de automatización.

Factor	Dispositivos Físicos	Granja de Dispositivos (Remoto)
Costo	Alto (compra, mantenimiento)	Pago por uso, escalable
Acceso y control	Total (red, logs, debugging, gestos)	Limitado a lo que permite el proveedor
Paralelismo	Limitado por la cantidad de dispositivos físicos	Alta escalabilidad
Configuración inicial	Compleja pero controlada	Configuración ya proporcionada
Integración CI/CD	Requiere hardware conectado al servidor	Integración directa con servicios en la nube

6. ¿Qué patrones de diseño ha implementado en sus proyectos que considere eficaces para minimizar la duplicación de código y mejorar la legibilidad al desarrollar automatizaciones para dispositivos móviles?

Patrón	Beneficio Principal	Aplicación Común en Automatización Móvil
Page Object Model (POM)	Separación de lógica y elementos UI	Manejo de pantallas / vistas
Factory	Creación flexible de objetos	Inicialización de Pages o Drivers según contexto
Singleton	Instancia única para recursos compartidos	Gestión de AppiumDriver, configuración global
Command	Reutilización de acciones encapsuladas	Flujos comunes como login, logout, registro
Strategy	Comportamientos dinámicos según plataforma/contexto	Soporte multi-plataforma (iOS vs Android)
Serenity Screenplay + Cucumber	Alta legibilidad y reutilización de tareas y preguntas	Automatización BDD móvil con Appium (iOS/Android)

- COMPONENTE PRÁCTICO (Automation)

Contexto: Se requiere implementar un nuevo script para una prueba automatizada. El caso de prueba involucra tres (3) vistas: Home del App, Login y el Home de producto. El flujo que se desea automatizar valida que el usuario Ingrese al Home del App donde da un tap al widget del producto para posteriormente ser redireccionado al Login. Allí ingresará un código de 6 dígitos y finalmente navegará de forma automática al Home del producto donde se encuentra el detalle y acciones que el usuario puede realizar sobre el producto.

- a) En la vista de Login se implementa un método para validar que todos los labels definidos en el diseño se encuentren presentes. A continuación, verá el código implementado.

```

1  ✓ public boolean verifyLabelsOnScreen() {
2  ✓      try {
3          // Espera para que la pantalla cargue los elementos
4          Thread.sleep(15000);
5
6          boolean flagLabelIniciarSesion = false;
7          boolean flagLabelHola = false;
8          boolean flagLabelIngresarContrasena = false;
9          boolean flagLabelOlvideContrasena = false;
10
11         // Recorre todos los labels en la pantalla
12  ✓     for (MobileElement label : screenLabels) {
13         String text = label.getAttribute("text");
14
15  ✓         if (text.equals("Iniciar Sesión")) {
16             flagLabelIniciarSesion = true;
17         }
18  ✓         if (text.contains("¡Hola")) {
19             flagLabelHola = true;
20         }
21  ✓         if (text.equals("Ingresa tu contraseña de 6 dígitos")) {
22             flagLabelIngresarContrasena = true;
23         }
24  ✓         if (text.equals("Olvidé mi contraseña")) {
25             flagLabelOlvideContrasena = true;
26         }
27     }
28
29     // Verifica que todos los labels estén presentes
30     if (flagLabelIniciarSesion && flagLabelHola
31  ✓     && flagLabelIngresarContrasena && flagLabelOlvideContrasena) {
32         return true;
33     }
34
35  ✓ } catch (Exception e) {
36     return false;
37 }
38 return false;|
39 }
40

```

Instrucción: De acuerdo con su experiencia indique si está de acuerdo con la forma en la que fue implementado el método. En caso de no estar de acuerdo. Indíquenos qué modificaciones haría o reescriba el método para ver reflejados sus cambios y justifique su respuesta.

De acuerdo con mi experiencia, con la forma de implementación del código presentado tiene varias mejorar por realizar:

- EL Thread.sleep(15000): es una mala práctica que este siempre espere 15 segundos, ya que hace que la prueba se más lenta, adicional a esto aun que los elementos ya estén cargados y se puedan interactuar con estos antes de 15 segundos este obliga a esperar los 15 segundos
Para ello una buena práctica seria usar los wait + condición ya solo espera a que el o los elementos estén cargados e interactúa inmediatamente con ellos

- El uso de las banderas booleanas repetitiva: código repetitivo y agrega complejidad, adicional en temas de escalabilidad y mantenibilidad en el tiempo se vuelve insostenible. Mejor usar una lista de textos esperados y se comparada con los textos encontrados.
 - Uso de equals: el riesgo que genera esto es si un label tiene un espacio de más, mayúsculas/minúsculas distintas o un carácter especial, la prueba puede fallar. Se puede usar contains si el label varia o equalsIgnoreCase si no mitigando el riesgo.
 - Código repetitivo y uso excesivo del if generando complejidad, mantenibilidad y escalabilidad.
Para ello es mejor usar Set o Map con textos esperados y validarlos en un bucle.
 - Uso de return false redundante el último return false (fuera del catch) es innecesario porque el flujo ya está cubierto.
 - Falta mensajes de error si la validación falla, no sabes qué label faltó. Solo devuelve false, para ellos lo ideal es logs o lanzar excepción con el detalle.
 - Falta de reutilización lo cual hace que no sea posible usar en otras partes del código si es requerido.
 - Problema de escalabilidad ya que por ahora son 4 pero si son 20 o 50 hace que el método se vuelve enorme y difícil de leer.
- b) En la vista del Home del App, se necesita crear dos métodos: El primero estará encargado de verificar si el usuario se encuentra o no en el Home del App y el segundo realizará la acción de dar tap en el botón de Rappipay para ingresar al login. A continuación, verá la definición de los elementos que se tuvieron en cuenta.

```

1  @AndroidFindBy(id = "com.grability.rappi:id/menu_image")
2  private MobileElement buttonMenu;
3  @AndroidFindBy(id = "com.grability.rappi:id/support_image")
4  private MobileElement buttonSupport;
5  @AndroidFindBy(className = "android.widget.TextView")
6  private List<MobileElement> widgetElements;
7  @AndroidFindBy(id = "com.grability.rappi:id/imageView_action")
8  private MobileElement buttonRappiPay;
9  @AndroidFindBy(id = "layout_primary_action")
10 private MobileElement BTN_RAPPIPAY_CO;
11 @AndroidFindBy(xpath = "//*[contains(@text, 'Restaurante')]")
12 private MobileElement RESTAURANT;
13 @AndroidFindBy(xpath = "//*[contains(@text, 'Mercado/Ex')]")
14 private MobileElement MARKET;
15 @AndroidFindBy(xpath = "//*[contains(@text, 'Turbo')]")
16 private MobileElement TURBO;
17

```

Instrucción: De acuerdo con su experiencia indique si está de acuerdo con la forma en la que se definieron los elementos. En caso de no estar de acuerdo. Indíquenos qué modificaciones haría o reescriba el código para ver reflejados sus cambios y justifique su respuesta.

De acuerdo con mi experiencia, con la forma de implementación funciona, sin embargo, tiene varias mejor que se pueden realizar:

- Uso de className = "android.widget.TextView" para widgetElements: muy genérico. Capturará todos los TextView de la pantalla haciendo que guarde información no requerida.

- Xpath con `//*` hace que la automatización se vuelva lenta ya que va a buscar en todo el DOM del código es mejor especificar ejemplo: `//a` o `//h1`
- Los xpath con texto son frágiles si cambia el idioma de la aplicación (ES → EN)
- Nombre de las variables no tienen estructura, mezclas mayúsculas/abreviaturas para ello usar naming, ejemplo `btnMenu`.
- Los nombres de algunas variables no tienen contexto para qué va a ser su uso ejemplo:
`private List<MobileElement> widgetElements;`
- Localizadores muy específicos (`com.grability.rappi:id/menu_image`) hacen que el Page Object sea poco portable si la app cambia la estructura.
- Si estamos usando localizadores como el Id no es una buena práctica
`com.grability.rappi:id/menu_image` es mejor colocar `menu_image`