**FullStack Development Test**

***Software Engineering***

*Technology*

This test is intended to evaluate various aspects of front-end & back-end development, like code clarity, API / endpoints definition, design simplicity, test coverage or correct use of the HTTP protocol, in a real-life like scenario, UI development, data management, infrastructure as a code, and software development principles.

**In this test you must create an App for a Product paid with Wompi.** It is the onboarding where we obtain customer payment data to make the transaction of specific product, obtain customer delivery data, show the result of the payment process, and update stock data1.

**General business process**

1. You must create a UI or page to show the product and units available in the stock from your store (with description and price), that will be bought by the customer.

2. Show to the customer the payment type button ”*Pay with credit card*”. That opens a Modal requesting Credit Card info.

3. Customer will complete Credit Card data and must be validated, (MasterCard and VISA number card detection logos is a plus) and Delivery information. *(Remember, credit card data must be fake but must follows the structure of credit cards),*

4. Next show the summary payment that includes (Product amount, Base fee added always and Delivery Fee) with payment

button in a backdrop component2.

5. When user click payment button:

1. Create a transaction in PENDING in your backend API and obtain a transaction number.

2. Call Wompi API to complete the payment.

3. Once payment is completed or failed:

a. Update the transaction with the transaction result in your Backend

b. Assign the product that will be delivered to the customer

c. Update the stock of your product.

6. Finally show the result of the transaction made, and redirect to the product page with the stock updated.

**Your responsibilities**

✓ The overall design of the API and information architecture (DB schema, folder structure, etc.)

✓ Deciding what data should be requested/responded on each endpoint. (*Add Postman Collection or Swagger URL public in*

*README.md*)

✓ Deciding what validations each endpoint should perform, thinking of real-life situations that could arise.

✓ Safely handling sensitive data.

✓ The API you create must have stock, transactions, customers and deliveries. They must perform different types of requests.

✓ You must create a UI or page to show the product and units available in the stock from your store.

✓ Your app must be resilient, the App should recover the progress made by the client in case of refresh.

✓ Attention to detail is key.

✓ This App follows 5-step screen business process.

1. **Product page** → **2. Credit Card/Delivery info** → **3. Summary** → **4. Final status** → **5. Product page**

**1. Frontend:** You must implement a single page app (SPA) built in either **ReactJS** or **VueJS**. NO other frameworks are allowed.

1. The **App must be mobile oriented**, and you as a candidate create a Front that will manage multiple screen sizes.

Making your App responsive but focus on mobile design. (Minimum screen size reference: iPhone SE (2020) 1334 x

750 pixels). UI interaction will work properly and must fit into UI boundaries.

2. Use of either Redux or Vuex **is mandatory,** following Flux Architecture alignments as possible. Additionally, store the

payment transaction data securely in the *application state* or *localStorage.*

3. UX Design could be created as your own consideration.

4. Use CSS frameworks of your preference, but we foster you to use flexbox or grid APIs1.

**2. Backend:** You must implement an API Backend services built in either **JavaScript/TypeScript** or **Ruby**. With **Nest.js**, **Grappe**

or **Sinatra**. Ruby on rails and other frameworks are NOT allowed.

1. Business logic must NOT be handled within the routing / controller layer, could be on a separate layer. Try Hexagonal

Architecture and Ports & Adapters pattern.

2. Try to use Railway Oriented Programming (ROP), for your Use Cases.

3. You can use any type of database. We recommend using **PostgreSQL** or **DynamoDB**. (*Add Data model design in*

*README.md*)

4. You can use any type of **ORM** and serialization library.

5. The database must be seeded with dummy Products. There's NO need for an endpoint to create new products.

**3. Unit tests is mandatory either Frontend and Backend** for the competition of the tests, more than 80% of the coverage. (*Add*

*the results in README.md and create them with Jest*)

**4. Publish the app using any Cloud provider**, we recommend AWS2 (*with the AWS free tier, AWS Lambda, ECS, EKS, CloudFront,*

*S3, RDS or DynamoDB*).

**Considerations**

We recommend create branches and pull requests by feature.

You must push your code to Github as a public repository only!, and please do NOT use words like “Wompi” (company name), in your repository, and do not share your solution with other developers or candidates.

1. Wompi API. For starters, you must read: 1. https://docs.wompi.co/docs/colombia/inicio-rapido/ 2. https://docs.wompi.co/docs/colombia/ambientes-y-llaves/

2. Use the following information to use API Keys or to access to an account created for this test. *Important note: This user could be*

*used by any people doing the same test and your user session could be closed by different location accessing at the same time, try to integrate with API*

*and API Keys.*

• Is important always the tests must be on Sandbox mode.

• Sandbox mode is in the main menu → Developers

• Wompi User

• **URL: https://login.staging.wompi.dev/**

• **Users:**

• **procesoseleccionbackend@yopmail.com**

• **procesoseleccionbackend2@yopmail.com**

• **procesoseleccionbackend3@yopmail.com**

• **procesoseleccionbackend4@yopmail.com**

• **procesoseleccionbackend5@yopmail.com**

• **Password for whole users: JobProcessWompi987\***

• API URL

• **UAT\_URL=https://api.co.uat.wompi.dev/v1**

• **UAT\_SANDBOX\_URL=https://api-sandbox.co.uat.wompi.dev/v1**

• API Keys Merchant Sandbox (remember, this API keys are not for specific by user, those are for merchant)

• **pub\_stagtest\_g2u0HQd3ZMh05hsSgTS2lUV8t3s4mOt7**

• **prv\_stagtest\_5i0ZGIGiFcDQifYsXxvsny7Y37tKqFWg**

• **stagtest\_events\_2PDUmhMywUkvb1LvxYnayFbmofT7w39N**

• **stagtest\_integrity\_nAIBuqayW70XpUqJS4qf4STYiISd89Fp**

3. Make sure you use the Sandbox environment. That means, there's NO need for any real money transactions, in the context

of this test.

1. Completion of Frontend App and Backend API

2. Link of GitHub repository with README.md updated.

3. Link of the AWS App deployed working connected with your Backend API

**Evaluation rubrics**

**1. [5 points]** for README completed correctly.

**2. [5 points]** Images that renders fast and avoid UI/UX Out-boundaries.

**3. [20 points]** Full functionality of the onboarding process for Credit card payment checkout.

**4. [20 points]** for API working correctly.

**5. [30 points]** for more than 80% of coverage with Unit testing Backend and Frontend

**6. [20 points]** for App and API deployed through any Cloud provider.

\_\_

**1. [5 *Bonus points*]** use of OWASP alignments, use of HTTPS and Security headers.

**2. [5 *Bonus points*]** for Apps completely responsive and working through different browsers.

**3. [10 *Bonus points*]** CSS skills.

**4. [10 *Bonus points*]** Clean code.

**5. [10 *Bonus points*]** for structuring your app using Hexagonal Architecture with Ports & Adapters pattern.

**6. [10 *Bonus points*]** for following ROP.

**Considerations**

To complete the test, you need at least 100 points.

If you obtain the minimum points will continue with the interview step.

Prueba de Desarrollo FullStack  
Ingeniería de Software  
Tecnología

Esta prueba está diseñada para evaluar varios aspectos del desarrollo front-end y back-end, como la claridad del código, la definición de API / endpoints, la simplicidad del diseño, la cobertura de pruebas o el uso correcto del protocolo HTTP en un escenario similar a la vida real, desarrollo de UI, gestión de datos, infraestructura como código y principios de desarrollo de software.

En esta prueba debes crear una aplicación para un producto pagado con Wompi. Es el proceso de incorporación donde obtenemos los datos de pago del cliente para realizar la transacción de un producto específico, obtener los datos de entrega del cliente, mostrar el resultado del proceso de pago y actualizar los datos del inventario.

Proceso general del negocio

1. Debes crear una interfaz de usuario o página para mostrar el producto y las unidades disponibles en el stock de tu tienda (con descripción y precio), que el cliente comprará.
2. Mostrar al cliente el botón de tipo de pago "Pagar con tarjeta de crédito", que abrirá un modal solicitando la información de la tarjeta de crédito.
3. El cliente completará los datos de la tarjeta de crédito, que deben ser validados (la detección de logotipos de tarjetas MasterCard y VISA es un plus), junto con la información de entrega. (Recuerda, los datos de la tarjeta deben ser falsos pero seguir la estructura de las tarjetas de crédito).
4. A continuación, muestra el resumen del pago que incluye (Monto del producto, tarifa base siempre añadida y tarifa de entrega) con el botón de pago en un componente de fondo.
5. Cuando el usuario haga clic en el botón de pago:
   1. Crea una transacción en estado PENDIENTE en tu API backend y obtén un número de transacción.
   2. Llama a la API de Wompi para completar el pago.
   3. Una vez que el pago se complete o falle:  
      a. Actualiza la transacción con el resultado en tu Backend.  
      b. Asigna el producto que será entregado al cliente.  
      c. Actualiza el inventario de tu producto.
6. Finalmente, muestra el resultado de la transacción realizada y redirige a la página del producto con el stock actualizado.

Tus responsabilidades

✓ El diseño general de la API y la arquitectura de la información (esquema de la base de datos, estructura de carpetas, etc.).

✓ Decidir qué datos se deben solicitar/responder en cada endpoint. (Agregar colección de Postman o URL pública de Swagger en README.md).

✓ Decidir qué validaciones debe realizar cada endpoint, pensando en situaciones reales que puedan surgir.

✓ Manejar de forma segura los datos sensibles.

✓ La API que crees debe tener stock, transacciones, clientes y entregas. Deben realizar diferentes tipos de solicitudes.

✓ Debes crear una interfaz de usuario o página para mostrar el producto y las unidades disponibles en el stock de tu tienda.

✓ Tu aplicación debe ser resiliente, la aplicación debe recuperar el progreso realizado por el cliente en caso de que se actualice la página.

✓ La atención al detalle es clave.

✓ Esta aplicación sigue un proceso de negocio de 5 pasos en pantalla.

1. Página del producto → 2. Información de tarjeta de crédito/entrega → 3. Resumen → 4. Estado final → 5. Página del producto

**Frontend**: Debes implementar una aplicación de una sola página (SPA) construida en ReactJS o VueJS. NO se permiten otros frameworks.

1. La aplicación debe estar orientada a dispositivos móviles, y tú, como candidato, debes crear un Front que gestione múltiples tamaños de pantalla, haciendo que tu aplicación sea responsive, pero enfocada en el diseño móvil. (Tamaño mínimo de pantalla de referencia: iPhone SE (2020) 1334 x 750 píxeles). La interacción de la UI debe funcionar correctamente y ajustarse a los límites de la UI.
2. Es obligatorio el uso de Redux o Vuex, siguiendo las alineaciones de la Arquitectura Flux tanto como sea posible. Además, almacena los datos de la transacción de pago de forma segura en el estado de la aplicación o en localStorage.
3. El diseño UX puede ser creado a tu propia consideración.
4. Usa los frameworks de CSS de tu preferencia, pero te recomendamos usar APIs de flexbox o grid.

**Backend**: Debes implementar servicios de API backend en JavaScript/TypeScript o Ruby, utilizando Nest.js, Grappe o Sinatra. No se permiten Ruby on Rails ni otros frameworks.

1. La lógica del negocio NO debe manejarse en la capa de routing/controlador, debe estar en una capa separada. Intenta usar Arquitectura Hexagonal y el patrón de Puertos y Adaptadores.
2. Intenta usar Railway Oriented Programming (ROP) para tus casos de uso.
3. Puedes usar cualquier tipo de base de datos. Recomendamos usar PostgreSQL o DynamoDB. (Agrega el diseño del modelo de datos en README.md).
4. Puedes usar cualquier tipo de ORM y biblioteca de serialización.
5. La base de datos debe estar pre-poblada con productos ficticios. NO es necesario un endpoint para crear nuevos productos.

Las pruebas unitarias son obligatorias tanto en el Frontend como en el Backend para la competencia de las pruebas, con más del 80% de cobertura. (Agrega los resultados en README.md y créalos con Jest).

Publica la aplicación usando cualquier proveedor en la nube, recomendamos AWS (con el nivel gratuito de AWS, AWS Lambda, ECS, EKS, CloudFront, S3, RDS o DynamoDB).

**Consideraciones**  
Recomendamos crear ramas y pull requests por funcionalidad. Debes subir tu código a GitHub como un repositorio público, ¡únicamente!, y por favor NO uses palabras como "Wompi" (nombre de la empresa) en tu repositorio, y no compartas tu solución con otros desarrolladores o candidatos.

1. **Wompi API**. Para comenzar, debes leer:
   1. <https://docs.wompi.co/docs/colombia/inicio-rapido/>
   2. <https://docs.wompi.co/docs/colombia/ambientes-y-llaves/>
2. Utiliza la siguiente información para usar las API Keys o acceder a una cuenta creada para esta prueba. Nota importante: este usuario podría ser usado por otras personas que están haciendo la misma prueba y tu sesión de usuario podría cerrarse por diferentes ubicaciones accediendo al mismo tiempo. Intenta integrarte con la API y las claves de API.

• Es importante que las pruebas siempre se realicen en modo Sandbox.  
• Modo Sandbox está en el menú principal → Desarrolladores.  
• **Usuario de Wompi**  
• URL: <https://login.staging.wompi.dev/>  
• Usuarios:  
• procesoseleccionbackend@yopmail.com  
• procesoseleccionbackend2@yopmail.com  
• procesoseleccionbackend3@yopmail.com  
• procesoseleccionbackend4@yopmail.com  
• procesoseleccionbackend5@yopmail.com  
• Contraseña para todos los usuarios: JobProcessWompi987\*  
• URL API  
• UAT\_URL=<https://api.co.uat.wompi.dev/v1>  
• UAT\_SANDBOX\_URL=<https://api-sandbox.co.uat.wompi.dev/v1>  
• Claves API Merchant Sandbox (recuerda, estas claves API no son específicas para cada usuario, son para merchant):  
• pub\_stagtest\_g2u0HQd3ZMh05hsSgTS2lUV8t3s4mOt7  
• prv\_stagtest\_5i0ZGIGiFcDQifYsXxvsny7Y37tKqFWg  
• stagtest\_events\_2PDUmhMywUkvb1LvxYnayFbmofT7w39N  
• stagtest\_integrity\_nAIBuqayW70XpUqJS4qf4STYiISd89Fp

1. Asegúrate de usar el entorno de Sandbox. Eso significa que NO es necesario realizar transacciones con dinero real en el contexto de esta prueba.

**Entregables**

1. Finalización de la aplicación frontend y la API backend.
2. Enlace al repositorio de GitHub con README.md actualizado.
3. Enlace de la aplicación de AWS desplegada y funcionando, conectada con tu API backend.

**Criterios de evaluación**

1. [5 puntos] README completado correctamente.
2. [5 puntos] Imágenes que se renderizan rápido y evitan desbordes de UI/UX.
3. [20 puntos] Funcionalidad completa del proceso de incorporación para el checkout de pago con tarjeta de crédito.
4. [20 puntos] API funcionando correctamente.
5. [30 puntos] Más del 80% de cobertura con pruebas unitarias en Backend y Frontend.
6. [20 puntos] Aplicación y API desplegadas a través de cualquier proveedor en la nube.

**Bonificaciones**

1. [5 puntos adicionales] Uso de alineaciones OWASP, uso de HTTPS y cabeceras de seguridad.
2. [5 puntos adicionales] Aplicaciones completamente responsivas y funcionando en diferentes navegadores.
3. [10 puntos adicionales] Habilidades en CSS.
4. [10 puntos adicionales] Código limpio.
5. [10 puntos adicionales] Estructuración de tu aplicación utilizando Arquitectura Hexagonal con el patrón de Puertos y Adaptadores.
6. [10 puntos adicionales] Uso de ROP.

**Consideraciones**  
Para completar la prueba, necesitas al menos 100 puntos.  
Si obtienes los puntos mínimos, continuarás con la etapa de entrevista.