Consumo de APIs

- Para esta prueba fueron utilizadas las siguientes tecnologías.

1. Cluster de kubernetes en cloud (Digital ocean - capa free) No utilice cluster de AWS o GCP por temas de costo.

2. Gitlab CI-CD, aunque el código para este proyecto que le compartiré se encuentra en GitHub, es una replica del usado para el proceso de CI CD en gitlab

3. Dado que poseeo una infraestructura de prueba que basada en microservicios, creación de imagenes, registro de imagenes en dockerHub, acceso a recursos estaticos a un S3 de AWS, acceso y despliegue a cluster mediante helm, preferi agregar dos nuevos servicios para la API solicitada y un pequeño frontend. Puder haber utilizado github Actions pero lo tengo ocupado en otro proyecto de prueba sin el uso de kubernetes.

4. La API fue diseñada en node js, y tiene las siguientes escificaciones:

    - Dominio: http://admin.ramselvin.com (Los dominios no tienen configurado ssl, ya que el sitio lo tengo solo para pruebas)
    - Url base repositories: `http://admin.ramselvin.com/admin/repositories`
    - Creación de un repository service que puede recibir parametros para realizar la petición a la API de github: `https://api.github.com/search/repositories?q=user:${user}&sort=${sort}&order=${order}&per_page=${per_page}`
    - La funcion ejecuta y procesa para solo devolver 7 atributos y enviar al forntend
    - Se agregaron algunas medidas de seguridad basicas, como rated limit, slow down, compression para header, cors, swagger para documentacion de API, aunque solo realiza ese unico proceso, se dejo estructurado de forma general.

5. Frontend app

    - Dominio: http://app.ramselvin.com
    - Se realizo una app simple pero contiene un sistema de login, las credenciales las enviare por correo.
    - La autenticacion consume el servicio realizado en python apuntando a : `http://api.ramselvin.com/auth`
    - Para el frontend utilice React, tailwindcss, typescript y `https://ui.shadcn.com/` para agilizar los componentes bases.
    - El frontend consume la API `http://admin.ramselvin.com/admin/repositories` para obtener la lista de los 10 repositorios solicitados.
    - El frontend use zustand para el manejo de estados, los datos obtenidos desde la API. se registran alli.

- Ejercicio 2

```
function f(x, y, z) {
  let a = x + y;
  let b = a * z;
  let c = Math.sin(b);
  return c;
}
```

- Se agrego documentacion basada en JSDoc
- Se cambio los nombres de argumentos y parametros para sser mas

descriptivos y apropiados
- Se agrego validacion en caso de no ser un tipo number
- Se agrego una excepcion. NOTA aqui dependera del contexto general y uso si se require retornar otro valor un bool

```
        /**
         * Calcula el seno del producto de la suma de dos angulos con un
factor de escala.
         *
         * @param {number} angle1 - El primer angulo.
         * @param {number} angle2 - El segundo ángulo.
         * @param {number} scaleFactor - El factor de escala.
         * @returns {number} - El seno del resultado del producto.
         */
        export const calculateSinOfProductOfSum = (angle1, angle2,
scaleFactor) => {
            if (typeof angle1 !== 'number'
                || typeof angle2 !== 'number'
                || typeof scaleFactor !== 'number'
            ) {
                return false;
                //throw new TypeError('Todos los argumentos deben ser
números.');
            }
            const sumAngles = angle1 + angle2;
            const product = sumAngles * scaleFactor;
            if (isNaN(product)) {
                throw new Error('El resultado del producto no es un
número.');
            }
            return Math.sin(product);
        }
```

- Ejercicio 3
- Se agrego documentacion basada en JSDoc
- Se creo funcion que recibe un numero y valida sea numero, entero y positivo
- Recorre verifica sea numero impar y agrega a un array
- Se agrego una excepcion. NOTA aqui dependera del contexto general y uso si se require retornar otro valor un bool

```
        /**
         * Retorna una matrix de numeros menores del numero proporcionado
         *
         * @param {number} number - El numero a evaluar.
        * @returns {Array} - El resultado en un array
        */
        export const processMatrixOddNumbers = (number) => {
            // Validar que el argumento sea un número
            if (typeof number !== 'number') {
                throw new TypeError('El argumento debe ser un número.');
            }

            // Validar que el número sea positivo
            if (number < 0) {
                throw new Error('El número debe ser positivo.');
            }

            // Validar que el número sea entero
            if (!Number.isInteger(number)) {
                throw new Error('El número debe ser un entero.');
            }

            const oddNumbers = [];
            for (let i = 1; i <= number; i++) {
                if (i % 2 !== 0) {
                    oddNumbers.push(i);
                }
            }
            return oddNumbers;
        }
```

Tecnologías:

1. Infraestructura ### Utilizaría las arquitecturas de Microservicios y orientada a eventos
    - Cluster de Kubernetes para la orquestacíon
    - Helm para gestionar el deploy y actualizar los manifiestos
    - Docker para la creación de imagenes
    - Un registry privado para las imagenes (Gitlab, DockerHub)
    - RabbitMQ para la comunicación entre producción y consumo de mensajes(eventos).
    - API gateway para el trafico y otras configuraciones - o Proxy para balanceo de carga
    - GitLab CI/CD para el ciclo de integración.
2. Lenguajes
    - Node js - Backend - para la mayoria de servicios, usaria al maximo la asíncronia, eventos, websocket, aplicaria medidas de seguridad. Manejo de errores personalizados.
    - Python - Servicios de Autenticación - scripts en infraestructura
    - React - Frontend
    - Typescript
3. Base de datos
    - MySQL - para el core y se debe garantizar las transacciones aplicando los principios ACID
    - MongoDB - para casos especificos no enfocados al core, como catalogos de productos.
4. Seguridad
    - Autenticación y autorización
    - JWT
    - Proveedores IAM
    - Protección para Inyección, CSRF, XSS, etc
    - Protección DDoS, Rated limit
    - Configuración y manejo de errores, Monitoreo continuo de logs, timeouts, compresión en peticiones
    - Validar y sanitizar las entradas.
5. Servicios ### Servicio de Ordenes
    - Solicitar Orden
    - Actualizar Orden ### Servicio de Inventario
    - Obtener Precio
    - Obtener Articulo
    - Reservar item
    - Eliminar Item ### Servicio de Cliente
    - Data Cliente ### Servicio de Entrega
    - Estimar Entrega
    - Iniciar Entrega
    - Check Entrega
    - Estatus Entrega ### Servicio de Pagos
    - Procesar Pago
    - Estatus Pago ### Servicio De Notificaciones
    - Correo
    - Mensaje ### Servicio Autenticación
    - Login
    - Register
    - Token
    - Verificación ## Servicio de Sincronización o actualización ### Servicio Frontend
    - Distintos Modulos ### Servicio Frontend APP Mobile
    - Distintos Modulos
6. Pasarelas de pago
    - Aqui se debe verificar segun la tasa de cliente y paises donde se piensa iniciar
    - La pasarela debe tener presencia en esos paises, asi como otros

aspectos a considerar.
    - De preferencia usaria stripe

7- Otros Servicios de terceros necesarios como envios, análisis de datos, registro de logs, metricas. Jest para pruebas funcionales.

Estructura de archivos

```
src|
config/
 - database.config.js
 - swagger.config.js
 - otros
controllers/
 - user.controller.js
 - product.controller.js
 - order.controller.js
 - index.js
 - otros
models/
 - user.model.js
 - product.model.js
 - order.model.js
 - index.js
 - otros
routes/
 - user.routes.js
 - product.routes.js
 - order.routes.js
 - index.js
 - otros
services/
 - auth.eervice.js
 - product.service.js
 - order.service.js
 - otros
middlewares/
 - auth.middleware.js
 - otros
utils/
 - date.utils.js
 - otros
class/
 - translation.js
 - error.handle.js
 - otros
test/
 - test.js
constants/
 - route.js
main.js
```

- controllers: Se usa para la lógica de negocio de la aplicación.
- models: Se usa para esquemas de datos y las operaciones de base de datos.
- routes: Se usa para rutas de la API y llaman a los controladores correspondientes.
- services: Se usa para lógica de negocio adicional que no pertenece a los controladores o servicios adicionales
- middlewares: Se usa para ejecutar antes de ejecuciones.
- config: Se usa para archivos de configuración para la base de datos, autenticación, etc.
- clases: Se usa para clases o facilidades genericas
- constants: Se usa para las constantes en general
- main.js: Se usa para inicializar la aplicación.

Patrones de diseño:

Patrones de Arquitectura que usaria * Microservices - Cada servicio independiente. * API Gateway - Para el enrutamiento, la seguridad, rated limit. * MVC - core de la app * Orientado ha eventos - para servici en flujo de pedidos, pagos, envios, monitorizacion y alertas.o * Circuit Breaker - para manejo adecuado de fallos, errores y posible sobrecarga.

Estilos de Arquitectura de API * Websocket * Rest * GraphQL * Serverles si usamos en estrutura cloud

Patrón Singleton: Lo usaria para manejar conexiones a la base de datos, configuraciones generales.

Patrón Factory: Lo usaria en este contexto para la creación de objetos complejos, Ej: creación de instancias de productos con sus diferentes atributos.

Patrón Repositorio: Dependiendo ya que tenemos servicios diferentes, pudiera usarlo en algunos modelos que se necesite.

Patrón Middleware: Lo usaria para manejar tareas comunes como la autenticación, la validación de datos y el registro de errores de manera general en toda la aplicación.

Pruebas: * Utilizaria Jest para las pruebas unitarias * Otras pruebas necesarias aplicaria de integración, de stress, de velocidad. * Otros sevicios para probar como apache benchmarks para las API, dive para la images.

Nota: Estas seria la estructura inicial del proyecto para comercio electronico, claro exisen otra variantes que se deben tener en cuenta. Como requerimientos funcionales, reglas del negocio, logica del negocio, etc

Prueba Técnica para la empresa Master

Prueba de ingreso - Desarrollador fullstack enfocado en backend - Master

Repositorio: https://github.com/elvingonzalezb/master-test

Ejercicio 1 https://github.com/elvingonzalezb/master-test/wiki/Ejercicio-1

Ejercicio 2 https://github.com/elvingonzalezb/master-test/wiki/Ejercicio-2-y-3

Modelado de bases de datos: https://github.com/elvingonzalezb/master-test/wiki/Modelado-de-bases-de-datos

Arquitectura del backend: https://github.com/elvingonzalezb/master-test/wiki/Estructura-comercio-electr%C3%B3nico

Nomenclatura: https://github.com/elvingonzalezb/master-test/wiki/Pol%C3%ADtica-de-Nomenclatura-para-el-Equipo-de-Desarrollo

Modelo general para la base de datos

Para la estructura solicitada tablas para vídeos, autores, colaboradores, comentarios, reviews y usuarios

```
-- Creación de la base de datos si no existe
CREATE DATABASE IF NOT EXISTS MASTER_TEST;

-- Utilizar la base de datos MASTER_TEST
USE MASTER_TEST;

-- Inicio de la transacción
BEGIN;

    CREATE TABLE users (
        user_id INT PRIMARY KEY,
```

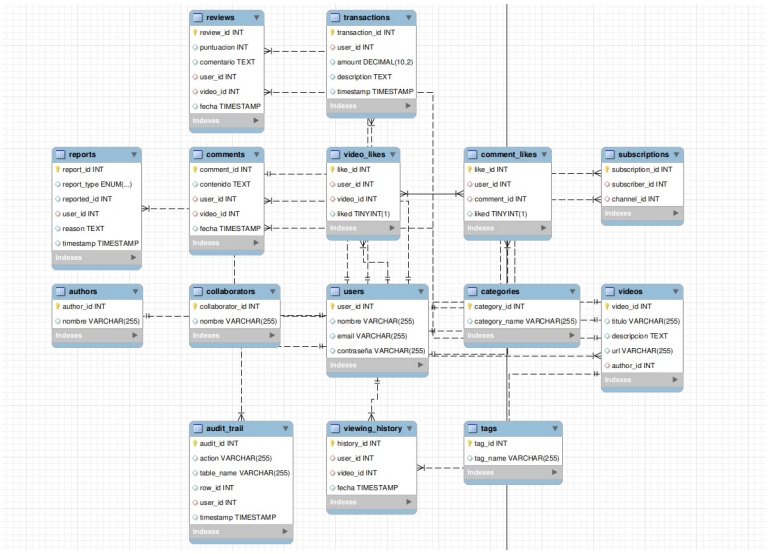
```
    nombre VARCHAR(255),
    email VARCHAR(255) UNIQUE,
    contraseña VARCHAR(255)
);

CREATE TABLE authors (
    author_id INT PRIMARY KEY,
    nombre VARCHAR(255)
);

CREATE TABLE collaborators (
    collaborator_id INT PRIMARY KEY,
    nombre VARCHAR(255)
);

CREATE TABLE videos (
    video_id INT PRIMARY KEY,
    titulo VARCHAR(255),
    descripcion TEXT,
    url VARCHAR(255),
    author_id INT,
    FOREIGN KEY (author_id) REFERENCES authors(author_id)
);

CREATE TABLE comments (
    comment_id INT PRIMARY KEY,
    contenido TEXT,
    user_id INT,
    video_id INT,
    fecha TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (video_id) REFERENCES videos(video_id)
);

CREATE TABLE reviews (
    review_id INT PRIMARY KEY,
    puntuacion INT,
    comentario TEXT,
    user_id INT,
    video_id INT,
    fecha TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (video_id) REFERENCES videos(video_id)
);

-- Rollback transacción
ROLLBACK;

-- Commit  transacción
COMMIT;
```

Otras tablas que considero importante para la aplicación.

```
-- Creación de la base de datos si no existe
CREATE DATABASE IF NOT EXISTS MASTER_TEST;

-- Utilizar la base de datos MASTER_TEST
USE MASTER_TEST;

-- Inicio de la transacción
BEGIN;

    CREATE TABLE categories (
        category_id INT PRIMARY KEY,
        category_name VARCHAR(255) UNIQUE
    );

    CREATE TABLE tags (
        tag_id INT PRIMARY KEY,
        tag_name VARCHAR(255) UNIQUE
    );


    CREATE TABLE subscriptions (
        subscription_id INT PRIMARY KEY,
```

```
    subscriber_id INT,
    channel_id INT,
    FOREIGN KEY (subscriber_id) REFERENCES users(user_id),
    FOREIGN KEY (channel_id) REFERENCES users(user_id)
);

CREATE TABLE video_likes (
    like_id INT PRIMARY KEY,
    user_id INT,
    video_id INT,
    liked BOOLEAN,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (video_id) REFERENCES videos(video_id)
);

CREATE TABLE comment_likes (
    like_id INT PRIMARY KEY,
    user_id INT,
    comment_id INT,
    liked BOOLEAN,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (comment_id) REFERENCES comments(comment_id)
);

CREATE TABLE viewing_history (
    history_id INT PRIMARY KEY,
    user_id INT,
    video_id INT,
    fecha TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (video_id) REFERENCES videos(video_id)
);

CREATE TABLE audit_trail (
    audit_id INT PRIMARY KEY AUTO_INCREMENT,
    action VARCHAR(255),
    table_name VARCHAR(255),
    row_id INT,
    user_id INT,
    timestamp TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

CREATE TABLE transactions (
    transaction_id INT PRIMARY KEY,
    user_id INT,
    amount DECIMAL(10, 2),
    description TEXT,
    timestamp TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

CREATE TABLE reports (
    report_id INT PRIMARY KEY,
    report_type ENUM('comment', 'video'),
    reported_id INT,
    user_id INT,
    reason TEXT,
    timestamp TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

-- Rollback transacción
ROLLBACK;

-- Commit  transacción
COMMIT;
```

- Es de vital importancia entender que el modelo esta basado en la representación generica de como debe funcionar una apliación para la gestion de videos.
- Esta puede variar dependiendo los requerimientos funcionales, reglas y logica comercial.



image

Política de Nomenclatura para el Equipo de Desarrollo

El presente documento es solo un propotipo inicial para documentar el estandar general de la empresa Master.

No es un documento completo, contiene algunos puntos importantes solicitados para la evaluación.

Se debe ajustar y agregar mas reglas y convenciones para manejo de errores y excepciones, seguridad, internacionalización, buenas prácticas generales, mantenimiento del estándar. etc.

Leyenda:
- M: Inicial nombre empresa - 0: Sección - 0.x: Subsección de una sección

M.0 General

M.0.1 Indentación * La indentación consiste en 4 espacios. * No usar tabulaciones. * Nunca mezclar espacios y tabulaciones.

```
function hello() {
    console.log('Hola, mundo!');
}
```

M.0.2 Longitud de una línea * El objetivo es tener líneas de hasta 120 caracteres, en lo posible. * En casos especificos es permitido longitud máxima de una línea de código es 140 caracteres, esto aplica tanto para código JS, Python, scripts SQL.

M.0.3 Variables

- Usar camelCase para el nombre de las variables.
- Longitud de entre 7 y 15 caracteres.
- Nunca mezclar espacios y tabulaciones.
- Nombres significativos que reflejen claramente su propósito.
- Evitar abreviaturas ambiguas.
- Utilizar prefijos como `has`, `is`, `can` para variables booleanas.
- Reducir al mínimo el uso de variables globales.
- Declarar variables en el ámbito más específico posible.

```
// Uso correcto
let paymentGateway = 'VISA';
let isActive = true;
let userName = 'Master';
let userAge = 30;
```

```
// Uso incorrecto
let nomPasarela = 'VISA';
let activo = true;
var globalVar = 'Global';
```

M.0.4 Constantes

- Usar UPPER_SNAKE_CASE para constantes.
- Longitud de entre 7 y 20 caracteres.

```
// Uso correcto
const BASE_URL_AUTH = 'http://api.master.com/auth/';

// Uso incorrecto
let url = 'http://api.master.com/auth/';
```

M.0.5 Clases

- Usar PascalCase para el nombre de las clases.
- Nombres descriptivos y claros que indentifiquen el propósito.
- No agregar abreviaturas sin sentido.
- Las clases deben tener una unica responsabilidad (Principio de Responsabilidad Única).
- Utilizar métodos de acceso (gets y sets) controlar el acceso a los miembros de la clase.

```
// Uso Correcto
class User {
  constructor(username, email) {
    this.username = username;
    this.email = email
  }

  // Get de usuario
  getInfo() {
    return `Este correo: ${this.email} pertenece a ${this.username}
`;
  }
}

// Uso Incorrecto
class use {
  constructor(u, e) {
    this.u = u;
    this.e = e;
  }

  // Uso Incorrecto
  info() {
    return `Este correo: ${this.e} pertenece a ${this.u} `;
  }
}
```

M.0.6 Funciones

- Usar camelCase para el nombre de las funciones.
- Nombres descriptivos que indiquen claramente lo que hace la función.
- No agregar abreviaturas sin sentido.
- Evitar funciones demasiado largas o complejas (mantenerlas entre 10 y 20 líneas si es posible).
- Limitar cantidad de return a 3 max 4

```
// Uso Correcto
/**
 * Calcula el seno del producto de la suma de dos angulos con un
factor de escala.
 *
 * @param {number} angle1 - El primer angulo.
 * @param {number} angle2 - El segundo ángulo.
 * @param {number} scaleFactor - El factor de escala.
 * @returns {number} - El seno del resultado del producto.
```

```
         */
        export const calculateSinOfProductOfSum = (angle1, angle2,
scaleFactor) => {
            if (typeof angle1 !== 'number'
                || typeof angle2 !== 'number'
                || typeof scaleFactor !== 'number'
            ) {
                return false;
                //throw new TypeError('Todos los argumentos deben ser
números.');
            }
            const sumAngles = angle1 + angle2;
            const product = sumAngles * scaleFactor;
            if (isNaN(product)) {
                throw new Error('El resultado del producto no es un
número.');
            }
            return Math.sin(product);
        }

        // Uso Incorrecto
        function f(x, y, z) {
          let a = x + y;
          let b = a * z;
          let c = Math.sin(b);
          return c;
        }
```

M.0.6 Documentación

- Usar generador de documentación JSDoc
- Aplicar para funciones, clases.

```
        // Uso Correcto
        /**
         * Calcula el seno del producto de la suma de dos angulos con un
factor de escala.
         *
         * @param {number} angle1 - El primer angulo.
         * @param {number} angle2 - El segundo ángulo.
         * @param {number} scaleFactor - El factor de escala.
         * @returns {number} - El seno del resultado del producto.
         */
        export const calculateSinOfProductOfSum = (angle1, angle2,
scaleFactor) => {
            if (typeof angle1 !== 'number'
                || typeof angle2 !== 'number'
                || typeof scaleFactor !== 'number'
            ) {
                return false;
                //throw new TypeError('Todos los argumentos deben ser
números.');
            }
            const sumAngles = angle1 + angle2;
            const product = sumAngles * scaleFactor;
            if (isNaN(product)) {
                throw new Error('El resultado del producto no es un
número.');
            }
            return Math.sin(product);
        }
```

M.0.7 Arrays

- Usar nombres plurales para los arrays.
- Usar los nombres de los arrays descriptivos y significativos.
- No usar nombres genéricos como `data`, `array`, etc.
- Utilizar corchetes (`[]`) para la declaración de arrays.

```
        // Uso Correcto
        {
          "users": [
            {"id": 1, "name": "Master"},
            {"id": 2, "name": "Elvin"}
```

```
  ],
  "products": [
    {"id": 101, "name": "Laptop"},
    {"id": 102, "name": "Phone"}
  ]
}

// Uso incorrecto
{
  "data": [
    {"n": 1, "nm": "Master"},
    {"n": 2, "nm": "Elvin"}
  ],
  "arr": [1, 2, 3]
}
```

M.0.8 JSON

- Mantener los nombres de las claves descriptivos y significativos.
- Usar comillas dobles (`" "`) para los nombres de las claves y valores de tipo string.
- Usar sintaxis de objeto (`{}`) para la definición de JSON.

```
// Uso Correcto
{
  "user": {
    "id": 1,
    "name": "Master",
    "email": "master@master.com"
  },
  "product": {
    "id": 101,
    "name": "Celular",
    "price": 850.000
  }
}

// Uso incorrecto
{
  "usr": {
    "i": 1,
    "n": "Master",
    "e": "master@master.com"
  },
  "prdct": {
    "i": 101,
    "nm": "Celular",
    "p": 850.500
  }
}
```

M.1 Base de datos M.1.0 General base de datos

- Usar nombres descriptivos y significativos para las tablas, columnas, índices y restricciones.
- Usar el formato snake_case para los nombres de las tablas y las columnas.
- Usar mayúsculas para las palabras clave de SQL (SELECT, INSERT, UPDATE, DELETE, otras).
- Usar comillas invertidas (`select`) para nombres de tablas, columnas o alias que contengan caracteres especiales o palabras clave reservadas.
- Usar comentarios descriptivos en el esquema de la base de datos para explicar la estructura, relaciones y cualquier otra información relevante.
- Usar índices para mejorar el rendimiento de consultas frecuentes.
- Usar transacciones para agrupar múltiples operaciones SQL y garantizar la consistencia de los datos.

```
// Uso Correcto
CREATE TABLE `users` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
```

```
  `username` VARCHAR(50) NOT NULL,
  `email` VARCHAR(100) NOT NULL,
  `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX `idx_username` ON `users` (`username`);

// Uso incorrecto
CREATE TABLE `Client` (
  `clientID` INT AUTO_INCREMENT PRIMARY KEY,
  `clientName` VARCHAR(50) NOT NULL,
  `Client_email` VARCHAR(100) NOT NULL,
  `RegisteredAt` TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX `client_name_idx` ON `Client` (`clientName`);

// Uso Correcto en transacciones
START TRANSACTION;

INSERT INTO users (username, email) VALUES ('elvin',
'elvin@elvin.com');
UPDATE account SET balance = balance - 100 WHERE user_id = 1;

COMMIT;

// Uso incorrecto
INSERT INTO users (username, email) VALUES ('elvin',
'elvin@elvin.com');

START TRANSACTION;

UPDATE account SET balance = balance - 100 WHERE user_id = 1;

COMMIT;
```

M.3 Git M.3.0 General git

- Flujo de trabajo
- Branch principal `master`
- Envio a producción mediante tag con release `1.0.0`
- Flujo de desarrollo branch principal `develop`
- A partir de esa branch se deben crear las ramas de trabajo
- En casos de `bugfix` - `feature` - `hotfix` se debe anteponer `bugfix/payment-not-processed`
- El ramas de tareas debe estar formada por el numero de tarea y una breve accion `#000-create-contact-form`
- Cada vez que inicie labores debe actualizar el repositorio remoto con su local

M.3.1 Resolución de conflictos con rebase

- pull de los cambios (actualizar el repositorio local con el repositorio principal)

```
git pull --rebase
```

- confirman la resolución

```
git add .
git rebase --continue
```

- Subir los cambios

```
git push -f origin <branch-id>
```

M.3.2 Resolución de conflictos con merge

- Ir a la rama develop

```
git checkout develop
```

- Actualizar

```
git pull origin develop
```

- Ir a la rama de la incidencia o de trabajo actual

```
git checkout origin <branch>
```

- Merge con develop

```
git merge origin/develop
```

- Verificar conflictos resolver y confirmar

```
git add .
git merge --continue
git push -f origin <branch-id>
```

links sugeridos * https://github.com/rwaldron/idiomatic.js/tree/master/translations/es_ES * https://mhdev.readthedocs.io/es/latest/js-style.html * https://www.w3schools.com/js/js_scope.asp * https://code.tutsplus.com/24-javascript-best-practices-for-beginners–net-5399t * https://owasp.org/