Prueba Técnica – sistemas de información base

La prueba que va a realizar consiste en validar las competencias técnicas del participante con el que podamos valorar su conocimiento con relación a las tecnologías/metodologías que se utilizarán en el cargo al que opta.
Es una prueba abierta, y la mayoría de las actividades pueden tener diferentes soluciones, todas ellas correctas. Se valorará positivamente la claridad de la respuesta y su sencillez.
La prueba tiene que realizarse de manera individual.

Nombre	:	
Fecha	:	(AAAA-MM-DD)

1. La empresa **MERCADER LTDA** es un negocio de retail que comercializa diferentes productos para el hogar, cuenta con diferentes canales de venta digitales y presenciales. El Gerente de BI de la compañía quiere generar un consolidado de los clientes con el fin de realizar diferentes estratégicas a nivel de mercadeo, sin embargo, actualmente la información de los clientes reposa en 7 bases de datos diferentes con datos de los clientes.

   La información que se requiere en el listado debe contener la siguiente información:

   a. Datos de Contacto (correo electrónico, celular, teléfono, dirección, ciudad, departamento, país), el dato de contacto que se requiere debe obedecer al dato más confiable dado que en las diferentes bases de datos pueden existir diferentes datos de contacto.
   b. Datos de identificación (documento, tipo de documento, nombres y apellidos).
   c. Autorización para manejo de datos personales por parte del cliente.

   Otros requerimientos:

   a. Se requiere que se automatice el proceso para que cuando ingresen nuevos clientes se actualice el listado por lo menos 1 vez al día.
   b. Se requiere realizar proceso de macheo de registros y limpieza (Que reglas implementaria).
   c. El listado debe mostrar el Golden Record del cliente.
   d. Se debe validar en la integración de datos cuando un documento es NIT dado que desde las diferentes bases de datos no hay validación del tipo de identificación.
   e. El Gerente de BI de la compañía requiere poder visualizar la información en un tablero de control o un informe de fácil manejo.

   Información adicional:

   Las bases de datos y tablas donde reposa la información de los clientes son las siguientes:

SODIMAC corona

Base de Datos	Esquema	Tablas	Campos	Nivel de Calidad de Datos 1: Baja 2: Media 3: Alta
PEDIDOS	PD	Clientes	Id, Documento, Tipo_Documento, Nombres, Apellidos.	1
PEDIDOS	PD	Direcciones	Id, Dirección, Ciudad, Departamento, País, correo.	1
DEVOLUCIONES	DV	Clientes	Id, Documento, Tipo_Documento, Nombres, Apellidos.	2
DEVOLUCIONES	DV	Direcciones	Id, Dirección, Ciudad, Departamento, País, correo.	2
ESPECIALISTAS	EP	Tbl_Clientes	Documento, Tipo_Documento, Nombres, Apellidos, Dirección, Ciudad, Departamento, País, correo.	1
ALQUILER	AL	T_Clientes	Documento, Tipo_Documento, Nombres, Apellidos, Dirección, Ciudad, Departamento, País, correo.	2
NOVIAS	NOV	Clientes	Id, Documento, Tipo_Documento, Nombres, Apellidos, Dirección, Ciudad, Departamento, Pais, correo.	1
EMPRESARIOS	EM	Clientes	Id, Documento, Tipo_Documento, Nombres, Apellidos.	3
EMPRESARIOS	EM	Direcciones	Id, Dirección, Ciudad, Departamento, País, correo.	3
CENTRAL DE AUTORIZACIONES	CA	Cliente	Id, Documento, Tipo_Documento, Nombres, Apellidos, Dirección, Ciudad, Departamento, País, correo, fecha de autorización, Autoriza (Si/No)	3

Nota: Las anteriores bases de datos se encuentran en primera forma normal, los tipos de datos de cada uno de los campos son los siguientes:

Campo	Tipo de Dato
Id	Llave Principal Autonumerico
Documento	Varchar (20)
Tipo_Documento	Varchar (20)
Nombres	Varchar (100)
Apellidos	Varchar (100)
Dirección	Varchar (200)
Ciudad	Varchar (100)
Departamento	Varchar (100)
País	Varchar (100)
Correo	Varchar (100)
Fecha de autorización	TimeStamp
Autoriza (Si/No)	Boolean

Punto a resolver.

- Diseñe la arquitectura de la solución, debe suministrar un artefacto que permita visualizar las capas de arquitectura, la mensajería, capa de datos y los sistemas que se proponen en la solución.

- Determine las actividades de desarrollo de acuerdo a los requerimientos relacionados en el punto anterior, indique de acuerdo a la metodología ágil la planeación que tendría en tiempos y recursos.
- Con el fin de detallar la solución en una segunda entrega, que preguntas realizaría al comité funcional que no estén resueltas en este documento.

2. Analice los siguientes fragmentos de código, los cuales son parte de un desarrollo, del análisis se espera entregue un diagrama de flujo o una explicación del entendimiento que tuvo para usted el código fuente proporcionado. Consumos que se hacen, Apis que consumen, reglas identificadas, etc.

Se entrega código del service, controller, route

Controller

```
import { NextFunction, Request, Response } from 'express';
import { auditResponse } from '../middlewares/audit.middleware';
import { getCliente360Service } from '../services/cliente360.services';
import { handleError } from '../services/error.service';
import { validateBody } from '../services/validate.services';

export async function getCliente360Controller(req: Request, res: Response,
next: NextFunction) {

  try {
    validateBody(req.query);
    Promise.all([getCliente360Service(req.query, res.locals, res)])
      .then(data => {
        return auditResponse(res, data[0]);
      }).catch(err => {
        return handleError(err, req, res, next)
      })
  } catch (error) {
    next(error)
    return res
  }
}
```

Route

```
import { Router } from 'express';
const microServiceRouter = Router();

import { getCliente360Controller } from
'../controllers/cliente360.controller';

microServiceRouter.route('/api/cliente360')
```

```
    .get(getCliente360Controller);

export default microServiceRouter;
```

Services

```
import oracledb from "oracledb";
import { AppLogger } from "../classes/appLogger.class";
import { commerce, country, originIP } from "../config/cmr.config";
import { QGetCliente360SP } from "../config/database.config";
import {
  closeDbConnection,
  executeQuery,
  fetchRowsFromCursor,
  getDbConnection,
} from "../database";
import { Databases } from "../enums/database.enums";
import { LogLevels } from "../enums/logLevels.enum";
import { createTransResponse } from "../utils/response.utils";

let traceId = 0;
let operationName = "";

export async function getCliente360Service(query: any, srvcInfo: any, res:
any) {
  const channel = query.channel;
  const formatBody = query;
  delete formatBody.channel;
  const validate: any = formatBody;
  validate["channel"] = channel;

  operationName = "Rq_getCliente360_SP";
  traceId = res.locals.tranId;
  AppLogger.log(
    LogLevels.INFO,
    getCliente360Service.name,
    `Se va a consultar cliente: ${JSON.stringify(validate)}`,
    operationName,
    traceId
  );

  const connection = await getDbConnection(Databases.HCHUBD_DB);
  let customer: any = {};
  let dbMessage = "";
  let dbContactable = 1;
```

```
  let resSVC: any = {};
  try {
    const data = await executeQuery(connection, QGetCliente360SP, {
      PCANAL: validate.channel,
      PTIPODOCUMENTO: validate.documentType,
      PIDENTIFICACION: validate.documentNumber,
      OUTCODIGO: { type: oracledb.NUMBER, dir: oracledb.BIND_OUT },
      OUTMENSAJE: { type: oracledb.STRING, dir: oracledb.BIND_OUT },
      CODIGOCONTACTABLE: { type: oracledb.NUMBER, dir: oracledb.BIND_OUT
},
      PDATOS: { type: oracledb.CURSOR, dir: oracledb.BIND_OUT },
    });

    dbMessage = data.outBinds.OUTMENSAJE;
    dbContactable = data.outBinds.CODIGOCONTACTABLE;

    if (data.outBinds.CODIGOCONTACTABLE == 0) {
      const rows = await fetchRowsFromCursor(data.outBinds.PDATOS);
      customer = rows[0];
      closeDbConnection(connection);
    } else if (data.outBinds.CODIGOCONTACTABLE == 1) {
      resSVC = {
        codigoContactable: data.outBinds.CODIGOCONTACTABLE.toString(),
        contactable: dbMessage,
      };
      closeDbConnection(connection);
    } else {
      operationName = "Rs_getCliente360_SP";
      AppLogger.log(
        LogLevels.INFO,
        getCliente360Service.name,
        `Estado del cliente: ${JSON.stringify(dbMessage)}`,
        operationName,
        traceId
      );
      throw { status: 404 };
    }
  } catch (err: any) {
    dbMessage = err.message;
    if (err.status == 404) {
      throw {
        status: 404,
        message: createTransResponse("Failed", "Not Found", dbMessage),
      };
    } else {
```

```
      throw { status: 500, message: err.message };
    }
  }

  if (res.statusCode != 504) {
    operationName = "Rs_getCliente360_SP";
    AppLogger.log(
      LogLevels.INFO,
      getCliente360Service.name,
      `Estado del cliente: ${JSON.stringify(dbMessage)}`,
      operationName,
      traceId
    );
  }
  if (dbContactable == 0) {
    const documentType =
documentHomologationService(customer.documentType);
    const hash = generateHashService(documentType,
validate.documentNumber);
    resSVC = await CMRPuntosService(validate, customer, hash, srvcInfo,
res);
    resSVC.optinVersion = customer.optinVersion;
  }
  return resSVC;
}

function documentHomologationService(documentType: string) {
  let type = "";
  switch (documentType) {
    case "cc":
      type = "01";
      break;
    case "nit":
      type = "02";
      break;
    case "ce":
      type = "03";
      break;
    case "pas":
      type = "05";
      break;
    default:
      throw {
        status: 400,
        message: createTransResponse(
```

```
          "Failed",
          "Bad request",
          "No se encontro tipo de documento"
        ),
      };
  }
  return type;
}

export function generateHashService(documentType: any, documentNumber:
any) {
  const value = country + documentType + documentNumber;
  const cipher = CryptoJS.SHA256(value);
  const cipherBase = cipher.toString(CryptoJS.enc.Base64);
  let hash = "";

  let replace = cipherBase.split("+").join("-");
  hash = replace.split("/").join("_");
  return hash;
}

export async function CMRPuntosService(
  validate: any,
  customer: any,
  hash: any,
  srvcInfo: any,
  res: any
) {
  let https = require("https");
  const myagent = new https.Agent({
    rejectUnauthorized: false,
  });

  const { CMR_ENDPOINT, CMR_API_KEY } = process.env as any;
  const URL_CMR_PUNTOS = CMR_ENDPOINT + hash + "?key=" + CMR_API_KEY;

  operationName = "Rq_getCMRPuntos";
  const traceId = res.locals.tranId;
  AppLogger.log(
    LogLevels.INFO,
    CMRPuntosService.name,
    `Se va a consultar cliente: {"customer":"
${country}${customer.documentType}${customer.documentNumber} "} , SHA256
{"customer":"${hash}"} `,
    operationName,
```

```
    traceId
  );

  let statusCodeDefault = 0;
  let dataResponse = "";
  let axiosRes: any = "";
  let checkCMR = false;

  const cmrResponse = await axios
    .get(URL_CMR_PUNTOS, {
      headers: {
        "X-Origin-Ip": originIP,
        "X-Channel": validate.channel,
        "X-Country": country,
        "X-Commerce": commerce,
      },
      httpsAgent: myagent,
    })
    .then((res: any) => { axiosRes = res.data; statusCodeDefault =
res.status; })
    .catch((err: any) => {
      statusCodeDefault = err.response.status;
      dataResponse = JSON.stringify(err.response.data.status);
      switch (statusCodeDefault) {
        case 400:
          throw {
            status: 400,
            message: `Message from CMR API. ${dataResponse}`,
          };
        default:
          break;
      }
      if (statusCodeDefault != 200 && statusCodeDefault != 404) {
        throw {
          status: 500,
          message: `Message from CMR API. ${err.response.data}`,
        };
      }
    });

  if (statusCodeDefault == 200 && axiosRes.status == "OK") {
    operationName = "Rs_getCMRPuntos";
    AppLogger.log(
      LogLevels.INFO,
      CMRPuntosService.name,
```

```
      `Estado del cliente: ${JSON.stringify(axiosRes)}`,
      operationName,
      traceId
    );
    checkCMR = true;
  }
  if (statusCodeDefault == 404) {
    operationName = "Rs_getCMRPuntos";
    AppLogger.log(
      LogLevels.INFO,
      CMRPuntosService.name,
      `Estado del cliente: ${JSON.stringify(dataResponse)}`,
      operationName,
      traceId
    );
    checkCMR = false;
  }
  customer["checkCMR"] = checkCMR;
  return customer;
}
```