Parte 1

a. Indique con sus propias palabras, qué tecnologías utilizaría para garantizar la solución. Justifique su respuesta.

Antes de abordar las tecnologías que usaría, propongo el uso de una arquitectura MVC dado que como es una arquitectura modular que promueve la separación de responsabilidades, facilita el mantenimiento y la escalabilidad. La arquitectura se vería así:

Modelo:

- **MongoDB** – Base de datos NoSQL que se ajusta a el problema de los datos debido a que como se trabaja con fondos de inversión estas estructuras podrían ser flexibles y dinámicas. MongoDB permite almacenar documentos tanto de las transacciones como de los fondos de inversión de manera eficiente y si se piensa en el aumento constante de clientes, la escalabilidad será mucho más sencilla.

- **Mongoose** – ODM para MongoDB para facilitar la integración y proporciona un esquema claro para definir los datos, las validaciones y middlewares.

Vista:

- **Angular** – Framework de frontend robusto que facilita la creación de interfaces interactivas y dinámicas. Además, permite el manejo de funcionales y/o estados a partir de servicios.

- **TailwindCSS** – Framework de CSS que, gracias a su estructura de estilos basada en clases, promueve una velocidad y facilidad a la hora de construir los estilos de la aplicación.

Controlador:

- **Node.js** – Alternativa para manejo de aplicaciones web que permite el manejo de operaciones asíncronas, lo que sería un excelente aliado para el sistema de transacciones para mantener una alta disponibilidad y velocidad.

- **Express.js** – Framework de Node.js que facilita la creación de APis Rest, necesarias para manejo de suscripciones, cancelaciones e historial de transacciones.

b. Diseñe un modelo de datos NoSQL que permita la solución al problema.

En modelo de datos NoSQL se basará en la construcción de 4 documentos:

- Fund Subscriptions: almacena suscripciones activas y pasadas del usuario.

```
const FundSchema = new mongoose.Schema(
  {
    fundId: {
      type: Number,
      required: true,
      unique: true,
      immutable: true,
    },
    name: {
      type: String,
      required: true,
    },
    minAmount: {
      type: Number,
      required: true,
    },
    category: {
      type: String,
      enum: ["FPV", "FIC"],
      required: true,
    },
  },
  { timestamps: true }
);
```

- Transactions: almacena historial de transacciones (aperturas y cancelaciones).

```
const TransactionSchema = new mongoose.Schema(
  {
    transactionId: {
      type: String,
      required: true,
    },
    fundId: {
      type: Number,
      ref: "Fund",
      required: true,
    },
    name: {
      type: String,
    },
    type: {
      type: String,
      enum: ["Subscription", "Cancellation"],
      required: true,
    },
    date: {
      type: Date,
      default: Date.now,
    },
    amount: {
      type: Number,
      required: true,
    },
  },
  { timestamps: true }
);
```

- User: almacena las opciones del usuario para manejar su dinero, historial de transacciones, fondos inscritos y preferencias para envió de notificaciones.

```
const UserSchema = new mongoose.Schema(
  {
    userId: {
      type: Number,
      unique: true,
      immutable: true,
    },
    name: {
      type: String,
      required: true,
    },
    email: {
      type: String,
    },
    sms: {
      type: String,
    },
    balance: {
      type: Number,
      default: 500000,
    },
    funds: [
      {
        fundId: {
          type: Number,
          ref: "Fund",
        },
        name: {
          type: String,
        },
        amount: {
          type: Number,
          required: true,
        },
        notificationPreferences: {
          type: String,
          enum: ["EMAIL", "SMS"],
          required: true,
        },
      },
    ],
    transactions: {
      type: [TransactionSchema],
      default: [],
    },
  },
  { timestamps: true }
);
```

Parte 2

Escriba las consultas SQL correspondientes, para ello, tenga en cuenta la base de datos llamada "BTG" la cual tiene las siguientes tablas (tenga en cuenta que se puede presentar el caso de que no todas las sucursales ofrecen los mismos productos).

Obtener los nombres de los clientes los cuales tienen inscrito algún producto disponible sólo en las sucursales que visitan.

```
SELECT DISTINCT c.nombre, c.apellidos
FROM Cliente c
JOIN Inscripción i ON c.id = i.idCliente
JOIN Producto p ON i.idProducto = p.id
JOIN Disponibilidad d ON p.id = d.idProducto
JOIN Sucursal s ON d.idSucursal = s.id
JOIN Visitan v ON c.id = v.idCliente AND s.id = v.idSucursal
WHERE NOT EXISTS (
  SELECT 1
  FROM Disponibilidad d2
  WHERE d2.idProducto = p.id
  AND d2.idSucursal != v.idSucursal
);
```