VIRTUALISA	**Virtualisa S.A**
	Entrevista para la contratación de nuevos recursos humanos **Desarrollador Full Stack MERN Ssr / Sr**

Dia / hora de la entrevista: 19 / 04 / 24

Nombre, Apellido: Nicolás Panneblanco

Preguntas de carácter general y abiertas

1. Situación académica: Carrera universitaria, Cursos, estado actual.
   Tecnicatura Superior en programación UTN Regional – Tecnicatura Superior en sistemas Informáticos

2. Porque elegiste ser un Desarrollador Full Stack y desde hace cuanto lo eres?

3. Experiencia laboral: Empresas y puestos en los que trabajó o trabaja.

   Municipalidad de San Martin de los Andes hace 4 años en un proyecto como, Full Stack.

4. Experiencia de trabajo en equipo, metodologías de trabajo, herramientas.
   Dos personas en el equipo de desarrollo.

5. ¿Cuál diría que es su punto más fuerte como desarrollador?

6. ¿Cuál diría que es su punto más débil como desarrollador?

7. ¿Cuáles son tus proyectos más destacados hasta ahora?

8. ¿Cómo te mantienes actualizado sobre las últimas tendencias y tecnologías en desarrollo web?

9. ¿Qué sabes de esta organización y de los productos que se desarrollan?

10. ¿Puedes contarme sobre una vez que enfrentaste un desafío importante en equipo y cómo lo superaste?

Preguntas para evaluar conocimiento técnico en general

1. ¿ Qué conocimientos tiene de UML? ¿donde plasmaría el diseño de la solución a un problema?
   Diagramas.

2. Herramientas utiliza/conoce para versionado de código, para la gestión y construcción de proyectos, para el versionado de componentes.
   Versionado con servidores on premise en la municipalidad

3. Describe la diferencia entre las pruebas unitarias, de integración y las pruebas funcionales.

4. Conoce lo que es un estándar de código.

5. Describe brevemente cada tecnología utilizada en el Stack MERN.

6. Proporcione un ejemplo específico de un problema técnico que hayas enfrentado y cómo lo solucionó.

7. Cuénteme sobre una vez en la que tuvo que aprender rápidamente una nueva tecnología para un proyecto.

Preguntas para evaluar conocimiento técnico MySql

1. Índices en MySQL:
    - ¿Cuál es la función principal de los índices en una base de datos MySQL y cómo afectan al rendimiento de las consultas? Proporciona ejemplos de situaciones en las que agregar un índice sería beneficioso y situaciones en las que no sería recomendado.
2. Normalización y Desnormalización:
    - Explica los conceptos de normalización y desnormalización en el contexto de una base de datos relacional como MySQL. ¿En qué situaciones considerarías la desnormalización y cuáles serían las implicaciones?
3. Consulta y Optimización:
    - Supongamos que tienes una tabla con millones de registros en MySQL. ¿Cómo optimizarías una consulta para mejorar su rendimiento? Menciona al menos dos estrategias y explica por qué serían efectivas.
4. Transacciones en MySQL:
    - Describe el concepto de transacciones en MySQL. ¿Cuándo y por qué considerarías el uso de transacciones en una aplicación MySQL? Proporciona ejemplos de escenarios en los que las transacciones son fundamentales.
5. Backups y Recuperación:
    - ¿Cuáles son las mejores prácticas para realizar copias de seguridad (backups) en MySQL? ¿Cómo restaurarías una base de datos desde un backup en caso de pérdida de datos? Considera la seguridad y la consistencia de los datos.

Preguntas para evaluar conocimiento técnico MongoDB

1. Índices en MongoDB:
    - ¿Puedes explicar la importancia de los índices en MongoDB y cómo se crean? Proporciona un ejemplo de un escenario en el que agregar un índice podría mejorar el rendimiento de una consulta.
2. Modelado de Datos:

- Describe las diferencias entre el modelado de datos en MongoDB y en una base de datos relacional. ¿En qué situaciones preferirías utilizar un modelo embebido frente a un modelo de referencia?

3. Consulta y Agregación:
   - Explica la diferencia entre una consulta y una operación de agregación en MongoDB. ¿En qué casos utilizarías la agregación en lugar de una consulta estándar?
4. Transacciones:
   - MongoDB introdujo el soporte para transacciones en versiones más recientes. ¿Cuándo y por qué considerarías el uso de transacciones en MongoDB? Proporciona un ejemplo de su aplicación.
5. Sharding:
   - ¿Qué es el sharding en MongoDB y cuándo sería apropiado implementarlo? Proporciona un escenario en el que el sharding podría ser beneficioso y describe cómo configurar y administrar un conjunto de clústeres con sharding.

Preguntas para evaluar conocimiento técnico Express y Node (Semi Sr / Sr)

1. Middleware en Express:
   - Explique el concepto de middleware en Express.js. ¿Puede proporcionar un ejemplo de cómo usar y crear middleware personalizado en una aplicación Express?
2. Manejo de Rutas en Express:
   - En Express, ¿cómo se manejan los parámetros de ruta y los parámetros de consulta? Proporcione un ejemplo práctico que demuestre su conocimiento en el manejo de rutas.
3. Node.js y Event Loop:
   - Describa el concepto de "event loop" en Node.js. ¿Cómo ayuda el modelo de I/O no bloqueante de Node.js a manejar muchas conexiones simultáneas de manera eficiente?
4. Gestión de Dependencias:
   - ¿Cómo gestionaría las dependencias en una aplicación Node.js? Mencione alguna herramienta o práctica común utilizada en la gestión de paquetes y dependencias.
5. API RESTfull con Express:
   - ¿Cuál es la diferencia entre GET y POST en una solicitud HTTP? Proporcione un ejemplo de cómo diseñaría una API RESTful simple utilizando Express y explique cómo manejar las solicitudes GET y POST en esta API.

     api/v1/users

     api/v1/users/1

     api/v1/users/1/stores

     api/v1/stores/1/users

Preguntas para evaluar conocimiento técnico React y Next (Semi Sr / Sr)

1. ºComponentes y Páginas en React y Next.js:
    - Explique el concepto de componentes en React. ¿Cuál es la relación entre componentes y páginas en Next.js? Proporcione un ejemplo práctico de cómo se estructurarían componentes y páginas en una aplicación Next.js.
2. Estado y Ciclo de Vida en Componentes con Next.js:
    - ¿Cómo funciona el estado en React y cómo se gestiona en componentes de páginas en Next.js? Describa el ciclo de vida de una página en Next.js y mencione al menos tres eventos del ciclo de vida y sus usos.
3. Gestión del Estado y Contexto con Next.js:
    - Explique las diferencias entre el estado local y el estado global en React y cómo se gestionan en una aplicación Next.js. ¿Cómo utilizaría el contexto en Next.js para compartir datos entre componentes?
4. React Hooks y Hooks Específicos de Next.js:
    - Describa el propósito de los hooks en React y proporcione ejemplos de al menos tres hooks incorporados. ¿Cómo utilizaría los hooks específicos de Next.js, como useRouter, para facilitar la navegación y el manejo de parámetros de ruta?
5. Routing y Enrutamiento con Next.js:
    - ¿Cómo implementaría la navegación entre diferentes vistas en una aplicación React con Next.js? Mencione las características específicas de enrutamiento que ofrece Next.js y cómo beneficiarían al desarrollo de una aplicación.

Ejercicios técnico

1. Consultas MongoDB

   Supongamos que tienes una colección llamada "usuarios" con documentos que tienen la siguiente estructura:

   ```
   {
     "_id": ObjectId("5f8a72e8b4f069001e0980c9"),
     "nombre": "Juan",
     "edad": 28,
     "correo": "juan@example.com",
     "activo": true
   }
   ```

   Actualización para cambiar el valor de "activo" a false para todos los usuarios mayores de 30 años:

    - db.usuarios.update({edad: {$gte: 30}}, {$set: {activo: false}})
    - db.usuarios.updateMany({edad: {$lt: 30}}, {$set: {activo: false}})
    - db.usuarios.updateMany({edad: {$gt: 30}}, {$set: {activo: false}})
    - db.usuarios.update({edad: {$lte: 30}}, {$set: {activo: false}})

2. Consultas MySql

Supongamos que tienes una base de datos MySQL con la siguiente estructura:

```
CREATE TABLE empleados (
 id INT PRIMARY KEY,
 nombre VARCHAR(255),
 edad INT,
 salario DECIMAL(10, 2)
);

CREATE TABLE asignaciones (
 id INT PRIMARY KEY,
 id_empleado INT,
 horas_trabajadas INT,
 FOREIGN KEY (id_empleado) REFERENCES empleados(id)
);

INSERT INTO empleados VALUES (1, 'Juan', 30, 5000.00);
INSERT INTO empleados VALUES (2, 'Ana', 25, 4500.00);
INSERT INTO asignaciones VALUES (1, 1, 40);
INSERT INTO asignaciones VALUES (2, 1, 30);
INSERT INTO asignaciones VALUES (3, 2, 35);
```

a) Escribe una consulta SQL para obtener el nombre y la edad de todos los empleados.

- SELECT id, nombre, edad FROM empleados;
- SELECT * FROM empleados;
- SELECT nombre, edad FROM empleados;
- SELECT nombre, salario FROM empleados;

b) Escribe una consulta SQL para calcular la cantidad total de horas trabajadas por todos los empleados.

- SELECT SUM(horas_trabajadas) FROM empleados;
- SELECT SUM(horas_trabajadas) FROM asignaciones;
- SELECT AVG(horas_trabajadas) FROM asignaciones;
- SELECT COUNT(*) FROM asignaciones;

3. Operaciones de Agregación y JOIN

Supongamos que tienes una base de datos MySQL con la siguiente estructura:

```
CREATE TABLE vehiculos (
 id INT PRIMARY KEY,
 dominio VARCHAR(10),
 marca VARCHAR(50),
 modelo VARCHAR(50),
 kilometraje INT
);

CREATE TABLE asignaciones (
 id INT PRIMARY KEY,
```

```
 id_empleado INT,
 id_vehiculo INT,
 horas_trabajadas INT,
 FOREIGN KEY (id_empleado) REFERENCES empleados(id),
 FOREIGN KEY (id_vehiculo) REFERENCES vehiculos(id)
);

INSERT INTO vehiculos VALUES (1, 'ABC123', 'Toyota', 'Camry', 50000);
INSERT INTO vehiculos VALUES (2, 'XYZ987', 'Honda', 'Civic', 60000);
INSERT INTO asignaciones VALUES (1, 1, 1, 40);
INSERT INTO asignaciones VALUES (2, 1, 2, 30);
INSERT INTO asignaciones VALUES (3, 2, 1, 35);
```

a) Escribe una consulta SQL para obtener la marca y el modelo de los vehículos asignados a un empleado específico (por ejemplo, Juan).

- SELECT marca, modelo FROM vehiculos INNER JOIN asignaciones ON vehiculos.id = asignaciones.id_vehiculo WHERE asignaciones.id_empleado = 1;
- SELECT marca, modelo FROM vehiculos LEFT JOIN asignaciones ON vehiculos.id = asignaciones.id_vehiculo WHERE asignaciones.id_empleado = 1;
- SELECT marca, modelo FROM vehiculos INNER JOIN asignaciones ON vehiculos.id = asignaciones.id_vehiculo WHERE asignaciones.id_empleado = 2;
- SELECT marca, modelo FROM vehiculos LEFT JOIN asignaciones ON vehiculos.id = asignaciones.id_vehiculo WHERE asignaciones.id_empleado = 2;

b) Escribe una consulta SQL para calcular el kilometraje total de todos los vehículos asignados.

- SELECT COUNT(*) FROM vehiculos;
- SELECT SUM(kilometraje) FROM asignaciones;
- SELECT AVG(kilometraje) FROM vehiculos;
- SELECT SUM(kilometraje) FROM vehiculos;

4. Express

Supongamos que tienes una aplicación Express.js con la siguiente estructura básica:

```
const express = require('express');
const app = express();

// Middleware 1
app.use((req, res, next) => {
 console.log('Middleware 1');
 next();
});

// Ruta 1
```

```
app.get('/ruta1', (req, res) => {
 console.log('Ruta 1');
 res.send('Respuesta desde Ruta 1');
});

// Middleware 2
app.use((req, res, next) => {
 console.log('Middleware 2');
 next();
});

// Ruta 2
app.get('/ruta2', (req, res) => {
 console.log('Ruta 2');
 res.send('Respuesta desde Ruta 2');
});

app.listen(3000, () => {
 console.log('Servidor escuchando en el puerto 3000');
});
```

a) ¿Cuál será el orden de los mensajes en la consola si se realiza una solicitud a la ruta '/ruta1'?

- Middleware 1, Ruta 1
- Ruta 1, Middleware 1
- Middleware 1, Middleware 2, Ruta 1
- Middleware 1, Ruta 1, Middleware 2

b) ¿Qué sucede si la función next() se omite en el Middleware 1?

- La aplicación funcionará correctamente.
- Se producirá un error indicando que la ruta no se puede encontrar.
- El Middleware 2 no se ejecutará.
- La aplicación se quedará en un estado de espera.

5. Node

Supongamos que tienes el siguiente código en Node.js utilizando el módulo 'fs' para leer un archivo:

```
const fs = require('fs');

console.log('Inicio de la lectura del archivo.');

fs.readFile('archivo.txt', 'utf8', (err, data) => {
 if (err) {
 console.error('Error al leer el archivo.');
 return;
 }
 console.log('Contenido del archivo:', data);
```

```
});

console.log('Fin de la lectura del archivo.');
```

a) ¿Cuál será el orden de los mensajes en la consola al ejecutar este código?

- Inicio de la lectura del archivo, Fin de la lectura del archivo, Contenido del archivo: [contenido]
- Inicio de la lectura del archivo, Contenido del archivo: [contenido], Fin de la lectura del archivo
- Contenido del archivo: [contenido], Inicio de la lectura del archivo, Fin de la lectura del archivo
- Fin de la lectura del archivo, Inicio de la lectura del archivo, Contenido del archivo: [contenido]

b) ¿Por qué se utiliza una función de retorno de llamada (callback) en lugar de una llamada a readFileSync?

- Mejora la legibilidad del código.
- Evita bloquear el hilo principal mientras se espera la lectura del archivo.
- No hay diferencia, ambas opciones son válidas.
- La llamada readFileSync es preferible para mejorar la concurrencia.

6. React

Supongamos que tienes el siguiente componente en React:

```
import React from 'react';

const UserProfile = ({ name, age, email }) => {
 return (
 <div>
 <h2>{name}</h2>
 <p>{`Edad: ${age}`}</p>
 <p>{`Correo: ${email}`}</p>
 </div>
 );
};

export default UserProfile;
```

a) ¿Cómo pasarías las propiedades (props) al componente UserProfile desde el componente padre?

- <UserProfile name="John" age={25} email="john@example.com" />
- <UserProfile {name="John" age={25} email="john@example.com"} />
- <UserProfile props={{name: "John", age: 25, email: "john@example.com"}} />
- <UserProfile [name="John", age={25}, email="john@example.com"] />

b) Supongamos que este componente se utiliza en una aplicación React con Next.js. ¿Cómo podrías cargar los datos del usuario durante la fase de servidor (SSR) antes de renderizar la página?

- Utilizando useEffect en el componente para hacer una solicitud HTTP al servidor.

- Utilizando getServerSideProps en la página que incluye el componente UserProfile.
- Cargando los datos directamente en el componente mediante una función síncrona.
- No es posible realizar carga de datos durante la fase de servidor en React con Next.js.

7. Next

Supongamos que estás desarrollando una aplicación con Next.js y tienes la siguiente estructura de archivos:

```
/pages
 /usuarios
 [id].js
```

El archivo [id].js contiene el siguiente código:

```
import React from 'react';
import { useRouter } from 'next/router';

const UsuarioDetalle = () => {
 const router = useRouter();
 const { id } = router.query;

 return (
 <div>
 <h2>Detalles del Usuario</h2>
 <p>ID del Usuario: {id}</p>
 </div>
 );
};

export default UsuarioDetalle;
```

a) ¿Qué tipo de página representa [id].js en Next.js?

- Página estática.
- Página dinámica con ruta personalizada.
- Página dinámica con parámetros en la ruta.
- Página de error personalizada.

b) ¿Cómo se podría acceder a la página [id].js con el ID del usuario en la URL?

- usuarios/1
- usuarios?id=1
- usuarios/[1]
- usuarios?id=[1]

Ejercicio práctico:

Realizar un sistema de gestión para una remisería.

La misma cuenta con una flota de vehículos (dominio, marca, modelo, kilometraje). Los mismos pueden estar disponibles para trabajar o en el taller.

Asimismo, la remisería registra el precio por kilometro a pagar a sus empleados por mes (es el mismo para todos los empleados, cambia todos los meses).

A su vez, los empleados (nombre, apellido, dni, tipo de licencia) pueden manejar uno o más vehículos. Las licencias particulares vencen cada 5 años y las profesionales son anuales.

Debe considerar que si tiene la licencia vencida el chofer no estará disponible para manejar.

El sistema debe registrar la cantidad de kilómetros recorridos de cada chofer por mes y el kilometraje del vehículo.

El sistema debe mostrar el sueldo a liquidar por mes de cada empleado.

Por último, los vehículos cada 15000km deben realizar el servicio y concurrir al taller.

Mostrar listado de:

Kilómetros por vehículo/chofer mensual.
Kilómetros por vehículo mensual.
Ranking de choferes más rentables por mes
Choferes con licencia vencida.
Vehículos en taller.

Exigencias de desarrollo y documentación:

1. Realizar Front-end con la siguiente tecnología:
    - HTML, CSS (Tailwindcss), JavaScript, DOM, etc.
    - Frameworks: React, Next.js
    - Typescript
2. Realizar Back-end con la siguiente tecnología
    - Node, Typescript.
    - Framework: Express.
    - Estándar de desarrollo REST.
3. Base de datos: MySql/Postgres.
4. Incluya las pruebas que considere.
5. Utilice la tecnología y metodología de versionado que crea más conveniente.
6. Incluir archivo MarkDown, Readme.md, en donde se solicita la documentación del proyecto.