**Fullstack Developer Assessment**

At **Sedibelo Technologies**, innovation meets excellence. Our dedicated team of software engineers is passionate about leveraging cutting-edge technology to craft bespoke solutions tailored to meet our clients' unique needs. From concept to deployment, we're committed to delivering top-tier software solutions that drive success in business and mining technology.

**About the Role:** We are seeking a skilled **Fullstack Developer** with expertise in both **front-end** and **back-end** technologies. The ideal candidate will have a passion for creating intuitive, efficient, and scalable applications that power critical operations, particularly within the mining industry. Working as part of a team of innovative creators, the candidate should possess a strong balance of technical expertise, curiosity, and a problem-solving mindset. This role offers the opportunity to work across all layers of technology infrastructure and create impactful business solutions.

Please complete the test below to advance to the next step of the recruitment process.

Tech Stack: Node.js, Vue.js (Bonus: Quasar), NoSQL Database (MongoDB/ArangoDB/IndexDB)

This test is designed to assess your proficiency in developing complex full-stack applications with an emphasis on architecture, performance, and scalability. Bonus points will be awarded for using Quasar for the front end and implementing security best practices.

---

**Task 1: Backend Simulation, Data Handling, and Security with Node.js**

**Step 1**: Simulate Secure Login Form Submission

Create a Node.js script to simulate posting the following form data securely:

```html

<form action="https://challenge.sedilink.co.za:12022" method="post">

<input type="text" name="username" id="username"/><br>

<input type="password" id="password" name="password"><br>

<input type="hidden" name="action" id="action" value="LOGIN">

<button type="submit">Submit</button>

</form>

```

- Use HTTPS to ensure the form submission is secure. Implement additional layers of security (e.g., CSRF protection) and explain your approach.

- On successful login, you should receive a JSON response with your session token and a list of users.

- Implement rate-limiting to prevent brute-force attacks.

Store the users array in `users.json`. Ensure this file is encrypted using appropriate encryption libraries.

**Step 2**: Data Handling, Deduplication, and Performance Optimization

- Using ` users.json `, create `uniqueUsers.json`, ensuring no duplicate entries and adding a UUID for each user. Optimize the process for large datasets (e.g., millions of entries).

- Implement file streaming techniques to minimize memory usage when handling large JSON files.

- Create a CSV file with the following columns:

- Name, Surname, and Number of Times Duplicated.

- Optimize the generation of the CSV file to handle millions of entries efficiently, explaining your performance considerations.

**Step 3**: Simulate Secure User Data Posting

Simulate secure posting of users from `uniqueUsers.json` to the URL `https://challenge.sedilink.co.za:12022`, ensuring the following fields are included:

```json

{

"name": "the relevant name",

"surname": "the surname",

"designation": "the designation",

"department": "the department",

"id": "the unique id you created",

"token": "the token you received when logging in"

}

```

- Implement retries and error handling for failed requests, ensuring idempotency.

**Step 4**: Engineering Department Reporting

From `uniqueUsers.json`, identify users in the Engineering department where Mechanics and Mechanic Assistants report to Michael Phalane. Write code to determine the total number of people reporting to Michael Phalane.

- Ensure your solution can scale to handle queries across a large dataset and explain the efficiency of your query mechanism.

- Use MongoDB or ArangoDB to optimize this query and demonstrate your understanding of NoSQL database query optimization.

**Step 4:** WebSocket Client (Bonus)

1. Create a WebSocket client that connects to `wss://challenge.sedilink.co.za:3006`.

2. After successfully connecting, send a string through the WebSocket.

3. The WebSocket server will return the string in reversed form.

4. Write code to verify that the returned string matches the reversed version of the string you originally sent.

**Task 2: Advanced Data Visualization and Performance Optimization with Vue.js/Quasar**

**Step 1**: Dynamic Pie Chart with Real-Time Updates

Create a web page using Vue.js (bonus if using Quasar) to display a pie chart based on data from `pieChart.json`.

- Ensure the chart is real-time and updates automatically if the data changes, simulating real-time data fetching via websockets.

- Implement responsive design and ensure the chart renders optimally on various devices and screen sizes.

**Step 2**: Dynamic Data Table with Advanced Filtering

Develop a dynamic data table that:

- Utilizes data from `users.json` and supports pagination for efficient handling of large datasets (10,000+ rows).

- Add a global search and filtering system that allows users to filter by multiple columns (e.g., name, designation, department).

- Integrate client-side caching and lazy-loading for improved performance with large datasets.

- Provide a dropdown menu with designations from `users.json`. Selecting a designation should filter the table by the corresponding users.

---

**Task 3**: Secure and Efficient CRUD Operations with IndexedDB

Implement secure CRUD operations with IndexedDB:

- Develop a mechanism to encrypt the data stored in IndexedDB to ensure sensitive information is secure.

---

**Task 4**: RESTful API Development and Architectural Considerations (Node.js)

Design and implement a RESTful API using Node.js that integrates with the above tasks. The API should:

- Have endpoints for:

- /uniqueUsers: Display the unique users created in Task 1.

- /orderedUsers: Display users ordered by department and designation, created in Task 3.

- /addUser: Add a new user to the unique users file.

- /updateUser: Update an existing user.

---

Submission Instructions:

- Ensure your code is well-structured, documented, and follows industry best practices for security, performance, and scalability.

- Provide detailed explanations of the architectural and performance decisions you've made, particularly around security, scalability, and optimization.

- Submit your solution along with instructions on how to set up and run the application.