**Technical Test**

**Q1: Tracing and Testing in Node.js**

**Ans:** Essentially, tracing in Node.js involves observing and logging the program flow of execution. It is essential for diagnosing performance problems and debugging issues. It is important to let developers know how different parts of an application interact and to pinpoint where bottlenecks or errors have occurred.  
  
**Importance**

• Diagnosing Performance Issues: Pinpoints slow parts of the application  
• Debugging: Details the flow of applications, which eases the operation of noticing bugs.  
• Monitoring: Due to the continuous tracing, the health of the application is maintained for a longer period.  
  
**Tools and Techniques:**

Node.js Built-in Tracing: By running `node --trace-events-enabled`, tracing is enabled.  
  
Example:  
Given in examples.js file.  
  
**Test Pyramid:**  
  
The test pyramid is a framework representing different levels of testing. It emphasizes having more low-level unit tests than high-level end-to-end tests.  
  
Levels:  
1. Unit Tests: These constitute the base of the pyramid and test individual functions or modules.  
2. Integration Tests: These tests ensure that different modules or services work together correctly.  
3. End-To-End Tests: These lie at the top of the pyramid and test the complete workflow of an application.  
  
Balanced Approach  
A balanced approach towards testing ensures a well-covered and efficient one. A huge amount of end-to-end tests may be slow and hence fragile. On the other hand, it is also risky to rely too heavily on unit-tests because that may allow integration problems to slip through the net.  
Benefits  
Quality: High quality in an application is achieved when thorough testing is done at all levels.  
Maintainability: Reduces hesitation when it comes to refactoring and working on improvements in the application since there is no fear of breaking features that have already been written.  
  
Example:

Given in examples.js file.  
  
**Q2: Node JS is Event Driven With Scalability**

a) Clusters and Worker Threads   
  
**Clusters:**  
The clusters feature of NodeJS allows for the generation of numerous NodeJS processes, all of which may be utilized in sharing the same server port. This, on its own, is quite beneficial in serving multiple requests concurrently.  
  
**Worker Threads:**  
Worker threads allow one to execute JavaScript in parallel threads. They are especially helpful for doing CPU-intensive tasks without blocking the main event loop.  
  
Use Cases:   
  
Clusters: Useful for web servers dealing with many concurrent connections.  
Worker Threads: Suitable for tasks like image processing or complex calculations.  
  
Example:

Given in examples.js file.  
  
b. How to Handle High Volume of Requests

**Event Loop and Non-Blocking I/O:**  
The event loop makes Node.js efficient in handling a large number of concurrent connections. Non-blocking I/O keeps the server responsive.  
  
**Strategies:**  
Load Balancing: By distributing requests among a number of processes or servers.  
Caching: Storing the results in memory using Redis so that the same data need not be queried soon again from the database.  
Rate Limiting: Preventing abuse by capping the number of requests a client can make.  
  
Example:

Given in examples.js file.  
  
Asynchronous Functions:  
Nonblocking functions are those that enable other operations while the functions are executing. Examples include reading files or issuing HTTP requests.  
  
Synchronous Functions:  
These are blocking functions; that is, they stop the execution of a program until they are completed. Examples include the synchronous reading of files or computation of complex calculations.  
  
Performance Impact:  
Asynchronous functions improve an application's responsiveness and scalability by not making other operations wait for the function to complete.  
  
Example:

Given in examples.js file.  
  
3. Discussion:  
- Pros: No blocking main thread for tasks, so CPU-intensive tasks are better.  
- Cons: Thread creation is expensive; thread handling is highly complex.  
  
1. WebSocket Server:

Example in File.  
  
2.Strategies for High Concurrent Connections :  
Load balancing easily occurs through tools like Nginx.  
• Efficient Memory Management: Memory needs to be tracked and addressed to avoid memory leaks.  
Scalable Architecture: This clustering concept is going to be useful in dealing with multiple connections.  
  
3. Caching:  
Implementation: I am going to implement it with the help of Redis or Memcached, which can store accessed data which is much more frequent in nature.  
  
3. **Why Unit Testing?**  
- Reliability: Ensures that the small atomic pieces work correctly.  
- Bug Detection: Catching bugs earlier than later in the development process.  
- Making Refactoring Easier: Will make refactoring easier and confident.  
1. Roman to Integer Conversion.  
Example in file.