Assignment 3.1
Introduction to Node.js

Deliverables 1

- Node.js Cheat Sheet

(1.) Event - Driven Architecture

- Node.js operates on an event - driven architecture, where actions or events trigger responses.
- Core to its design is the event loop, which handles asynchro- - nous operations efficiently.

(2.) Asynchronous Programming:

- Node.js uses non-blocking, asynchronous programming to handle multiple requests concurrently.
- This allows for scalable and efficient handling of I/O operations, crucial for web applications.

(3.) npm (Node Package Manager):

- npm is the default package manager for Node.js, providing access to a vast ecosystem of open-source libraries.
- It simplifies dependency management and package installation, enhancing development productivity.

(4.) Modules:

- Node.js follows a modular approach, where functionalities are encapsulated into reusable modules.

- Modules promote code organization, maintainability and reusability, fostering a modular architecture.

5. Common JS Module System:

- Node.js implements the Common JS module system for module loading and dependency management.
- Modules use 'require()' to import dependencies and 'module.exports' to export functionalities.

6. Built-in Modules:

- Node.js provides a set of built-in modules for common tasks like file system operations ('fs'), HTTP server creation ('http'), and more.
- These modules streamline development and reduce reliance on external dependencies.

7. Package.json:

- package.json is a metadata file for Node.js projects, containing project details, dependencies and scripts.
- It serves as a central configuration file, facilitating project setup, version management, and script execution.

8. Callback functions:

- Callback functions are fundamental in Node.js for handling asynchronous operations.
- They are passed as arguments to asynchronous functions and executed upon completion, ensuring non-blocking behaviour.

9. Event Emitters:

- Node.js utilizes event emitters to implement the observer pattern for handling events.
- Custom events can be defined and emitted, allowing for decoupled and scalable event-driven architectures.

10. Streams:

- Streams are used for handling large amount of data efficiently with minimal memory footprint.
- They enable reading from or writing to data sources incrementally, enhancing performance and scalability.

11. Error Handling:

- Node.js emphasizes error-first callback conventions for handling errors in asynchronous operations.

- Errors are typically propagated using callback functions or event emitters, promoting robust error handling practices.

12. Concurrency and Scalability:
- Node.js excels in building highly concurrent and scalable applications due to its non-blocking I/O model.
- It efficiently utilizes system resources and handles large numbers of simultaneous connections with minimal overhead.

13. Middleware
- Middleware functions are widely used in Node.js frameworks like Express for request processing.
- They enable modularization of request handling logic, enhancing code organization and reusability.

14. Promise and Async / Await :
- Promises and Async / Await provide alternative approaches to asynchronous programming, offering cleaner syntax and improved error handling.
- They simplify asynchronous code by avoiding callback nesting and enabling sequential execution of

asynchronous tasks.

15. Security Considerations:

- Node.js applications should adhere to security best practices to mitigate common vulnerabilities such as injection attacks, cross-site scripting (XSS), and cross-site request forgery (CSRF).
- Proper input validation, authentication, authorization, and secure coding practices are essential for building secure Node.js applications.