**Introduction to Express:**

- Express.js is a web application framework for Node.js that simplifies the process of building web applications and APIs.

- It's known for its minimalistic and unopinionated approach, providing a basic foundation for web development.

- Express is widely used to handle HTTP requests, routing, and middleware for processing requests and responses.

- It's open-source and offers great flexibility, allowing developers to make architectural and design decisions.

**Routing in Express:**

- Routing in Express refers to the process of mapping URL paths to specific request handlers or controllers.

- Routes are defined using HTTP methods (e.g., GET, POST) and URL patterns.

- Express provides a clean and easy-to-use API for defining routes, making it simple to create RESTful APIs and web application routes.

- Routes are crucial for directing incoming requests to the appropriate code that should handle them.

**Module Introduction in Express:**

- In Express, modules are often referred to as middleware. They are reusable pieces of code that can be integrated into an application to add specific functionality.

- Middleware can be built-in Express middleware, third-party middleware packages, or custom middleware functions.

- Modules enhance the application by handling tasks such as request processing, data parsing, authentication, logging, and more.

- The modular nature of Express allows developers to extend and customise their applications by integrating various middleware modules.

**Introduction to Middleware:**

- Middleware, in the context of Express, refers to functions that process requests and responses within an application.

- Middleware functions are executed in the order they are defined in the application, and they have access to the request and response objects.

- Middleware can be used for tasks like data parsing, authentication, error handling, logging, and security enhancements.

- The order and choice of middleware functions play a significant role in controlling the request/response flow.

**Middleware Lifecycle:**

- Middleware functions in Express are executed in the order they are added to the application using `app.use()` or specific route handlers.

- The order of execution is crucial, as each middleware function can modify the request or response object and pass control to the next middleware.

- The middleware lifecycle allows for operations to be performed before or after route handlers, depending on their placement in the application.

**Serving Static Files:**

- Express allows you to serve static files, such as HTML, CSS, JavaScript, images, and more, using the `express.static` middleware.

- Serving static files enhances the performance of web applications by reducing the load on the server.

- This feature is especially useful for delivering client-side assets to the browser, making it an essential part of many web applications.

**Request Lifecycle in Express:**

- The request lifecycle in Express describes the journey of an incoming HTTP request through the application.

- It starts with the request hitting the Express server and proceeds through middleware execution, route handling, and response generation.

- Understanding the request lifecycle is essential for debugging and optimising applications.

- Express allows for customising this lifecycle by adding middleware for various processing steps.

**Blocking vs. Non-blocking Code:**

- In Express, blocking code refers to code that can cause delays or "block" the event loop, leading to slower application performance.

- Non-blocking code, on the other hand, allows the application to continue processing other tasks without waiting for a specific operation to complete.

- Express encourages non-blocking code to ensure the server remains responsive to incoming requests, improving scalability and performance.

**Body Parser:**

- In Express, the `body-parser` middleware is used for parsing request bodies, including JSON and URL-encoded data.

- It extracts data sent in the request body and makes it accessible to the application.

- This is crucial for handling form submissions, AJAX requests, and API endpoints that require data input.

- Properly parsing request bodies is essential for processing and validating incoming data.

You can use these theory concepts to create a comprehensive and informative document in Google Docs, covering the essential aspects of Express.js.