CHAPTER 1: OVERVIEW OF COMPANY

1.History

- My internship is about Designing Hardware and Programming, I am doing this internship in Hermes iot Solutions.
- Established in 2018 at GIDC electronics zone Gandhinagar.
- It's a new startup manufacturing solar grid tied inverter which are very rearin India.
- Company has its own R&D center working on their own iot platform "solarguru", to monitor and improve the efficiency of solar power generation. They are specially working on AI.
- They are leading manufacturer and exporter of Solar gird tied it infrastructure. Is Hermes iot exponentially growing company in field of innovative solar technology. We have set in place the largest of service providers nationwide, unmatched by anyone else in it field. The best of our technically qualified engineers to efficiently cater your needs of any services.

Figure 1.1 Infrastructure of company

1.1Vision

To be the most admired and responsible solar power company enabling solar everywhere with an international footprint, delivering sustainable value to all stakeholders.

1.2Mission

Ensuring cost leadership and growing profitability across all segments in the solar space in focused geographies, delivering benchmark customer experience & thereby earning customer affection.

Chapter 2. INTRODUCTION TO INTERNSHIP

2.1 INTERNSHIP SUMMARY

- This internship is about working in R & D research and development department and work on different projects. I have completed my internshipfrom Hermes Iot Solution. Internship is scheduled for 3months. The project which is given to me by the company is on issue management Station which is used to Solve issue, using different Api. In this internship I learning about how to make a whole project which include software programming.
- The primary objective of an IT company intern is to gain hands-on experience and exposure to various aspects of IT, including software development, web development, data analysis, cybersecurity, and more. They may also be involved in research, testing, and documentation of various projects, as well as providing technical support to the company's clients or end-users

2.2 OBJECTIVE

- The objective of this NestJS project is to develop a highly scalable, robust, and efficient server-side application that meets the client's requirements and adheres to industry standards. Using the modular architecture and built-in modules provided by NestJS, the goal is to create a reliable and maintainable codebase that can easily handle the anticipated traffic and user demands.

- . Additionally, the ap
- plication will be built with security in mind, ensuring that all user data is kept safe and protected. By leveraging the various features and benefits of NestJS, the aim is to deliver a high-quality server-side application that meets or exceeds the client's expectations.

2.3TOOLS & TECHNOLOGY

Here are the Tools which are used in this Internship: -

2.3.1Sublime: -

Sublime Text is a popular text editor used by developers to write and edit code for a variety of programming languages. It has a clean and intuitive user interface, which allows for efficient navigation and editing of code. Sublime Text offers a wide range of features such as syntax highlighting, auto-completion, code folding, multiple selections, and macros, which enhance productivity and save time. It supports a vast number of plugins and packages that can be easily installed to add additional functionality and customization to the editor. Sublime Text is highly customizable, with options to modify fonts, color schemes, key bindings, and other settings to suit individual preferences. It is lightweight and fast, with quick startup times and minimal system resource usage, making it a preferred choice for developers who need to work with large codebases. Sublime Text is available for Windows, macOS, and Linux, and has a portable version that can be run from a USB driveIt has an active and supportive community, with regular updates and bug fixes, and extensive documentation and tutorials available online.

2.3.2 Atom: -

Atom is a popular open-source text editor developed by GitHub, designed for developers to write and edit code for a variety of programming languages. It offers a modern and intuitive user interface with a highly customizable layout and numerous themes, making it visually appealing and easy to use. Atom offers several features such as syntax highlighting, code folding, auto-completion, find and replace, and multiple cursors, which enhance productivity and save time. It supports a vast number of packages and plugins that can be easily installed to add additional functionality and customization to the editor.

Atom has integrated Git control, allowing developers to manage their code and collaborate with others using GitHub without leaving the editor. It is highly customizable, with options to modify fonts, color schemes, key bindings, and other settings to suit individual preferences. Atom has integrated Git control, allowing developers to manage their code and collaborate with others using GitHub without leaving the editor. It is highly customizable, with options to modify fonts, color schemes, key bindings, and other settings to suit individual preferences. Atom is lightweight and fast, with quick startup times and minimal system resource usage, making it a preferred choice for developers who need to work with large codebases.

2.3.3Code Lite: -

CodeLite is an open-source, cross-platform IDE (Integrated Development Environment) designed for developing C, C++, PHP, and Node.js applications. It offers a modern and intuitive user interface, with customizable layouts and numerous themes, making it visually appealing and easy to useCodeLite offers several features such as syntax highlighting, code folding, auto-completion, debugging, refactoring, and project management, which enhance productivity and save time.It supports a wide range of plugins and packages that can be easily installed to add additional functionality and customization to the IDE.CodeLite has integrated support for Git, allowing developers to manage their code and collaborate with others using

Git without leaving the IDE.It is highly customizable, with options to modify fonts, color schemes, key bindings, and other settings to suit individual preferences.CodeLite is lightweight and fast, with quick startup times and minimal system resource usage, making it a preferred choice for developers who need to work with large codebases.CodeLite is available for Windows, macOS, and Linux, and has a portable version that can be run from a USB drive.

2.3.3 SNIPPETS

JavaScript snippets are small pieces of code that can be executed in the Chrome browser's developer console.where you can write and save your code.You can also run JavaScript snippets from saved files on your local machine by dragging and dropping them into the developer console.JavaScript snippets can be very useful for quickly testing and debugging small pieces of code or automating repetitive tasks.Some common use cases for JavaScript snippets in Chrome include modifying the behavior of a website, extracting data from a webpage, and testing API endpoints.Chrome also allows you to add third-party libraries like jQuery to your snippets, which can help simplify your code and make it more readable.It's important to be careful when using JavaScript snippets, as they can potentially harm your system or compromise your privacy if they are written maliciously or without proper understanding.Finally, there are many online resources available that provide JavaScript snippets for common tasks or functions. These can be a great starting point for learning how to use JavaScript in Chrome.To access the developer console in Chrome, you can either right-click on a webpage and select "Inspect" or press the F12 key on your keyboard.

2.3.5 VSTUDIO

Visual Studio is a comprehensive integrated development environment (IDE) developed by Microsoft for building software applications.It supports multiple programming languages including C++, C#, Visual Basic, .NET Framework, TypeScript, JavaScript, and many more.Visual Studio includes a range of tools and features that make it easier for developers to build, test, and debug their code.One of the main features of Visual Studio is its IntelliSense technology, which provides suggestions and auto-completion for code as you type.Visual Studio also supports version control systems like Git and provides seamless integration with popular code repositories like GitHub

The IDE also includes built-in debugging tools that allow developers to quickly identify and fix issues in their code.Visual Studio has a range of project templates that allow developers to quickly start new projects and provides a wide range of tools to help with building applications, including visual designers for creating user interfaces. Visual Studio also includes a range of extensions and add-ons that can be downloaded from the Visual Studio Marketplace, providing additional functionality and integration with other tools.

2.3.6 Technology:

2.3.6.1 C Language

C language is a general-purpose computer programming language. It was created in the 1970s by Dennis Ritchie and Bell Labs, and remainsvery widely used and influential. By design, C's features cleanly reflectthe capabilities of the targeted CPUs. It has found lasting use in operating systems, device drivers, protocol stacks, though decreasingly for application software, and is common in computer architectures that range from the largest supercomputers to the smallestmicrocontrollers and embedded systems.

C is an imperative procedural language supporting structured programming, lexical variable scope, and recursion, with a static typesystem. It was designed to be compiled to provide low-level access tomemory and language constructs that map efficiently to machine instructions, all with minimal runtime support. Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant C program written with portability in mind can be compiled for a wide variety of computer platforms and operating systems with few changes to its source code.

2.3.6.2Html

HTML is a markup language that is used to define the structure and content of a web page.HTML code consists of a series of elements, which are enclosed in tags and can be nested inside one another to create complex structures.HTML uses a hierarchical structure to organize content, with the main content of the page enclosed within the <body> tag.HTML provides a wide range of elements for creating different types of content, including headings, paragraphs, lists, tables, forms, images, videos, and more.HTML is often combined with CSS (Cascading Style Sheets) and JavaScript to create rich, interactive web pages and applications.HTML is a constantly evolving language, with new features and updates being added regularly by the World Wide Web Consortium (W3C).HTML is a fundamental skill for anyone interested in web development, and is often one of the first languages that developers learn when getting started in the field.

2.3.6.3 CSS

CSS (Cascading Style Sheets) is a style sheet language used to describe the presentation and layout of HTML (Hypertext Markup Language) and XML (Extensible Markup Language) documents, including web pages and web applications. It is used to add style, layout, and visual effects to web pages, and works in conjunction with HTML and JavaScript to create engaging user interfaces.CSS is used to separate the presentation of a web page from its content, making it easier to manage and update the style of a website.CSS provides a wide range of style rules, including font size, color, layout, positioning, and animation.CSS uses selectors to target

specific HTML elements and apply style rules to them.CSS has a hierarchical structure, with styles cascading down from parent elements to child elements, and from more general styles to more specific ones.CSS allows developers to define reusable styles using classes and IDs, making it easier to maintain consistency across a website.CSS can be embedded directly into an HTML document using the <style> tag, or it can be linked to an external stylesheet using the <link> tag.

CSS supports media queries, which allow developers to apply different styles to a website based on the size of the screen or device it is being viewed on.CSS is constantly evolving, with new features and updates being added regularly by the World Wide Web Consortium (W3C).Overall, CSS is a powerful and essential tool for web developers, allowing them to create engaging and visually appealing web pages and applications.

2.3.6.4 JAVA SCRIPT

JavaScript is a high-level, dynamic, and interpreted programming language used primarily to create interactive front-end web applications. It is often used in conjunction with HTML and CSS to create dynamic and interactive web pages and web applications.JavaScript is a client-side programming language, meaning that it runs in the browser rather than on the server.It is a scripting language, which means it is interpreted line-by-line and doesn't require compilation before execution.JavaScript is a dynamically typed language, meaning that variable types are determined at runtime rather than during compilation.JavaScript provides a wide range of built-in objects and functions, including arrays, strings, numbers, and dates, that can be used to perform complex operations on data.JavaScript supports event-driven programming, which means that code is executed in response to user actions such as clicks and keyboard inputs.JavaScript is often used in conjunction with libraries and frameworks such as jQuery, React, and AngularJS, which provide additional functionality and simplify the development process.JavaScript has a growing presence on the server-side as well, with the development of server-side frameworks such as Node.js.JavaScript is an essential skill for web developers, and is widely used in the development of web applications and interactive user interfaces.

2.3.6.5 NODE JS

Node.js is an open-source, cross-platform, server-side JavaScript runtime environment that allows developers to build scalable, high-performance web applications using JavaScript. It provides a platform for building network applications that can handle a large number of concurrent connections with high throughput and low latency.Here are a few key points about Node.js:

1. Node.js is built on the V8 JavaScript engine, which is the same engine used by the Google Chrome web browser.
2. It uses an event-driven, non-blocking I/O model, which allows it to handle a large number of connections with low overhead and high throughput.
3. Node.js provides a rich set of built-in modules and libraries, including the HTTP module for building web servers, the File System module for working with files and directories, and the Express.js framework for building web applications.
4. Node.js allows developers to write server-side code in JavaScript, which makes it easier to build full-stack applications using a single language.

5. Node.js is highly scalable and can be used to build high-performance

applications that can handle a large number of concurrent connections.

6. Node.js has a large and active community of developers and contributors, which ensures that it is constantly evolving and improving.

7. Node.js is widely used by companies such as Netflix, LinkedIn, and PayPal for building high-performance web applications.

8. Overall, Node.js is a powerful and versatile platform for building scalable, high-performance web applications using JavaScript, and is an essential tool for full-stack web developers.

Chapter 3 Internship Planning

During the twelve weeks of my internship, as part of the R&D Electronicsdepartment I was assigned multiple projects and roles. The task which wasgiven to me was a project Weather Monitoring Station.

3.1 GIVEN TASK

At the very first week of my internship, my guide has setup a project plan for all interns. The project which was given to me was Weather monitoringStation. In which my tasks are: -

1. langugae Selection
2. Understanding languages
3. Practice and repetition
4. foundational skills Building
5. Advanced skills development

3.2 LANGUAGE SELECTED

When selecting a language for an IT project, there are several factors that should be taken into consideration Project requirements: The language you choose should be able to meet the requirements of your project. This includes the functionality you need to build, performance requirements, and compatibility with other technologies and systems

Development team skills: Consider the skill level of your development team.
Choosing a language that your team is familiar with can help speed up development and reduce the learning curve.

Community support: A language with a large and active community can provide resources and support when you encounter issues or need help. This can also ensure the language remains up-to-date and relevant.

Scalability: Consider the scalability of the language. Will it be able to handle the growth of your project? Will it be able to accommodate future requirements?Cost: The cost of licensing, tools, and resources required for development should also be considered.

Security: Security should be a key consideration when selecting a language. The language should have strong built-in security features, and it should be regularly updated to address any known security vulnerabilities.

3.3 UNDERSTANDING LANGUAGES

Understanding programming languages is a critical aspect of working on an IT project. Here are some key concepts to keep in mind:

Syntax: Programming languages have their own unique syntax, which defines how code is written and structured. Understanding the syntax of a language is essential for writing code that can be executed by a computer.

Data types: Programming languages have built-in data types, such as integers, strings, and arrays, which are used to represent different types of information. Understanding data types is important for working with data and performing calculations or operations on that data.

Control structures: Control structures are used to define the flow of a program. These include conditionals, such as if/else statements, and loops, such as for and while loops.
Functions: Functions are reusable blocks of code that perform a specific task. Understanding how to create and use functions is essential for writing efficient and modular code.
Libraries and frameworks: Many programming languages have libraries and frameworks that provide pre-built functions and tools for common tasks, such as interacting with databases or creating user interfaces. Understanding how to use these libraries and frameworks can save time and improve the quality of your code.

Debugging: Debugging is the process of identifying and fixing errors in code. Understanding how to use debugging tools and techniques is essential for troubleshooting issues and improving the quality of your code.

Version control: Version control systems are used to manage changes to code over time, allowing multiple developers to work on the same codebase without conflicts. Understanding version control is essential for working effectively in a team environment.

3.4 PRACTICE AND REPITATION

Practice and repetition are crucial for success in IT projects. Here are some ways to incorporate practice and repetition into your development process:

Code exercises: Code exercises can help you practice specific programming concepts and reinforce your understanding of syntax and data structures. There are many online resources available that offer coding challenges and exercises, such as HackerRank and LeetCode.Collaborative coding: Collaborative coding with other developers can provide an opportunity to learn from others and practice working in a team environment. Pair programming, for example, involves two developers working together on the same codebase, taking turns coding and reviewing each other's work.Code reviews: Code reviews are an important practice for improving code quality and catching errors. By reviewing and critiquing each other's code, developers can learn from one another and identify areas for improvement.

Test-driven development: Test-driven development involves writing tests for code before it is written, and then writing code that passes those tests. This can help ensure code is functional

and reduce the need for manual testing. Continuous integration and deployment: Continuous integration and deployment involve automating the build and deployment process for code changes. This can help ensure code changes are quickly and consistently incorporated into the project, and can help identify issues early on. By incorporating these practices into your development process, you can improve your skills and knowledge, and develop more efficient and effective IT projects.

3.5 FOUNDATIONAL SKILL BUILDING

Foundational skills are essential for success in IT projects. Here are some skills you can focus on building: Problem-solving: The ability to solve problems is crucial for IT projects. It involves breaking down complex problems into smaller, more manageable components and coming up with solutions that work within project constraints. Critical thinking: Critical thinking involves evaluating information, identifying biases and assumptions, and making informed decisions based on evidence. It is important for identifying potential issues and developing effective solutions. Communication: Effective communication is key for collaborating with team members, stakeholders, and clients. This includes being able to explain technical concepts to non-technical stakeholders, as well as actively listening to feedback and incorporating it into your work. Time management:

Time management is critical for meeting project deadlines and delivering high-quality work. It involves prioritizing tasks, setting realistic goals, and managing your time effectively. Attention to detail: Attention to detail is important for catching errors and ensuring code is functional and efficient. It involves carefully reviewing code and documentation, and being meticulous in your work. Continuous learning: The technology landscape is constantly evolving, so it is important to stay up-to-date with new technologies and tools. This involves regularly seeking out new learning opportunities, such as online courses, workshops, and conferences.

In addition to foundational skills, advanced skills are also important for success in IT projects. Here are some skills you can focus on developing:Object-oriented programming: Object-oriented programming (OOP) is a programming paradigm that uses objects to represent data and functionality. It is widely used in software development, and is an important skill for working on large-scale projects.Web development: Web development involves building websites and web applications using a variety of languages and frameworks, such as HTML, CSS, JavaScript, and React.

It is an important skill for developing user-facing applications.Database design and management: Databases are used to store and manage data in IT projects, so understanding how to design and manage databases is essential. This involves understanding data modeling, database normalization, and SQL.DevOps: DevOps is a set of practices that combines software development and IT operations to improve the speed and quality of software delivery. It involves automating processes, such as testing and deployment, and using tools such as Git and Docker.

Cloud computing: Cloud computing involves using remote servers to store, manage, and process data, rather than local servers or personal computers. It is an important skill for developing scalable and flexible applications. Machine learning and artificial intelligence: Machine learning and artificial intelligence (AI) are rapidly growing fields, and are being used in a variety of IT projects. Understanding how to develop and use machine learning algorithms can help you build intelligent applications that can analyze data and make predictions.

CHAPTER 4 SELECTING BASIC LANGUAGE

Choosing the right programming language for an IT project depends on various factors, such as project requirements, team expertise, scalability.

4.1 BASIC LANGUAGES

"Basic" is a family of programming languages that were designed to be easy to learn and use for beginners. Here are some key points about Basic languagesBASIC stands for "Beginner's All-purpose Symbolic Instruction Code".

It was developed in the mid-1960s by John Kemeny and Thomas Kurtz at Dartmouth College in New Hampshire.Basic languages are interpreted, rather than compiled, which means that the code is executed line by line rather than being translated into machine code before execution.

Basic languages are known for their simple syntax and easy-to-understand structure, making them popular among beginners.One of the most well-known Basic languages is Microsoft's Visual Basic, which was released in 1991 and became a popular tool for creating Windows applications.

Other popular Basic languages include QBASIC, GW-BASIC, and Small Basic.Basic languages have been used to create a wide range of applications, including games, business applications, and educational software.

Despite being designed for beginners, Basic languages can also be used to create complex programs and systems.Basic languages have had a significant impact on the history of programming, and many programmers today got their start with Basic.

HTML is a markup language used for creating web pages, while CSS is a style sheet language used for formatting and styling those web pages.HTML provides the structure and content of a web page, including text, images, and multimedia elements. CSS provides the visual design and layout of the web page, including colors, fonts, and positioning.

Both HTML and CSS are essential for creating modern, responsive websites that can be viewed on a variety of devices and screen sizes.

HTML is based on a system of tags and attributes that define the structure of a web page, while CSS uses selectors and declarations to apply styles to specific elements on the page.

4.2 HTML

Frontend web page HTML is the code used to create the visual structure and content of a website.HTML provides the building blocks of a web page, such as text, images, headings, and links.Frontend web developers use HTML to optimize the website for usability, accessibility, and search engine optimization.HTML can be styled using CSS and enhanced using JavaScript and other web development technologies.

Effective use of HTML is critical for creating engaging and user-friendly web pages.HTML is constantly evolving, with new features being added regularly to improve performance and functionality.There are many resources available for learning HTML, including online tutorials, courses, and documentation.HTML is an essential skill for any frontend web developer.

4.3 IMPLEMENTATION OF HTML

Start with a clear plan and vision for the website or web page you want to create. Define the purpose, audience, and content of the website.Use an HTML editor or integrated development environment (IDE) to write and edit your HTML code. Popular options include Visual Studio Code, Sublime Text, and Atom.

Structure your HTML code using semantic tags, such as <header>, <nav>, <main>, <section>, and <footer>, to improve accessibility, SEO, and maintainability.Use HTML attributes, such as class and id, to add styling and functionality to your HTML elements.
Include essential metadata, such as the page title, description, and keywords, in the head section of your HTML code.Optimize your HTML code for performance by minimizing unnecessary whitespace, reducing the number of HTTP requests, and using compressed or minified files.
Test your HTML code in multiple web browsers and on different devices to ensure cross-browser compatibility and responsive design.Continuously improve and update your HTML code as new technologies and best practices emerge.
Use version control tools, such as Git, to track changes and collaborate with other developers on your HTML code.Stay up-to-date with the latest HTML standards and specifications to ensure your code is future-proof and compliant with web standards.

4.4 EXAMPLE

```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
</head>
<body>
</body>
</html>
```

The <!DOCTYPE html> declaration at the beginning specifies that this is an HTML5 document.The <html> element is the root element of the HTML document, and the lang="en" attribute specifies the language of the document as English.
The <head> element contains metadata about the document, such as the character encoding and the title of the page.The <meta> elements provide information about the document. The charset="UTF-8" attribute specifies the character encoding of the document as UTF-8, which supports a wide range of characters and languages.
The http-equiv="X-UA-Compatible" attribute tells Internet Explorer to use the latest rendering engine available. The viewport meta tag sets the viewport width to the device width, which is necessary for responsive design.The <title> element sets the title of the page, which is displayed in the browser tab.
The <body> element contains the visible content of the web page, such as text, images, and multimedia elements. In this case, the body is empty, but you can add any content you want between the opening and closing body tags.

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <title>Table Example</title>
</head>
<body>
  <table>
    <thead>
      <tr>
        <th>Name</th>
        <th>Age</th>
        <th>Gender</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>John Smith</td>
        <td>30</td>
        <td>Male</td>
      </tr>
      <tr>
        <td>Jane Doe</td>
        <td>25</td>
        <td>Female</td>
      </tr>
      <tr>
        <td>Bob Johnson</td>
        <td>40</td>
        <td>Male</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

The <html> element is the root element of the HTML document, and the lang="en" attribute specifies the language of the document as English.The <head> element contains metadata about the document, such as the character encoding and the title of the page.

The <meta> element provides information about the document. The charset="UTF-8" attribute specifies the character encoding of the document as UTF-8, which supports a wide range of characters and languages.

The <body> element contains the visible content of the web page, such as text, images, and multimedia elements. In this case, it contains a single <table> element.The <table> element creates a table on the web page, with the <thead> and <tbody> elements specifying the header and body sections of the table, respectively.

The <tr> element creates a table row, and the <th> and <td> elements create table header and table data cells, respectively. In this example, the table has three columns: Name, Age, and Gender, and each row contains data for each column for each person.

<!DOCTYPE html>

```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Sign Up Form</title>
</head>
<body>
   <form>
      <label for="username">Username:</label>
      <input type="text" id="username" name="username"><br><br>

      <label for="password">Password:</label>
      <input type="password" id="password" name="password"><br><br>

      <label for="email">Email:</label>
      <input type="email" id="email" name="email"><br><br>

      <label for="birthdate">Birthdate:</label>
      <input type="date" id="birthdate" name="birthdate"><br><br>

      <input type="submit" value="Sign Up">
   </form>
</body>
</html>
```

<!DOCTYPE html>: This declares the document type and version of HTML that the document uses.<html lang="en">: This is the opening tag of the HTML document and specifies the language of the document as English.

<head>: This is the container for the metadata of the HTML document, such as the title, charset, and other tags that provide additional information about the document.<meta charset="UTF-8">: This sets the character encoding of the document to UTF-8, which supports a wide range of characters and symbols.

<title>Sign Up Form</title>: This sets the title of the document, which is displayed in the browser's title bar and used by search engines and other tools to identi<form>: This creates a form that allows users to enter data and submit it to a server for processing.<label for="username">Username:</label>: This creates a label for the username field and links it to the corresponding input field using the for attribute.
<input type="text" id="username" name="username">: This creates an input field for the username and sets its type to "text", its ID to "username", and its name to "username".fy the page.
<body>: This is the container for the visible content of the HTML document, such as text, images, and other elements.br>
: This creates line breaks between the input fields.<input type="password" id="password" name="password">: This creates an input field for the password and sets its type to "password", its ID to "password", and its name to "password".
<input type="email" id="email" name="email">: This creates an input field for the email

address and sets its type to "email", its ID to "email", and its name to "email".<input type="date" id="birthdate" name="birthdate">: This creates an input field for the birthdate and sets its type to "date", its ID to "birthdate", and its name to "birthdate".<input type="submit" value="Sign Up">: This creates a submit button that users can click to submit the form data to the server, and sets its value to "Sign Up".

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Login Page</title>
</head>
<body>
   <form>
      <label for="username">Username:</label>
      <input type="text" id="username" name="username"><br><br>

      <label for="password">Password:</label>
      <input type="password" id="password" name="password"><br><br>

      <input type="submit" value="Log In">
   </form>
</body>
</html>
```

The <!DOCTYPE html> declaration is used to specify that this is an HTML5 document.The <html> element is the root element of an HTML page.The lang="en" attribute in the <html> tag specifies the language of the document as English.

The <head> element contains metadata about the document, such as the title and character encoding.The <meta> element sets the character encoding to UTF-8, which can display all Unicode characters.The <title> element sets the title of the document, which appears in the browser's title bar or tab.The <body> element contains the visible content of the page.The <form> element is used to create a form for user input.

The for attribute in the <label> elements specifies which form element the label is associated with.The type attribute in the <input> elements specifies the type of form element, such as "text" or "password".The id attribute in the <input> elements specifies a unique identifier for the form element.The name attribute in the <input> elements specifies a name for the form element, which is used when submitting the form data to the server.

The
 element creates a line break between form elements.The value attribute in the <input> element specifies the default value for the form element.The <input type="submit"> element creates a submit button for the form, which submits the form data to the server when clicked.

4.5 CSS

CSS stands for Cascading Style Sheets, and it is a language used to style HTML and XML documents.CSS allows you to control the appearance of web pages, including the layout, colors, fonts, and more.

CSS uses a variety of selectors to target specific elements on a page, such as HTML tags, classes, and IDs.You can use CSS to apply styles to multiple elements at once, reducing the amount of code needed and making maintenance easier.

CSS also provides a way to create responsive designs that adjust to different screen sizes, making your website more accessible across devices.CSS has a wide range of properties and values that you can use to style your web pages, including background color, font size, margin, padding, and more.

You can include CSS in your HTML documents using either internal or external stylesheets, depending on your needs.CSS also provides a way to create animations and transitions, allowing you to add dynamic effects to your web pages.CSS frameworks like Bootstrap and Foundation provide pre-built CSS styles and components that can help you create responsive, modern-looking web designs more quickly.CSS is constantly evolving, with new features and capabilities being added all the time to make web design more flexible and powerful.

4.6 IMPLEMENTATION OF CSS

CSS can be implemented in an HTML document using the <style> tag, which is placed within the <head> section of the document.Another way to implement CSS is by using an external stylesheet. This involves creating a separate CSS file with a .css extension and linking to it from the HTML document using the <link> tag.

CSS can also be implemented inline, which means adding the style directly to the HTML element using the "style" attribute. However, this is generally not recommended as it can make the HTML code difficult to read and maintain.
Selectors are used to target specific HTML elements to apply the desired styles. These can include element selectors, class selectors, ID selectors, and more.CSS properties are used to define the styles that are applied to the targeted elements. These can include properties such as color, font-size, background-color, and more.
Cascading is an important feature of CSS, which determines the order in which styles are applied to an element. Styles defined in an external stylesheet, for example, will override styles defined inline.

CSS frameworks such as Bootstrap and Foundation can be used to simplify and speed up the process of creating a website, by providing pre-built styles and components that can be easily customized to fit the specific needs of a project.

4.7 EXAMPLES

```
selector {
   margin: value;
   padding: value;
}
```

where selector is the element or group of elements you want to apply the styling to, and value is the size of the margin or padding you want to apply. You can specify the size in different units such as pixels.
For example, to set a margin of 20 pixels and a padding of 10 pixels to all paragraphs in your

```
p {
     margin: 20px;
     padding: 10px;
        }
```

The margin property sets the space around the <p> element. In this code, it sets 20 pixels of margin on all sides (top, right, bottom, and left).The padding property sets the space between the <p> element's content and its border. In this code, it sets 10 pixels of padding on all sides.
The 20px and 10px values are examples and can be adjusted to change the amount of margin and padding.The use of the shorthand property margin and padding allows for setting all sides at once.This code can be customized for other HTML elements and their respective CSS properties.

```
selector {
margin-top: value;
margin-right: value;
margin-bottom: value;
margin-left: value;
padding-top: value;
padding-right: value;
padding-bottom: value;
padding-left: value;
     }
```

margin-top, margin-right, margin-bottom, and margin-left: These properties specify the margin of an element, which is the space outside the border of the element. The value can be a length, a percentage, or the keywords auto or inherit.
padding-top, padding-right, padding-bottom, and padding-left: These properties specify the padding of an element, which is the space between the element's content and its border. The value can be a length, a percentage, or the keywords auto or inherit.
selector: This is a placeholder for the actual selector you want to target with these margin and padding properties. A selector can be an HTML element, a class, an ID, or any combination of these.
Overall, this code sets the margin and padding values for the selected element. The margin-top, margin-right, margin-bottom, and margin-left properties set the margin on each side of the element, while the padding-top, padding-right, padding-bottom, and padding-left properties set the padding on each side of the element. By adjusting these values, you can control the spacing between elements on your web page.

```
#myDiv {
   margin-top: 10px;
```

```
        margin-right: 20px;
        margin-bottom: 30px;
        margin-left: 40px;
          }
```

myDiv: This is a selector that targets the HTML element with the ID "myDiv".margin-top: 10px;: This sets the top margin of the element to 10 pixels.
margin-right: 20px;: This sets the right margin of the element to 20 pixels.margin-bottom: 30px;: This sets the bottom margin of the element to 30 pixels.margin-left: 40px;: This sets the left margin of the element to 40 pixels.

```
#myDiv {
   padding-top: 5px;
   padding-right: 10px;
   padding-bottom: 15px;
   padding-left: 20px;
}
```

"#myDiv" is a selector that targets the element with an ID of "myDiv". This selector can be used to style only that specific element.

"padding-top", "padding-right", "padding-bottom", and "padding-left" are property-value pairs that set the padding of the element on each side. The values are 5px, 10px, 15px, and 20px respectively.

Padding is the space between the content of the element and its border. So, in this case, there will be 5 pixels of space between the top of the content and the top border of the element, 10 pixels of space between the right of the content and the right border of the element, and so on.

The padding values can be specified in different units such as pixels, ems, rems, and percentages. In this example, pixels are used.

Padding is often used to create space between an element's content and its border. This can improve the readability and visual appearance of the content.

CHAPTER 5. UNDERSTANDING BACKEND

Backend languages are used for server-side development and are responsible for managing data and processing requests from the client-side

5.1 BACKEND LANGUAGES

Backend languages are essential because they form the backbone of web applications, allowing for the management and processing of data and requests from the client-side Data Management: Backend languages provide the necessary tools for managing and processing data, ensuring that it is organized, stored securely, and easily retrievable.

Server-Side Processing: Backend languages enable server-side processing of requests, allowing web applications to handle complex logic and operations that cannot be performed on the client-side.Scalability: Backend languages are crucial for ensuring that web applications can scale effectively, handling increased traffic and user requests without compromising performance or stability.

Security: Backend languages play a critical role in ensuring the security of web applications, preventing unauthorized access to data and protecting against cyber threats such as hacking and malware.Integration: Backend languages enable the integration of various services and APIs, making it possible for web applications to interact with external systems and services, such as payment gateways, databases, and social media platforms.

Performance: Backend languages have a significant impact on the performance of web applications, as they determine how quickly and efficiently data is processed and served to the client-side.Customization: Backend languages offer a high level of customization, allowing developers to build web applications that meet specific business requirements and user needs.

5.2 JAVA SCRIPT

JavaScript is a popular high-level programming language used primarily for creating interactive and dynamic web pages. It was initially developed in the mid-1990s and has since become an essential tool for web developers.JavaScript is a client-side language, meaning it runs on the user's web browser rather than on the server, which allows for real-time manipulation of web page content without the need for a page refresh. It is used to add interactivity and functionality to web pages, such as form validation, animation, and user interface design.

JavaScript is a popular scripting language used for creating dynamic and interactive web pages.It is a client-side language that runs on the user's web browser, allowing for real-time manipulation of web page content.

JavaScript is a client-side language, meaning it runs on the user's web browser rather than on the server, which allows for real-time manipulation of web page content without the need for a page refresh. It is used to add interactivity and functionality to web pages, such as form validation, animation, and user interface design.

JavaScript is an object-oriented language, which means it allows developers to create and manipulate objects and their properties and methods. It also supports functional programming, allowing for the creation and manipulation of functions as first-class objects.JavaScript can also be used as a server-side language with technologies like Node.js, allowing developers to build full-stack applications using a single language.

It is an object-oriented language, allowing for the creation and manipulation of objects and their properties and methods.JavaScript is constantly evolving, with new features and updates being released regularly to improve performance, security, and functionality. It is also widely supported by all major web browsers, making it a reliable and accessible language for web development.

JavaScript is a versatile language, with a vast library of frameworks and tools that make it suitable for building complex and scalable applications.JavaScript is known for its flexibility and ease of use, making it a popular language among developers of all skill levels.

It is constantly evolving, with new features and updates being released regularly to improve performance, security, and functionality.JavaScript is widely supported by all major web browsers, making it a reliable and accessible language for web development.

5.3 IMPLEMENTATION OF JAVA SCRIPT

JavaScript code is typically included in HTML documents using script tags, either in the head or body section of the document.External JavaScript files can also be linked to HTML documents using the script tag with the "src" attribute.

JavaScript can be used to manipulate HTML elements by accessing and modifying their properties and attributes.Event listeners can be added to HTML elements using JavaScript to respond to user interactions such as clicks, mouseovers, and key presses.

JavaScript can be used to validate form inputs, ensuring that user-entered data meets specific criteria before being submitted to a server.AJAX (Asynchronous JavaScript and XML) can be used to load and exchange data with a server without requiring a page refresh, allowing for a more dynamic and responsive user experience.

JavaScript can also be used on the server-side with Node.js, allowing developers to write full-stack applications using a single language.Various JavaScript frameworks and libraries, such as React, Angular, and jQuery, can be used to simplify and speed up the development process and provide additional functionality.

JavaScript code can be minified and compressed to reduce file size and improve website performance.Browser compatibility issues can be addressed using JavaScript polyfills or fallbacks to ensure consistent behavior across different browsers.

5.4 EXAMPLE

```
<form>
  <label for="name">Name:</label>
  <input type="text" id="name" name="name"><br>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email"><br>

  <label for="password">Password:</label>
  <input type="password" id="password" name="password"><br>

  <button type="submit" onclick="validateForm()">Submit</button>
</form>

<script>
function validateForm() {
  var name = document.forms[0]["name"].value;
  var email = document.forms[0]["email"].value;
  var password = document.forms[0]["password"].value;

  if (name == "") {
    alert("Name must be filled out");
    return false;
  }

  if (email == "") {
    alert("Email must be filled out");
    return false;
  }

  if (password == "") {
    alert("Password must be filled out");
    return false;
  }

  if (password.length < 8) {
    alert("Password must be at least 8 characters");
    return false;
  }
}
</script>
```

This is a piece of HTML and JavaScript code that creates a form with three input fields for name, email, and password, and a submit button. The form also includes a JavaScript function called validateForm() that is triggered when the submit button is clicked.

The validateForm() function is used to check whether the user has filled out all three fields and whether the password meets certain requirements. If any of the fields are left blank or if the password is less than 8 characters long, an alert box is displayed with an error message and the function returns false, preventing the form from being submitted.

To check the values entered by the user, the function uses the value property of the input fields. It accesses the name, email, and password fields by using the document.forms[0] object, which refers to the first form on the page (in this case, the only form). The [] notation is used to access the value of the specific input field.

Overall, this code provides a basic validation function for a simple form with three input fields. However, it is important to note that client-side validation (i.e., validation done using JavaScript in the browser) is not a substitute for server-side validation, which should always be done to ensure the security and integrity of data.

```
<form id="signup-form">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name"><br>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email"><br>

  <label for="password">Password:</label>
  <input type="password" id="password" name="password"><br>

  <label for="confirm-password">Confirm Password:</label>
  <input type="password" id="confirm-password" name="confirm-password"><br>

  <button type="submit" onclick="submitForm()">Sign Up</button>
</form>

<script>
function submitForm() {
  var form = document.getElementById("signup-form");
  var name = form.elements["name"].value;
  var email = form.elements["email"].value;
  var password = form.elements["password"].value;
  var confirm_password = form.elements["confirm-password"].value;

  if (name == "") {
    alert("Name must be filled out");
    return false;
  }

  if (email == "") {
    alert("Email must be filled out");
    return false;
  }
```

```
  if (password == "") {
    alert("Password must be filled out");
    return false;
  }

  if (password.length < 8) {
    alert("Password must be at least 8 characters");
    return false;
  }

  if (password != confirm_password) {
    alert("Passwords do not match");
    return false;
  }


submission
}
</script>
```

This is another example of an HTML form with validation using JavaScript. It includes four input fields for name, email, password, and confirm password, as well as a submit button. The form also includes a JavaScript function called submitForm() that is triggered when the submit button is clicked.

The submitForm() function performs a series of checks on the input values to ensure that they are valid. It first retrieves the form element using the document.getElementById() method and assigns it to a variable called form. It then uses the form.elements object to access the values of each input field by name and assigns them to separate variables.

The function then checks whether each field has been filled out, and displays an alert with an error message if it hasn't. It also checks whether the password is at least 8 characters long and whether the password and confirm password fields match. If any of these conditions are not met, the function returns false, preventing the form from being submitted.

It's worth noting that this code is an improvement over the previous example in that it includes a confirm password field to ensure that the user has entered their password correctly. However, like before, it's important to remember that client-side validation is not a substitute for server-side validation.

```
<head>
        <title>Calculator</title>
        <script>
```

```
            function calculate() {
                var num1 = parseFloat(document.getElementById("num1").value);
                var num2 = parseFloat(document.getElementById("num2").value);
                var operator = document.getElementById("operator").value;
                var result = 0;

                switch(operator) {
                    case "+":
                        result = num1 + num2;
                        break;
                    case "-":
                        result = num1 - num2;
                        break;
                    case "*":
                        result = num1 * num2;
                        break;
                    case "/":
                        if(num2 == 0) {
                            alert("Cannot divide by zero!");
                            return;
                        }
                        result = num1 / num2;
                        break;
                    default:
                        alert("Invalid operator!");
                        return;
                }

                document.getElementById("result").value = result;
            }
        </script>
</head>
<body>
        <h1>Calculator</h1>
        <form>
            <label for="num1">Number 1:</label>
            <input type="number" id="num1" name="num1"><br>

            <label for="operator">Operator:</label>
            <select id="operator" name="operator">
                <option value="+">+</option>
                <option value="-">-</option>
                <option value="*">*</option>
                <option value="/">/</option>
            </select><br>

            <label for="num2">Number 2:</label>
            <input type="number" id="num2" name="num2"><br>

            <button type="button" onclick="calculate()">Calculate</button><br>
```

```
            <label for="result">Result:</label>
            <input type="number" id="result" name="result" readonly>
        </form>
</body>
</html>
```

The <head> section contains a <title> element that sets the title of the page and a <script> element that defines a JavaScript function called calculate().

The calculate() function retrieves the values of two input elements with the IDs num1 and num2, as well as a select element with the ID operator. It then uses a switch statement to perform the appropriate arithmetic operation based on the selected operator. The result is stored in a variable called result and is displayed in an input element with the ID result.

The <body> section contains a heading element <h1> that displays the title "Calculator", followed by a form element that contains three input elements for the two numbers and the operator, as well as a button element that calls the calculate() function when clicked.Finally, there is an input element with the ID result that displays the result of the calculation.

```
<!DOCTYPE html>
<html>
<head>
        <title>My Single Webpage</title>
</head>
<body>
        <h1>Welcome to my webpage!</h1>
        <p>This is a single webpage created using HTML and JavaScript.</p>
        <script>
                // Inline JavaScript code
                alert('Hello, World!');
        </script>
</body>
</html>
```

This code creates a very basic HTML webpage with a title, a heading, a paragraph, and a single line of JavaScript code.The <head> section of the webpage includes a <title> tag, which sets the title of the webpage that appears in the browser tab. In this case, the title is "My Single Webpage".

The <body> section contains a <h1> tag with the text "Welcome to my webpage!" and a <p> tag with the text "This is a single webpage created using HTML and JavaScript."

The code also includes an inline script using <script> tags. This script simply displays an alert message with the text "Hello, World!" when the page loads. An alert message is a pop-up message box that displays on the screen when the code is executed.Overall, this code creates a very basic HTML webpage with a single line of JavaScript code that displays an alert message.

5.5 NODE JS

Node.js is an open-source, cross-platform, server-side runtime environment built on Chrome's V8 JavaScript engine. It allows developers to use JavaScript on the server-side to build scalable and high-performance applications. Node.js was first released in 2009 and has since become very popular in the development community due to its speed and ease of use.

Node.js uses an event-driven, non-blocking I/O model, which makes it highly efficient and lightweight. It can handle large amounts of I/O operations with minimal resources, making it ideal for building real-time applications such as chat apps, gaming platforms, and social networks. It is also used for building APIs, web applications, and server-side utilities.

Node.js comes with a large number of built-in modules and libraries that can be used to build a variety of applications, such as the HTTP, HTTPS, and Express modules, which are used for building web applications, and the FS and Path modules, which are used for working with the file system.

To get started with Node.js, you first need to install it on your computer. You can download and install it from the official Node.js website. Once you have installed Node.js, you can start writing JavaScript code using a text editor or an IDE. You can then run your code using the Node.js command-line interface (CLI) by typing node filename.js. Node.js will then execute your code and provide you with the output.

Overall, Node.js is a powerful platform for building server-side applications and is widely used by developers around the world. Its ease of use, scalability, and speed make it a popular choice for building modern web applications.

Node.js is an open-source, cross-platform, server-side JavaScript runtime environment that executes JavaScript code outside a web browser. It allows developers to use JavaScript on both the front-end and back-end of web applications, creating a seamless development experience.

Node.js uses an event-driven, non-blocking I/O model, which means that it can handle a large number of simultaneous connections efficiently without blocking the execution of other tasks. This makes it ideal for building highly scalable, real-time applications such as chat applications, real-time collaboration tools, and online gaming platforms.

Fast performance: Node.js is built on the V8 JavaScript engine, which is known for its high performance and speed. This makes Node.js a popular choice for building real-time, high-performance applications.

Easy scalability: Node.js allows developers to scale applications horizontally by adding more instances of the application, rather than vertically by increasing the resources on a single server. This makes it easy to scale applications as traffic increases.
Large and active community: Node.js has a large and active community of developers, which has contributed to the development of a wide range of modules and packages that can be used to extend the functionality of Node.js applications.

Cross-platform support: Node.js is a cross-platform runtime environment, which means that it can be used to build applications that can run on different operating systems such as Windows, macOS, and Linux.Node.js is commonly used to build web applications, RESTful APIs, real-time applications, and microservices. It is also used in the development of tools and utilities such as build tools, testing frameworks, and command-line interfaces.

To work with Node.js, developers typically use a package manager such as npm or Yarn to install and manage dependencies, and a code editor such as Visual Studio Code or Sublime Text to write and edit code. Node.js also has a built-in command-line interface that can be used to run scripts and manage applications.

5.5.1 EXPRESS.JS

Express is a popular Node.js framework for building web applications and APIs. It is fast, flexible, and provides a robust set of features for web and mobile applications.
Express provides a simple API for building HTTP servers and handling requests and responses.

It supports middleware, which allows you to easily add functionality to your application, such as logging, authentication, and error handling.You can also create routes for handling specific requests and defining endpoints for your API.

To get started with Express, you first need to install it using Node.js's package manager, npm. Once installed, you can create a new Express application using the following code:

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

// Define routes and middleware here

app.listen(3000, () => console.log('Server started on port 3000'));
```

This code imports the Express library, creates a new application instance, and defines a callback function to start the server on port 3000. You can then define routes and middleware for handling requests and responses.For example, the following code defines a route for handling GET requests to the root URL ('/') and returns a simple message:

```
app.get('/', (req, res) => {
res.send('Hello, World!');});
```

This code uses the app.get() method to define a route handler for GET requests to the root URL. When a request is received, the callback function is executed, which sends a response back to the client with the message "Hello, World!".

Express also provides a variety of middleware functions for handling requests and responses. For example, the following code uses the express.json() middleware to parse JSON data in request bodies

```
app.use(express.json());

app.post('/api/users', (req, res) => {
 const user = req.body;
// Handle the user data here
});
```

This code uses the app.use() method to add the express.json() middleware to the application. The middleware parses any JSON data in the request body and makes it available in the req.body object. The application then defines a route handler for POST requests to the '/api/users' endpoint, which can then access the user data in the req.body object.

Overall, Express provides a powerful and flexible framework for building web applications and APIs in Node.js. With its easy-to-use API and robust set of features, it has become one of the most popular Node.js frameworks for web development.

5.5.2 JSON

JSON stands for JavaScript Object Notation, and it is a lightweight data interchange format that is easy for humans to read and write and for machines to parse and generate.

It is based on a subset of the JavaScript programming language, and is often used to transmit data between a s erver and web application as an alternative to XML.
JSON data is represented using key-value pairs, with the keys being strings and the values being any valid JSON data type (such as strings, numbers, arrays, or other objects).

JSON supports nested data structures, allowing objects and arrays to be used as values for other objects or arrays.It is often used in web development for transmitting data between the server and client-side JavaScript code, as well as for storing configuration data and other information in text files.

JSON has become a widely adopted standard for data exchange due to its simplicity, flexibility, and compatibility with many programming languages and tools.

Example

```
{
   "name": "John Doe",
    "age": 30,
      "email": "johndoe@example.com",
      "address": {
       "street": "123 Main St",
         "city": "Anytown",
         "state": "CA"}
```

5.6 TYPESCRIPT

TypeScript is an open-source programming language that is a superset of JavaScript. It adds optional static typing and other features to JavaScript, which makes it easier to write and maintain large-scale applications.

Static Typing: TypeScript supports static typing, which helps to catch errors during development. Static typing also improves code quality, maintainability, and readability, and it reduces the number of errors that occur at runtime.

Object-Oriented Features: TypeScript supports object-oriented programming (OOP) concepts like classes, interfaces, and inheritance. These features allow developers to write cleaner and more modular code.Compiler: TypeScript code is compiled into JavaScript code, which can be run on any browser or platform.

The TypeScript compiler is highly configurable, which makes it easy to customize the build process according to the needs of the project.

Compatibility with JavaScript: TypeScript is fully compatible with existing JavaScript code, which makes it easy to integrate TypeScript into an existing JavaScript project.
TypeScript also supports all the features of the latest version of JavaScript, which means that developers can use the latest language features while still enjoying the benefits of static typing.Tooling: TypeScript has excellent tooling support, which includes editors, IDEs, and other development tools.

TypeScript integrates seamlessly with popular development tools like Visual Studio Code, which provides features like code completion, debugging, and syntax highlightingNestJS and TypeScript are closely related to each other.

NestJS is built using TypeScript. This means that it relies on TypeScript to provide features such as static typing, classes, and interfaces.TypeScript is a superset of JavaScript. This means that it provides all of the features of JavaScript, as well as additional features that are not available in the core language.

One of the main benefits of using TypeScript is that it provides compile-time type checking. This helps to catch errors before the code is even executed, which can save time and prevent bugs.NestJS takes advantage of TypeScript's features to provide a framework for building scalable, maintainable, and robust server-side applications.

NestJS provides decorators that allow developers to easily define controllers, providers, and modules. These decorators are similar to annotations in other languages and provide a way to add metadata to classes and functions.TypeScript's class-based syntax and interface definitions are well-suited to the structure of NestJS applications. This makes it easy to organize code and create reusable components.

TypeScript's support for generics and interfaces allows developers to create strongly-typed APIs and data models. This makes it easier to reason about the code and prevents errors caused by type mismatches.

NestJS and TypeScript are both open source projects with large and active communities. This means that there are plenty of resources and tools available for developers who want to learn more about them.

5.7 EXAMPLE

```
class Person {
  name: string;
  age: number;

  constructor(name: string, age: number) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
  }
}

const person = new Person('John', 30);
person.greet();
```

Outputs "Hello, my name is John and I am 30 years old."

This is a TypeScript code that defines a class called "Person". The class has two properties: "name" and "age" of types string and number, respectively. It also has a constructor that takes in two parameters, "name" and "age", and sets them to the corresponding properties of the class using the "this" keyword.

The class also has a method called "greet()", which logs a greeting message to the console using the "console.log()" method. The message includes the name and age of the person.

In the last two lines of the code, a new instance of the Person class is created using the "new" keyword and the constructor arguments "John" and 30. Then, the "greet()" method of this instance is called using the dot notation.When the code is executed, it will output the message "Hello, my name is John and I am 30 years old." to the console.

This code demonstrates the use of TypeScript's class syntax and type annotations, as well as the object-oriented programming concepts of encapsulation and method invocation. It also showcases the benefits of using TypeScript over plain JavaScript, as TypeScript helps catch type-related errors at compile-time rather than runtime

```
import React from 'react';

 interface Props {
name: string;
age: number;
```

```
 }

const Person: React.FC<Props> = ({ name, age }) => {
return (
<div>
<p>Name: {name}</p>
<p>Age: {age}</p>
</div>
);
 };
```

This is a React functional component written in TypeScript. import React from 'react'; - imports the React library so we can use its features to create a component.interface Props { name: string; age: number; } - defines an interface for the props that will be passed to this component. It specifies that the component requires a name prop of type string and an age prop of type number.

const Person: React.FC<Props> = ({ name, age }) => { ... } - creates a new functional component named Person. The React.FC type specifies that this is a functional component. The <Props> generic tells TypeScript to use the Props interface we defined earlier to check the props being passed to the component. The component takes in an object with two properties, name and age, which are destructured from the props object.

return (...); - returns the JSX elements to be rendered on the page when this component is used. In this case, it returns a div containing two p elements that display the name and age props passed to the component.

So, this component takes in a name prop and an age prop, both of which are required and have specific data types, and it renders a div with two p elements that display the name and age values respectively.

CHAPTER 6. CREATING API

NestJS is a powerful and popular Node.js framework that allows you to build efficient, scalable, and maintainable server-side applications. One of the key features of NestJS is its ability to quickly create APIs.

6.1 API

API stands for Application Programming Interface.An API provides a way for different software components or systems to communicate and exchange data with each other.
APIs can be used to expose functionality of an application or service to third-party developers or systems, enabling them to build applications or integrate systems using that functionality.

APIs can be used to access or manipulate data, trigger actions, or retrieve information from a system or application.APIs can be used over the internet or within a local network, and can be accessed through various protocols such as HTTP, WebSocket, or GraphQL.
APIs can be designed using different architectural styles such as REST, SOAP, or gRPC.
APIs can be created for various purposes, such as providing access to a database, integrating different systems, building a mobile application, or developing a web application.

APIs can be secured using authentication and authorization mechanisms to ensure that only authorized users or systems can access them.APIs can be documented using various tools and standards such as OpenAPI, Swagger, or RAML to provide information about the API's endpoints, parameters, and response formats.

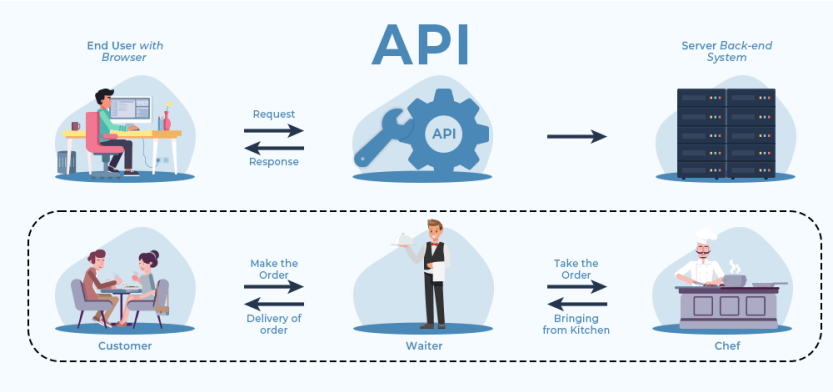

Figure 6.1 API Block Diagram

6.2 NEST JS

NestJS is a back-end web application framework that is built with Node.js and TypeScript. It is designed to be scalable and efficient, while also providing developers with a flexible and modular structure to build their applications.

Modular design: NestJS follows a modular architecture which makes it easy to create and maintain large-scale applications.Built with TypeScript: NestJS is built on top of TypeScript, which adds a lot of features to JavaScript, such as static typing, interfaces, and decorators.

Dependency Injection: NestJS makes use of dependency injection which allows developers to easily manage the dependencies between different modules and components.

MVC framework: NestJS follows the Model-View-Controller (MVC) design pattern, which separates the application into three main components: the model, the view, and the controller.Support for multiple protocols: NestJS provides support for multiple protocols such as HTTP, WebSockets, and gRPC.

Built-in testing: NestJS comes with built-in testing capabilities that allow developers to easily test their code and catch errors before they reach production.Middleware: NestJS provides a middleware system which allows developers to add custom logic to the request/response cycle of their application.

Scalable: NestJS is designed to be scalable and can handle high levels of traffic and data processing.Active community: NestJS has a very active community that contributes to its development, creates plugins and modules, and provides support to other developers.

Overall, NestJS is a powerful and flexible framework that allows developers to easily build scalable and efficient back-end applications. Its modular design, support for TypeScript, and built-in testing capabilities make it an ideal choice for building complex applications.

6.2.1 CONTROLLER

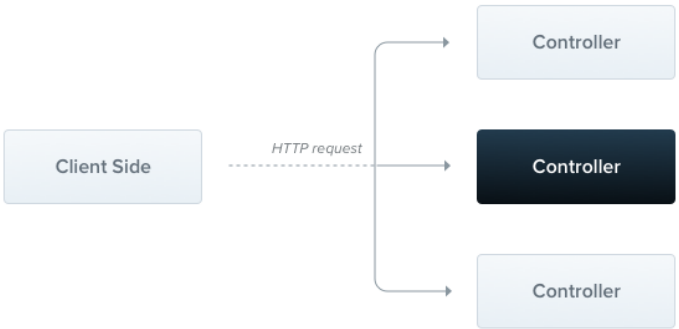

Figure 6.2.1 Controller

In NestJS, a controller is a class responsible for handling incoming HTTP requests and generating HTTP responses. Controllers act as a mediator between the server's API routes and the application's logic, receiving the request data, processing it, and returning the response to the client.

Here are some key points about controllers in NestJS:Controllers are defined as classes decorated with @Controller() and a path prefix. For example: @Controller('users').Each controller method is decorated with the HTTP verb it handles (@Get(), @Post(), @Put(), @Delete(), etc.) and a route relative to the controller's path prefix.

Controllers can use dependency injection to inject services, providers, or other dependencies they need to process the request. For example: constructor(private userService: UserService) {}.

Controllers can use pipes to validate or transform incoming data before it reaches the method. Pipes are used to sanitize data, perform type conversion, or perform custom validation.

Controllers can use guards to restrict access to certain routes based on user roles, permissions, or authentication status.Controllers can return various types of responses, such as plain text, HTML, JSON, or binary data.

Controllers can use interceptors to modify the response or request objects, such as adding headers, logging, or transforming the response data.NestJS supports many built-in decorators to handle common HTTP features, such as query parameters, request headers, cookies, or file uploads.

Controllers can be organized into modules, which are independent units of the application that can be easily added or removed from the server.

EXAMPLE

```
import { Controller, Get, Param } from '@nestjs/common';

@Controller('cats')
export class CatsController {
@Get()
findAll(): string {
return 'This action returns all cats';
}

@Get(':id')
findOne(@Param('id') id: string): string {
return `This action returns a #${id} cat`;
}
}
```

@Controller('cats'): This decorator specifies that this controller will handle requests for the /cats route.export class CatsController: This is the definition of the CatsController class.

@Get(): This decorator specifies that this method will handle GET requests to the /cats route (because it has no parameters).findAll(): string: This is the implementation of the /cats GET route handler. It simply returns a string that says it returns all cats.

@Get(':id'): This decorator specifies that this method will handle GET requests to the /cats/:id route (because it has a parameter called id).

findOne(@Param('id') id: string): string: This is the implementation of the /cats/:id GET route handler. It takes a single parameter called id which is extracted from the request URL using the @Param decorator. It returns a string that says it returns a specific cat with the given ID.

6.2.2 Providers

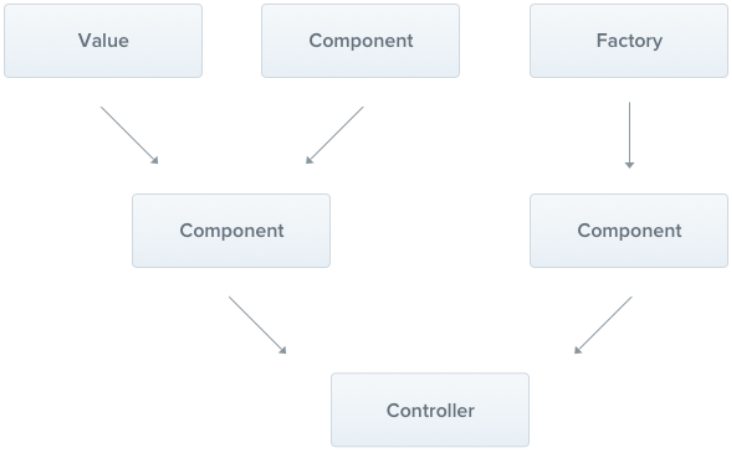

Figure 6.2.2 Providers

In NestJS, a controller is a class responsible for handling incoming HTTP requests and generating HTTP responses. Controllers act as a mediator between the server's API routes and the application's logic, receiving the request data, processing it, and returning the response to the client.

Here are some key points about controllers in NestJS:
Controllers are defined as classes decorated with @Controller() and a path prefix. For example: @Controller('users').

Each controller method is decorated with the HTTP verb it handles (@Get(), @Post(), @Put(), @Delete(), etc.) and a route relative to the controller's path prefix.

Controllers can use dependency injection to inject services, providers, or other dependencies they need to process the request. For example: constructor(private userService: UserService) {}.

Controllers can use pipes to validate or transform incoming data before it reaches the method. Pipes are used to sanitize data, perform type conversion, or perform custom validation.Controllers can use guards to restrict access to certain routes based on user roles, permissions, or authentication status.Controllers can return various types of responses, such as plain text, HTML, JSON, or binary data.

Controllers can use interceptors to modify the response or request objects, such as adding headers, logging, or transforming the response data.NestJS supports many built-in decorators to handle common HTTP features, such as query parameters, request headers, cookies, or file uploads.Controllers can be organized into modules, which are independent units of the application that can be easily added or removed from the server.

Example

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class CatsService {
private readonly cats: string[];

constructor() {
this.cats = ['Mittens', 'Fluffy', 'Whiskers'];
}

findAll(): string[] {
return this.cats;
}

findOne(id: number): string {
return this.cats[id];
}
}
```

This code is an example of a NestJS provider, which is a class that can be injected into other classes to provide certain functionality. In this case, the CatsService class provides a simple in-memory database of cat names.The @Injectable() decorator is used to mark the class as an injectable provider in NestJS.The private readonly cats: string[] property is an array that holds the names of the cats.The constructor initializes the cats array with three cat names.The findAll() method returns all of the cat names in the cats array.

The findOne(id: number) method takes an id parameter, which is used to find a specific cat name in the cats array. The method returns the cat name at the specified index in the array.Overall, this CatsService provider can be injected into other classes to provide access to a simple in-memory database of cat names.

6.2.3 MODULE

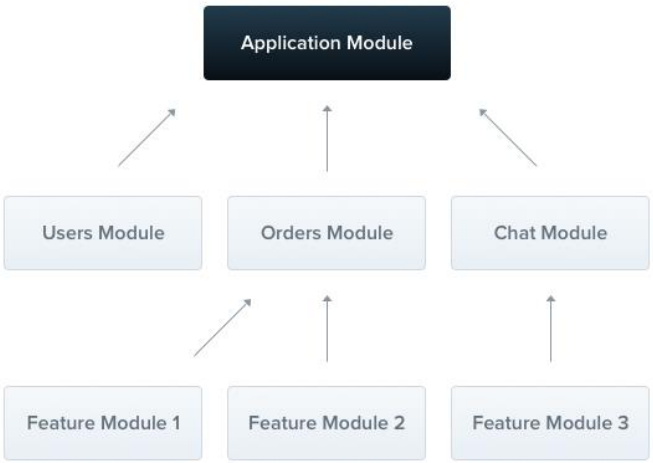

Figure 6.2.3 Module

Modules are a fundamental building block in NestJS applications. They encapsulate related functionality and enable you to organize your code into logical units.A module is simply a TypeScript class decorated with the @Module() decorator. This decorator configures the module and specifies its dependencies.

Modules can declare providers, controllers, and other modules as dependencies, enabling you to build complex, modular architectures.NestJS provides a number of different ways to configure modules, including configuring providers using constructor injection and configuring modules using the imports array.

Modules can be loaded asynchronously, which enables you to perform complex setup logic such as connecting to a database or performing authentication.Modules can be used to organize your application into different layers, such as a presentation layer, a business logic layer, and a data access layer.

NestJS provides a number of built-in modules that you can use to add functionality to your application, such as the HttpModule for handling HTTP requests and responses.Modules can be reused across multiple applications, enabling you to build libraries and share code between different projects.

Overall, modules are a powerful feature of NestJS that enable you to build scalable, maintainable applications by organizing your code into logical units and managing dependencies between them.

EXAMPLE

```
import { Module } from '@nestjs/common';
import { CatsController } from './cats.controller';
import { CatsService } from './cats.service';

@Module({
controllers: [CatsController],
providers: [CatsService],
})
export class CatsModule {}
```

This code is an example of a module in NestJS. A module is a fundamental building block of a NestJS application that groups related functionality using controllers and providers.Here, the CatsModule is defined and it has two properties inside the @Module decorator:

controllers: This is an array of controllers that are associated with this module. In this case, the CatsController is the only controller associated with this module.providers: This is an array of providers that are associated with this module. In this case, the CatsService is the only provider associated with this module.

When a module is imported into another module, all the providers, controllers, and other elements exported by the imported module become part of the importing module.So, if the CatsModule is imported in another module, the CatsController and CatsService will be available for use in that module.Overall, this code sets up a CatsModule with a CatsController and a CatsService, which can be used to handle HTTP requests related to cats in a NestJS application.

6.2.4 MDDLEWARE

6.2.4 Middleware

Middleware is a type of software that acts as a bridge between different applications, systems, or components, allowing them to communicate and interact with each other.
In the context of web development, middleware is code that runs between the server receiving a request and sending a response. It can intercept, modify, or augment the request or response as needed.

Middleware functions are typically chainable, which means that multiple middleware functions can be executed in sequence to handle different aspects of the request or response. Middleware

can be used for a variety of purposes, such as authentication, logging, error handling, caching, compression, and more.

Middleware can be written in various programming languages and frameworks, and is commonly used in popular web development frameworks such as Express (for Node.js), Django (for Python), and Ruby on Rails (for Ruby).

Middleware can be built into the framework or library being used, or can be created as custom middleware functions specific to the application being built.

Example

```
// This is a middleware function
const logMiddleware = (req, res, next) => {
  console.log(`${req.method} ${req.path}`);
  next();
};

// Use the middleware function
app.use(logMiddleware);
```

This code defines a middleware function called logMiddleware. The middleware function takes three parameters: req (request object), res (response object), and next (a callback function that signals to move to the next middleware or route handler).

The logMiddleware function logs the HTTP method and path of the incoming request using console.log(). After logging, it calls the next() function to move to the next middleware or route handler.

The middleware function is then used by the Express app with app.use() method. This method registers the middleware function globally so that it will be executed for all incoming requests.

When a request is made to the app, the middleware function will be called first before any other route handlers. It logs the request details and then calls next() to move to the next middleware function or route handler.

6.3 IMPLEMENTATION OF NEST JS IN PROJECT

NestJS can be used in issue management systems to build robust and scalable web applications that can handle large volumes of data and complex business logic.

Ticket tracking: NestJS can be used to build ticket tracking systems that allow users to create, manage, and track issues, bugs, and feature requests. It can handle complex workflows, notifications, and integrations with other systems.

Analytics and reporting: NestJS can be used to build analytics and reporting tools that provide insights into the status, progress, and performance of projects and issues. It can handle data visualization, real-time updates, and custom queries.

Collaboration and communication: NestJS can be used to build collaboration and communication tools that enable teams to work together on issues, share files, and exchange messages. It can handle real-time chat, file uploads, and integrations with external services.

Automation and workflows: NestJS can be used to build automation and workflow tools that streamline the process of issue management, reducing manual work and improving efficiency. It can handle task scheduling, rule-based triggers, and integrations with other systems.

Security and access control: NestJS can be used to build secure and access-controlled issue management systems that protect sensitive data and enforce compliance. It can handle authentication, authorization, and encryption.

Overall, NestJS provides a powerful and flexible platform for building issue management systems that meet the specific needs of businesses and organizations.Its modular architecture, strong typing, and rich ecosystem of modules and libraries make it an ideal choice for building complex, scalable, and maintainable web applications.

CH 7 IMPLEMENTATION

7.1 IMPLEMENTATION NEST JS FOR PROJECT

NestJS can be used in issue management systems to build robust and scalable web applications that can handle large volumes of data and complex business logic.

Ticket tracking: NestJS can be used to build ticket tracking systems that allow users to create, manage, and track issues, bugs, and feature requests. It can handle complex workflows, notifications, and integrations with other systems.

Analytics and reporting: NestJS can be used to build analytics and reporting tools that provide insights into the status, progress, and performance of projects and issues.

It can handle data visualization, real-time updates, and custom queries.

Collaboration and communication: NestJS can be used to build collaboration and communication tools that enable teams to work together on issues, share files, and exchange messages. It can handle real-time chat, file uploads, and integrations with external services.

Automation and workflows: NestJS can be used to build automation and workflow tools that streamline the process of issue management, reducing manual work and improving efficiency. It can handle task scheduling, rule-based triggers, and integrations with other systems.

Security and access control: NestJS can be used to build secure and access-controlled issue management systems that protect sensitive data and enforce compliance. It can handle authentication, authorization, and encryption.

Overall, NestJS provides a powerful and flexible platform for building issue management systems that meet the specific needs of businesses and organizations. Its modular architecture, strong typing, and rich ecosystem of modules and libraries make it an ideal choice for building complex, scalable, and maintainable web applications.

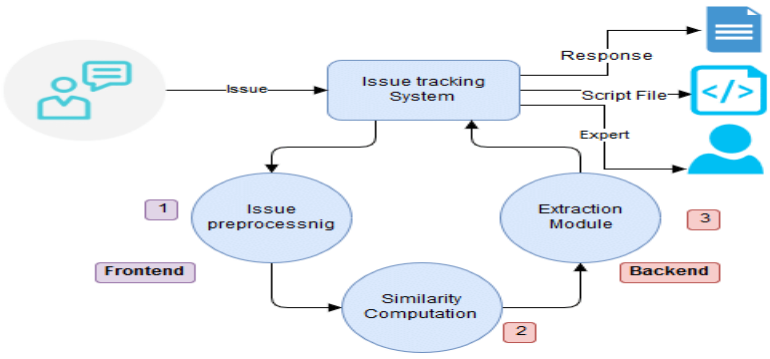

Figure 7.1 Issue Managmente

7.2 OBJECTIVES OF PROJECT

To provide a centralized platform for tracking and managing issues across an organization, enabling stakeholders to view and update the status of issues in real-time.

To improve communication and collaboration among team members by providing a shared platform for discussing issues and assigning tasks.
To increase visibility into the issue resolution process by providing analytics and reporting features that enable stakeholders to track key performance indicators such as issue resolution time and customer satisfaction.
To reduce the time and effort required to resolve issues by providing automated workflows that route issues to the appropriate team members and escalate issues when necessary. To improve the accuracy and completeness of issue data by providing standardized data entry forms and enforcing data validation rules.
To enable integration with other systems, such as customer relationship management (CRM) and project management tools, to provide a comprehensive view of issue management across the organization.
To enable customization and flexibility to meet the unique needs of different teams and departments within the organization.
To improve overall customer satisfaction by resolving issues more quickly and effectively, and by providing a transparent and responsive issue resolution process.

7.3 PROJECT

Controller: -

```
import { Controller, Get, Post, Body, Param, Put, Delete, BadRequestException, Query,
Patch, NotFoundException } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
//import console from 'console';
import { Model } from 'mongoose';
import { Issue, IssueCriticality } from './issue.model';
import { IssueService } from './issue.service';

@Controller('issue')
export class IssueController {
[x: string]: any;
constructor(@InjectModel('Issue') private readonly issueModel: Model<Issue>,private
readonly issueService: IssueService) {}
// Create issue
@Post("/issues")
async create(@Body() issue: Issue): Promise<Issue> {
const createdIssue = new this.issueModel(issue);
return createdIssue.save();
}

// Get issue by ID
@Get(':id')
async findById(@Param('id') id: string): Promise<Issue> {
```

```
return this.issueModel.findById(id).exec();
}

// Assign issue to support person

@Put('/:id/assign')
async assign(@Param('id') id: string, @Body('assignedTo') assignedTo: string):
Promise<Issue> {
return this.issueService.assign(id, assignedTo);
}

// Resolve issue
@Put('/:id/resolve')
async resolve(@Param('id') id: string, @Body('resolvedBy') resolvedBy: string):
Promise<Issue> {
return this.issueService.resolve(id, resolvedBy);
}
// To Delete issue
@Delete(':id')
async deleteIssue(@Param('id') id: string): Promise<void> {
await this.issueService.deleteIssue(id);
}
@Post()
async createIssue(@Body() issue: Issue): Promise<Issue> {
const existingIssue = await this.issueModel.findOne({ title: issue.title }).exec();
if (existingIssue) {
throw new Error('Issue with same title already exists');
} else {
const createdIssue = new this.issueModel(issue);
return createdIssue.save();
}
}

//To chnage end point
@Patch(':id')
async updateIssue(@Param('id') id: string, @Body() fields: Partial<Issue>): Promise<Issue>
{
return this.issueService.updateIssue(id, fields)
.then(updatedIssue => updatedIssue || Promise.reject(new NotFoundException(`Issue with ID
"${id}" not found`)));
}

//to get string
@Get()
async getByParams(@Query() fields: Partial<Issue>): Promise<Issue[]> {
const issues = await this.issueService.getByParams(fields);
return issues;
```

Issue Model: -

```
import * as mongoose from 'mongoose';
export enum IssueCriticality {
 Critical = 'Critical',
 High = 'High',
 Low = 'Low',
}

export const IssueSchema = new mongoose.Schema({
assetId: { type: String },
title: {type:String},
deviceInstanceId: { type: String },
virtualDeviceInstanceId: { type: String },
alertId: { type: String },
description: { type: String, required: true },
criticality: { type: String, enum: ['Critical', 'High', 'Low'], required: true },
openDateTime: { type: Date, default: Date.now },
openedBy: { type: String },
enteredDateTime: { type: Date, default: Date.now },
enteredBy: { type: String },
assignedDateTime: { type: Date },
assignedTo: { type: String },
assignedBy: { type: String },
resolvedDateTime: { type: Date },
resolvedBy: { type: String },
resolveTo:{ type: String },
resolve: {type: String },
getIssueByTitle: {type: String },
});

export interface Issue extends mongoose.Document {
id?: string;
title?:string;
assetId?: string;
deviceInstanceId?: string;
virtualDeviceInstanceId?: string;
alertId?: string;
description: string;
criticality: 'Critical' | 'High' | 'Low';
openDateTime: Date;
openedBy?: string;
enteredDateTime: Date;
enteredBy?: string;
assignedDateTime?: Date;
assignedTo?: string;
```

```
assignedBy?: string;
resolvedDateTime?: Date;
resolvedBy?: string;
resolveTo: String ;
resolve:string;
getIssueByTitle:string;

}
```

Issue Module:-

```
import { Module } from '@nestjs/common';
import { MongooseModule } from '@nestjs/mongoose';
import { IssueController } from './issue.controller';
import { IssueService } from './issue.service';
import { Issue, IssueSchema } from './issue.model';

@Module({
  imports: [MongooseModule.forFeature([{ name: 'Issue', schema: IssueSchema }])],
  controllers: [IssueController],
  providers: [IssueService],
})

export class IssueModule {}
```

Issue Service

```
import { BadRequestException, ConflictException, Injectable, NotFoundException, Query }
from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
```

```
import { Model } from 'mongoose';
import { Issue, IssueCriticality } from './issue.model';
import { IssueModule } from './issue.module';

@Injectable()
export class IssueService {
  [x: string]: any;
  constructor(@InjectModel('Issue') private readonly issueModel: Model<Issue>) { }

  async findById(id: string): Promise<Issue> {
    return this.issueModel.findById(id).exec();
  }

  async findOne(id: string): Promise<Issue> {
    return this.issueModel.findById(id).exec();
  }

  async create(issue: Issue): Promise<Issue> {

    const createdIssue = new this.issueModel(issue);
    return createdIssue.save();
  }
  async assign(id: string, assignedTo: string): Promise<Issue> {
    const issue = await this.issueModel.findById(id).exec();
    if (!issue) {
      throw new NotFoundException('Issue not found');
    }
    issue.assignedBy = "dinesh";
    issue.assignedDateTime = new Date();
    return issue.save();
  }
  async resolve(id: string, resolveTo: string): Promise<Issue> {
    const issue = await this.issueModel.findById(id).exec();
    if (!issue) {
      throw new NotFoundException('Issue not found');
    }
    issue.resolvedBy = resolveTo;
    issue.resolvedDateTime = new Date();
    return issue.save();
  }

  async deleteIssue(id: string): Promise<void> {
    const result = await this.issueModel.deleteOne({ _id: id }).exec();
    if (result.deletedCount === 0) {
      throw new NotFoundException(`Issue with ID ${id} not found`);
    }
  }

  async getIssuesByPriority(priority: string): Promise<Issue[]> {
    return this.issueModel.find({ priority }).sort({ createdAt: -1 }).exec();
```

```
  }
  async createIssue(issue: Partial<Issue>): Promise<Issue> {
    // Check if an issue with the same title already exists
    const existingIssue = await this.issueModel.findOne({ title: issue.title }).exec();
    if (existingIssue) {
      throw new BadRequestException(`An issue with the title '${issue.title}' already exists`);
    }

    // Create the new issue
    const createdIssue = await this.issueModel.create(issue);
    return createdIssue.toObject();
  }
  //to update string
  async updateIssue(id: string, fields: Partial<Issue>): Promise<Issue> {
    const issue = await this.issueModel.findOne({ _id: id }).exec();
    if (!issue) {
      throw new NotFoundException(`Issue with ID "${id}" not found`);
    }

    Object.assign(issue, fields);
    const updatedIssue = await issue.save();

    return updatedIssue;
  }

   //to get string
   async getByParams(fields: Partial<Issue>): Promise<Issue[]> {
    const query = {};
    for (const key in fields) {
      if (fields[key]) {
        query[key] = fields[key];
      }
    }

    const issues = await this.issueModel.find(query).exec();

    return issues;

  }

}
```

7.3.1Application

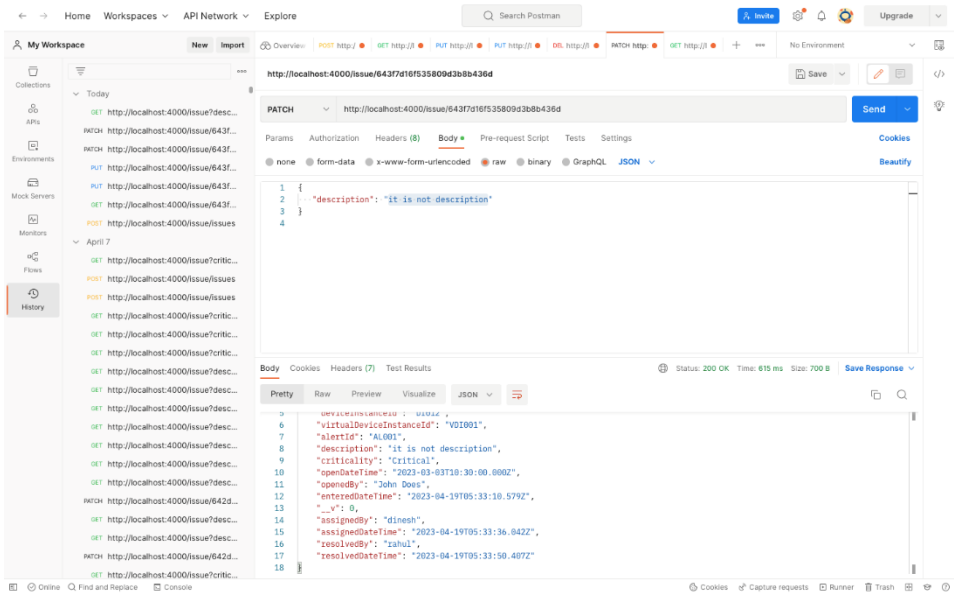

Figure 7.3.1 API

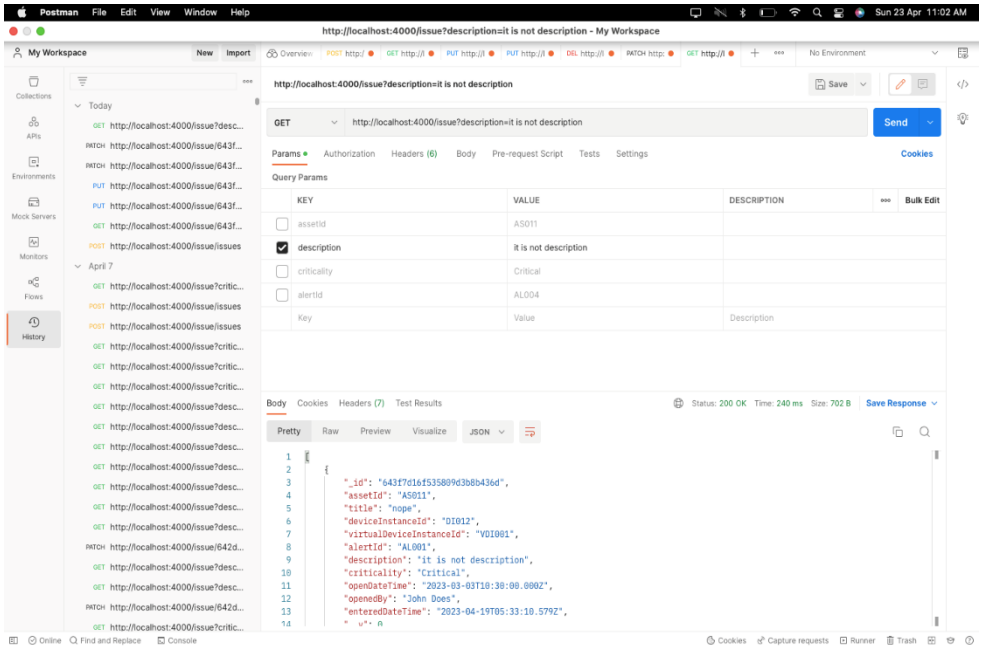

Figure 7.3.2 API 2

CHAPTER 8 CONCLUSION AND FUTURE ENHANCEMENT

8.1 CONCLUSION

NestJS provides a solid framework for building scalable and maintainable web applications, including those that manage issues.NestJS uses decorators to simplify routing and request handling, making it easier to build RESTful APIs.
By using dependency injection, NestJS makes it easier to manage the various components of an issue management system, such as databases, models, and services.NestJS has a robust ecosystem of plugins and modules that can be used to extend the functionality of an issue management system.

With its built-in support for TypeScript, NestJS provides a type-safe way to define APIs, making it easier to catch errors early in the development process.NestJS provides powerful tools for logging and error handling, which are essential for tracking and resolving issues.

8.2 FUTURE ENHANCEMENT

AI/ML integration: Using machine learning algorithms, an issue management system could analyze past issues and provide suggestions for resolution or prevention of similar issues in the future.Mobile app integration: Developing a mobile application for an issue management system would allow users to report and manage issues on-the-go.

Chatbot integration: Integrating a chatbot into an issue management system could provide users with immediate assistance and support, reducing the response time to reported issues.

Dashboard and Reporting: Adding a dashboard and reporting functionality can provide a visual representation of the issues, which can be analyzed to identify trends and patterns.Integration with other tools: Integrating with other tools like version control systems, CI/CD pipelines, or project management tools can provide additional context to the issues and help prioritize the issues.

Notification system: Implementing a notification system can notify users or teams about the status of an issue, or the next steps required for its resolution.Customer Portal: Implementing a customer portal can allow customers to report issues, view their status, and track their progress

8.3REFERENCES

https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://www.w3schools.com/js/DEFAULT.asp
https://www.geeksforgeeks.org/javascript/
https://www.typescriptlang.org/
https://www.mongodb.com/
https://nestjs.com/
https://expressjs.com/
https://nodejs.org/en