Task Manager - Website in JavaScript

Technical University of Cluj-Napoca

Halmagiu Catalin Gabriel

1. Introduction
   1. Overview of Web Development
   2. Importance of JavaScript, Node.js, MySQL, and Angular
2. Working with MySQL Database
   1. Designing the Database Schema
   2. Connecting Node.js with MySQL
3. Building the Node.js API
   1. Creating the Project Structure
   2. Setting up Express.js
   3. Establishing Database Connection
   4. Creating API Endpoints
   5. Handling HTTP Requests and Responses
4. Developing the Angular Frontend
   1. Creating Components and Services
   2. Building User Interfaces with Angular Templates
   3. Implementing Data Binding and Event Handling
   4. Integrating API Calls with the Backend
5. Conclusion

1. Introduction
2. Overview of Web Development

A website establishes an online presence that is reachable to a worldwide audience 24/7 and acts as a virtual representation of your company or personal brand. By giving your audience useful information about your goods, services, or concepts, it serves as a hub for information and helps you establish credibility and trust with your audience. A website promotes your offerings, encourages direct interaction, and facilitates e-commerce transactions through marketing tactics and consumer engagement features. Additionally, it provides insightful analytics and information that will aid in your data-driven decision-making while enabling scalability and accessibility for ongoing development and participation in the digital world.

Today, we live in a digital era where websites play a crucial role in our daily lives. From online shopping to social networking, websites have become an integral part of our online experience. In this referat, we will explore the fundamental concepts behind building a website using JavaScript, Node.js for the API, and Angular for the front end.

1. Importance of JavaScript, Node.js, MySQL, and Angular

A website is a collection of web pages that are accessed through a web browser. It serves as an interface between users and the information or services they seek. Websites are built using various programming languages and technologies to create a visually appealing, interactive, and functional

JavaScript is a widely used programming language for building websites. It provides the necessary tools and frameworks to create dynamic and interactive web pages. JavaScript enables developers to manipulate web page elements, handle events, validate forms, and perform other essential functions on the client-side.

Node.js is a powerful runtime environment that allows JavaScript to be executed on the server-side. It provides a server-side JavaScript execution environment, enabling developers to build scalable and efficient web applications. Node.js enables us to create APIs (Application Programming Interfaces) to handle data requests from the front end and interact with databases.

Angular is a popular framework for building web applications. It is developed and maintained by Google and provides a comprehensive set of tools and features to create dynamic and responsive user interfaces. Angular follows the Model-View-Controller (MVC) architectural pattern, making it easier to manage complex web applications.

MySQL is an important database management system known for its high performance, scalability, and versatility. It offers robustness, compatibility with various technologies, and strong security features. With its wide support and optimization capabilities, MySQL is a reliable choice for businesses and developers seeking efficient data storage and retrieval.

1. Working with MySQL Database
2. Designing the Database Schema

Designing the database schema is a crucial step in creating a well-structured and efficient database system. It involves defining the tables, their columns, data types, relationships, and constraints that will store and organize the data. MySQL Workbench provides a visual interface to assist in this process. Users can create and manage tables, define primary keys, set up relationships between tables (such as one-to-one or one-to-many), and establish constraints like unique values or foreign keys. The graphical representation allows for a clear visualization of the database structure, making it easier to design an optimized and scalable schema for the specific needs of the application or project.

In order to successfully establish a database's structure and relationships, several critical actions must be taken during design. Here is an explanation of the design process for the MySQL database used by the task manager application:

Start by determining the two primary components of the program, namely "users" and "tasks." These objects stand for various data types that need to be managed and stored.

The database should now contain two new tables called "users" and "tasks." The "users" table will contain data about app users, whereas the "tasks" table will contain information about the tasks that users have created.

Choose the appropriate columns and their matching data types for each table.

Determine the table's main key. In this instance, the primary key for both the "users" and the "tasks" tables will be the "id" column.

The "users" and "tasks" tables should be related to one another using a foreign key constraint. Each task should be associated with the user who created it via the "user\_id" column in the "tasks" table, which should correspond to the "id" column in the "users" table.

The final result:

A screenshot of a computer

Description automatically generated with low confidence

1. Connecting Node.js with MySQL

To connect Node.js with MySQL, you can use the MySQL module, which is a popular choice for working with MySQL databases in Node.js.

Connecting Node.js with MySQL involves several steps. First, you need to install the MySQL module using npm, which allows you to interact with the MySQL database from your Node.js application. Once the module is installed, you can require it in your Node.js script.

Next, you'll set up the necessary database connection details such as the host, username, password, and database name. These details will be used to establish a connection between Node.js and the MySQL database.

To manage connections efficiently, you can create a connection pool. A connection pool allows you to reuse and manage connections instead of establishing a new connection for every query. This helps improve performance and scalability.

Once the connection pool is set up, you can acquire a connection from the pool whenever you need to execute a query. With the acquired connection, you can then execute SQL queries against the MySQL database using the appropriate methods provided by the MySQL module.

When executing queries, it's important to handle the query results and any errors that may occur during the execution. You can process the query results and perform any necessary actions based on the returned data. Additionally, proper error handling ensures that your application can respond appropriately in case of any issues.

After executing a query, it's good practice to release the connection back to the connection pool. Releasing the connection allows it to be reused by other parts of your application, improving resource utilization.

If you no longer need to perform any further database operations, you can choose to close the connection pool to release all associated resources.

By following these steps and implementing best practices, you can successfully connect Node.js with MySQL, allowing your application to interact with the database efficiently and effectively.

1. Building the Node.js API
2. Creating the Project Structure

Creating the Project Structure: Set up the project structure for your Node.js API. This typically involves creating directories for different components such as routes, controllers, models, and utilities. Organizing your codebase helps maintain a clean and manageable structure.

1. Setting up Express.js

Setting up Express.js: Install and set up Express.js, a popular web framework for Node.js. Express.js simplifies the process of creating APIs by providing a robust set of tools and middleware. Configure your Express.js application to handle HTTP requests and responses.

1. Establishing Database Connection

Establishing Database Connection: Connect your Node.js API to the MySQL database established earlier. Utilize the MySQL module or library to establish a connection and handle database operations. Set up the necessary configuration parameters such as the host, username, password, and database name to establish the database connection.

1. Creating API Endpoints

Creating API Endpoints: Define the API endpoints that your application will expose to handle various client requests. These endpoints are typically defined in the routes directory and mapped to appropriate controller functions. Create routes for different HTTP methods (GET, POST, PUT, DELETE) and define the corresponding logic in controller functions to interact with the database, process data, and send responses back to the client.

1. Handling HTTP Requests and Responses

Handling HTTP Requests and Responses: Implement the logic in the controller functions to handle incoming HTTP requests and generate appropriate responses. This involves processing request data, validating input, interacting with the database, and constructing the response objects.

Ensure proper error handling by catching and handling exceptions. Return the appropriate HTTP status codes and response bodies based on the outcome of the request. You can also apply middleware functions at the route or application level to handle tasks like authentication, input validation, and request logging. By following these steps, you can build a robust Node.js API that handles HTTP requests, communicates with the database, and generates appropriate responses for your client applications.

1. Developing the Angular Frontend
2. Creating Components and Services

Creating Components and Services: Start by creating the necessary components and services in your Angular project. Components are the building blocks of your user interface, while services handle data retrieval, API calls, and other business logic. Organize your components and services based on the features and functionality of your application.

Components: Components in Angular are reusable building blocks that encapsulate the presentation and behaviour of a part of your user interface. They consist of a TypeScript class that defines the component's properties and methods, and an HTML template that determines how the component is rendered. Components represent different parts of your application's UI, such as header, sidebar, forms, or individual sections of a page.

To create a component, you use the Angular CLI (Command Line Interface) command ng generate component or the shorthand ng g c. This command generates the necessary files and folders for your component, including the TypeScript class, HTML template, CSS styles, and a spec file for testing.

Services: Services in Angular are responsible for handling data retrieval, API calls, and other business logic that is not specific to a particular component. Services are typically used to centralize data access and provide a common set of functionality across multiple components. They help keep components focused on their specific UI responsibilities and promote code reusability and maintainability.

To create a service, you can use the Angular CLI command ng generate service or ng g s. This command generates a TypeScript class for the service. Services can be injected into components or other services through dependency injection, allowing them to be easily shared and accessed.

1. Building User Interfaces with Angular Templates

Building User Interfaces with Angular Templates: Use Angular templates to define the structure and layout of your user interfaces. Templates combine HTML markup with Angular directives, allowing you to dynamically display data and interact with components. Design and implement the user interface components to provide an intuitive and visually appealing experience.

1. Implementing Data Binding and Event Handling

Data binding is a fundamental concept in Angular that allows you to establish a connection between your component's properties and the user interface elements. This enables you to display and update data dynamically without manual intervention. Angular provides several types of data binding, including one-way binding, two-way binding, and event binding.

One-way data binding allows you to display component data in the user interface. You can bind a component property to an HTML element using interpolation, property binding, or attribute binding. Interpolation uses double curly braces ({{ }}) to directly display the value of a component property within the HTML markup. Property binding uses square brackets ([ ]) to bind the value of a component property to an HTML attribute or property. Attribute binding uses square brackets as well, but it allows you to bind a component property to an HTML attribute directly.

Two-way data binding combines the features of one-way data binding and event binding. It establishes a bidirectional data flow between the component and the user interface element. Changes in the user interface update the component property, and changes in the component property update the user interface. Two-way data binding is achieved using the ngModel directive, which is commonly used with form inputs like textboxes, checkboxes, and dropdowns.

Event binding allows you to respond to user interactions, such as button clicks, mouse events, or keyboard events. You can bind a component method to an HTML event using parentheses (()). When the event occurs, the associated component method is executed, allowing you to perform specific actions or update data accordingly.

By leveraging data binding and event handling, you can create dynamic and responsive user interfaces in Angular. The data binding mechanisms ensure that your UI reflects the latest component data, while event handling allows you to capture user interactions and trigger appropriate actions or data updates within your application.

1. Integrating API Calls with the Backend

Integrating API Calls with the Backend: Use Angular's HTTP module or a dedicated HTTP client library (e.g., HttpClient) to make API calls to your backend server. Create service methods to interact with the backend API, handle request parameters and headers, and process responses. Map the API responses to appropriate data structures and update the UI accordingly.