**Beginner Guide for Full Stack Web Development**

Table of Contents

[Purpose of the Document: 3](#_Toc151132298)

[1.Introduction about Full stack web development 3](#_Toc151132299)

[1.1 What is Web development? 4](#_Toc151132300)

[1.2 What is Front End? 4](#_Toc151132301)

[1.3 What is a Back End? 4](#_Toc151132302)

[1.4 What is Middleware? 4](#_Toc151132303)

[1.5 What is a Full Stack? 5](#_Toc151132304)

[1.6 Application Data Flow: 5](#_Toc151132305)

[2. Technology & Software Tools: 5](#_Toc151132306)

[3. Sample Demo Application: 5](#_Toc151132307)

[3.1 TMS Business Requirement: 6](#_Toc151132308)

[3.2 Features 6](#_Toc151132309)

[4. Application Architecture 11](#_Toc151132310)

[4.1 Layer Explanation: 11](#_Toc151132311)

[a) Data Store Layer 11](#_Toc151132312)

[b) Data Access Layer 11](#_Toc151132313)

[c) Business Layer 12](#_Toc151132314)

[d) Data Service Layer 12](#_Toc151132315)

[e) Presentation Layer 12](#_Toc151132316)

[4.2 TMS Architecture Diagram: 13](#_Toc151132317)

[5. Development Process 13](#_Toc151132318)

[5.1 Allocation of user stories: 15](#_Toc151132319)

[5.2 Analysis& Design Document Preparation: 15](#_Toc151132320)

[5.3 Review & Approval: 15](#_Toc151132321)

[5.4 Unit Testing: 15](#_Toc151132322)

[5.5 Feature Demo: 16](#_Toc151132323)

[5.6 Code Commit to feature branch: 16](#_Toc151132324)

[5.7 PR Raise to develop branch: 16](#_Toc151132325)

[5.8 Code Review, Approval & Code Merge: 17](#_Toc151132326)

[5.9 Development Release: 17](#_Toc151132327)

[5.10 Verification of feature in Development server: 17](#_Toc151132328)

[5.11 Update of a DevOps Task: 17](#_Toc151132329)

[5.12 Test Release Document preparation: 18](#_Toc151132330)

[6. How to develop a TMS? 18](#_Toc151132331)

[6.1 Database Design: 18](#_Toc151132332)

[6.2 Feature Spec Preparation: 21](#_Toc151132333)

[7. Debugging 45](#_Toc151132334)

[7.1 Rest Data Access: 46](#_Toc151132335)

[7.2 Rest Business 46](#_Toc151132336)

[7.3 Gateway 46](#_Toc151132337)

[7.4 Web Client 46](#_Toc151132338)

[8.Integration Testing: 47](#_Toc151132339)

[9.Release & Deployment: 47](#_Toc151132340)

[10.Feature Demonstration Guideline: 47](#_Toc151132341)

[Conclusion: 48](#_Toc151132342)

# Purpose of the Document:

Any individual interested in developing a web application using the Full Stack web development process can refer to this material without skipping any sections.

1.Introduction about Full stack web development

## 1.1 What is Web development?

* Web development is the process of producing internet-accessible websites and web applications.
* It includes a wide range of activities and skills related to the creation and maintenance of websites, such as web design, web programming, web content development, client-side and server-side scripting, and network security setup.

## 1.2 What is Front End?

* This includes designing a website's user interface and user experience (UI/UX).
* Front-end developers work on a website's visual features, such as layout, design, and interactivity.
* They create the client-side of web applications that run in a user's web browser using technologies like HTML, CSS, and JavaScript.

## What is a Back End?

* The back-end layer is like the brain of a system.
* It manages the database, which is like a memory, and checks that the information is stored.

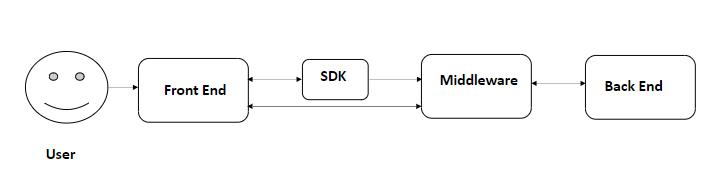
## 1.4 What is Middleware?

* API is an abbreviation for **"Application Programming Interface."**
* They oversee data processing, user authentication, and server-side scripting via the server, database, and application logic.
* We have middleware as Data-Access-Layer, Business-Layer and Gateway-Layer
* We use JavaScript and Node.js, the programming languages we use for back-end development.
* It's a set of rules and protocols that enable one software application to communicate with another.
* APIs specify the techniques and data formats that developers can use to request and exchange data between software systems. An API, in essence, acts as a bridge that allows different apps to interact and collaborate, even if they were developed by separate teams or companies.
* Simply the communication between back end and front-end.

## 1.5 What is a Full Stack?

* A full-stack developer is skilled in both front-end and back-end programming.
* They can help with everything from developing user interfaces to managing server databases and writing server-side software.

## 1.6 Application Data Flow:



The middleware layer acts like a bridge between your application and the database, organizing data access, managing business logic, and connecting to the database through a gateway.

# 2. Technology & Software Tools:

* Node – V18.16.0
* Postman – V10.19.7
* Microsoft Visual Studio Code – V1.83.1
* swagger editor online - V3.1.0
* pgAdmin - v7.8
* postgreSQL - v16.0

# 3. Sample Demo Application:

This Full Stack Developer Guide can assist in creating a prototype demo application called Task Management System (TMS). TMS serves as a guide for developers and people who are interested in developing a model application.

## 3.1 TMS Business Requirement:

A business requirement is nothing more than the customer's business needs. It refers to the expectations and conditions that need be met or satisfied to achieve the objectives of a business for building an application.

## 3.2 Features

A feature refers to the valuable functionality of a product that delivers specific benefits or serves a particular purpose.

It includes functionalities like a user login, a search bar etc. within an application.

**Task** and **Search** are the two features. Task performs various functions such as CRUD operations, while Search retrieves the required data.

**3.2.1 Task**

This feature is used to create tasks that the user can manage, assign to team members, and monitor. It also allows the user to analyze their work items daily.

**A) Functionality: Create Task**

**Objective:**

The user can create a task and assign it to another team member. So, that the tasks can be tracked.

**Business Rules:**

* Users must have the capability to create tasks, and the system should enforce the prevention of task duplication.
* The task duplication validation is done by the title field and the assigned person.
* The user cannot be able to create more than one task with the same title to the same person.

**Acceptance Criteria:**

* Users should have the capability to create tasks through a user-friendly interface within the system.
* Upon task creation, the system should promptly store the task details in the database.
* The system must provide a table view that displays all created tasks, including the newly created task.
* The table view should present relevant details of each task, such as title, description, and other key information.
* The task information displayed in the table view should accurately match the details provided during task creation.
* The table view should be dynamically updated to include the latest task entries.

**B) Functionality: View Task**

**Objective**

Users can view the created tasks in the table view.

**Business Rules**

* The system must present a comprehensive grid-formatted table that displays all created tasks.
* This table should provide a consolidated view of relevant task details, allowing users to easily access and review information for effective task management.

**Acceptance Criteria**

* The system should provide users with a dedicated section or interface to view the list of tasks created.
* The list of tasks should be presented in a clear and organized manner, displaying relevant details such as title, description, and other key information.
* The task list should include all tasks created by the user, and it should dynamically update to reflect any additions or changes.
* Each task in the list should be uniquely identified and selectable for further interaction or review.
* The displayed information for each task should accurately match the details provided during task creation.
* The system should allow users to sort or filter the task list based on different criteria, enhancing the usability of the task viewing functionality.

**C) Functionality: Edit Task**

**Objective**

The user can be able to edit the task details which are created.

**Business Rules**

* The system shall facilitate user-editable task values, allowing users to modify all the fields during task updation.
* Furthermore, these edits should be seamlessly reflected in the database, ensuring accurate and up-to-date data storage.

**Acceptance Criteria**

* The system must provide users with functionality to edit tasks that have been created.
* Upon selecting the task for editing, the system should present an editable form or interface containing the existing task details.
* Users should be able to modify the relevant information for the selected task, such as title, description, or any other editable fields.
* After editing, the system should promptly update and store the changes made to the task in the database.
* A confirmation message or notification should be displayed to the user, confirming the successful editing of the task.
* The edited task details should be accurately reflected in any views or interfaces displaying the task information.
* Users should have the flexibility to make multiple edits to the same task, with the system consistently updating and storing the latest changes.

**D) Functionality: Delete Task**

**Objective**

The user can delete the task if it is created mistakenly or not necessary.

**Business Rules**

* The system should empower users with the ability to delete tasks in the event of accidental creation.
* This feature ensures that users can promptly rectify mistakes by removing erroneously created tasks from the system, contributing to a more flexible and user-friendly task management experience.

**Acceptance Criteria**

* The system should include a feature allowing users to delete tasks that were created mistakenly.
* Users must be able to locate and select the task they wish to delete within the system.
* Upon selecting the task for deletion, the system should prompt the user for confirmation to ensure intentional deletion.
* Upon confirmation, the system should promptly and should not permanently remove the selected task from the database.
* A confirmation message should be displayed to the user, confirming the successful deletion of the task.
* Deleted tasks should no longer be visible in any task listings.
* The system should prevent accidental deletion by implementing appropriate safeguards, such as confirmation dialogs or authorization checks.

**3.2.2 Search Task**

The user can be able to search the task by using the task code field. The output search data will be shown in the table. If it is not available, then it should show no data found.

**Objective**

The user can search for a task with task code.

**Business Rules**

* The system should let users easily find tasks by entering their unique task codes in a search field.
* If no match is found, clear indications should be provided.
* This feature is intended to make it straightforward for users to locate tasks using their task codes.

**Acceptance Criteria**

* The system should provide a search functionality allowing users to find a specific task using its unique task code.
* Users should be able to input the task code in a designated search field within the system's user interface.
* The search mechanism should be case-insensitive, ensuring flexibility in entering task codes.
* Upon initiating the search, the system should promptly retrieve and display the task matching the entered task code.
* If no matching task is found, the system should provide a clear indication that no results were found.
* The search results should include relevant details of the task, such as title, description, and other key information.
* Users should be able to click on the search result to access detailed information about the selected task.

# 4. Application Architecture

The Application Consists of a total of 5 layers.

## 4.1 Layer Explanation:

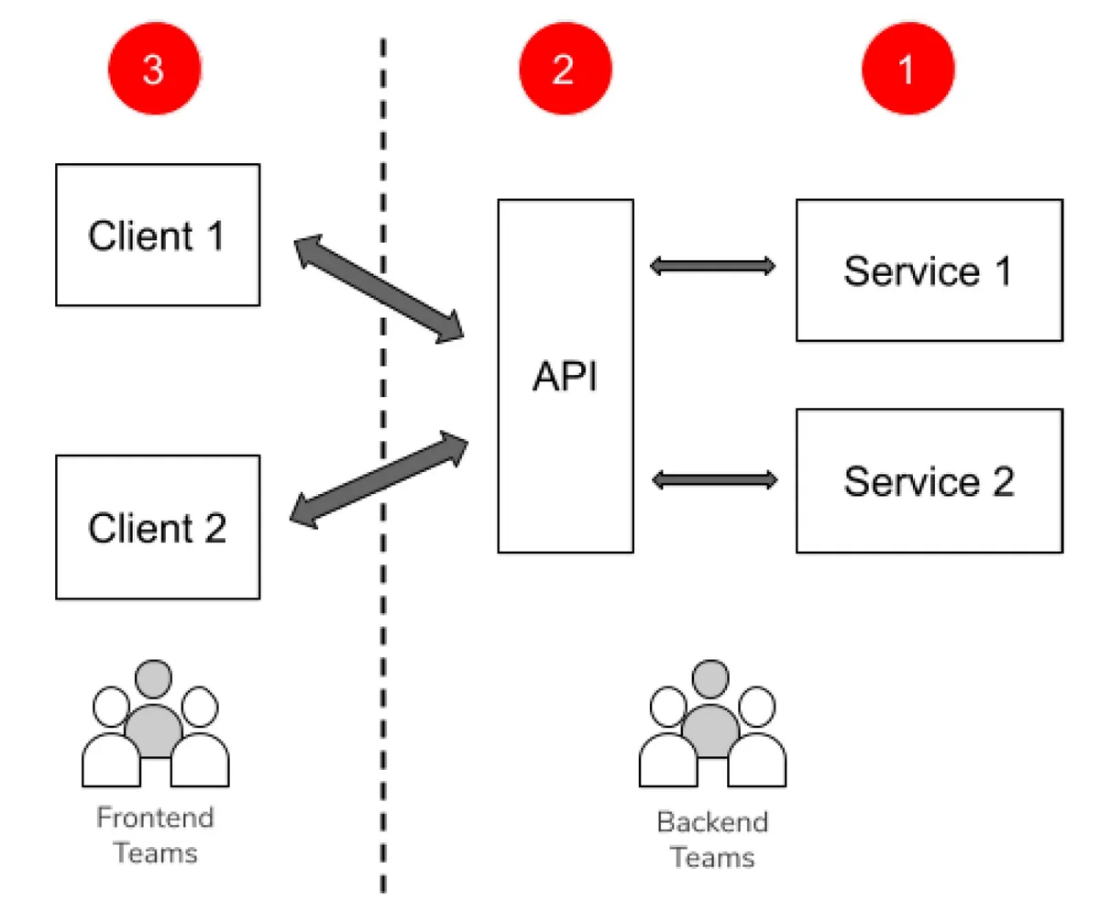
## a) Data Store Layer

This layer is responsible for storing user data in the database, ensuring that all relevant

information is securely stored and easily retrievable.

## b) Data Access Layer

* This layer both retrieves and stores user information in the database, managing the seamless exchange of data between the user interface and the database storage system.
* API functions similarly to a hotel server. Assume you are the client (UI), and you are requesting some dish(data) to the server (API), which receives the input and finds the dish(data) from the kitchen(database) and returns it to the correct requested client.



## c) Business Layer

The business layer houses application-specific business logic, such as duplicate validation rules, ensuring that essential business rules and functionalities are implemented within the application.

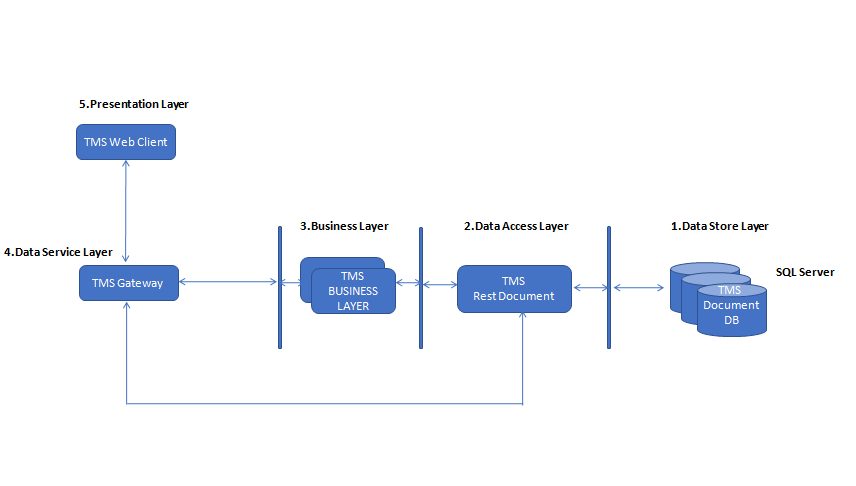
## d) Data Service Layer

* A gateway serves as a connection point between the business layer and the data access layer, acting as an intermediary that is exclusively exposed and independently connects to each respective service.
* The gateway has the capability to directly communicate with both the business layer and the data access layer.

## e) Presentation Layer

* In the Presentation layer, users exclusively interface with the application, visualizing data retrieved from the database through an Application Programming Interface (API).
* This layer ensures a seamless user experience, shielding the complexities of data retrieval and manipulation, while the API facilitates efficient communication between the user interface and the underlying database. Bi-directional communication is facilitated seamlessly across all layers, allowing for effective interaction between the Web-client gateway, business layer, and data access layer.

## 4.2 TMS Architecture Diagram:



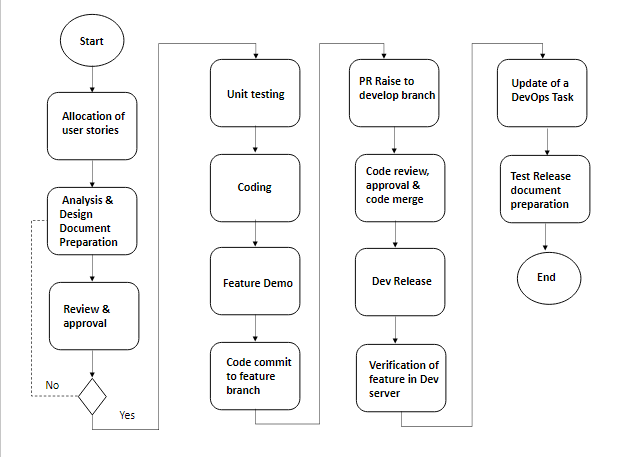
# 5. Development Process

* The Development Process Flowchart is to streamline work processes, improve productivity, and ensure consistent and standardized execution of work across the Engineering Team.
* Developers should only code after the unit testing code is completed.

**In a TDD approach, the development process typically follows these steps:**

1. **Write a Test:** Before implementing a new feature or fixing a bug, developers write a test that defines the expected behavior of the code.
2. **Run the Test:** Execute the test to ensure that it fails, as the corresponding code has not been implemented yet.
3. **Write Code:** Implement the code necessary to pass the test.
4. **Run Tests Again:** Execute all tests, including the newly written one, to verify that the changes did not introduce regressions.
5. **Refactor (if needed):** Refactor the code for better design or efficiency while ensuring that all tests still pass.
6. **Repeat:** Repeat the cycle for each new feature or code change.

* By emphasizing unit testing before coding, TDD aims to improve code quality, promote early bug detection, and ensure that the codebase remains maintainable and reliable over time.
* However, the strict order suggested in the statement might not be universally applicable, as development methodologies and project requirements can vary. Some developers may choose to integrate testing at different stages, and the adoption of TDD is a matter of preference and project context.



## 5.1 Allocation of user stories:

* The allocation of user stories is a crucial step in the development process.
* It involves the distribution of specific user stories to individual team members.
* User stories represent the pieces of functionality, features from the perspective of end-users.
* If the assigned user story is not understood, communicate to the product analyst for better clarification.

## 5.2 Analysis& Design Document Preparation:

* During this phase, the development team examines the project's requirements to create a comprehensive plan that guides the entire development process.
* During the analysis phase, the development team engages in in-depth discussions with stakeholders and subject matter experts to gather a clear understanding of the feature objectives, functionalities, and constraints.
* This process involves showcasing both functional and non-functional requirements, which outline what the feature should achieve and how it should perform.
* This document encapsulates the comprehensive analysis of requirements, functional and architectural design that outlines a clear roadmap for the development.

## 5.3 Review & Approval:

* At this stage, the team responsible for creating the deliverables submits their work to the Development Lead responsible for reviewing and approving the work.
* Team lead must review the document and guide the developers to work on changes if needed.
* Following the document modification, implementation can start.

## 5.4 Unit Testing:

* The developer must create a unit test case during the analysis and design stage.
* Each unit test should focus on a specific unit of code and a particular aspect of its functionality.
* By catching defects early, unit testing contributes to efficient development processes.

## 5.5 Feature Demo:

* During the development phase, it is the developer's responsibility to present a demo.
* Both the product analyst and the development lead get involved in the demo.
* They should feel completely satisfied with the features that were developed.
* This reduces bugs and makes it easier for developers to understand what is expected from them.

## 5.6 Code Commit to feature branch:

* Once a new feature or task has been completed, developers commit the developed code in their dedicated feature branch in the Azure DevOps.
* Developers write and commit code changes directly to the feature branch and not in the develop branch.
* Each commit should be focused on a specific logical change and come with a clear commit.
* If your changes affect any documentation, such as README files, make sure to update them accordingly to reflect the latest changes accurately.

## 5.7 PR Raise to develop branch:

* A pull request (PR) is a way to propose and discuss changes made in a feature branch before integrating them into the development branch.
* Initiate a pull request from your feature branch to the development branch.
* Provide a clear and informative title that summarizes the purpose of the pull request.
* In the PR description, provide a detailed explanation of the changes you have made.
* Assign one or more team members as reviewers for your pull request. Reviewers will review your code.

## 5.8 Code Review, Approval & Code Merge:

* To ensure that the code meets established coding standards, is well-designed, and free from logical errors.
* Code Merge also known as code integration, involves combining individual code changes from multiple feature branches into the Develop branch.

## 5.9 Development Release:

* The code that the developer raises should be reviewed by the development lead.
* Once the PR is accepted by the lead, Code will be automatically deployed in the development server followed by CI/CD process.

## 5.10 Verification of feature in Development server:

* Once the code has been sent to the development server, the developer must verify that it is really deployed.
* To ensure that the developed functionality works as expected.
* It should fulfil the functionality that the customer has requested.

## 5.11 Update of a DevOps Task:

* Developer should work for the user story assigned by the Product analyst/ Scrum Master.
* Initially the user story of the status will be in New.
* When the developer begins working on a specific story, it should be updated as In-Progress.
* Once the developed feature functions perfectly working in the development server, upgraded its status to closed.
* The task's discussion points must be updated regularly for the reporting person to see the task's current state.

## 5.12 Test Release Document preparation:

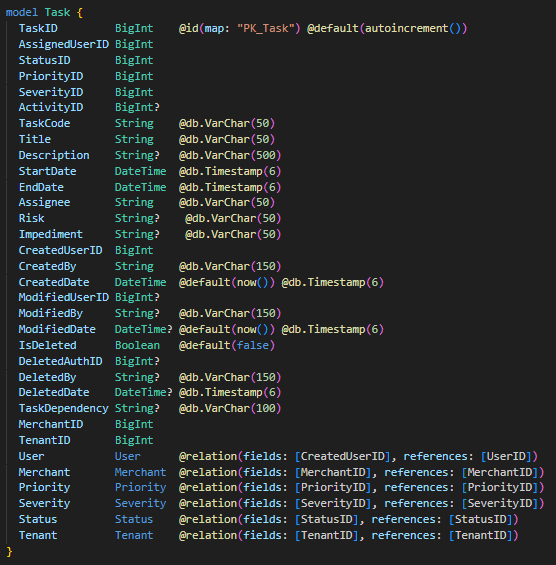
* The developer is responsible for creating the test release document after the task is closed.
* This release document serves as a means of deploying the most recent code to the testing environment.
* This document contains: Feature, components need to deploy, release branch name and roll back plan.

# 6. How to develop a TMS?

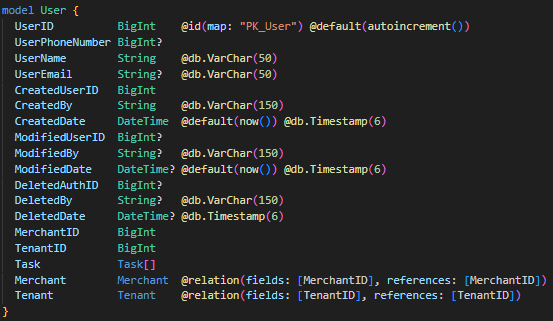
## 6.1 Database Design:

**6.1.1 Table Definition**

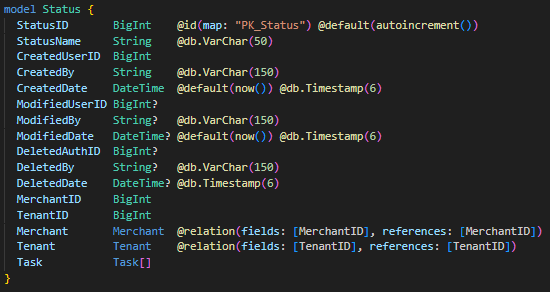
**a) Task Table:**



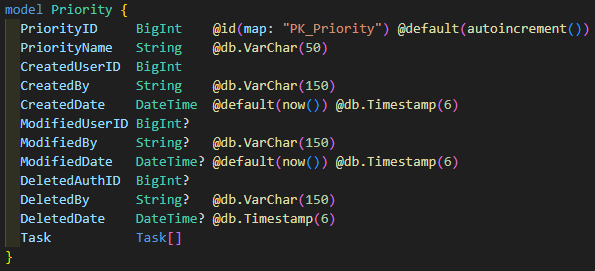
**b) User Table:**



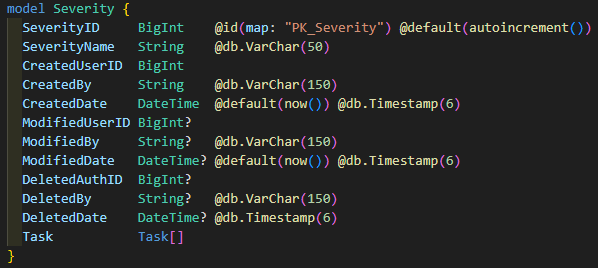
**c) Status Table:**



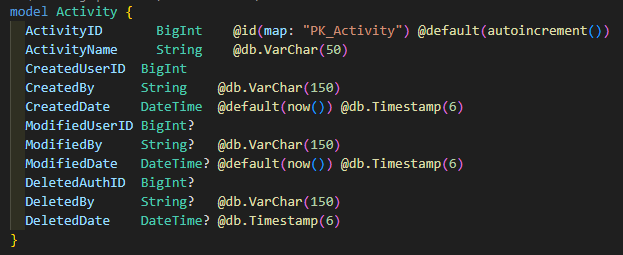
**d) Priority Table:**



**e) Severity Table:**



**f) Activity Table:**



* Once the schema is finalized put the schema in the Prisma file of the data access layer base code.
* Then run the command **“npm i".**
* Run the command **“npm run schema-generate".**
* After that run the command “**npm run schema-push".**

## 6.2 Feature Spec Preparation:

* The Design Specification is a crucial step-by-step guide for our developers, dividing the system into API, Business, and Web client.
* It ensures clear understanding, avoiding confusion during development, covering details like API functions, data organization, and security measures.
* The Business part deals with how the app works, including logic, error handling, and data retrieval.
* The Web client section provides a user-friendly interface plan. We've established communication rules between different parts and emphasized the importance of clear explanations.
* This guide not only directs developers but also assists in making necessary changes for a successful feature. Additionally, we define endpoints and URLs for each API and business operation.

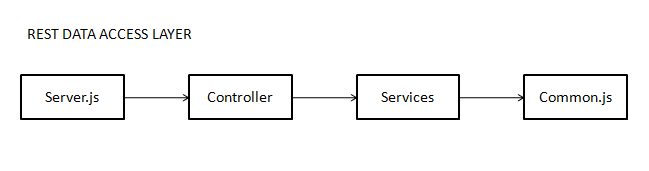
**6.2.1 Task Feature:**

**a.1) Specification Document**



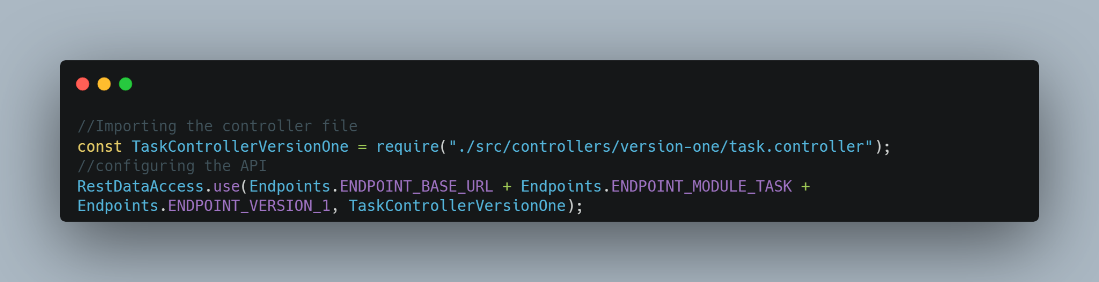
**b.1) Data Access Layer**

* The data access layer serves as the intermediary between the database, handling tasks such as connecting, storing, and retrieving data, and the web client, facilitating the seamless transfer of data to and from the database for presentation to users.



**Server.js File:**

* **Role**: Acts as the entry point for the server application.
* **Functionality**: Imports and calls the controller file.
* **Explanation**: The server.js file is responsible for setting up the server and initiating the application. It imports the controller file, which contains the logic for handling various API requests.



In the above snippet, we import the necessary API and files.

**Controller File:**

* **Role**: Manages the flow of control between the server.js file and the service file.
* **Functionality**: Contains functions that correspond to different API routes or methods and calls the appropriate functions in the service file.
* **Explanation**: The controller file acts as an intermediary between the server.js file and the service file. It organizes the application's logic into functions, each of which corresponds to a specific API endpoint or functionality. These functions call the corresponding methods in the service file to perform the actual operations.

This is a sample controller file where we invoke each API.

**Service File:**

* **Role**: Implements the actual functionality of the API.
* **Functionality**: Contains the business logic for each API endpoint or method.
* **Explanation**: The service file holds the core functionality of the API.
  + - It is where the actual processing of API requests takes place.
    - Each method in the service file corresponds to a specific API endpoint, performing the necessary operations, such as interacting with a database, processing data, or executing other business logic.

**Create Task Functionality:**



**Read Tasks Functionality:**



**Read One task Functionality:**



**Update Tasks Functionality:**



**Delete Tasks Functionality:**



In the provided snippets, you can observe examples of how to write APIs for a specific table task.

**Common.js File:**

* It exports functions or objects using module.exports, making them accessible to the controller module.
* The CommonJS pattern helps structure the API codebase, making it modular and organized, with clear responsibilities assigned to different parts of the application.
* In summary, this architecture follows a common pattern in Node.js API development where the server.js file initializes the server, the controller file manages the flow of control, and the service file contains the business logic for the API.
* This separation of concerns helps in organizing code, improving maintainability, and adhering to the principles of modularity and abstraction.

**Endpoint.js File:**

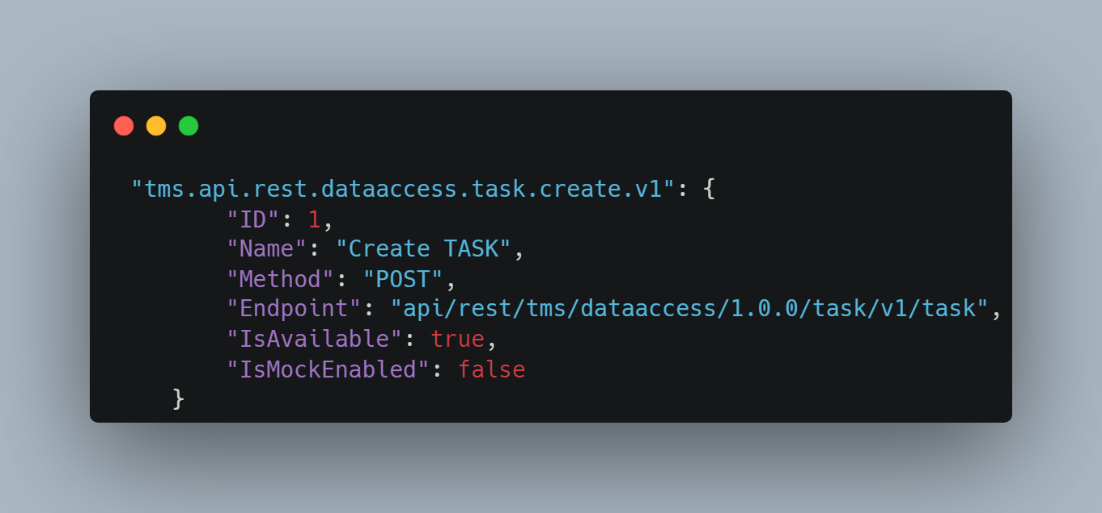
The 'Endpoint.js' file houses the necessary endpoints required for the API.

**Message.js File:**

"The 'message.js' file contains the necessary messages required for the API responses."

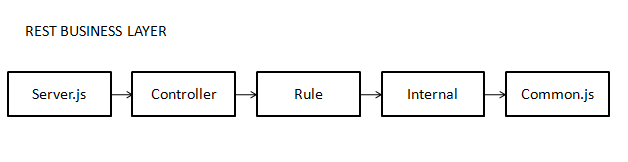
**API.Config File:**

The 'api.config' file includes all the API configurations, allowing for the enabling and disabling of specific APIs.



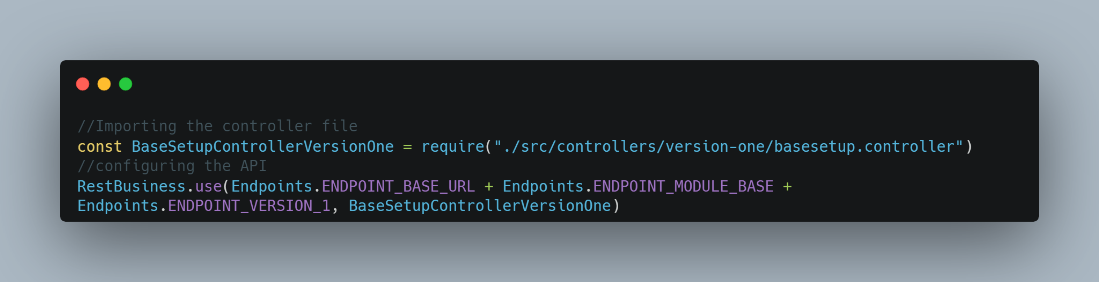
**c.1) Rest Business Layer**

In the business layer, we incorporate essential business logic, such as duplicate validations, to ensure the integrity and correctness of data processing.



**Server.js:**

* **Role**: Acts as the entry point for the server application.
* **Functionality**: Imports and calls the controller file.
* **Explanation**: The server.js file is responsible for setting up the server and initiating the application.
  + It imports the controller file, which contains the logic for handling various API requests.



In the above snippet, we import the necessary API and files.

**Controller File:**

* **Role**: Manages the flow of control between the server.js file and the service file.
* **Functionality**: Contains functions that correspond to different API routes or methods and calls the appropriate functions in the service file.
* **Explanation**: The controller file acts as an intermediary between the server.js file and the service file. It organizes the application's logic into functions, each of which corresponds to a specific API endpoint or functionality.
  + These functions call the corresponding methods in the service file to perform the actual operations.



**Rule File:**

* **Role**: Implements the actual functionality of the API.
* **Functionality**: Contains the business logic for each API endpoint or method.
* **Explanation**: The service file holds the core functionality of the API. It is where the actual processing of API requests takes place. Each method in the service file corresponds to a specific API endpoint, performing the necessary operations, such as interacting with a database, processing data, or executing other business logic.



This Function calls the Api for Base Setup

**Internal file:**

The Internal file maintains a record of the APIs being called from the data access layer.



This field is used to call the necessary API From Data Access Layer

**Common.js:**

* It exports functions or objects using module.exports, making them accessible to the controller module.
* The CommonJS pattern helps structure the API codebase, making it modular and organized, with clear responsibilities assigned to different parts of the application.

**Endpoint.js:**

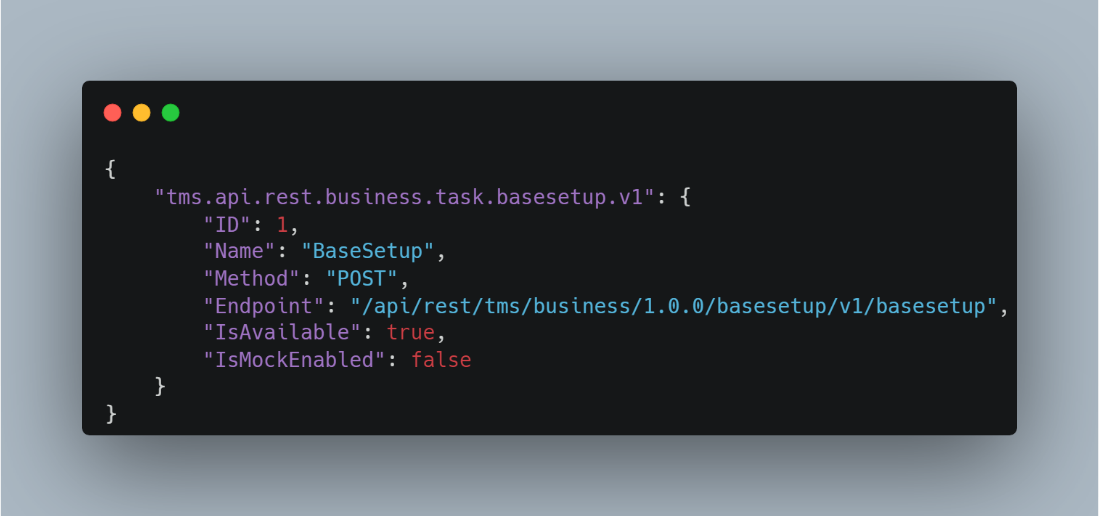
The 'Endpoint.js' file houses the necessary endpoints required for the API.

**Message.js:**

"The 'message.js' file contains the necessary messages required for the API responses."

**API.Config:**

The 'api.config' file includes all the API configurations, allowing for the enabling and disabling of specific APIs.



**d.1) Gateway Layer**

**Communication Hub**:

* The gateway serves as a central point of communication, facilitating interactions between different components of the system.
* This includes handling requests from web clients and managing the flow of data between the client, business layer, and REST data access layer.

**Internal Processing**:

* The gateway is responsible for coordinating and managing internal processing.
* This involves running the business logic and interacting with the REST data access layer to retrieve or manipulate data.

**Service Exposure**:

* The gateway exposes services to external entities, such as web clients.
* This means that external systems or clients interact with the system primarily through the gateway.
* The gateway abstracts the complexities of the internal architecture, providing a simplified and standardized interface for external consumers.

**Layered Architecture:**

* Your description implies a layered architecture where the gateway sits at the forefront, interfacing with the web client, and orchestrating the flow of requests to the underlying business and data access layers.
* This separation of concerns helps in maintaining a modular and scalable system.

**Security and Access Control**:

* The gateway can play a role in enforcing security measures, such as authentication and authorization.
* It ensures that only authorized users or systems can access the services provided by the business and data access layers.

**Abstraction of Complexity**:

* By exposing services only through the gateway, you create a level of abstraction.
* This allows you to modify or enhance the internal implementation of the business and data access layers without affecting the external interfaces.
* Clients interact with the system through a stable and well-defined gateway interface.

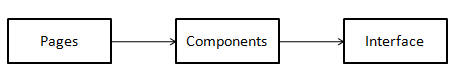
The gateway in your system architecture acts as a communication hub, manages internal processing, exposes services to external clients, and provides a layer of abstraction for the underlying components. This architectural pattern helps in creating a more modular, secure, and maintainable system.



The "gateway.config.yml" file consolidates configuration settings for both the data access layer and the business layer, encompassing the respective endpoints for each layer.

**e.1) Web-Client**

UI functionalities for Web-Client



**Pages:**

* Pages typically refer to the individual views or components that represent different sections or routes of a web application.
* React is often used to build single-page applications (SPAs), where the content of the web page is dynamically updated without requiring full page reloads.
* Each page in a React application corresponds to a specific URL route and is responsible for rendering the content and functionality associated with that route.
* Pages are a way to modularize your application. Each page component is responsible for a particular section of your website.

**Components**:

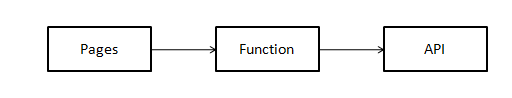
* Components are a fundamental building block of the user interface.
* Components are reusable, self-contained pieces of code that define how a part of the user interface should look and behave.
* React applications are typically constructed by composing various components together.
* Components are designed to be reusable. You can create a component once and use it multiple times throughout your application.
* Each component is responsible for a specific part of the user interface or a particular feature.
* You have a top-level component often referred to as the "App" component that contains or renders other child components.
* Components can also have their own internal state, which represents data that can change over time.

**Interface**:

* Interfaces are used to define the structure and types of data that components and other parts of your application expect or provide.
* Interfaces are a way to define the contract of what data should look like.
* Interfaces in a React web client is for defining the structure of props that a component expects.



API Functionalities from Web-client to Database



**Pages**:

In the pages, we call components that, in turn, connect to the API using functions. The data fetched is then retrieved from the database and displayed in the UI for the user.

**Function**:

The function includes the API query necessary to fetch data related to the information that should be displayed to the user.



**API**:

The API file comprises all the required APIs needed in the UI to retrieve data based on the user's request.



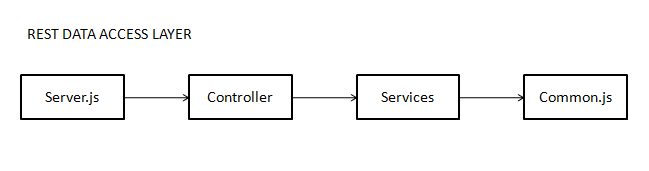
**6.2.2 Search Feature:**

**a.1) Specification Document**

****

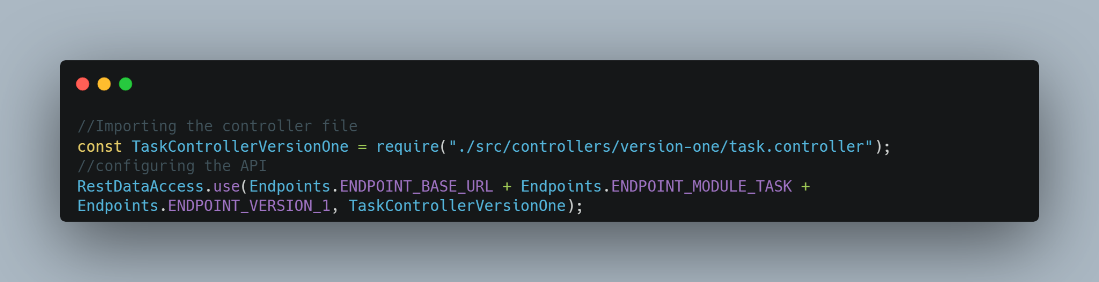
**b.1) Data Access Layer**

The data access layer serves as the intermediary between the database, handling tasks such as connecting, storing, and retrieving data, and the web client, facilitating the seamless transfer of data to and from the database for presentation to users.



**Server.js:**

* **Role**: Acts as the entry point for the server application.
* **Functionality**: Imports and calls the controller file.
* **Explanation**: The server.js file is responsible for setting up the server and initiating the application. It imports the controller file, which contains the logic for handling various API requests.



In the above snippet, we import the necessary API and files.

**Controller File:**

* **Role**: Manages the flow of control between the server.js file and the service file.
* **Functionality**: Contains functions that correspond to different API routes or methods and calls the appropriate functions in the service file.
* **Explanation**: The controller file acts as an intermediary between the server.js file and the service file. It organizes the application's logic into functions, each of which corresponds to a specific API endpoint or functionality. These functions call the corresponding methods in the service file to perform the actual operations.

This is a sample controller file where we invoke each API.

**Service File:**

* **Role**: Implements the actual functionality of the API.
* **Functionality**: Contains the business logic for each API endpoint or method.
* **Explanation**: The service file holds the core functionality of the API. It is where the actual processing of API requests takes place. Each method in the service file corresponds to a specific API endpoint, performing the necessary operations, such as interacting with a database, processing data, or executing other business logic.

**Read Tasks**



In the provided snippets, you can observe examples of how to write APIs for a specific table task.

**Common.js:**

* It exports functions or objects using module.exports, making them accessible to the controller module.
* The CommonJS pattern helps structure the API codebase, making it modular and organized, with clear responsibilities assigned to different parts of the application.

In summary, this architecture follows a common pattern in Node.js API development where the server.js file initializes the server, the controller file manages the flow of control, and the service file contains the business logic for the API. This separation of concerns helps in organizing code, improving maintainability, and adhering to the principles of modularity and abstraction.

**Endpoint.js:**

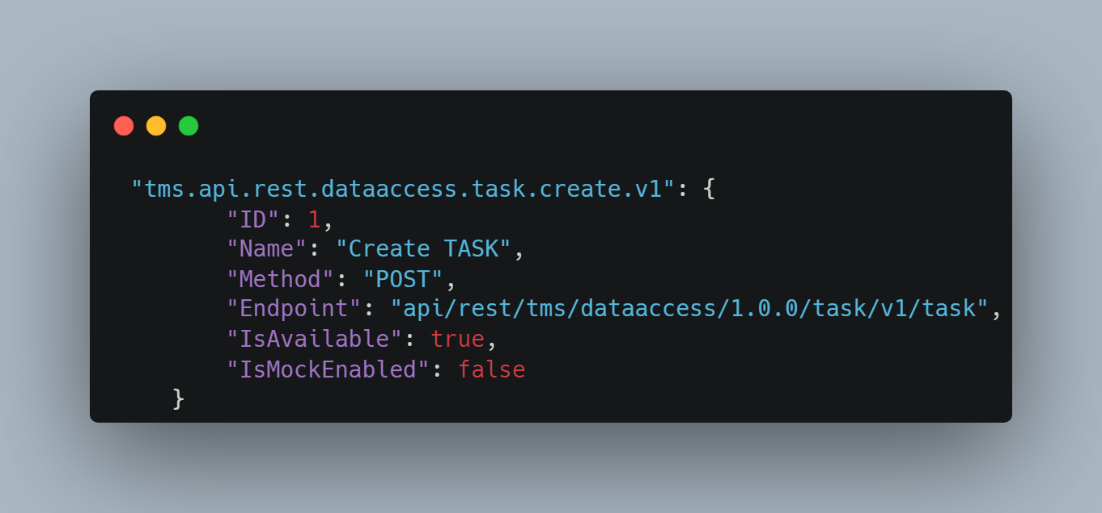
The 'Endpoint.js' file houses the necessary endpoints required for the API.

**Message.js:**

"The 'message.js' file contains the necessary messages required for the API responses."

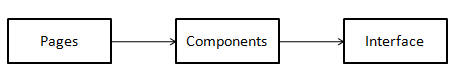
**API.Config:**

The 'api.config' file includes all the API configurations, allowing for the enabling and disabling of specific APIs.



**c.1) Web-Client**

UI functionalities for Web-Client



**Pages**:

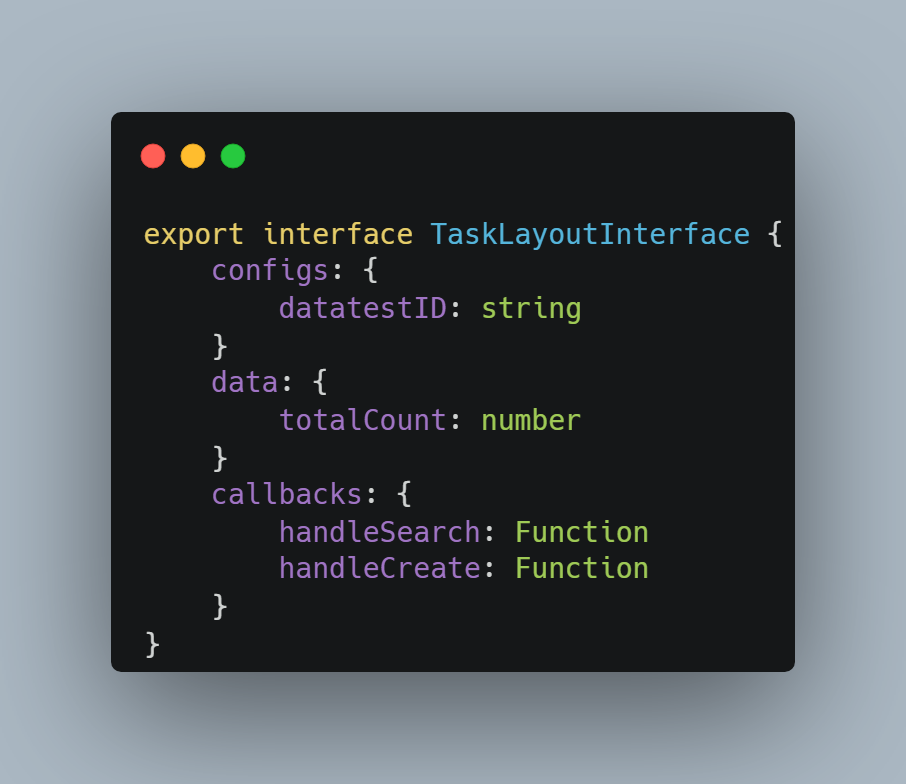
* Pages typically refer to the individual views or components that represent different sections or routes of a web application.
* React is often used to build single-page applications (SPAs), where the content of the web page is dynamically updated without requiring full page reloads.
* Each page in a React application corresponds to a specific URL route and is responsible for rendering the content and functionality associated with that route.
* Pages are a way to modularize your application. Each page component is responsible for a particular section of your website.

**Components**:

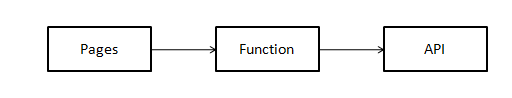
* Components are a fundamental building block of the user interface.
* Components are reusable, self-contained pieces of code that define how a part of the user interface should look and behave.
* React applications are typically constructed by composing various components together.
* Components are designed to be reusable. You can create a component once and use it multiple times throughout your application.
* Each component is responsible for a specific part of the user interface or a particular feature.
* You have a top-level component often referred to as the "App" component that contains or renders other child components.
* Components can also have their own internal state, which represents data that can change over time.

**Interface**:

* Interfaces are used to define the structure and types of data that components and other parts of your application expect or provide.
* Interfaces are a way to define the contract of what data should look like.
* Interfaces in a React web client is for defining the structure of props that a component expects.



API Functionalities from Web-client to Database

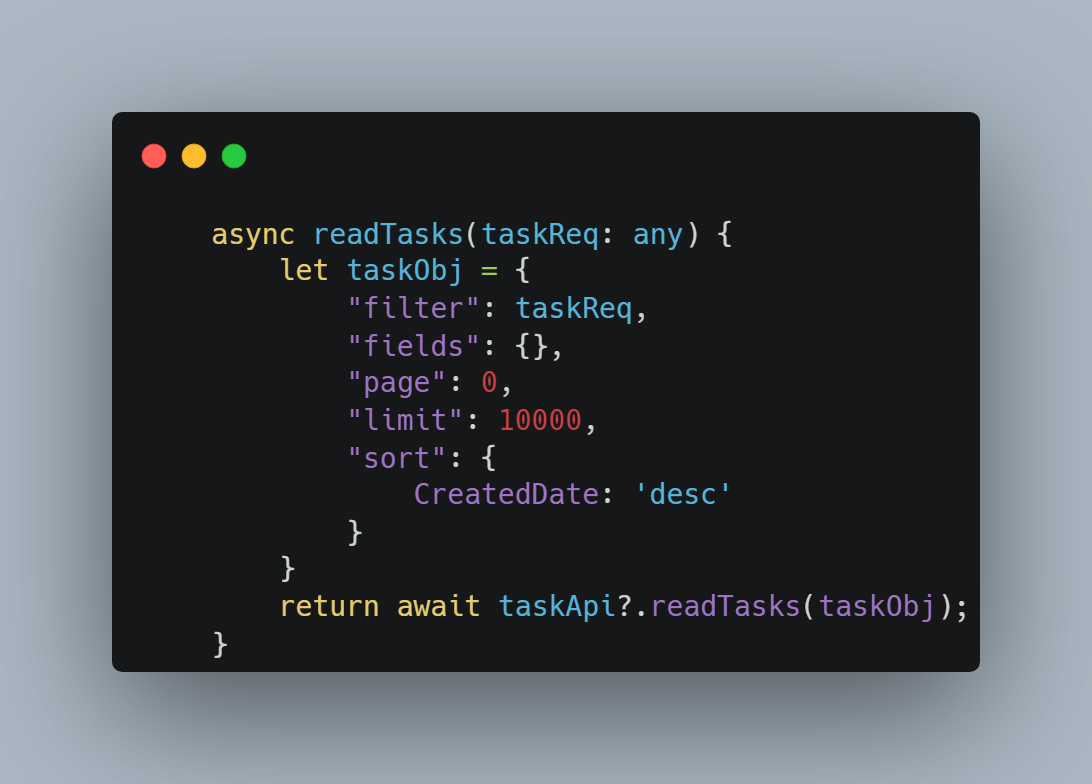


**Pages**:

In the pages, we call components that, in turn, connect to the API using functions. The data fetched is then retrieved from the database and displayed in the UI for the user.

**Function**:

The function includes the API query necessary to fetch data related to the information that should be displayed to the user.



**API**:

The API file comprises all the required APIs needed in the UI to retrieve data based on the user's request.



# 7. Debugging

Debugging is the process of finding and fixing errors or bugs in the source code of any software. When software does not work as expected, computer programmers study the code to determine why any errors occurred.

## 7.1 Rest Data Access:

* Before running **“npm run local”** command run **“npm i"** command.
* The **“npm i"** command should be run in the root path of the code where the **“package. Json”** file is available.
* If you do not add your controller in the **sever.js** file, then the code will crash.
* If your database is down, then there is also a possibility of the app crashing.
* Else, your response will be with the status code of 500.
* Only the data & filter type input are acceptable for the DAL layer.
* Using the .env and **config.js** files you can connect your data access layer to the database.
* In the **.env** file you have the database connection string.
* In the **config.js** file you have the database connection string, **username**, **and password.**

## Rest Business

* Configure the business layer to the running data access layer otherwise it will crash.
* **The** **“npm i"** command should be run in the root path of the code where the **“package. Json”** file is available.
* If you do not add your controller in the **sever.js** file, then the code will crash.
* If your database or data access layer is down, then there is also a possibility of the app crashing.
* Else, your response will be with the status code of 500.
* In the **config.js** file you have the database connection string, username **and password.**

## Gateway

* The only common error observed in the gateway is a 'bad gateway.'
* To resolve this issue, it is necessary to inspect the gateway configuration or ensure that all other services are running smoothly.

## Web Client

* If the file import statement is wrong, it shows error.
* Function argument is not correctly passed it shows error.
* Use dev tools to check the console, errors, and network for Api calls.
* We used the material UI library in this project.
* Other code level issues you can see through the screen.

# 8.Integration Testing:

Refer to the document below for guidance on integration testing.

****

# 9.Release & Deployment:

Refer to the document below for guidance on Release & Deployment.



# 10.Feature Demonstration Guideline:

Refer to the document below for guidance on Feature Demonstration Guideline.

****

# Conclusion:

This document explains thorough exploration into the complex field of full-stack web development, including various important components and approaches. About using the application tool, an overview of creating an architecture diagram, and knowing the steps involved in the development process. Additionally described the features and methods. Lastly, release and deployment document, as well as the debugging ideas. Additionally, Guidelines for giving demonstrations to internal and external stakeholders.