MICROSERVICE BACKEND IN TYPESCRIPT FOR POLKACHAIN EXPLORER

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

by

Malay Srivastava (191352)

Under the supervision of

Dr. Vipul Sharma

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology,
Waknaghat, Solan-173234, Himachal Pradesh

DECLARATION

I hereby declare that this submission is my own work carried out at Antier Solutions Pvt. Ltd., Mohali from February 2023 to May 2023 and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma from a university or other institute of higher learning, except where due acknowledgment has been made in the text.

SUBMITTED BY:

Malay Srivastava

191352

Computer Science & Engineering and Information Technology Department.

Jaypee University of Information Technology, Waknaghat, Solan

CERTIFICATE

I hereby declare that the work presented in this report entitled **MICROSERVICE BACKEND IN TYPESCRIPT FOR POLKACHAIN EXPLORER** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of work carried out over a period from February 2023 to May 2023 under the supervision of **Veer Pratap Singh (Senior Software Developer).** The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Malay Srivastava
191352

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Mr. Veer Pratap Singh
Senior Software Developer
Antier Solutions
Dated: 13-05-2023

Dr. Vipul Sharma
Assistant Professor (SG)
Department of Computer Science & Engineering
Dated:

ACKNOWLEDGEMENT

This report is not just a result of hard work by me but there has been a joint contribution by a lot of other people who I would like to thank.

I would like to thank Ms.Amandeep Kaur, Manager of HR in Talent Acquisition, of Antier Solutions, Mohali for giving me an opportunity to be a part of this organization.

I also would like to thank Mr. Veer Pratap Singh for mentoring me throughout my project and helping me learn new concepts and technologies. It is indeed with a great sense of pleasure and immense sense of gratitude that I acknowledge the help of these individuals.

I am highly indebted to Mr. Pankaj Kumar, Training & Placement Coordinator of our college for the facilities provided to accomplish this job. I would also like to thank the Head of our Department Dr.Vivek Kumar Sehgal and the faculty for teaching us the skills required for this job.

Special thanks to my mentor in the college Dr. Vipul Sharma who supported me throughout the whole and guided me to achieve the best.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Malay Srivastava
191352

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: __________________________ **Department:** ________________ **Enrolment No** ________

Contact No. _____________________________ **E-mail.** ____________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _______________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

CQL	Cassandra Query Language

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Creating a micro service backend is quite simple but the challenge comes when the code has to be tested and optimized,structured, cleaned and maintained and thus here we follow the Eslint Typescript backend structure.

The backend structure has followed the MVC architecture and followed the eslint rules to provide strictness in code. Focus on more functional components than class components.
Talking about the database, we have used the no sql type database CQL providing fast read query.

Polkadot functions are implemented to interact with the blockchain and other libraries are used for testing and printing-saving logs.

1. INTRODUCTION

1.1 Company

Antier Solutions is a technology company that provides blockchain development services and solutions. The company is headquartered in Mohali, Punjab, India, and has additional offices in Canada, the United Kingdom, and the United States.

Antier Solutions offers a range of blockchain development services, including cryptocurrency exchange development, smart contract development, blockchain consulting, and ICO development. The company also provides white label cryptocurrency exchange solutions for businesses looking to launch their own exchanges.

In addition to blockchain development services, Antier Solutions also offers web development, mobile app development, and digital marketing services. The company has worked with a variety of clients, from startups to large enterprises, across a range of industries including finance, healthcare, and e-commerce.

antier

Decentralizing the World

1.2 Introduction

It is a backend service that implements Read and add operations based on the polka blockchain structure. Functions at each layer have their own unit test. There is also an implementation of interface that checks the type of data exchanged between the structures.

1.3 Objectives

To create testable, structured, clean and maintainable web applications by using industrial best practices.

1.4 Motivation

To apply industrial best practices and create a fast, scalable and secure service..

1.5 Libraries/Frameworks Used

i. Polkadot API: Polkadot API is a JavaScript library that provides a simple and easy-to-use interface to interact with the Polkadot network. It allows you to query data, send transactions, and subscribe to events on the Polkadot network.

ii. Web3.js: Web3.js is a collection of libraries that allow you to interact with Ethereum and other Ethereum-compatible networks, including Polkadot. It provides functionalities such as sending transactions, signing messages, and querying data from the blockchain.

iii. Express.js: Express.js is a popular Node.js framework for building web applications and APIs. You can use it to create a RESTful API to interact with the Polkadot network and expose your data to the web.

iv. React.js: React.js is a JavaScript library for building user interfaces. You can use it to create a frontend for your Polkadot explorer and display the data in a user-friendly way.

v. MongoDB: MongoDB is a popular NoSQL database that can be used to store data related to the Polkadot network. You can use it to store blocks, transactions, accounts, and other data related to your Polkadot explorer.

vi. WebSocket: WebSocket is a protocol that allows you to establish a two-way communication channel between a client and a server. You can use it to subscribe to real-time updates from the Polkadot network and receive notifications when new blocks or transactions are added to the blockchain.

MOVING FORWARD WITH NODEJS

All the backend framework such as implementing http request, sending response to server, writing program logic etc is written in NodeJs.

1.6 Technical Requirements

VSCode is an IDE to write clean code .

- **Postman** API platform for building and using APIs.
- **Docker** server provides a database management system with querying and connectivity capabilities

1.6.1 Hardware Configuration

Table 1 : Hardware Configuration

Processor	Apple M1 chip, 8-core CPU
RAM	8 GB
Hard Disk	256 GB SSD
Monitor	13”
Mouse	
Keyboard	

Table 2 : Software Configuration

Operating System	Ubuntu
Language	Typescript
Runtime environment	ExpressJs
Package Manager	NodeJs

2. LITERATURE SURVEY

1) NodeJS Documentation

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment built on the V8 engine of Google Chrome.

2) Cassandra Documentation

Cassandra is a distributed NoSQL database that is designed to handle large amounts of data across multiple servers. It provides high availability and fault tolerance, and is optimized for read and write performance.

3) Docker

Docker is a popular containerization platform that allows you to create, deploy, and run applications in a lightweight, portable, and isolated environment called a container.

4) Git and Github

Official documentary that familiarizes you with the concepts of a version control system i.e Git and how it works with GitHub.

3. SYSTEM DESIGN

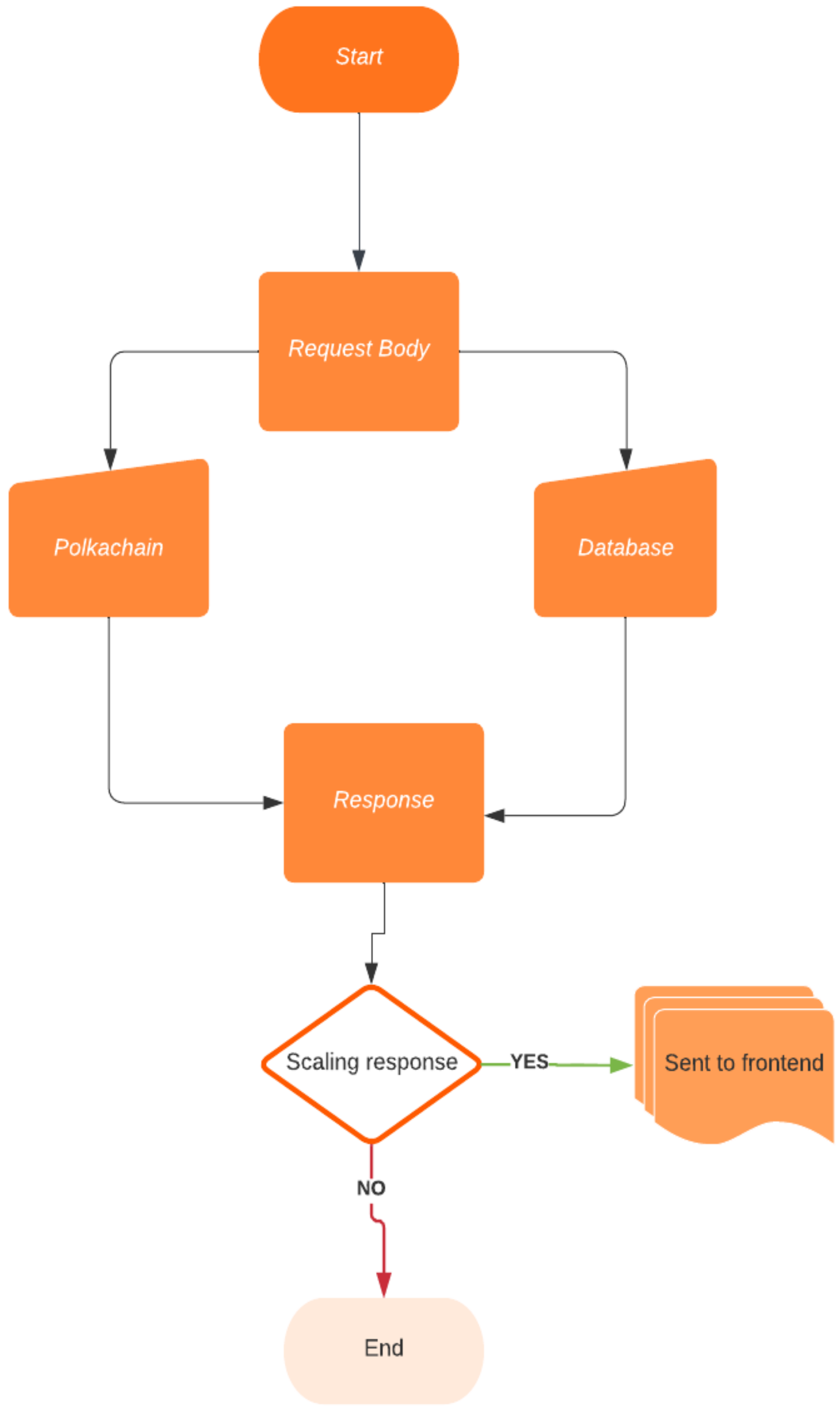


Fig. 1 - System Design

3.1 METHODOLOGY

3.1.1 Identification of features

The micro service features:

- Explorer: Polkadot-JS includes an explorer that allows you to view the state of the Polkadot network. You can view the latest blocks, transactions, and events, and explore the network topology. You can also view information about specific parachains and their associated tokens.

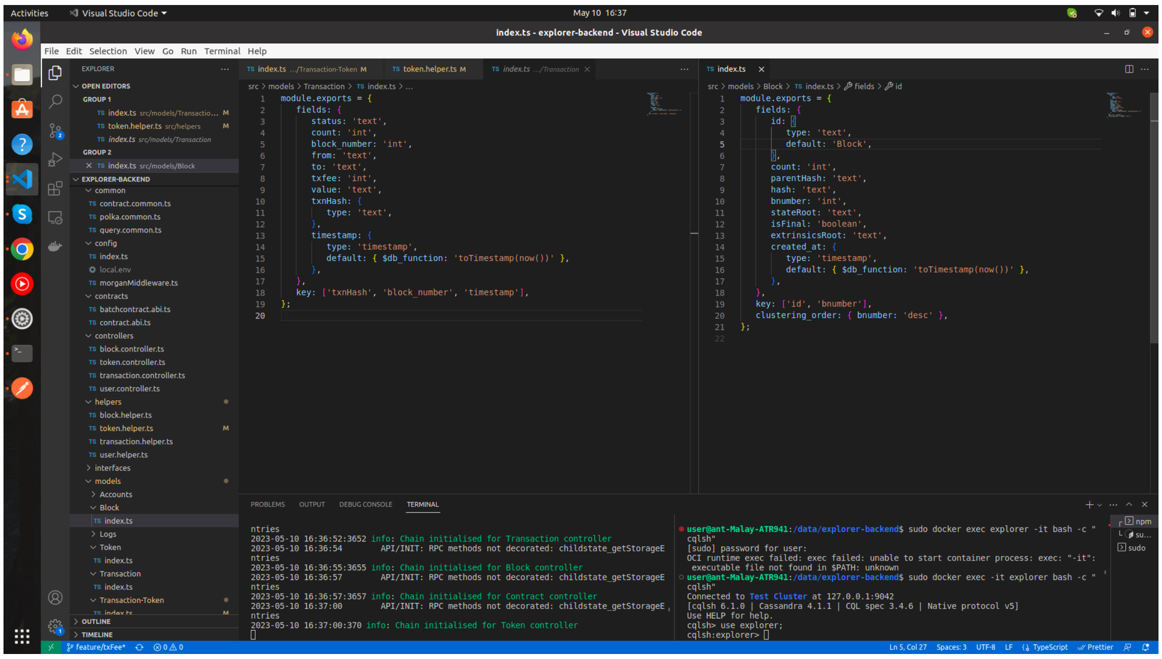
- Accounts: Polkadot-JS allows you to create and manage accounts on the Polkadot network. You can generate new account keys, import existing keys, and manage your balances and transactions.

- Tools: Polkadot-JS includes several tools for interacting with the Polkadot network. For example, you can use the extrinsic tool to send transactions, the storage tool to view and manipulate storage values, and the chain state tool to view the state of the chain at a particular block height.

- Developer tools: Polkadot-JS includes several developer tools for building applications on the Polkadot network. For example, you can use the Polkadot-JS API to interact with the Polkadot network programmatically, and the Polkadot-JS UI library to build custom user interfaces.

- Integration: Polkadot-JS can be integrated with other tools and platforms, such as Metamask, to provide a seamless user experience for interacting with the Polkadot network.

3.1.2 CQL Schema



```
module.exports = {
   fields: {
      status: 'text',
      count: 'int',
      block_number: 'int',
      from: 'text',
      to: 'text',
      txfee: 'int',
      value: 'text',
      txnHash: {
         type: 'text',
      },
      timestamp: {
         type: 'timestamp',
         default: { $db_function: 'toTimestamp(now())' },
      },
   },
   key: ['txnHash', 'block_number', 'timestamp'],
};
```

```
module.exports = {
   fields: {
      id: {
         type: 'text',
         default: 'Block',
      },
      count: 'int',
      parentHash: 'text',
      hash: 'text',
      bnumber: 'int',
      stateRoot: 'text',
      isFinal: 'boolean',
      extrinsicsRoot: 'text',
      created_at: {
         type: 'timestamp',
         default: { $db_function: 'toTimestamp(now())' },
      },
   },
   key: ['id', 'bnumber'],
   clustering_order: { bnumber: 'desc' },
};
```

```
ntries
2023-05-10 16:36:52:3652 info: Chain initialised for Transaction controller
2023-05-10 16:36:54        API/INIT: RPC methods not decorated: childstate_getStorageE
ntries
2023-05-10 16:36:55:3655 info: Chain initialised for Block controller
2023-05-10 16:36:57        API/INIT: RPC methods not decorated: childstate_getStorageE
ntries
2023-05-10 16:36:57:3657 info: Chain initialised for Contract controller
2023-05-10 16:37:00        API/INIT: RPC methods not decorated: childstate_getStorageE
ntries
2023-05-10 16:37:00:370 info: Chain initialised for Token controller
```

```
user@ant-Malay-ATR941:/data/explorer-backend$ sudo docker exec explorer -it bash -c "
cqlsh"
[sudo] password for user:
OCI runtime exec failed: exec failed: unable to start container process: exec: "-it":
 executable file not found in $PATH: unknown
user@ant-Malay-ATR941:/data/explorer-backend$ sudo docker exec -it explorer bash -c "
cqlsh"
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.1.0 | Cassandra 4.1.1 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
cqlsh> use explorer;
cqlsh:explorer>
```

Fig. 2 - CQL Schema for Cassandra Database

3.1.3 Study Material

LINUX

Linux is a free and open-source operating system that is based on the Unix operating system. It was first created in 1991 by Linus Torvalds, and since then has become one of the most widely used operating systems, powering everything from servers and supercomputers to mobile phones and embedded devices.

Overall, Linux is a highly customizable and powerful operating system that provides a wide range of tools and features for both developers and users. Its open-source nature and large community of developers make it a popular choice for building and running a wide variety of applications and systems.

1. Here are some commonly used Linux commands:
2. ls: Lists the contents of a directory.
3. cd: Changes the current directory.
4. pwd: Displays the current working directory.
5. mkdir: Creates a new directory.
6. rm: Removes a file or directory.
7. cp: Copies files or directories.
8. mv: Moves or renames files or directories.

9. touch: Creates an empty file or updates the access and modification times of an existing file.
10. cat: Displays the contents of a file.
11. nano/vim: A text editor for editing files in the terminal.
12. grep: Searches for a pattern in a file or output.
13. chmod: Changes the permissions of a file or directory.
14. chown: Changes the ownership of a file or directory.
15. tar: Archives and compresses files or directories.
16. curl/wget: Downloads files from the internet.
17. ps: Lists the running processes.
18. top: Displays real-time system resource usage information.
19. df: Shows disk space usage for file systems.
20. du: Shows disk

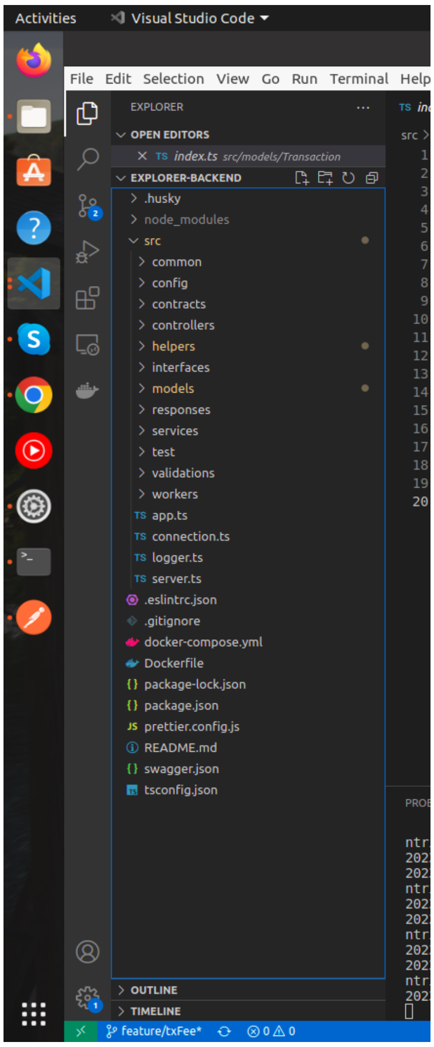
MVC Structure

1. In Node.js, the Model-View-Controller (MVC) architectural pattern is commonly used to organize code into separate and distinct components. Here's how it works:

2. Model: The model represents the data and business logic of the application. It defines how the data is stored, retrieved, and manipulated. In Node.js, the model can be implemented as a module that interacts with a database or other data source.

3. View: The view is responsible for presenting the data to the user. It can be implemented as a template engine that generates HTML, CSS, and JavaScript code. The view is typically passive and does not contain any business logic.

4. Controller: The controller acts as the intermediary between the model and the view. It receives input from the user through the view, processes the input using the model, and updates the view with the results. In Node.js, the controller can be implemented as a module that handles HTTP requests and responses.



```
EXPLORER-BACKEND
> .husky
> node_modules
v src
  > common
  > config
  > contracts
  > controllers
  > helpers
  > interfaces
  > models
  > responses
  > services
  > test
  > validations
  > workers
  app.ts
  connection.ts
  logger.ts
  server.ts
.eslintrc.json
.gitignore
docker-compose.yml
Dockerfile
package-lock.json
package.json
prettier.config.js
README.md
swagger.json
tsconfig.json
```

Fig. 3 - Packet Structure

By following the MVC pattern, Node.js applications can be structured in a way that separates concerns and makes the code easier to maintain and scale.

NPM Packages

Each and every nodejs program is made of packages. All the program in nodejs enviroment start running in the main package

math/rand:- In package rand, environment is deterministic i.e. when run rand. In return same number, and if we want different results each time we use, rand.Seed

With import use ()-for clarity[Factored statement] and " " with packages When exporting names use Capital letter with its package- ex: Pi(math.Pi) We can use fmt: formatted i/o package to format all this

Functions:func()

JavaScript (JS) and ECMAScript 6 (ES6) are both used for writing functions in Node.js. However, ES6 introduced some new features for writing functions that make the code more concise and expressive. Here are some differences between JS and ES6 functions in Node.js:

Arrow Functions:
ES6 introduced the arrow function syntax, which provides a more concise way to define functions. Instead of using the "function" keyword, you can use the "=>" operator. Here's an example:

JS:

```
function add(x, y) {
  return x + y;
}
```

ES6:

```
const add = (x, y) => x + y;
```

Import:

In Node.js, you can use the "require" function to import modules or files. The "require" function is a built-in function in Node.js that allows you to include external modules and make them available in your code.

File Watchers:

nodemon:
"nodemon" is a package that monitors changes to your Node.js application and automatically restarts the server when changes are detected. This can be useful during development, as it saves you the hassle of manually restarting the server after making changes. Here's an example of using "nodemon" to monitor a Node.js file:

Variables

In Node.js, variables are declared using the var, let, or const keywords.

The var keyword declares a variable globally, or locally to an entire function, regardless of block scope. However, it is generally recommended to use let and const instead of var, as they provide better scoping and are block-scoped.

The let keyword declares a variable that is block-scoped, meaning it is only accessible within the block where it is defined. For example:

```
let x = 10;
if (true) {
 let x = 20;
 console.log(x); // outputs 20
}
console.log(x); // outputs 10
```

The const keyword is similar to let, but once a variable is defined with const, it cannot be reassigned. This makes const useful for defining constants or values that should not be changed. For example:

```
const pi = 3.14;
pi = 3; // Error: Assignment to constant variable.
```

It's important to note that while const prevents reassignment of a variable, it does not prevent mutation of an object or array. For example:

```
const arr = [1, 2, 3];
arr.push(4); // mutation is allowed
console.log(arr); // outputs [1, 2, 3, 4]
```

Interfaces:

Interfaces can be used to define the shape of objects and their properties. This can help catch type errors early on and make your code more readable. For example:

```
interface User {
 id: number;
 name: string;
 email: string;
}

function getUserById(id: number): User {
 // fetch user by ID and return it as a User object
}
```

Type annotations:

Type annotations can be used to specify the type of a variable or function parameter. This can also help catch type errors early on. For example:

\`\`\`

```
function addNumbers(a: number, b: number): number {
  return a + b;
}
```

Third-party libraries:

There are many third-party libraries available for use with Node.js and TypeScript. Some popular choices include Express for building web applications, TypeORM for database interactions, and Jest for testing.

The `use` keyword:

The `use` keyword is commonly used in middleware functions in Node.js applications. Middleware functions can be used to add functionality to an HTTP request/response cycle. For example:

```
import express, { Request, Response, NextFunction } from 'express';

const app = express();

function logRequest(req: Request, res: Response, next: NextFunction) {
  console.log(`${req.method} request to ${req.path}`);
  next();
}

app.use(logRequest);
```

In this example, the `logRequest` function is middleware that logs the method and path of each incoming request. The `use` method is used to apply this middleware to all incoming requests.

These are just a few examples of how TypeScript can be used to write components in Node.js. By taking advantage of TypeScript's powerful type system and libraries, you can write more robust and maintainable code for your Node.js applications.

FOR

A for loop in Node.js is used to execute a block of code repeatedly for a specified number of times or for a given range of values. Here's an example of a basic for loop in Node.js:

```
const arr = [1, 2, 3, 4, 5];

for (const num of arr) {
  console.log(num);
}

const arr = [1, 2, 3, 4, 5];

for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

IF

In this example, the if statement checks whether the value of the num variable is greater than 5. If it is, the code inside the curly braces is executed, which logs a message to the console.

Here's a breakdown of the different parts of the if statement:

if: This is the keyword that starts the if statement.
(num > 5): This is the condition that the if statement checks. If it is true, the code inside the curly braces is executed; if it is false, the code is skipped.
{ ... }: This is the body of the if statement, which contains the code that is executed if the condition is true.
You can also use an else statement to execute a different block of code if the condition is false:

SWITCH

The switch statement checks the value of the day variable and executes a different block of code depending on its value. If day is 1, the first message is logged; if it is 2, the second message is logged; and so on. If the value of day doesn't match any of the cases, the default block is executed.

Here's a breakdown of the different parts of the switch statement:

switch: This is the keyword that starts the switch statement.
day: This is the variable that the switch statement checks.
case: These are the different cases that the switch statement checks. If the value of day matches one of the cases, the corresponding block of code is executed.
break: This keyword is used to end each case block and prevent the execution of subsequent cases.
default: This block is executed if the value of day doesn't match any of the cases.
{ ... }: This is the body of each case block, which contains the code that is executed if the value of day matches the corresponding case.

Promise

In Node.js, the Promise object is used to represent a value that may not be available yet, but will be resolved at some point in the future. You can create a new Promise object using the Promise constructor, which takes a function as its argument.

```
myPromise
  .then(result => {
    console.log(result);
  })
  .catch(error => {
    console.error(error);
  });
```

Overall, Promise objects are a powerful way to work with asynchronous operations in Node.js, allowing you to handle successful and failed outcomes separately, and chain together multiple operations in a readable way.

CLASSES

In TypeScript, you can define classes in Node.js using the same syntax as in modern JavaScript. Here's an example:

```
class Person {
  firstName: string;
  lastName: string;

  constructor(firstName: string, lastName: string) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getFullName(): string {
    return `${this.firstName} ${this.lastName}`;
  }
}
```

To use this class in your Node.js application, you can create a new instance of the class using the new keyword:

```
const person = new Person('John', 'Doe');
console.log(person.getFullName()); // Output: John Doe
```

ARRAYS

Arrays are an essential part of JavaScript and can be used in Node.js just like in a browser-based JavaScript environment. You can define an array in Node.js using the same syntax as in JavaScript:

```
const myArray = [1, 2, 3, 4, 5];
```

In this example, we define an array named `myArray` with five elements.

You can access elements in an array using their index. The index of the first element is 0, and the index of the last element is the length of the array minus 1. Here's an example:

```
console.log(myArray[0]); // Output: 1
console.log(myArray[4]); // Output: 5
```

You can add elements to an array using the `push` method:

```
myArray.push(6);
console.log(myArray); // Output: [1, 2, 3, 4, 5, 6]
```

You can remove elements from an array using the `pop` method:

```
myArray.pop();
console.log(myArray); // Output: [1, 2, 3, 4, 5]
```

You can iterate over an array using a `for` loop or the `forEach` method:

```
for (let i = 0; i < myArray.length; i++) {
  console.log(myArray[i]);
}
myArray.forEach(item => console.log(item))
```

SLICES

In Node.js, you can use the same syntax as in JavaScript to slice arrays. Slicing is a way to extract a portion of an array without modifying the original array. Here's an example:

```
const myArray = [1, 2, 3, 4, 5];
const mySlice = myArray.slice(1, 4);
console.log(mySlice); // Output: [2, 3, 4]
```

In this example, we define an array named `myArray` with five elements. We then use the `slice` method to extract a portion of the array starting at index 1 and ending at index 3. The resulting slice is assigned to a variable named `mySlice`.

The `slice` method takes two arguments: the starting index (inclusive) and the ending index (exclusive) of the slice. If the second argument is omitted, the slice will include all elements from the starting index to the end of the array.

You can also use negative indices to slice from the end of the array. For example:

```
const myArray = [1, 2, 3, 4, 5];
const mySlice = myArray.slice(-3);
console.log(mySlice); // Output: [3, 4, 5]
```

Slicing is a useful technique when you want to work with a subset of an array without modifying the original array. It's commonly used in functional programming and in situations where you need to extract a specific range of elements from an array.

RANGE

A Range is a form of for loop that iterates over a slice/map.

For each iteration it returns an index and copy of value at that index.

We can also skip the index or value by assigning _.

Syntax:

```
function range(start: number, end: number, step: number = 1): number[] {
  const result = [];
  for (let i = start; i <= end; i += step) {
    result.push(i);
  }
  return result;
}

console.log(range(1, 5)); // Output: [1, 2, 3, 4, 5]
console.log(range(1, 10, 2)); // Output: [1, 3, 5, 7, 9]
```

MAPS

Maps are a built-in data structure in Node.js (and in JavaScript in general) that allow you to store key-value pairs. A key-value pair is a set of two linked data items: a key that is used to retrieve the value, and the value itself. Here's an example of how to use a Map in Node.js:

```
const myMap = new Map();
myMap.set("key1", "value1");
myMap.set("key2", "value2");

console.log(myMap.get("key1")); // Output: "value1"
console.log(myMap.get("key2")); // Output: "value2"
```

In this example, we create a new Map object called `myMap` using the `new Map()` syntax. We then use the `set()` method to add two key-value pairs to the Map: `"key1"` and `"value1"`, and `"key2"` and `"value2"`.

To retrieve the value associated with a specific key, we use the `get()` method, passing in the key as an argument.

PANIC: run-time error

Variadic Functions

In Node.js, you can create a variadic function by using the rest parameter syntax, which allows you to pass an arbitrary number of arguments to a function. Here's an example:

```
function sum(...numbers: number[]): number {
  let total = 0;
  for (const number of numbers) {
    total += number;
  }
  return total;
}

console.log(sum(1, 2, 3)); // Output: 6
console.log(sum(4, 5, 6, 7)); // Output: 22
```

Function Values

```
function createGreeting(name) {
  return function() {
    console.log(`Hello, ${name}!`);
  }
}
```

```
const sayHelloToJohn = createGreeting('John');
sayHelloToJohn(); // Output: Hello, John!
```

A closure is a function value that references variables from outside its body.

The function may access and assign to the referenced variables; in this sense the function is "bound" to the variables.

METHODS

In Node.js, as well as in JavaScript, a method is simply a function that is defined as a property of an object. The main difference between a function and a method is that a method is associated with an object, and can access and modify the object's properties.

Here's an example of defining a method in Node.js:

```
const person = {
  name: 'John',
  age: 30,
  greet: function() {
    console.log(`Hello, my name is ${this.name}, and I'm ${this.age} years old.`);
  }
};
```

```
person.greet(); // Output: Hello, my name is John, and I'm 30 years old.

class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

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

const john = new Person('John', 30);
john.greet(); // Output: Hello, my name is John, and I'm 30 years old.
```

We can define methods on type. We can define methods on non-struct types also.
We can only declare a method with a receiver whose type is defined in the same package as method including built in types like int.

There are two reasons to use a pointer receiver:

The first is so that the method can modify the value that its receiver points to.

The second is to avoid copying the value on each method call.
This can be more efficient

if the receiver is a large struct.

All methods on a given type should have either value or pointer receivers, but not a mixture of both.

Receiver Arguments:

- Value receiver argument can only reference methods with value receiver whereas pointer receiver argument references methods with both value and pointer receiver: METHOD SETS

- We use value receivers when we don’t want changes to be reflected in the original value, while using slices, maps, etc

- We can use a pointer receiver when we want the changes to be reflected or when we want to access methods either way or when the struct is quite large to avoid duplicate copies.

INTERFACES

Interface type is defined as method signature of a particular underlying base.

A value of interface type can hold any value that implements those method s i.e same methods with different type is implemented by interface]

To define an interface in Node.js, you can use the interface keyword. Here's an example:

Interfaces are implemented implicitly.
There is no explicit declaration of intent, no "implements" keyword.

```
interface Person {
  name: string;
  age: number;
  greet(): void;
}
```

Zero value of interface is nil.

Abstract type underlying which is our concrete type (struct, float, etc): can be thought of as a tuple

of a value and a concrete type: (value, type)

In case of pointer receiver: (&{Hello}, *main.T)

If the concrete value inside the interface itself is nil, the method will be called with a nil receiver & doesn't trigger a null pointer exception. Interface value that holds a nil concrete value is itself non-nil.

A nil interface value holds neither value nor concrete type.
Callin a method on a nil interface is a run-time error because there is no type in side the
Interface tuple to indicate which *concrete* method to call.

The interface type that specifies zero methods is known as the *empty interface*: interface{}.
An empty interface may hold values of any type. Every type implements at least zero methods.Empty interfaces are used by code that handles values of unknown type.
var i interface{}
i=42 (type->int)

TYPE ASSERTION

Type assertion in Node.js is a way to tell the compiler that you know more about the type of a variable than the compiler does. It is also known as type casting.

Type assertion is used when you have a value of one type and you need to convert it to another type. This is especially useful when working with JavaScript code that may not have types.

In Node.js, type assertion is done using the as keyword. Here's an example:

```
const myNumber: any = 42;
const myString: string = myNumber as string;
console.log(myString); // Output: undefined
```

Type SWITCH

Type switches in Node.js are used to determine the type of a value at runtime. It allows you to write code that can handle different types of values without knowing their types at compile-time.

To use a type switch in Node.js, you first define a switch statement with an expression that you want to check the type of. Inside the switch statement, you define cases for each type that you want to handle.

Here's an example:

```
function doSomething(value: string | number): void {
  switch (typeof value) {
    case 'string':
      console.log('value is a string');
      break;
    case 'number':
      console.log('value is a number');
      break;
    default:
      console.log('value is of an unknown type');
  }
}

doSomething('hello'); // Output: value is a string
doSomething(42); // Output: value is a number
doSomething(true); // Output: value is of an unknown type
```

Type switches can be very useful when working with dynamic data in Node.js. They allow you to write more flexible and robust code that can handle different types of values without crashing or throwing errors.

ASYNC AND AWAIT

Async/await is a way to write asynchronous code in a synchronous style in Node.js. It allows you to write code that looks synchronous, but actually runs asynchronously under the hood. Async/await is built on top of Promises, and it makes it easier to work with them.

To use async/await in Node.js, you first need to define an asynchronous function using the async keyword. Inside the async function, you can use the await keyword to wait for a Promise to resolve before continuing execution.

```
async function fetchData() {
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();
  return data;
}

fetchData().then(data => {
  console.log(data);
}).catch(error => {
  console.error(error);
});
```

Readers

To read files efficiently in Node.js, you can use the stream module. Streams are a way to handle large amounts of data in a more efficient manner by processing it piece by piece instead of loading it all into memory at once.

One type of stream in Node.js is a Readable stream, which represents a source of data that can be read from. You can create a Readable stream for a file using the fs.createReadStream() method. Here's an example:

```
const fs = require('fs');

const stream = fs.createReadStream('file.txt');
```

```
stream.on('data', (chunk) => {
  console.log(`Received ${chunk.length} bytes of data.`);
});

stream.on('end', () => {
  console.log('Finished reading file.');
});

stream.on('error', (error) => {
  console.error(`Error reading file: ${error.message}`);
});
```

REST- Represntation State Transfer

REST (Representational State Transfer) is an architectural style used to build web services that uses HTTP methods like GET, POST, PUT, and DELETE to perform the operations on resources. A RESTful API is a web-based API that uses REST architecture to interact with web-based clients.

RESTful APIs use the following HTTP methods to perform operations:

- GET: Used to retrieve a resource from the server.

- POST: Used to create a new resource on the server.
- PUT: Used to update an existing resource on the server.
- DELETE: Used to delete a resource from the server.

RESTful APIs are stateless, meaning that each request is independent and self-contained. The server does not store any state information about the client session. Instead, all necessary data is sent along with the request itself.

To use a RESTful API, clients send requests to a server, specifying the HTTP method, the resource to operate on, and any necessary data. The server then returns a response, which may include data, status information, or error messages.

RESTful APIs are widely used in web development, particularly in mobile app development. They offer a flexible and scalable way to interact with data stored on remote servers.

Response Status Codes

1. 200:OK, Success
2. 201: Success+Created
3. 202: Accepted, request received but not completed
4. 204: No content
5. 400: Bad Request, incorrect syntax
6. 404: Not found
7. 405: Method Not Allowed
8. 500: Internal Server Error

HTTP package

The http package provides a client and a server. The server is made of handlers. The handler takes a request and based on that it returns a response.

1. HTTP protocols

Create : Post-> new data

Read : Get-> retrieve data

Update: Put-> update data

Delete: Delete-> delete data

In Node.js, the built-in `http` package provides an easy way to create a RESTful API server.

To create a RESTful API server using the `http` package, you can follow these steps:

1. Import the `http` package:

```
const http = require('http');
```

2. Create a server object using the `http.createServer()` method:

```
const server = http.createServer();
```

3. Use the `server.on()` method to handle requests:

```
server.on('request', (req, res) => {
  // Handle request here
});
```

4. Set the response status and headers:

```
res.writeHead(200, {'Content-Type': 'text/plain'});
```

5. Write the response body:

```
res.write('Hello World!');
```

6. End the response:

```
res.end();
```

Here's an example of a simple RESTful API server using the `http` package:

```
const http = require('http');

const server = http.createServer();

server.on('request', (req, res) => {
  if (req.method === 'GET' && req.url === '/') {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.write('Hello World!');
    res.end();
  } else {
    res.writeHead(404, {'Content-Type': 'text/plain'});
    res.write('Not Found');
    res.end();
  }
});

server.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

This example creates a server that responds to GET requests to the root path (`/`) with a "Hello World!" message, and responds to all other requests with a "Not Found" message. The server listens on port 3000.

Functions as handlers

In Node.js, functions can be used as request handlers for HTTP servers. These functions are often referred to as "middleware" functions.

To use a function as a request handler, you can simply pass the function as a callback to the server's request event:

```
const http = require('http');

function requestHandler(req, res) {
  // handle the request
}

const server = http.createServer();
server.on('request', requestHandler);

server.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

In this example, the requestHandler function is passed as the callback for the server's request event. Whenever the server receives a request, it will invoke the requestHandler function with the request and response objects.

HTTPtest Package

In Node.js, you can use the http module to test HTTP servers. The http module provides a ClientRequest class that can be used to make HTTP requests to a server.

Here's an example of how to use the http module to test an HTTP server:

```
const http = require('http');
const assert = require('assert');
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(3000, () => {
  console.log('Server listening on port 3000');
});

const req = http.request({
  host: 'localhost',
  port: 3000,
  path: '/'
}, (res) => {
  let data = '';
  res.on('data', (chunk) => {
    data += chunk;
  });
  res.on('end', () => {
    assert.strictEqual(res.statusCode, 200);
    assert.strictEqual(data, 'Hello World\n');
    console.log('Test passed!');
    server.close();
  });
});

req.on('error', (err) => {
  console.error(err);
});
req.end();
```

Layered architecture: Layers are autonomous from one another and interact via interfaces.

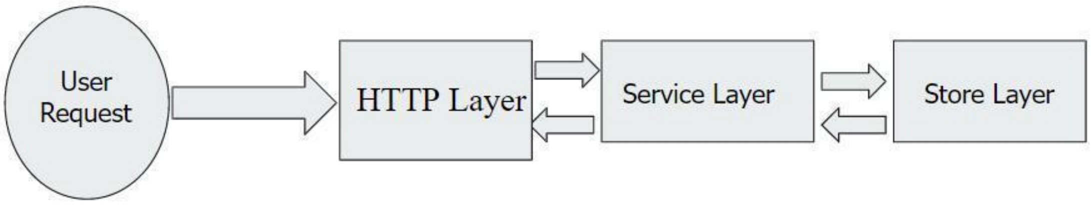


Fig. 4 - Layered Architecture

Basically, this aids in the modularization, readability, and maintainability of our application.

HTTP layer, Service layer, and Store layer are the three layers of this.

Table 3 - HTTP Layer, Service Layer & Store Layer

1. HTML layer	checks the request body, headers, and query/path parameters for validity.
2. Service layer	: Implements business logic and communicates with datastore layer
3. Store layer	executes database-level queries.

1. Each layer uses an interface (methods with defined input parameters and output types) to communicate with the layer above it or below it.

2. The interface, database, and server for each layer are all mocked during testing, depending on the circumstances.

Dependency Injection:

It is a way of writing code where the dependencies of a specific object or struct are provided at the time the object is initialized.

We are able to specify when to reuse the same dependency instance and when to create new ones.
Our structs are less closely coupled to their dependencies because they are no longer in charge of establishing them.

Factory method

It is a design pattern that addresses the issue of creating product objects without identifying the concrete classes for those objects. Instead of directly calling the new operator, it defines a method that can be used to create objects.

i. Simple factory
ii. Interface factories

MICRO SERVICES

Small, independent services that communicate over clearly defined APIs make up the architectural and organizational strategy of software development known as "microservices." Small, self-contained teams own and operate these services.

Applications can be expanded and developed more easily thanks to microservices designs, which also speed up the time it takes to market new features.

Microservices' advantages

- **Adaptable Scaling**

  The demand for each microservice's underlying app feature can be scaled independently of the others. This enables teams to maintain service availability during times of high demand, accurately estimate the cost of a feature, and size infrastructure appropriately.

- **Simple Deployment**

  Continuous integration and delivery are made possible by microservices, making it simple to test new concepts and roll them back if they don't work. The low cost of failure makes it possible to try more things and experiment more quickly with code modifications and new feature time-to-market.

- **Reusable Code**

Teams can use functions for many purposes by dividing software into discrete, well-defined modules. A service created for one function can be used as a foundation for another feature. This allows an application to self-bootstrap since developers may add new features without having to write code from scratch.

Docker

Docker is a platform for developing, deploying, and running applications inside containers. Containers are lightweight, standalone executables that can run in any environment with the Docker runtime installed, making it easy to build and deploy applications across different environments.

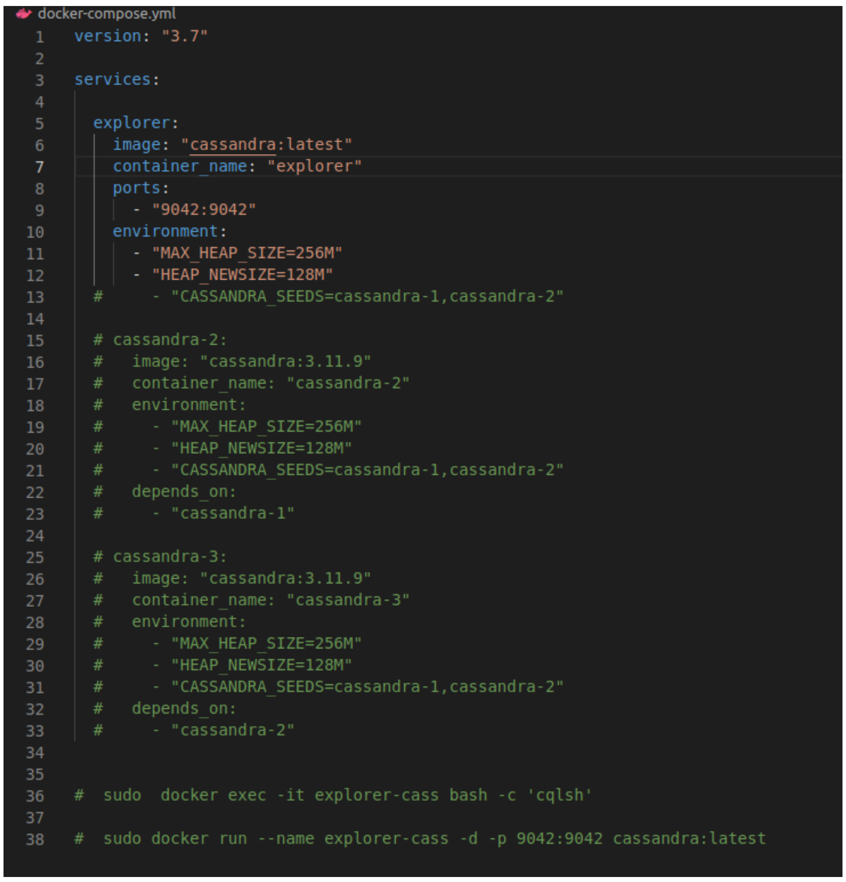
Docker allows you to package an application with all its dependencies into a single container, which can then be run on any Docker-compatible system. This makes it easy to move applications between development, testing, and production environments, and ensures that the application runs consistently across different environments.

Docker provides a simple command-line interface for building, managing, and deploying containers, and supports a wide range of operating systems, programming languages, and application architectures. Docker also provides a number of tools and services for managing and scaling containerized applications, including Docker Compose for managing multi-container applications, and Docker Swarm for managing container clusters.

Some common use cases for Docker include:

- Developing and testing applications in a consistent, isolated environment
- Packaging and deploying applications in a portable, containerized format
- Running legacy applications on modern infrastructure
- Scaling and managing containerized applications in production environments
- Building and distributing pre-configured environments for specific use cases

Overall, Docker provides a powerful and flexible platform for building and deploying applications, and is widely used in modern software development and deployment workflows.



docker-compose.yml

```yaml
version: "3.7"

services:

  explorer:
    image: "cassandra:latest"
    container_name: "explorer"
    ports:
      - "9042:9042"
    environment:
      - "MAX_HEAP_SIZE=256M"
      - "HEAP_NEWSIZE=128M"
  #     - "CASSANDRA_SEEDS=cassandra-1,cassandra-2"

  # cassandra-2:
  #   image: "cassandra:3.11.9"
  #   container_name: "cassandra-2"
  #   environment:
  #     - "MAX_HEAP_SIZE=256M"
  #     - "HEAP_NEWSIZE=128M"
  #     - "CASSANDRA_SEEDS=cassandra-1,cassandra-2"
  #   depends_on:
  #     - "cassandra-1"

  # cassandra-3:
  #   image: "cassandra:3.11.9"
  #   container_name: "cassandra-3"
  #   environment:
  #     - "MAX_HEAP_SIZE=256M"
  #     - "HEAP_NEWSIZE=128M"
  #     - "CASSANDRA_SEEDS=cassandra-1,cassandra-2"
  #   depends_on:
  #     - "cassandra-2"


# sudo  docker exec -it explorer-cass bash -c 'cqlsh'

# sudo docker run --name explorer-cass -d -p 9042:9042 cassandra:latest
```

Fig. 5 - Docker yml File

POLKADOT

Polkadot is a next-generation blockchain protocol designed to connect different blockchain networks and enable them to work together seamlessly. It is a multi-chain network that allows for interoperability between different blockchain platforms, making it possible to transfer value, data, and assets between them.

The Polkadot protocol was created by the Web3 Foundation, a non-profit organization dedicated to developing decentralized technologies. The project was founded by Gavin Wood, who was also a co-founder of Ethereum.

Polkadot uses a unique consensus mechanism called "Nominated Proof-of-Stake" (NPoS), which allows users to nominate validators to participate in block production and earn rewards. The system is designed to be highly scalable and efficient, with the ability to process up to 1 million transactions per second.

One of the key features of Polkadot is its ability to support multiple parallel chains, known as "parachains." These parachains can be customized for different use cases, such as smart contracts, decentralized finance (DeFi), or identity management, and can communicate with each other through the Polkadot relay chain.

Polkadot also includes a governance system that allows token holders to vote on proposed changes to the protocol, including upgrades and amendments. This helps ensure that the protocol remains decentralized and community-driven.

HUSKY (Git Hooks)

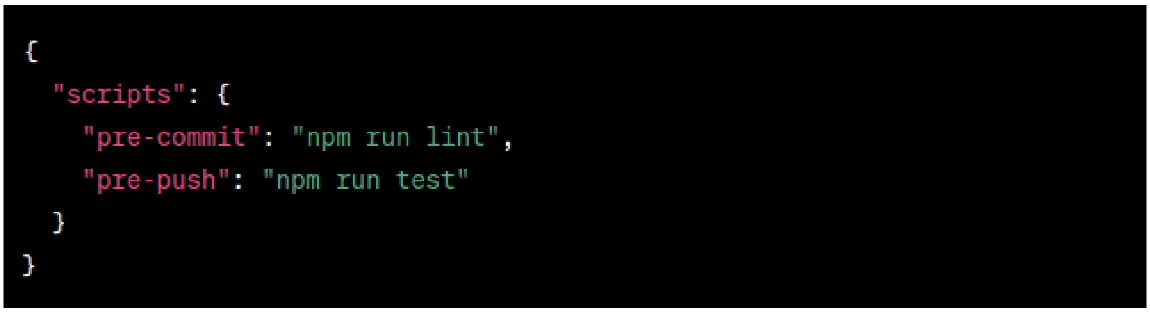
Husky is a popular Node.js library that helps you to manage Git hooks easily in your project. Git hooks are scripts that can be executed automatically by Git whenever certain actions occur, such as committing or pushing code.

Husky simplifies the process of configuring Git hooks by providing a convenient API to create and manage them. It allows you to define your hooks as scripts in your package.json file and then automatically installs them as Git hooks in your project's .git/hooks directory.

Some examples of hooks that you can manage with Husky include:

Pre-commit: Runs before a commit is created and can be used to perform linting, formatting, or other checks on the code being committed.
Pre-push: Runs before a push is executed and can be used to run tests, check for vulnerabilities, or other checks on the code being pushed.

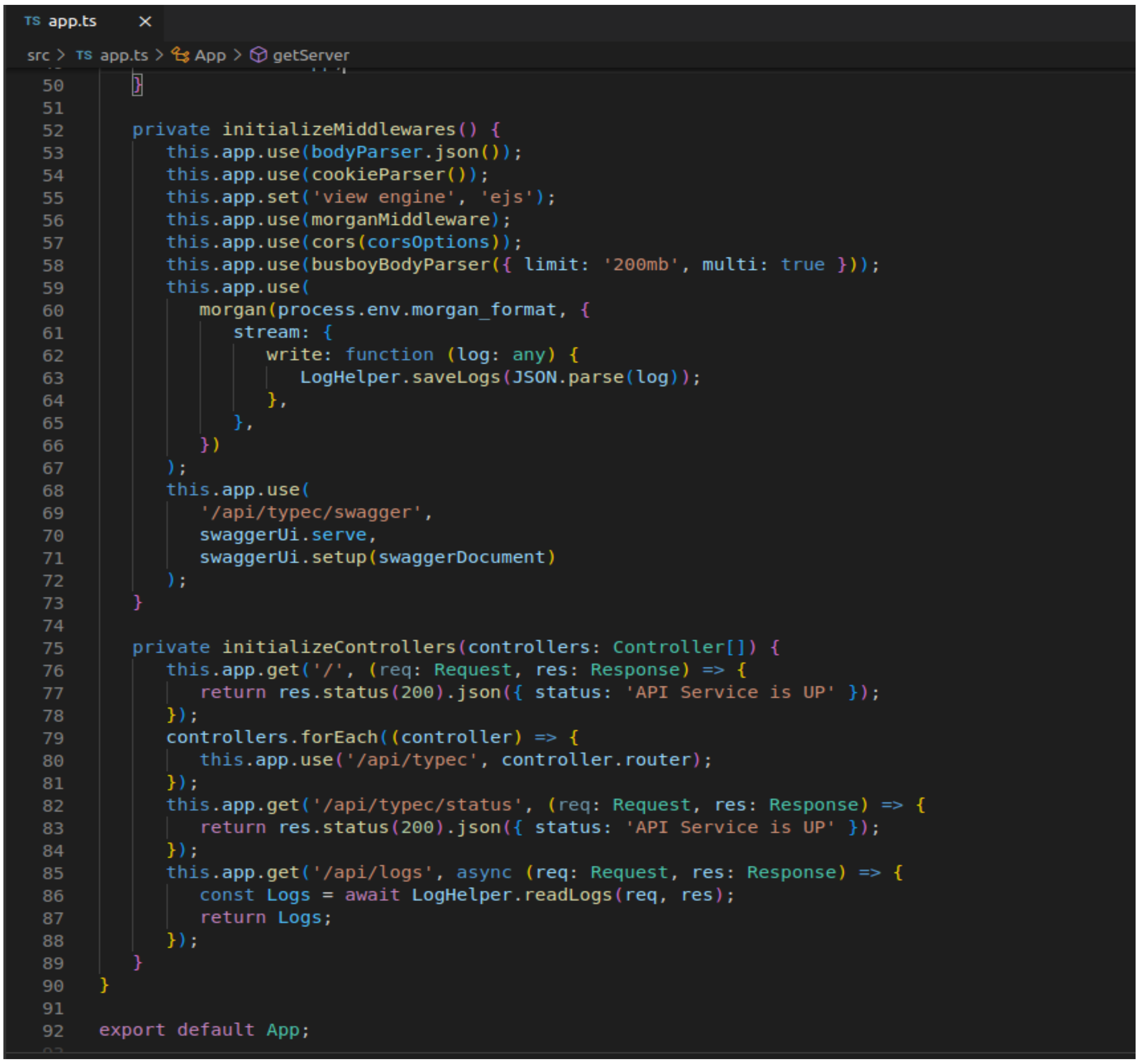
Post-merge: Runs after a merge is completed and can be used to perform additional setup or configuration tasks.



```
{
  "scripts": {
    "pre-commit": "npm run lint",
    "pre-push": "npm run test"
  }
}
```

CODE

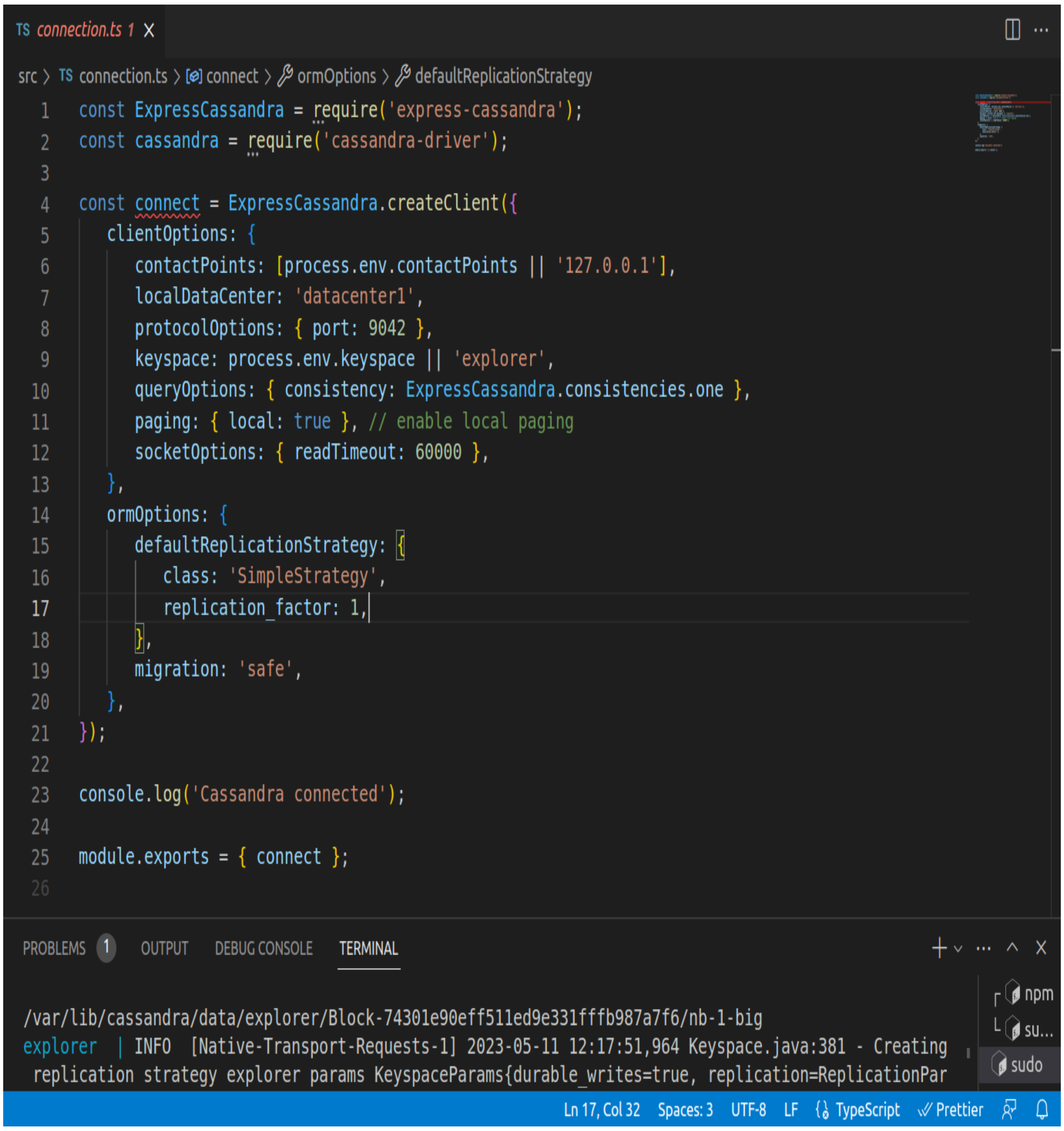
App.ts



```
    }

    private initializeMiddlewares() {
        this.app.use(bodyParser.json());
        this.app.use(cookieParser());
        this.app.set('view engine', 'ejs');
        this.app.use(morganMiddleware);
        this.app.use(cors(corsOptions));
        this.app.use(busboyBodyParser({ limit: '200mb', multi: true }));
        this.app.use(
            morgan(process.env.morgan_format, {
                stream: {
                    write: function (log: any) {
                        LogHelper.saveLogs(JSON.parse(log));
                    },
                },
            })
        );
        this.app.use(
            '/api/typec/swagger',
            swaggerUi.serve,
            swaggerUi.setup(swaggerDocument)
        );
    }

    private initializeControllers(controllers: Controller[]) {
        this.app.get('/', (req: Request, res: Response) => {
            return res.status(200).json({ status: 'API Service is UP' });
        });
        controllers.forEach((controller) => {
            this.app.use('/api/typec', controller.router);
        });
        this.app.get('/api/typec/status', (req: Request, res: Response) => {
            return res.status(200).json({ status: 'API Service is UP' });
        });
        this.app.get('/api/logs', async (req: Request, res: Response) => {
            const Logs = await LogHelper.readLogs(req, res);
            return Logs;
        });
    }
}

export default App;
```

Initialization of middleware and controllers.
Other functionality, like Winston logs and cors, was also added.

Cassandra connection:
Here we connect cassandra service running on docker to our micro service backend.

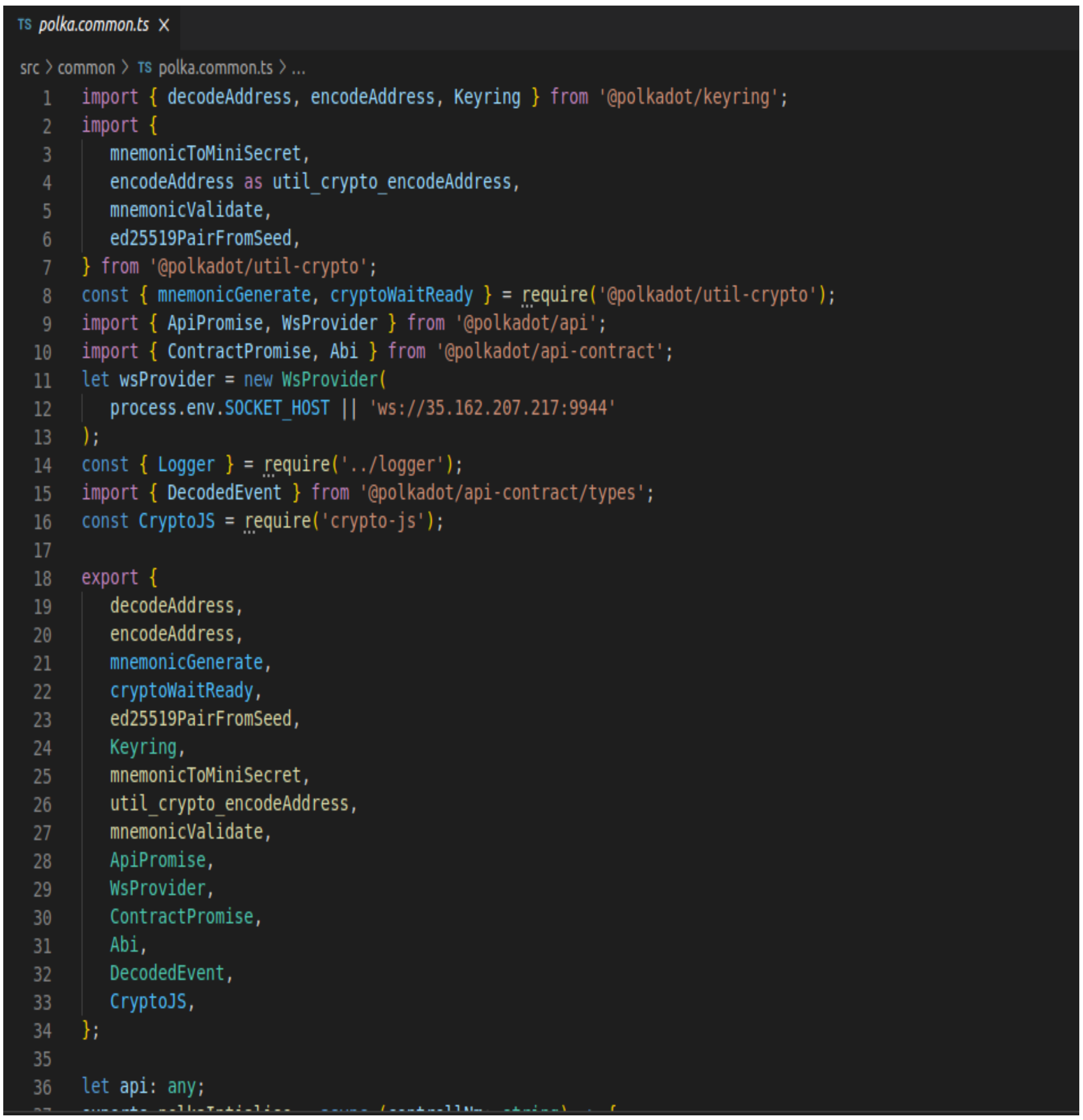


```
const ExpressCassandra = require('express-cassandra');
const cassandra = require('cassandra-driver');

const connect = ExpressCassandra.createClient({
   clientOptions: {
      contactPoints: [process.env.contactPoints || '127.0.0.1'],
      localDataCenter: 'datacenter1',
      protocolOptions: { port: 9042 },
      keyspace: process.env.keyspace || 'explorer',
      queryOptions: { consistency: ExpressCassandra.consistencies.one },
      paging: { local: true }, // enable local paging
      socketOptions: { readTimeout: 60000 },
   },
   ormOptions: {
      defaultReplicationStrategy: {
         class: 'SimpleStrategy',
         replication_factor: 1,
      },
      migration: 'safe',
   },
});

console.log('Cassandra connected');

module.exports = { connect };
```

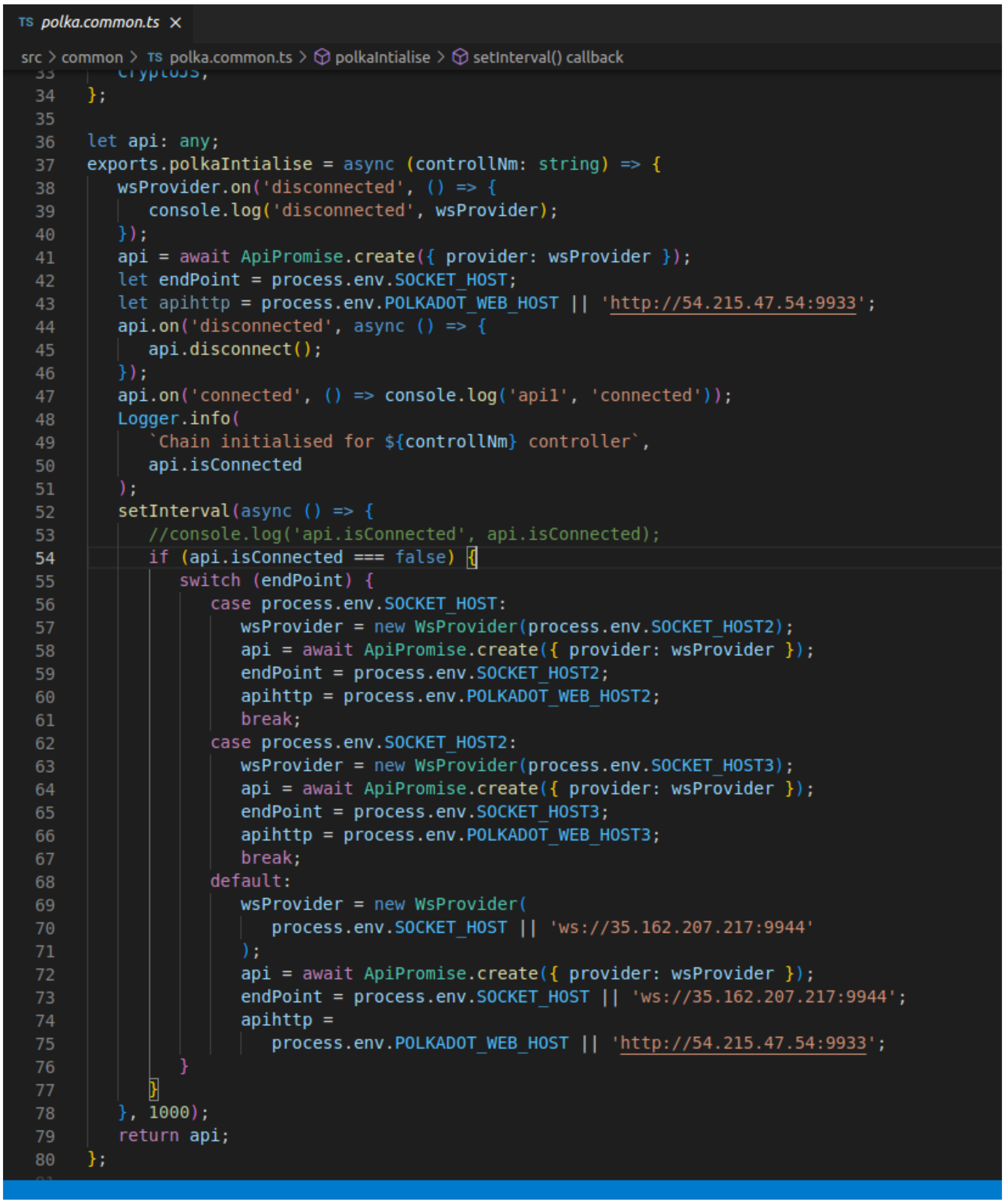
```
/var/lib/cassandra/data/explorer/Block-74301e90eff511ed9e331fffb987a7f6/nb-1-big
explorer  | INFO  [Native-Transport-Requests-1] 2023-05-11 12:17:51,964 Keyspace.java:381 - Creating
 replication strategy explorer params KeyspaceParams{durable_writes=true, replication=ReplicationPar
```

Polka Chain Initialisation:

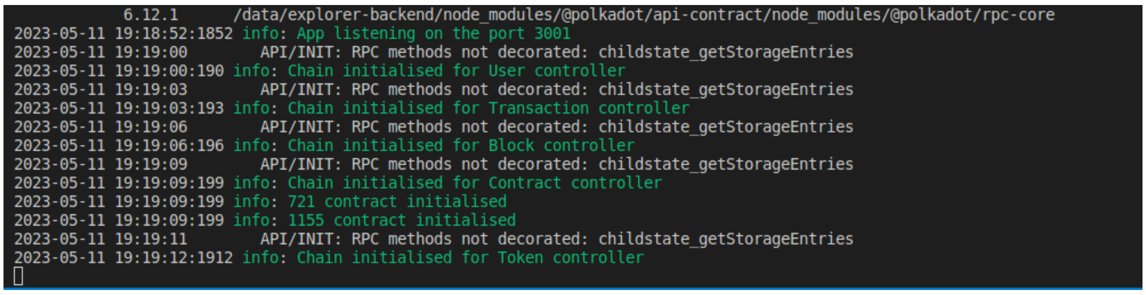
Here we initialize the polkachain testnet chain with the service.
The values of endpoints have beer derived from the environment file i.e, .env.



```typescript
import { decodeAddress, encodeAddress, Keyring } from '@polkadot/keyring';
import {
  mnemonicToMiniSecret,
  encodeAddress as util_crypto_encodeAddress,
  mnemonicValidate,
  ed25519PairFromSeed,
} from '@polkadot/util-crypto';
const { mnemonicGenerate, cryptoWaitReady } = require('@polkadot/util-crypto');
import { ApiPromise, WsProvider } from '@polkadot/api';
import { ContractPromise, Abi } from '@polkadot/api-contract';
let wsProvider = new WsProvider(
  process.env.SOCKET_HOST || 'ws://35.162.207.217:9944'
);
const { Logger } = require('../logger');
import { DecodedEvent } from '@polkadot/api-contract/types';
const CryptoJS = require('crypto-js');

export {
  decodeAddress,
  encodeAddress,
  mnemonicGenerate,
  cryptoWaitReady,
  ed25519PairFromSeed,
  Keyring,
  mnemonicToMiniSecret,
  util_crypto_encodeAddress,
  mnemonicValidate,
  ApiPromise,
  WsProvider,
  ContractPromise,
  Abi,
  DecodedEvent,
  CryptoJS,
};

let api: any;
```


TS polka.common.ts ×
src > common > TS polka.common.ts > polkaIntialise > setInterval() callback

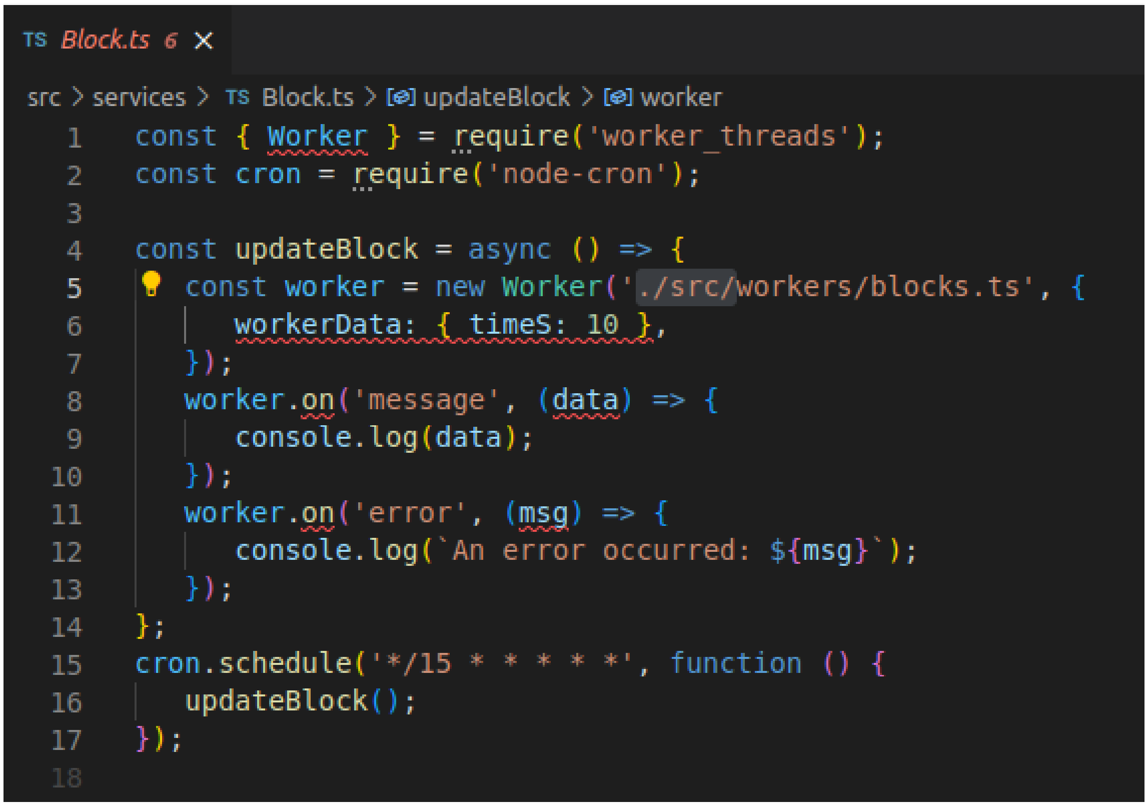
```
    CryptoJS,
};

let api: any;
exports.polkaIntialise = async (controllNm: string) => {
  wsProvider.on('disconnected', () => {
    console.log('disconnected', wsProvider);
  });
  api = await ApiPromise.create({ provider: wsProvider });
  let endPoint = process.env.SOCKET_HOST;
  let apihttp = process.env.POLKADOT_WEB_HOST || 'http://54.215.47.54:9933';
  api.on('disconnected', async () => {
    api.disconnect();
  });
  api.on('connected', () => console.log('api1', 'connected'));
  Logger.info(
    `Chain initialised for ${controllNm} controller`,
    api.isConnected
  );
  setInterval(async () => {
    //console.log('api.isConnected', api.isConnected);
    if (api.isConnected === false) {
      switch (endPoint) {
        case process.env.SOCKET_HOST:
          wsProvider = new WsProvider(process.env.SOCKET_HOST2);
          api = await ApiPromise.create({ provider: wsProvider });
          endPoint = process.env.SOCKET_HOST2;
          apihttp = process.env.POLKADOT_WEB_HOST2;
          break;
        case process.env.SOCKET_HOST2:
          wsProvider = new WsProvider(process.env.SOCKET_HOST3);
          api = await ApiPromise.create({ provider: wsProvider });
          endPoint = process.env.SOCKET_HOST3;
          apihttp = process.env.POLKADOT_WEB_HOST3;
          break;
        default:
          wsProvider = new WsProvider(
            process.env.SOCKET_HOST || 'ws://35.162.207.217:9944'
          );
          api = await ApiPromise.create({ provider: wsProvider });
          endPoint = process.env.SOCKET_HOST || 'ws://35.162.207.217:9944';
          apihttp =
            process.env.POLKADOT_WEB_HOST || 'http://54.215.47.54:9933';
      }
    }
  }, 1000);
  return api;
};
```



```
            6.12.1      /data/explorer-backend/node_modules/@polkadot/api-contract/node_modules/@polkadot/rpc-core
2023-05-11 19:18:52:1852 info: App listening on the port 3001
2023-05-11 19:19:00          API/INIT: RPC methods not decorated: childstate_getStorageEntries
2023-05-11 19:19:00:190 info: Chain initialised for User controller
2023-05-11 19:19:03          API/INIT: RPC methods not decorated: childstate_getStorageEntries
2023-05-11 19:19:03:193 info: Chain initialised for Transaction controller
2023-05-11 19:19:06          API/INIT: RPC methods not decorated: childstate_getStorageEntries
2023-05-11 19:19:06:196 info: Chain initialised for Block controller
2023-05-11 19:19:09          API/INIT: RPC methods not decorated: childstate_getStorageEntries
2023-05-11 19:19:09:199 info: Chain initialised for Contract controller
2023-05-11 19:19:09:199 info: 721 contract initialised
2023-05-11 19:19:09:199 info: 1155 contract initialised
2023-05-11 19:19:11          API/INIT: RPC methods not decorated: childstate_getStorageEntries
2023-05-11 19:19:12:1912 info: Chain initialised for Token controller
```

Services:

In Node.js, a worker service is a module that runs in the background and performs a specific task or set of tasks without blocking the main event loop of the application. Worker services are typically used for computationally intensive or long-running tasks that would otherwise block the event loop and make the application unresponsive.

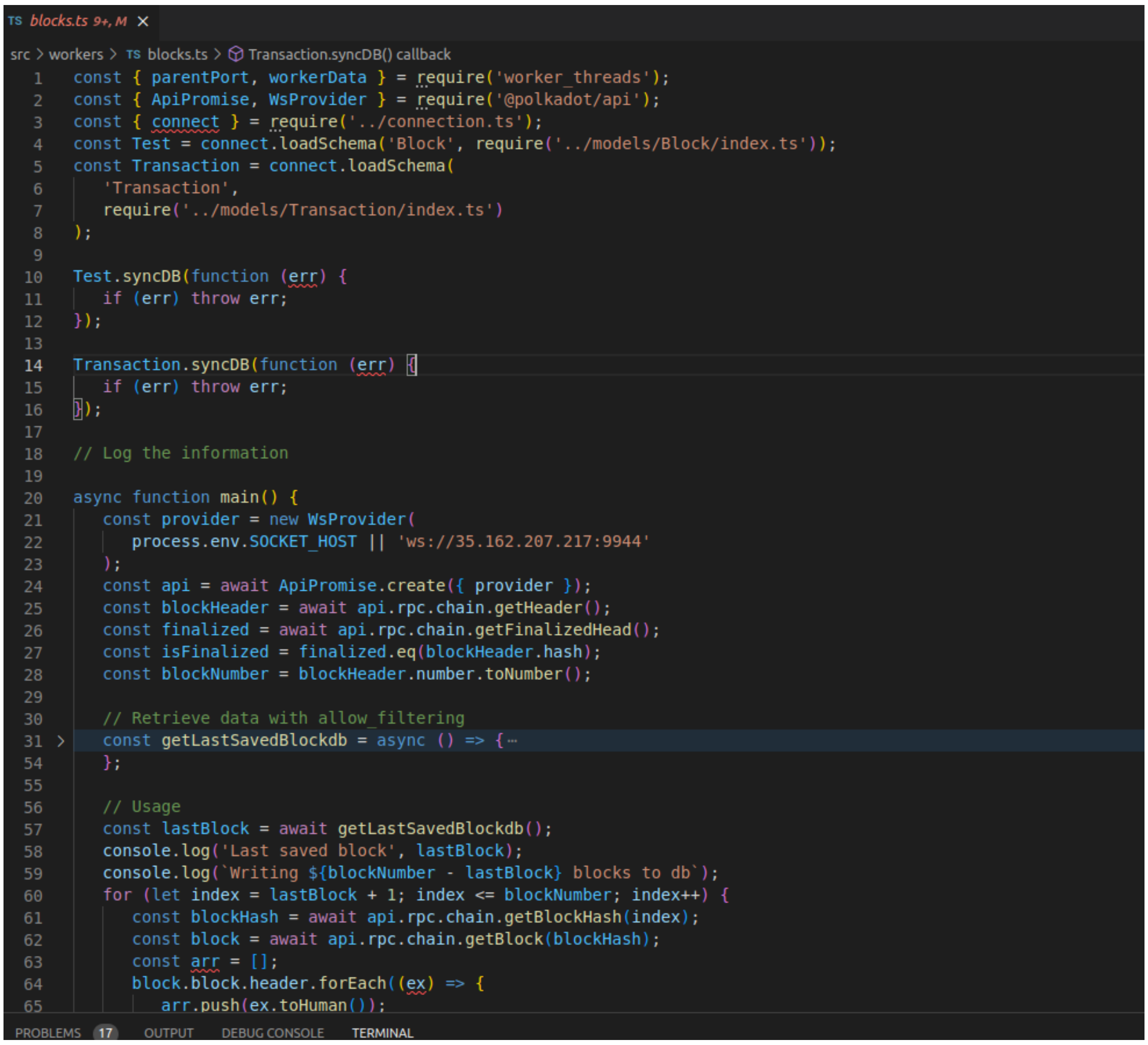


```
const { Worker } = require('worker_threads');
const cron = require('node-cron');

const updateBlock = async () => {
  const worker = new Worker('./src/workers/blocks.ts', {
    workerData: { timeS: 10 },
  });
  worker.on('message', (data) => {
    console.log(data);
  });
  worker.on('error', (msg) => {
    console.log(`An error occurred: ${msg}`);
  });
};
cron.schedule('*/15 * * * * *', function () {
  updateBlock();
});
```

Workers:

Here are some common characteristics of worker services in Node.js:

Worker services are typically implemented using the Worker API, which allows you to create and manage background threads in Node.js.
Worker services are often designed to be run as separate processes or clusters, allowing you to take advantage of multi-core CPUs and distribute the workload across multiple threads.

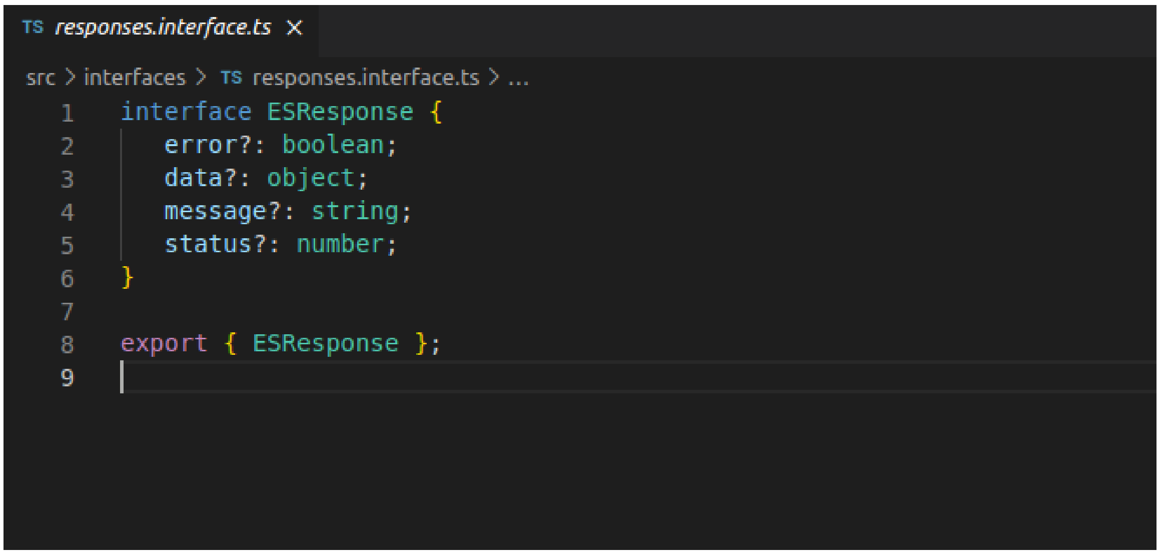


TS blocks.ts 9+, M ×
src > workers > TS blocks.ts > Transaction.syncDB() callback
PROBLEMS 17 OUTPUT DEBUG CONSOLE TERMINAL

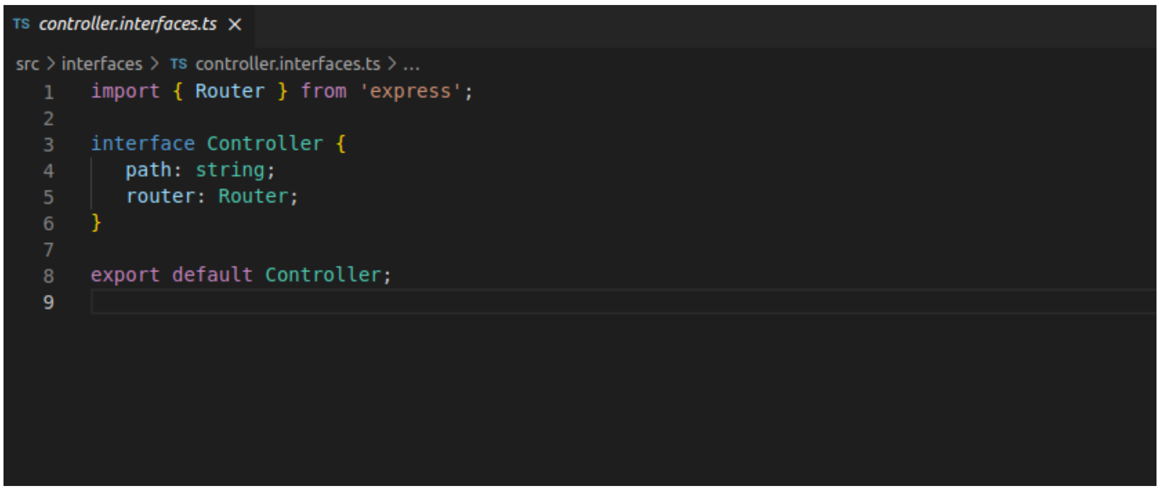
```
const { parentPort, workerData } = require('worker_threads');
const { ApiPromise, WsProvider } = require('@polkadot/api');
const { connect } = require('../connection.ts');
const Test = connect.loadSchema('Block', require('../models/Block/index.ts'));
const Transaction = connect.loadSchema(
  'Transaction',
  require('../models/Transaction/index.ts')
);

Test.syncDB(function (err) {
  if (err) throw err;
});

Transaction.syncDB(function (err) {
  if (err) throw err;
});

// Log the information

async function main() {
  const provider = new WsProvider(
    process.env.SOCKET_HOST || 'ws://35.162.207.217:9944'
  );
  const api = await ApiPromise.create({ provider });
  const blockHeader = await api.rpc.chain.getHeader();
  const finalized = await api.rpc.chain.getFinalizedHead();
  const isFinalized = finalized.eq(blockHeader.hash);
  const blockNumber = blockHeader.number.toNumber();

  // Retrieve data with allow_filtering
  const getLastSavedBlockdb = async () => {⋯
  };

  // Usage
  const lastBlock = await getLastSavedBlockdb();
  console.log('Last saved block', lastBlock);
  console.log(`Writing ${blockNumber - lastBlock} blocks to db`);
  for (let index = lastBlock + 1; index <= blockNumber; index++) {
    const blockHash = await api.rpc.chain.getBlockHash(index);
    const block = await api.rpc.chain.getBlock(blockHash);
    const arr = [];
    block.block.header.forEach((ex) => {
      arr.push(ex.toHuman());
```

Interfaces:

In Node.js, an interface is a contract or agreement between two or more modules, specifying the methods, properties, and behavior that they must implement in order to work together. Interfaces are often used to define the API or public interface of a module, allowing other modules to interact with it in a consistent and predictable way.



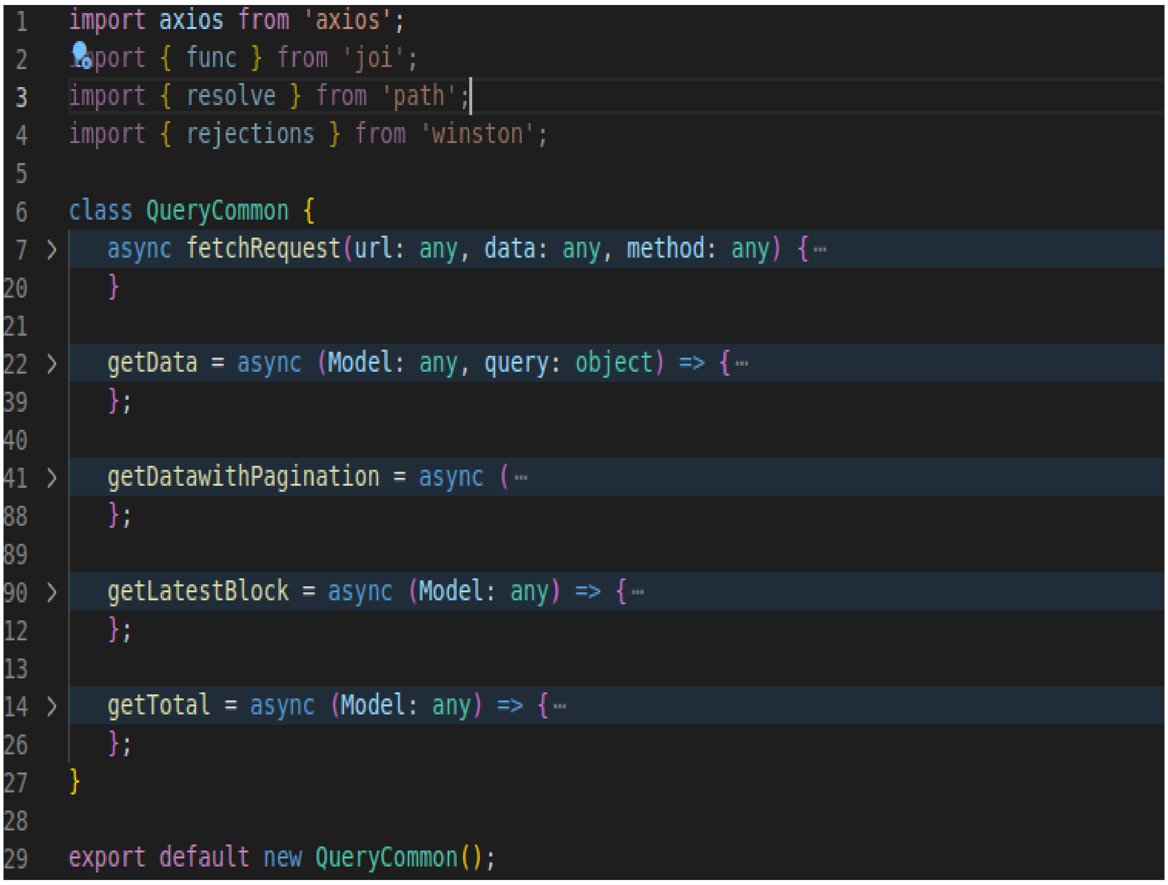
```
interface ESResponse {
    error?: boolean;
    data?: object;
    message?: string;
    status?: number;
}

export { ESResponse };
```



```
import { Router } from 'express';

interface Controller {
    path: string;
    router: Router;
}

export default Controller;
```

Helpers:

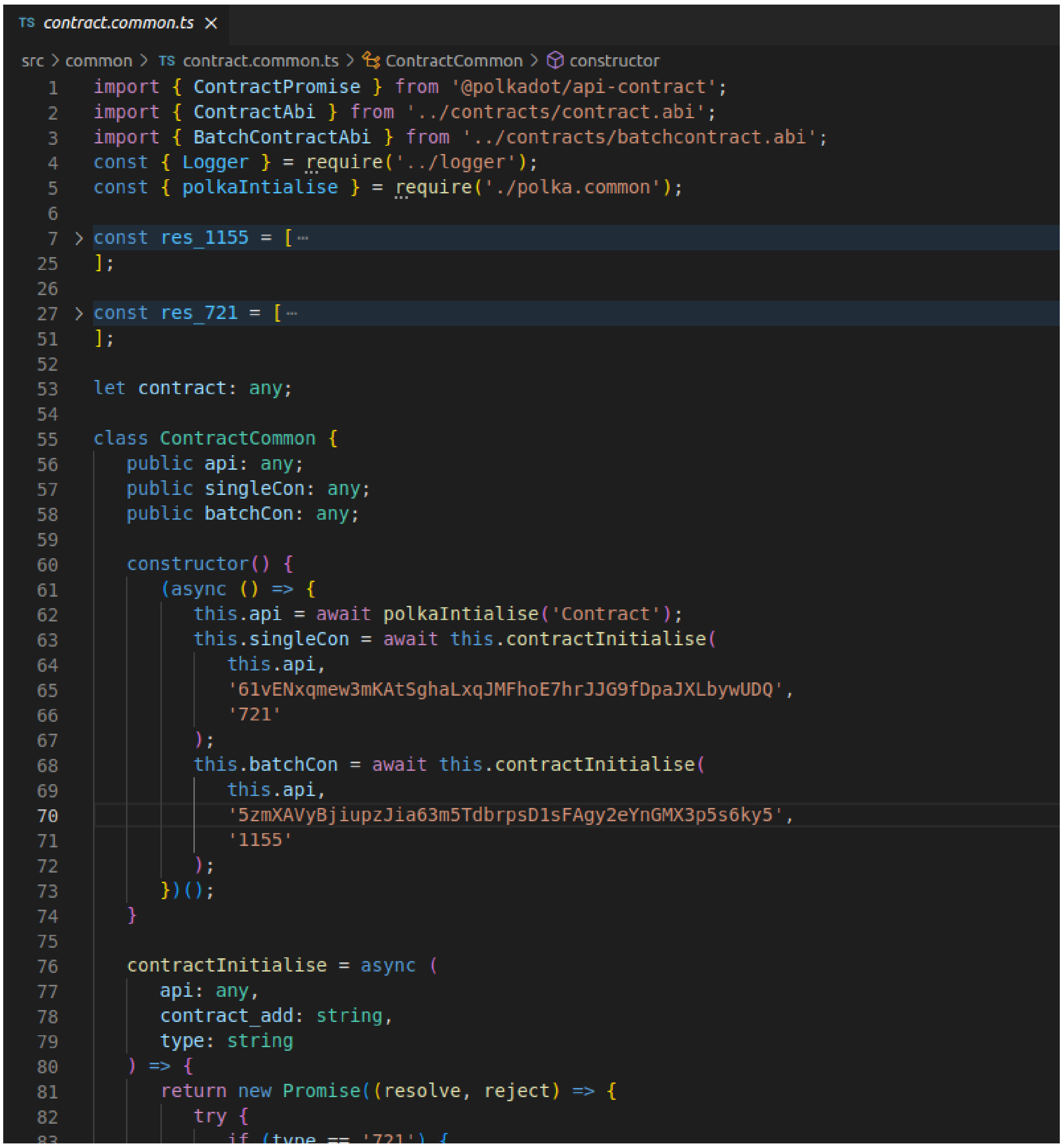
In Node.js, helpers are utility functions or modules that provide common functionality or assist in specific tasks. Helpers are often used to avoid repeating code, simplify complex tasks, or provide common functionality across multiple modules or applications.



```
import axios from 'axios';
import { func } from 'joi';
import { resolve } from 'path';
import { rejections } from 'winston';

class QueryCommon {
  async fetchRequest(url: any, data: any, method: any) {…
  }

  getData = async (Model: any, query: object) => {…
  };

  getDatawithPagination = async (…
  };

  getLatestBlock = async (Model: any) => {…
  };

  getTotal = async (Model: any) => {…
  };
}

export default new QueryCommon();
```

Contract Initialised:

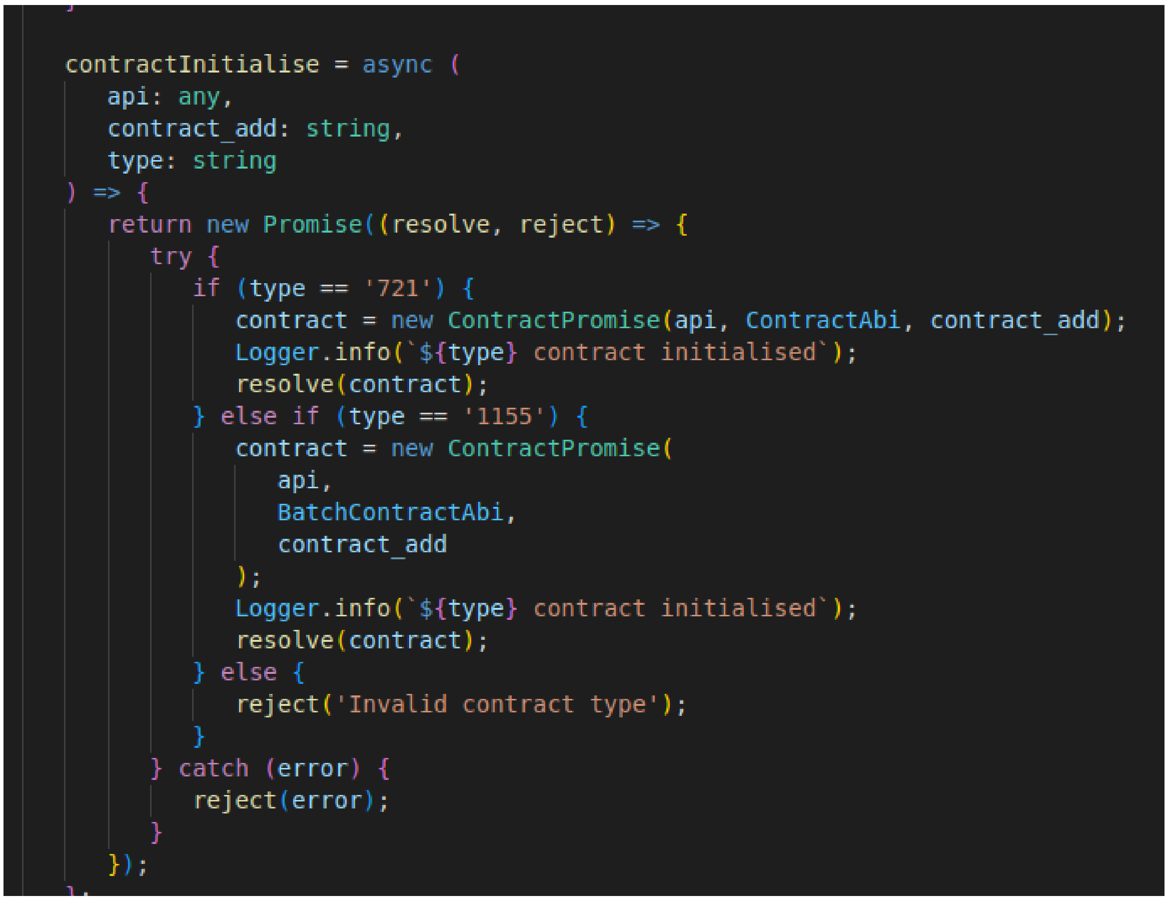
Here we initialized smart contracts for our query functions to be used on polkachain.



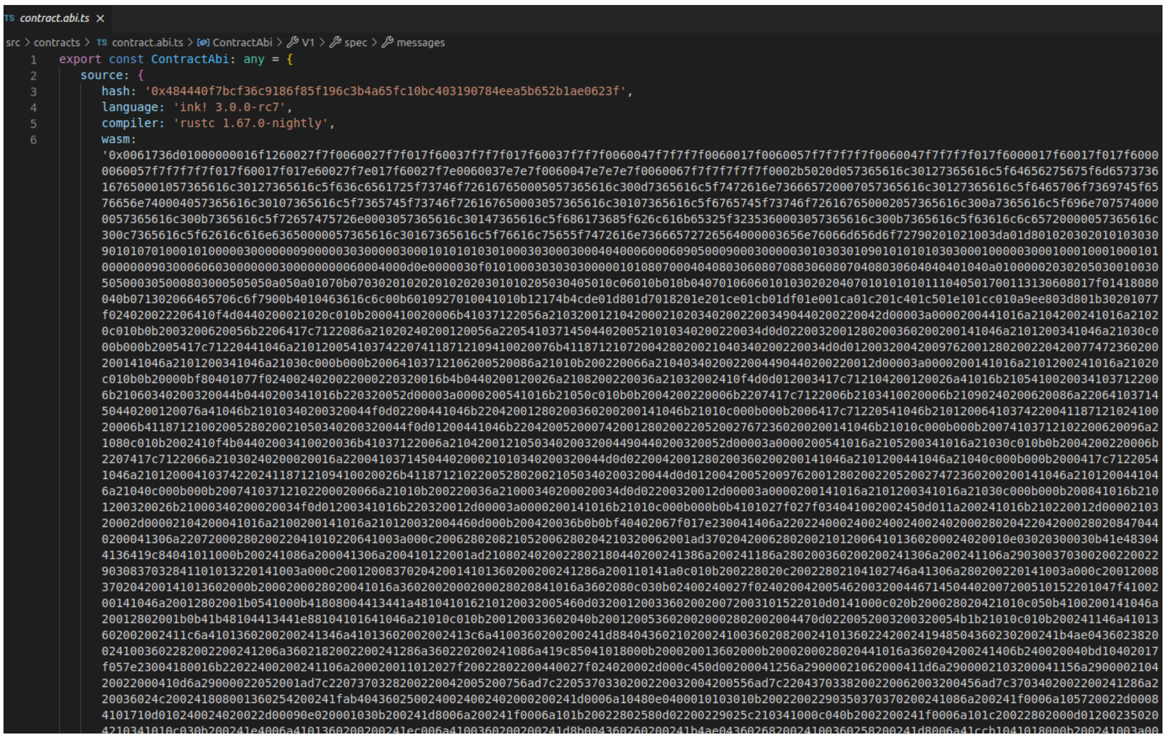
TS contract.common.ts ×

src > common > TS contract.common.ts > ContractCommon > constructor

```
import { ContractPromise } from '@polkadot/api-contract';
import { ContractAbi } from '../contracts/contract.abi';
import { BatchContractAbi } from '../contracts/batchcontract.abi';
const { Logger } = require('../logger');
const { polkaIntialise } = require('./polka.common');

const res_1155 = [⋯
];

const res_721 = [⋯
];

let contract: any;

class ContractCommon {
  public api: any;
  public singleCon: any;
  public batchCon: any;

  constructor() {
    (async () => {
      this.api = await polkaIntialise('Contract');
      this.singleCon = await this.contractInitialise(
        this.api,
        '61vENxqmew3mKAtSghaLxqJMFhoE7hrJJG9fDpaJXLbywUDQ',
        '721'
      );
      this.batchCon = await this.contractInitialise(
        this.api,
        '5zmXAVyBjiupzJia63m5TdbrpsD1sFAgy2eYnGMX3p5s6ky5',
        '1155'
      );
    })();
  }

  contractInitialise = async (
    api: any,
    contract_add: string,
    type: string
  ) => {
    return new Promise((resolve, reject) => {
      try {
        if (type == '721') {
```



```
  }

  contractInitialise = async (
    api: any,
    contract_add: string,
    type: string
  ) => {
    return new Promise((resolve, reject) => {
      try {
        if (type == '721') {
          contract = new ContractPromise(api, ContractAbi, contract_add);
          Logger.info(`${type} contract initialised`);
          resolve(contract);
        } else if (type == '1155') {
          contract = new ContractPromise(
            api,
            BatchContractAbi,
            contract_add
          );
          Logger.info(`${type} contract initialised`);
          resolve(contract);
        } else {
          reject('Invalid contract type');
        }
      } catch (error) {
        reject(error);
      }
    });
  };
```



contract.abi.ts

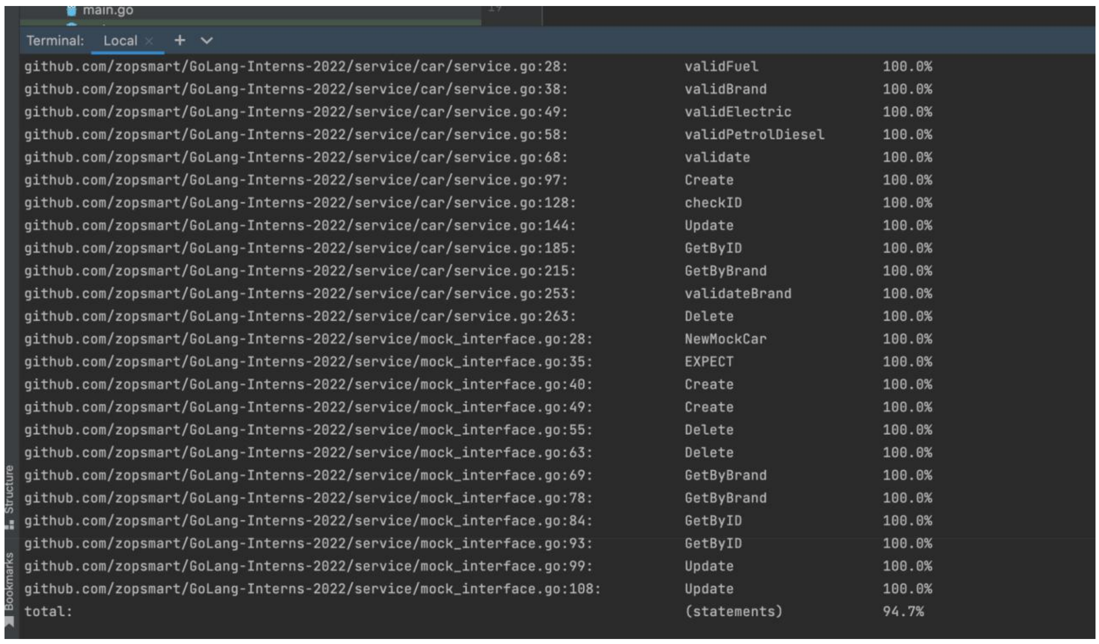
src > contracts > contract.abi.ts > ContractAbi > V1 > spec > messages

```
export const ContractAbi: any = {
  source: {
    hash: '0x484440f7bcf36c9186f85f196c3b4a65fc10bc403190784eea5b652b1ae0623f',
    language: 'ink! 3.0.0-rc7',
    compiler: 'rustc 1.67.0-nightly',
    wasm:
    '0x0061736d01000000016f1260027f7f0060027f7f017f60037f7f7f017f60037f7f7f0060047f7f7f7f0060017f0060057f7f7f7f7f0060047f7f7f7f017f6000017f60017f017f6000
    0060057f7f7f7f7f017f60017f017e60027f7e017f60027f7e0060037e7e7f0060047e7e7e7f0060067f7f7f7f7f7f0002b5020d057365616c30127365616c5f64656275675f6d6573736
    167650001057365616c30127365616c5f636c6561725f73746f726167650005057365616c300d7365616c5f7472616e736665720007057365616c30127365616c5f6465706f7369745f65
    76656e740004057365616c30107365616c5f7365745f73746f726167650003057365616c30107365616c5f6765745f73746f726167650002057365616c300a7365616c5f696e707574000
    0057365616c300b7365616c5f72657475726e0003057365616c30147365616c5f686173685f626c616b65325f3235360003057365616c300b7365616c5f63616c6c65720000057365616c
    300c7365616c5f62616c616e63650000057365616c30167365616c5f76616c75655f7472616e736665727265640000003656e76066d656d6f72790201021003da01d801020302010103030
    901010701000101000003000000090000030300000300010101010301000303000300040400060000609050009000300000301030301090101010103030001000003000100010001000101
    0000000903000606030000000300000000060004000d0e0000030f0101000303030300000101080700040408030608070803060807040803060404040104 0a0100000203020503001003 0
    505000305000803000505050a050a01070b070302010202010202030101020503040501 0c06010b010b040701060601010302020407010101010111040501700113130608017f01418080
    040b071302066465706c6f7900b4010463616c6c00b6010927010041010b12174b4cde01d801d7018201e201ce01cb01df01e001ca01c201c401c501e101cc010a9ee803d801b30201077
    f024020022206410f4d0440200021020c010b2000410020006b41037122056a2103200121042000210203402002200349044020022004 2d00003a0000200441016a2104200241016a2102
    0c010b0b2003200620056b2206417c7122086a21020240200120056a220541037145044020052101034020022003 4d0d0220032001280200360200200141046a2101200341046a21030c0
    00b000b2005417c71220441046a2101200541037422074118712109410020076b4118712107200428020021040340200220034d0d012003200420097620012802002204200774723602 00
    200141046a2101200341046a21030c000b000b200641037121062005 20086a21010b200220066a2104034020022004490440200220012d00003a0000200141016a2101200241016a21020
    c010b0b20000bf80401077f024002402002200022000220320016b4b0440200120026a2108200220036a21032002410f4d0d012003417c712104200120026a41016b2105410020034103712200
    6b21060340200320044b0440200341016b220320052d00003a0000200541016b21050c010b0b2004200220006b2207417c7122006b2103410020006b21090240200620086a22064103714
    50440200120076a41046b21010340200320044f0d02200441046b22042001280200360200200141046b21010c000b000b2006417c71220541046b21012006410374220041187121024100
    20006b4118712100200528020021050340200320044f0d01200441046b220420052000742001280200220520027672360200200141046b21010c000b000b20074103712102200620096a2
    1080c010b2002410f4b04402003410020036b41037122006a2104200121050340200320044904402003200 52d00003a0000200541016a2105200341016a21030c010b0b2004200220006b
    2207417c7122066a21030240200020016a220041037145044020002101034020032004 4d0d022004200128020036020020 0141046a2101200441046a21040c000b000b2000417c7122054
    1046a2101200041037422024118712109410020026b4118712102200528020021050340200320044d0d012004200520097620012802002205200274723602002001 41046a210120044104
    6a21040c000b000b20074103712102200020066a21010b200220036a21000340200020034d0d02200320012d00003a0000200141016a2101200341016a21030c000b000b200841016b210
    1200320026b21000340200020034f0d01200341016b220320012d00003a0000200141016b21010c000b000b0b4101027f027f034041002002450d011a200241016b210220012d00002103
    20002d00002104200041016a2100200141016a210120032004460d000b200420036b0b0bf40402067f017e230041406a22022400024002400240024020002802042204200028020847044
    0200041306a2207200028020022041010220641003a000c200628020821052006280204210320062001ad37020420062802002101200641013602000240 20010e03020300030b41e48304
    4136419c84041011000b200241086a200041306a200410122001ad21080240200228021804402002 41386a200241186a280200360200200241306a200241106a290300370300200220022
    9030837032841101013220141003a000c2001200837020420014101360200200241286a200110141a0c010b200228020c20022802104102746a41306a280200220141003a000c20012008
    370204200141013602000b2000200028020041016a3602002000200028020841016a3602080c030b02400240027f024020042005462003200446714504402007200510152201047f41002
    00141046a20012802001b0541000b41808004413441a481041016210120032005460d0320012003360200200720031015 22010d0141000c020b200028020421010c050b4100200141046a
    20012802001b0b41b48104413441e88104101641046a21010c010b200120033602040b20012005360200200028020020044 70d022005200320032005 4b1b21010c010b200241146a41013
    602002002411c6a4101360200200241346a41013602002002413c6a4100360200200241d88404360210200241003602082002410136022420024194850436023020 0241b4ae0436023820
    02410036022820022002 41206a36021820022002 41286a360220200241086a419c85041018000b20002001 3602000b2000200028020441016a360204200241406b240020040bd10402017
    f057e23004180016b22022400200241106a200020011012027f20022802200440027f024020002d000c450d00200041256a29000021062000411d6a29000021032000 41156a2900002104
    20022000410d6a29000022052001ad7c2207370328200220042005200756ad7c22053703302002200320042005 56ad7c22043703382002200620032004 56ad7c3703402002200241286a2
    20036024c200241808001360254200241fab4043602500240024002402000200241d0006a10480e0400010103010b20022002290350370370200241086a200241f0006a105720022d0008
    4101710d010240024020022d00090e020001030b200241d8006a200241f0006a101b20022802580d02200229025c210341000c040b2002200241f0006a101c20022802000d01200235020
    4210341010c030b200241e4006a4101360200200241ec006a4100360200200241d8b004360260200241b4ae04360268200241003602582002 41d8006a41ccb1041018000b200241003a00
```

4. PERFORMANCE ANALYSIS

1. Unit Test Coverage

Performed unit test coverage and found all 44 tests ran successfully i.e PASS with a total coverage of 94.7%.



```
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:28:              validFuel           100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:38:              validBrand          100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:49:              validElectric       100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:58:              validPetrolDiesel   100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:68:              validate            100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:97:              Create              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:128:             checkID             100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:144:             Update              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:185:             GetByID             100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:215:             GetByBrand          100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:253:             validateBrand       100.0%
github.com/zopsmart/GoLang-Interns-2022/service/car/service.go:263:             Delete              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:28:           NewMockCar          100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:35:           EXPECT              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:40:           Create              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:49:           Create              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:55:           Delete              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:63:           Delete              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:69:           GetByBrand          100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:78:           GetByBrand          100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:84:           GetByID             100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:93:           GetByID             100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:99:           Update              100.0%
github.com/zopsmart/GoLang-Interns-2022/service/mock_interface.go:108:          Update              100.0%
total:                                                                          (statements)        94.7%
```

2. Linter Check

Performed a linter check using command ***golangci-lint run*** which makes sure that the program is properly formatted and follows standard code guidelines such as no gocognit complexity or funlen to be 0 etc. There were **no** linter errors found in this project.

5. CONCLUSION

5.1 Results Achieved

The main aim of the training was to be able to understand and implement the concepts of **GoLang, MySQL, Unit Testing,** being able to create a web application successfully performing basic CRUD operations and can be tested using postman using the **three layered architecture.**

5.2 Applications Contributions

GoLang have been part of a variety of real world/ open source applications, some of the which are listed below.

Docker, a set of tools for deploying Linux containers, Kubernetes container management system

1. Dropbox, who migrated some of their critical components from Python to Go
2. Ethereum, The go-ethereum implementation of the Ethereum Virtual Machine, blockchain for the Ether cryptocurrency
3. Gitlab, a web-based DevOps lifecycle tool that provides a Git-repository, wiki, issue-tracking, continuous integration, deployment pipeline features etc.

5.3 Limitations

The application implements only the backend part but front end can be done for the same to make the application more attractive and user friendly.

5.4 Future Work / Scope

1. Front-end for application
2. Make the program more extensive