CHAPTER - 1
OVERVIEW OF COMPANY

1.1 HISTORY

Founded in 1998 by three visionary entrepreneurs, RKIT Software Pvt. Ltd. has emerged as a prominent player among India's top ten accounting software firms. Initially specializing in bespoke software solutions, the company strategically transitioned to offering a range of packaged accounting software solutions. Today, with a user base exceeding 75,000, our firm's flagship general-purpose accounting software stands as a market leader and preferred choice across various sectors.

Fig 1.1 Company Logo

Fig 1.2 Product Logo

Our primary offering, '*Miracle*', represents a scalable solution catering to the diverse needs of accountants, small enterprises, and large corporations alike. Complementing this, our product portfolio includes '*MyInvestor*', '*Miracle-G'*, '*MiracleMandi*' and '*Miracle-Petro*'.

Based in Rajkot, Gujarat, RKIT Software Pvt. Ltd. operates sales of software from almost all key locations of India. We have established a robust dealer network covering Gujarat, Maharashtra, Madhya Pradesh, Chhattisgarh, Rajasthan, Assam, and Tamil Nadu, with ongoing expansion plans to penetrate additional Indian states. Furthermore, our firm has successfully extended its presence into Oman, where we operate a growing dealer network, and we are actively pursuing opportunities for expansion into other Arab countries.

1.2 SCOPE OF WORK

- Miracle (flagship accounting software)
- My-Investor (additional patch for stock -market users)
- Miracle-Petro (additional patch for petrochemical industries)
- Miracle-Report (mobile application for report generation)
- Miracle-Textile (additional patch for businesses of textile)

1.3 DETAILED WORK OF THE COMPANY

1.3.1 HR Department

Human resources is responsible for scheduling interviews, managing hiring initiatives, and onboarding new workers.

1.3.2 Marketing Team

Managing marketing campaigns, developing content, and ensuring that the company's website is optimised for search engines. Creating and maintaining the company's brand.

1.3.3 Development Team

Creating a product that meets current industry standards and is simple to use, browse, and use.

1.3.4 Client service

Handle and resolve customer concerns in a timely and effective way. Support professionals communicate with consumers across a range of channels, including phone, email, and social media, and ensure that any legitimate customer complaints are promptly addressed.

1.3.5 Research Team

Analysing Product and attempting to alter and add new features depending on customer input.

1.4 SUMMARY

1.4.1 Website

https://www.rkitsoftware.com

1.4.2 Industries

IT Services and Product

1.4.3 Developer Capacity

200 employees

1.4.4 Main Branch

Rajkot, Gujarat

CHAPTER - 2
INTERNSHIP INTRODUCTION

2.1 INTERNSHIP SUMMARY

In January 2024, I commenced my internship journey at RKIT Software Pvt. Ltd. Operating from the Rajkot office, I found myself immersed in a stimulating work environment that fuelled my inspiration on a daily basis. The team members exhibited qualities of amiability, helpfulness, customer-centricity, empathy, and integrity, fostering a conducive atmosphere for professional growth. This stint provided me with a platform to showcase my capabilities as a diligent employee, a reliable colleague, and an eager learner, while also allowing me to acquire invaluable office experience, a facet lacking in my prior roles.

One of the paramount skills I honed during my internship was adaptability. Engaging with diverse technologies was a routine occurrence in the corporate landscape, and I embraced this challenge wholeheartedly throughout my tenure.

As I undertook various tasks and demonstrated proficiency, I was entrusted with additional responsibilities, which I enthusiastically embraced. The constructive feedback and encouragement I received from my colleagues served as a catalyst for my confidence and growth. I express my heartfelt gratitude to all my peers at Protium for affording me the opportunity to evolve both personally and professionally.

2.2 PURPOSE / OBJECTIVE

Engaging in this internship has proven to be an invaluable learning journey, allowing me to forge new connections and establish meaningful professional relationships. In today's competitive job market, academic achievements alone are insufficient to excel in the workplace. Real-world experience is paramount. Internships offer a unique opportunity to garner practical insights, expand one's skill set, and assess alignment with chosen career paths.

The primary objective of this project is to acquire comprehensive knowledge and proficiency in a spectrum of technical domains, including *C#*, *.Net Framework*, *ASP .Net Core, MySQL, System Designing* and *Caching*. The overarching aim is to develop expertise in constructing and testing APIs using industry-standard tools like *Swagger* and *Postman*,

while also designing and implementing corresponding front-end solutions using *Javascript* and *jQuery*.

2.3 INTERNSHIP PLANNING

2.3.1 Week 1 to Week 4

- In depth study of Database Modeling & Query Processing (MySql & Workbench 8.0 CE)
- Overview of .Net Framework
- HTTP Action Methods
- CORS & its implementation
- Authorization & Security Control
- Exception Handling
- Versioning in Web API
- Web API which implements all above topic (URL Shortener)

2.3.2 Week 5 to Week 8

- Generics & it's use case
- File System Handling
- Data Serialization
- Database Integration with Web API
- Query based Database Access
- ORM based Database Access
- Create Base Library in C# Advance
- LINQ
- Dynamic, Object & Lambda Expression
- Cryptography
- Web API which implements all above topics (Expense Tracker)
- Testing & Improving Web API
- HLD Overview & its's importance

2.3.3 Week 9 to Week 12

- Caching in Redis
- HTTP Caching
- Difference between .Net Core & .Net Framework
- ASP .Net Core
- Request Pipelining
- Middleware & Filters
- How, Why & When to use Dependency Injection ?
- Logging
- LLD & it's implementation
- Craft Structure of efficient ASP .Net Core Web API

CHAPTER - 3

.NET FRAMEWORK WEB API TRAINING

3.1 WEB API PROJECT

.Net Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices. It develops RESTful (Representational State Transfer) applications. Web API consists Controllers, Models, Filters, Middleware, Global.asax file and .csproj file which is a project file.

3.2 ACTION METHOD RESPONSE

HTTP consists following methods, all implementation & source code is available in github repository.

Method	URI	Operation	Description	Request body	Response body
GET	/api/movies	Read	Get all movie records	None	Movie records
GET	/api/movies/{id}	Read	Get a movie record by ID	None	Movie record
POST	/api/movies	Create	Add a new movie record	Movie record	Movie record
PUT	/api/movies/{id}	Update	Update an existing movie record	Movie record	None
DELETE	/api/movies/{id}	Delete	Delete a movie record	None	None

Fig 3.1 HTTP Action Methods

3.2.1 Get Method

Get Method is used for the purpose of getting / extracting from the database.

3.2.2 Post Method

Post method is used for the purpose of addition / creation in the database.

3.2.3 Put Method

Put method is used for the updation within the database. Any database record can be updated by using put method. Here, updation means we have to provide all details of the object, put method is not capable to make changes on specific fields, if we want to update only one field in record still we have to provide other all fields too.

3.2.4 Delete Method

Delete method is used for the deletion of the record in the database.

3.2.5 Patch Method

Patch method is used for the updation within the database. Any database record can be updated by using patch method. Here, updation means we have to provide only details of the object which we want to update, other details will be as itis as before.

3.2.6 Response Status Codes

3.2.6.1 1xx – Informational

100 – Continue

3.2.6.2 2xx – Successful

200 – Ok

201 – New resource created.

204 – No Content

3.2.6.3 3xx – Redirection

301 – Moved permanently resource which is asked

302 – Moved temporarily resource which is asked

3.2.6.4 4xx – Client Errors

400 – Bad Request

401 – Unauthorized (Invalid authentication)

403 – Authentication is success but authorization has error.

404 – Requested resource is not available or found.

405 – Not Supported

3.2.6.5 5xx – Server Side Errors

500 – Internal server error

503 – Service Unavailable

504 – Gateway Timeout

3.3 SECURITY

Achieving security in Web APIs involves implementing several key techniques:

3.3.1 Cross-Origin Resource Sharing (CORS)

Browser security protocols, like the same-origin policy, restrict web pages from making AJAX requests to different domains, preventing potential data breaches. However, there are scenarios where allowing cross-origin requests is necessary.

CORS, a W3C standard, enables servers to selectively permit cross-origin requests while maintaining security standards.

3.3.2 Authentication

Authentication serves as a crucial layer of defense against unauthorized access to applications and websites. By verifying user credentials, it prevents unauthorized users from accessing sensitive information, including through tools like Postman and Fiddler. For Web API access, user credentials must be passed in the request header. Failure to provide valid credentials results in a 401 (Unauthorized) status code, indicating support for Basic Authentication.

3.3.3 Authorization

Authorization dictates the permissions granted to users regarding web pages, functionality, and data. It determines whether a user can perform specific actions or access particular functionalities. Authorization in Web APIs is typically implemented using authorization filters, which execute before controller actions. The built-in authorization filter, Authorize Attribute, verifies user authentication and returns a 401 (Unauthorized) status code if authentication fails, without invoking the action.

3.3.4 Jwt Token

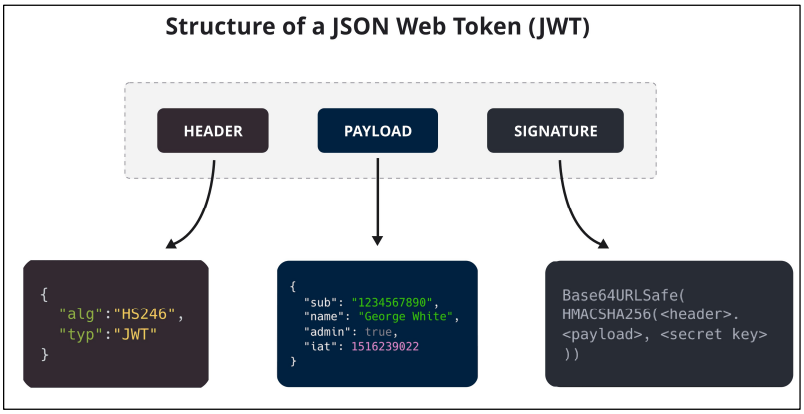


Fig 3.2 Structure of Jwt Token

JWTs are widely utilized to authenticate users securely. Issued by an authentication server, JWTs are consumed by the client-server to secure its APIs. By utilizing JWTs, authenticated users can be identified and their access to resources can be managed effectively.

3.4 CACHING

Caching is a technique of storing frequently used data or information in a local memory, for a certain time period. So, next time, when the client requests the same information, instead of retrieving the information from the database, it will give the information from the local memory. Common Cache-Control directives are shown in the following table :

Table 3.1 HTTP Caching Properties

Directive	Action
public	A cache may store the response.
private	The response must not be stored by a shared cache. A private cache may store and reuse the response.
max-age	The client doesn't accept a response whose age is greater than the specified number of seconds. Examples: max-age=60 (60 seconds), max-age=2592000 (1 month)
Age	An estimate of the amount of time in seconds since the response was generated or successfully validated at the origin server.
Expires	The time after which the response is considered stale.

3.4.1 Memory Cache

Memory cache is suitable for scenarios where you want to store data in memory for quick access within a single application. It's commonly used for caching data that is expensive to compute or retrieve from a data source.

3.4.2 Distributed Caching

Distributed caching is useful in scenarios where you have multiple instances of your application or multiple servers and need to share cached data among them. Redis is the most popular example of distributed caching.

3.4.3 HttpContext.Cache

HttpContext.Cache is specific to Asp .Net applications and is useful for caching data that is tied to a particular request or needs to be shared among different parts of the application during the request lifecycle.

3.4.4 Object Cache

ObjectCache provides a common interface for different caching implementations. It's beneficial when you want to abstract your code from a specific caching provider and

work with a common set of caching features.

3.4.5 Output Cache

Output caching is used to cache the entire output of a page or a specific action method in an ASP.NET application. This is beneficial when the content of a page or action is relatively static and can be reused for multiple requests.

3.5 VERSIONING

Web API Versioning is required as the business grows and business requirement changes with the time. As Web API can be consumed by multiple clients at a time, Versioning of Web API will be necessarily required so that Business changes in the API will not impact the client that are using/consuming the existing API. Web API Versioning can be done by using the following methods.

- URI based Versioning
- Query String Versioning
- Custom Header Versioning
- Accept Header Versioning

3.6 USE OF SWAGGER

Swagger is an open-source framework that provides a comprehensive set of guidelines, specifications, and tools tailored for the development and description of RESTful APIs. This framework empowers developers to create interactive and easily understandable API documentation, both for machines and humans. API specifications encapsulate crucial details such as supported operations, input parameters, output formats, authentication requirements, available endpoints, and necessary licenses. Swagger simplifies the generation of such specifications by leveraging annotations within the source code, enabling automatic documentation creation.

This versatile framework facilitates the entire lifecycle of RESTful web services, from design and documentation to testing and consumption. It accommodates different development methodologies, including both top-down and bottom-up approaches. In the top-down approach, also known as design-first, Swagger aids in the preliminary design of APIs even before code implementation. Conversely, in the bottom-up approach, or code-first method, Swagger generates comprehensive documentation from existing API code.

The advantages of utilizing Swagger in API development are numerous:

- Intuitive User Interface
- Human and Machine Readability
- Interactive Documentation
- Multilingual Support
- Flexible Formats

Overall, Swagger serves as a valuable tool for developers and organizations aiming to enhance the efficiency, clarity, and accessibility of their RESTful API development processes.

3.7 DEPLOYMENT

Web API deployment involves packaging the API along with dependencies, deploying it to a chosen environment (on-premises, cloud, or serverless), and exposing endpoints for client interaction. Continuous integration and deployment streamline the process, while monitoring ensures reliability and performance post-deployment.

CHAPTER - 4
ADVANCE C# TRAINING

4.1 GENERICS

Generics in C# and Web API development facilitate code reusability and type safety. They enable the creation of classes, methods, and interfaces that work with any data type rather than being specific to one. In Web API development, generics are vital for handling diverse data structures like collections and DTOs across controllers, services, and repositories. By employing generics, developers can build adaptable and strongly typed components, enhancing maintainability and scalability. For example, generic methods in controllers streamline CRUD operations for various resources without code duplication. Additionally, generics are utilized in middleware and filters for consistent processing of requests and responses. Overall, leveraging generics empowers developers to write cleaner, more maintainable code while ensuring type safety and minimizing errors.

4.2 FILE SYSTEM

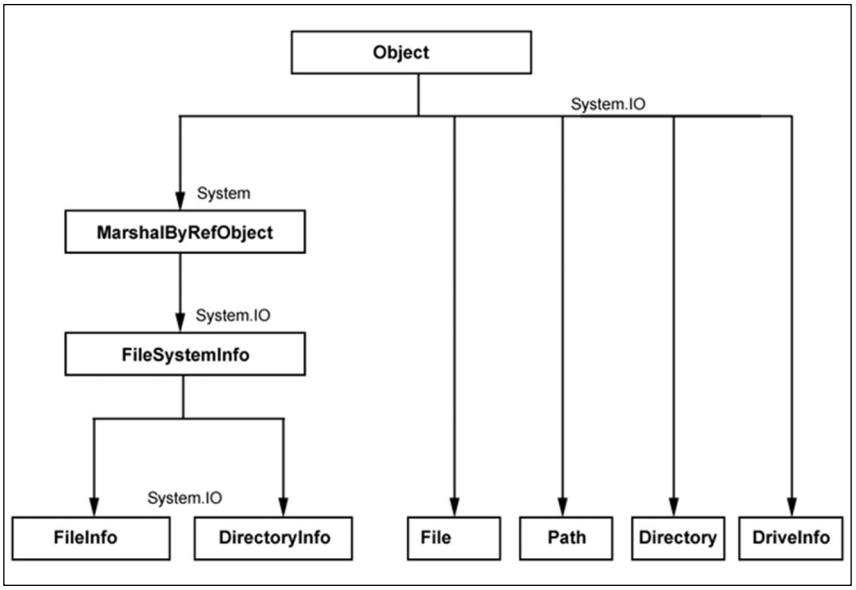


Fig 4.1 Classes for File System in C#

The file system, integral to computing, orchestrates the organization and storage of data on various storage devices. It comprises directories and files, structured hierarchically, with directories containing files and other directories. *FileInfo* and *DirectoryInfo* classes in C# provide essential tools for navigating and manipulating this structure. *FileInfo* represents

individual files, offering details like name, size, and creation time, while *DirectoryInfo* handles directories, facilitating operations such as creation, deletion, and accessing contents. Together, these components enable efficient data management and interaction within software applications.

4.3 DATA SERIALIZATION

Serialization involves converting an object or data structure into a format that can be easily transmitted over a network or stored in a persistent storage medium like a file or a database. This process is necessary when you need to transfer an object's state between different components of a distributed system or when persisting data for later use. During serialization, the object's properties or fields are converted into a linear stream of bytes or a text-based format, which can then be sent or stored.

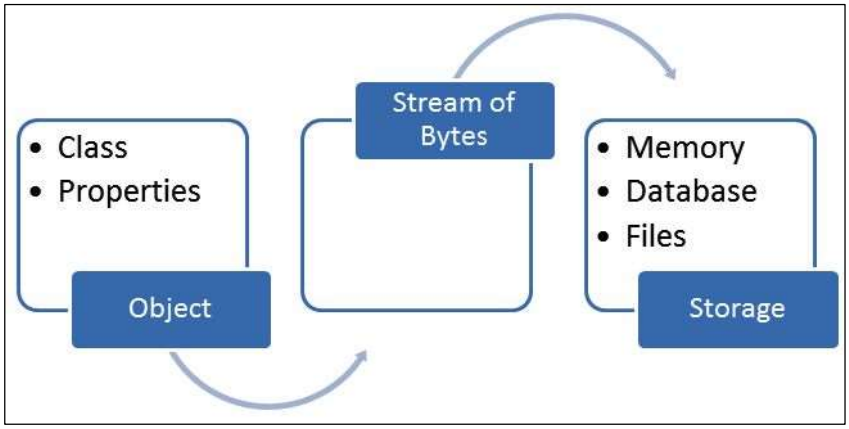


Fig 4.2 Serialization Process

Deserialization is the reverse process of serialization. It involves taking a serialized data stream and reconstructing the original object or data structure from it. This is crucial when receiving data over a network or retrieving data from a storage medium. The deserialization process interprets the serialized data, extracts the relevant information, and recreates the object with its original state.

4.4 LAMDA EXPRESSION

Lambda expressions in C# provide a significant way to create anonymous functions inline, enhancing code readability and expressiveness. They are commonly used for tasks like sorting, filtering, and defining delegates without the need for separate method declarations. Lambda expressions offer a flexible and concise syntax, making them a powerful tool for

improving code efficiency and maintainability.

4.5 LINQ

LINQ, or Language Integrated Query, is a set of features in the Microsoft .NET framework that provides a unified and expressive way to query and manipulate data from various sources. LINQ allows developers to use a consistent syntax for querying data, regardless of whether the data is stored in databases, XML files, collections, or other data sources. Key components of LINQ include:

4.5.1 Query Expressions

LINQ introduces a declarative syntax that allows developers to write queries using familiar keywords such as from, where, select, group by, and order by. These queries resemble SQL statements and are used to filter, project, and sort data.

4.5.2 Standard Query Operators

LINQ provides a set of standard query operators, such as Where, Select, *OrderBy*, and *GroupBy*, which can be used to perform common operations on data collections. These operators are applicable to a wide range of data sources, promoting code reuse and consistency**.**

4.5.3 Deffered Execution

LINQ queries are lazily executed, meaning that the actual execution of the query is deferred until the results are explicitly requested. This allows for more efficient processing, especially when dealing with large datasets.

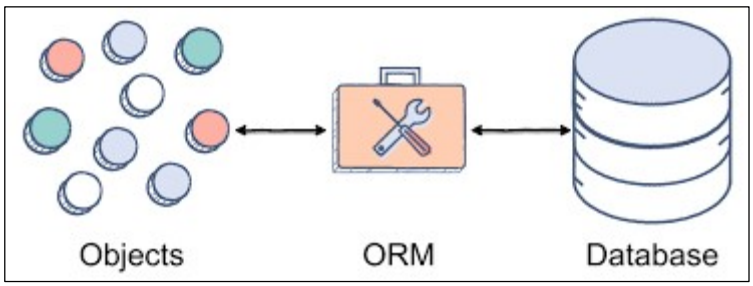
4.5.4 Lambda Expression

LINQ heavily utilizes lambda expressions, which are concise, inline functions.

4.6 ORM

ORM (Object-Relational Mapping) acts as a mediator between object-oriented models and relational database models, facilitating seamless interaction between the two paradigms. ORM enables developers to manipulate database objects using familiar object-oriented constructs from programming languages, reducing the need for direct SQL queries. ORM frameworks, such as *OrmLite*, abstract the complexities involved in database operations, providing APIs for tasks like establishing connections, executing queries, and managing transactions. Metadata serves as a crucial component in ORM, providing the necessary information to map database tables to corresponding objects in the programming language.

OrmLite, also known as Object-Relational Mapping Lite, is a lightweight ORM framework designed specifically for .NET environments.



4.3 Working of ORM

StackService.OrmLite is a specific implementation or variant of *ORMlite,* tailored to work within the Stack-Service ecosystem. *StackService.OrmLite* simplifies database operations within the StackService platform, allowing developers to interact with databases using C# objects. *StackService.OrmLite* likely provides additional features or optimizations specific to the Stack-Service environment, enhancing performance or compatibility within the ecosystem.

4.7 SECURITY & CRYPTOGRAPHY

To counter the risk of data breaches during transmission, encrypting all sensitive data before network transfer is crucial. This involves implementing an interceptor on the server-side to handle decryption of incoming requests and encryption of outgoing responses. A custom middleware facilitates this process, encrypting outgoing response data and decrypting incoming request data. By selectively applying encryption to specific API routes, control over data security measures is enhanced. Password and confidential information about the user is stored in database in the encrypted format for the security reason. AES algorithm and Triple DES algorithm are best choice for the password while RSA algorithm is utilized to implement digital signature verification as it consists private key and public key.

4.8 DYNAMIC TYPE

Dynamic data type enables dynamic typing, allowing for late binding and type determination at runtime. It supports interoperability with dynamic languages and scenarios where types are not known until runtime. Dynamic variables lack compile-time type checking, providing flexibility but with the potential for runtime errors.

CHAPTER - 5

.NET CORE WEB API TRAINING

5.1 ASP .NET CORE REQUEST PROCESSING PIPELINE

In .Net Core Framework, request and response follows specific path or way for processing that is known as request processing pipeline. Organised structured implementation of request processing pipeline can build robust and user – centric secured and performance-oriented Web API. Execution order of request processing pipeline is demonstrated here.

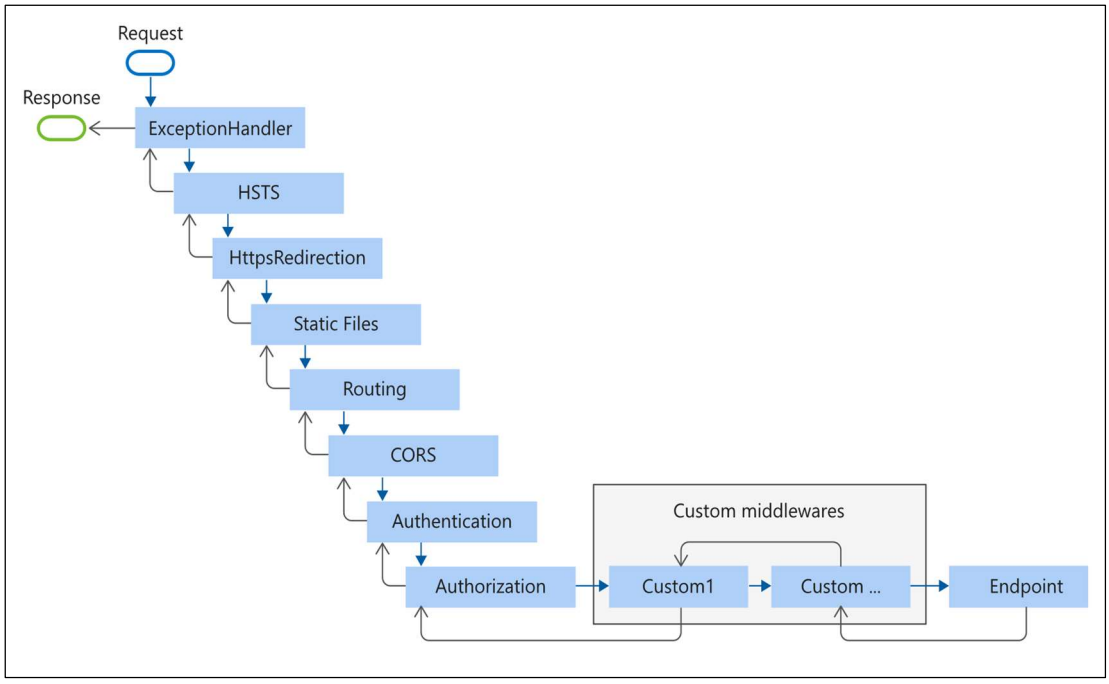


Fig 5.1 Request Processing Pipeline

5.1.1 Middleware

Middleware is software which handles requests through pipelines, it processes request & response one by one. When middleware doesn't pass the response to next middleware, it is calling short-circuiting of the middleware. There are three types of middleware.

5.1.1.1 Terminal Middleware

Middleware which deals with client.

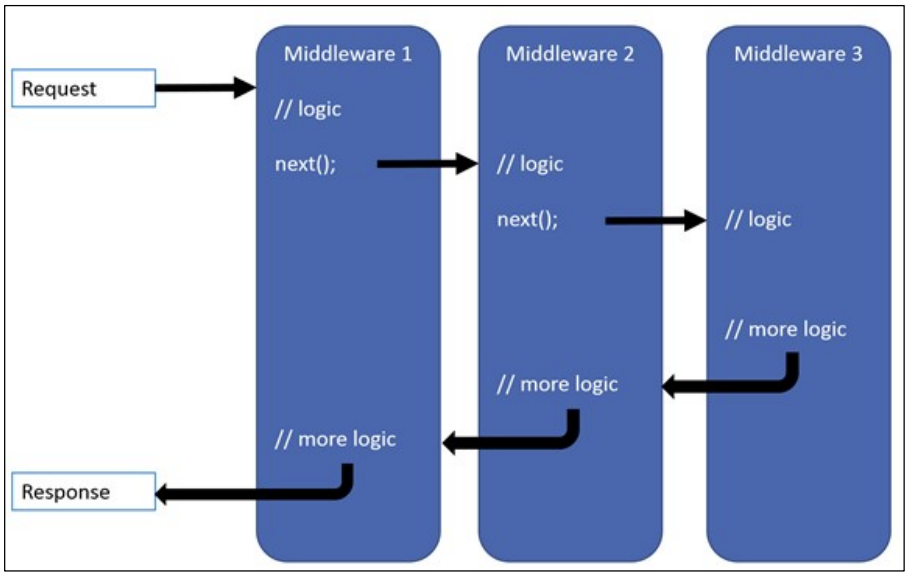


Fig 5.2 Middleware Execution

5.1.1.2 Non-Terminal Middleware

Which sends the request to next middleware.

5.1.1.3 Custom Middleware

Developer Specific Middleware.

5.1.2 Routing

Routing is way of matching request URL with source like controller's action etc. There are two types of routing which are explained below:

5.1.2.1 Conventional Routing

It is used for global routing purposes. It gives less flexibility/control compare to attribute based routing. It is safe to use conventional routing since it creates less or no confusion or conflicts.

5.1.2.2 Attribute-Based Routing

It is used for particular action method of controller. It gives more control over action methods compare to conventional routing. We can also use routing constraints to restrict data comes from URL.

5.1.3 Filters

Filter is an implementation of request pipeline which can we add at any particular method or controller or we can declare that globally. Filter can be synchronous or asynchronous. Async filter can be used for logging, caching, authorization with additional services like data processing and recourse cleanup. .Net Core also provides built-in filters which are capable to implement:

- Authorization
- Response Caching
- Short – Circuiting the request
- Exception

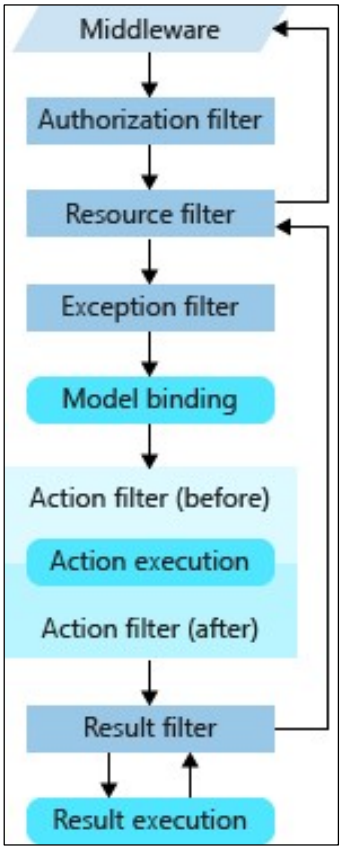


Fig 5.3 Filter Execution

5.1.4 Action Method

5.1.4.1 As Json Data

```
return Ok( new
{
    key1 = "value1",
    key2 = "value2"
});
```

5.1.4.2 Redirect to another page or link

```
return Redirect("https://www.google.com/");
```

5.1.4.3 Return Content

```
return Content("This is plain text", "text/plain");
```

5.1.4.4 Return File

return File(fileBytes,

"application/pdf",

"C_Sharp_Advance_Doc.pdf");

5.1.4.5 Return Status Code

return Ok();

return NotFound();

return Unauthorized();

5.2 DEPENDENCY INJECTION

5.2.1 Built-in IoC Container

Dependency Injection container is a container which contains interfaces and implementation of that interfaces. In ASP .NET DI container is already provided which is also known as built in IoC container. It means we can now take refence of the interfaces in application & that interface will take reference of implementation which is mapped in DI container. Built in IoC container is used for providing support of dependency injection in application to smooth the process. Dependency injection also adds future scope for updating within application without major changes.

5.2.2 Understanding Service Lifetime

5.2.2.1 Singleton Service Lifetime

Service will be registered only once throughout the application or server life, then it utilizes reference of that instance.

5.2.2.2 Scoped Service Lifetime

Reference implementation of service will be initialized each time. Instance of the service reference will not be saved.

5.2.2.3 Transient Service Lifetime

Reference implementation of service will be saved and that can be utilized furthermore.

5.2.3 Extension Method for Registration

We can register any service with its implementation by using extension method. That will register services by taking reference of *IServiceCollection*. Code of Extension Method is demonstrated below:

```
public static IServiceCollection AddMyService
                                    (this IServiceCollection services)
{
    services.AddSingleton<IStudentService, StudentService>();
    return services;
}
```

5.2.4 Constructor Injection

Constructor injection is a phynonyma in which dependency injection is provided through constructor. When instance of class is created it also gives support of required components or dependencies. Here is an example of service is demonstrated where almost all types of constructor injection are implemented.

5.3 LOGGING

5.3.1 Logging API

Microsoft.Extensions.Logging includes the necessary classes and interfaces for logging. The most important are the *ILogger*, *ILoggerFactory*, *ILoggerProvider* interfaces and the *LoggerFactory* class. The following figure shows the relationship between logging classes.

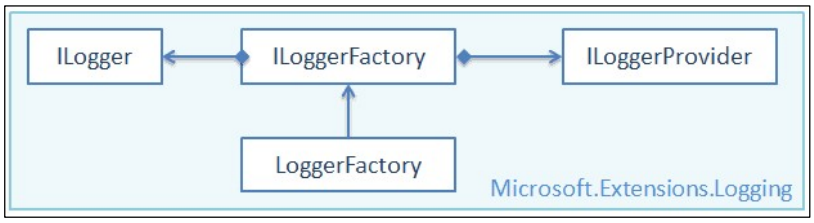


Fig 5.4 Logging API

5.3.2 Logging Providers

There are several logging providers which provides customizable implementation with database support & file integration. *nLog* is a powerful and robust logging provider which can be installed as nuget package.

CHAPTER - 6
SYSTEM DESIGNING

6.1 LOW LEVEL DESIGNING

6.1.1 SOLID Principle

SOLID principles are fundamental guidelines in object-oriented design, emphasizing principles like single responsibility and dependency inversion. Applying SOLID principles in ASP.NET Web API development ensures that each component has a clear purpose, promoting modularity and maintainability.

The Single Responsibility Principle advocates for each class or module to have only one responsibility, enhancing code clarity and testability.

The Open/Closed Principle suggests that software entities should be open for extension but closed for modification, encouraging developers to use abstraction and inheritance to accommodate change.

The Liskov Substitution Principle emphasizes the importance of maintaining substitutability of derived classes for their base classes, ensuring consistency and predictability in the system.

The Interface Segregation Principle advises against creating large, monolithic interfaces, instead favouring smaller, more focused interfaces to prevent clients from depending on methods they do not use.

The Dependency Inversion Principle encourages designing components to depend on abstractions rather than concrete implementations, promoting loose coupling and facilitating easier maintenance. Adhering to SOLID principles in ASP.NET Web API development fosters modularity and maintainability, leading to robust and scalable APIs that are easier to maintain and extend over time.

6.1.2 DbFactory Design Pattern

The dbFactory design pattern abstracts database connection management, offering a centralized factory for creating database-specific objects. By encapsulating database creation logic, the dbFactory pattern simplifies database interactions in ASP.NET Web API development, facilitating seamless integration with different database providers.

6.1.3 Singleton Design Pattern

The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance.

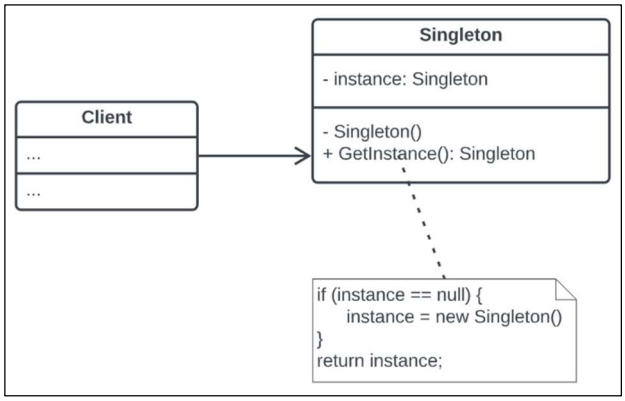


Fig 6.1 Singleton Design Pattern

In ASP.NET Web API, the Singleton pattern can be applied to manage resources that need to be shared across multiple requests, such as caching objects or configuration settings. Care must be taken when implementing Singletons in a multi-threaded environment to avoid race conditions and ensure thread safety.

6.1.4 Flyweight Design Pattern

The Flyweight pattern minimizes memory usage by sharing common state between multiple objects. In the context of ASP.NET Web API, the Flyweight pattern can be applied to optimize memory usage when dealing with large datasets or frequently accessed resources. For example, in a Web API serving static content, the Flyweight pattern can be used to cache and reuse common resources like images or CSS files across multiple requests.

6.1.5 Circuit Breaker Design Pattern

The Circuit Breaker pattern provides a mechanism to handle faults in distributed systems by temporarily blocking requests to a failing component.

In ASP .NET Web API development, the Circuit Breaker pattern can be used to improve resilience and prevent cascading failures when calling external services or APIs. By monitoring the health of external dependencies and selectively opening or closing the circuit based on predefined thresholds, the Circuit Breaker pattern helps maintain system stability and performance.

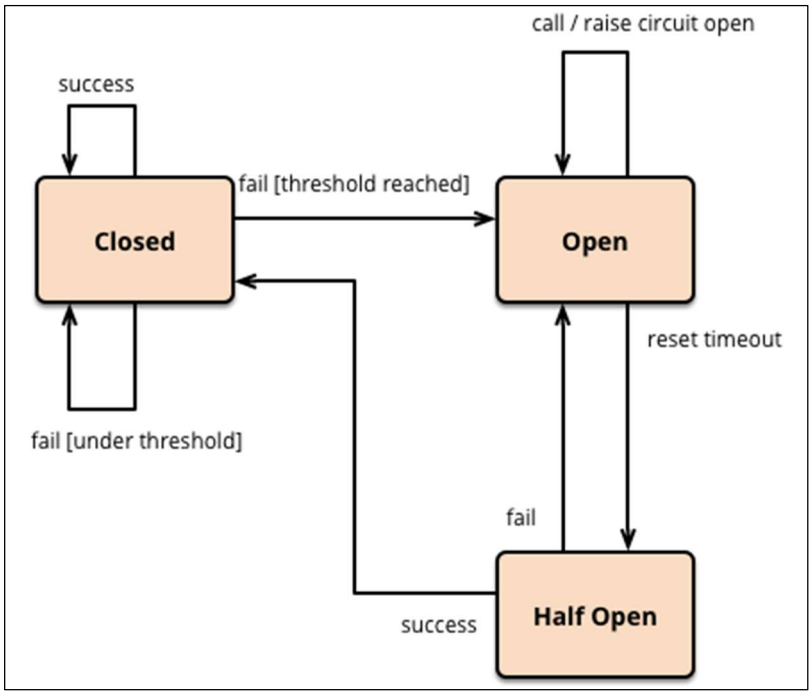


Fig 6.2 Circuit Breaker Design Pattern

6.2 HIGH LEVEL DESIGNING

High-level designing of a web API within a microservice architecture involves decomposing the system into small, independently deployable services. Utilizing asynchronous messaging patterns enhances scalability and responsiveness. Incorporating Redis as a cache layer improves performance by storing frequently accessed data in memory. Security measures such as authentication and encryption ensure data protection. Monitoring and observability tools enable real-time monitoring of service health. Containerization simplifies deployment and scaling. Overall, these practices facilitate the development of robust and scalable APIs that meet modern application demands.

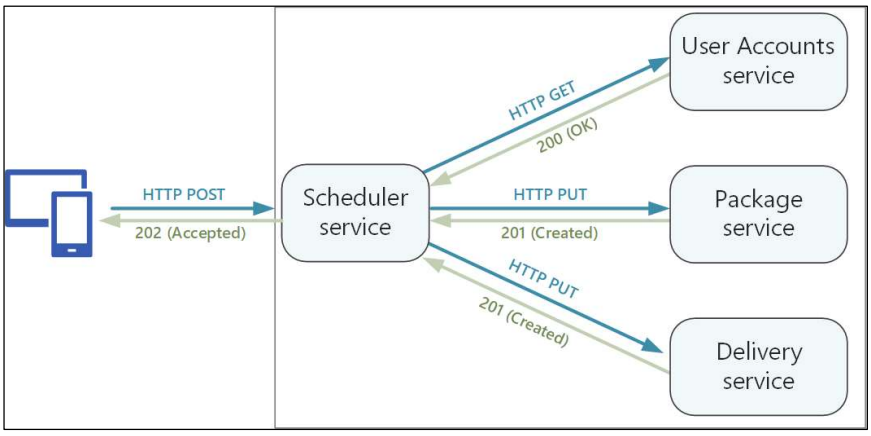


Fig 6.3 Microservice HLD

Example of HLD design with services is referenced above which is prepared by Microsoft for understanding purpose which insists to use dependency injection to follow LLD rules properly.

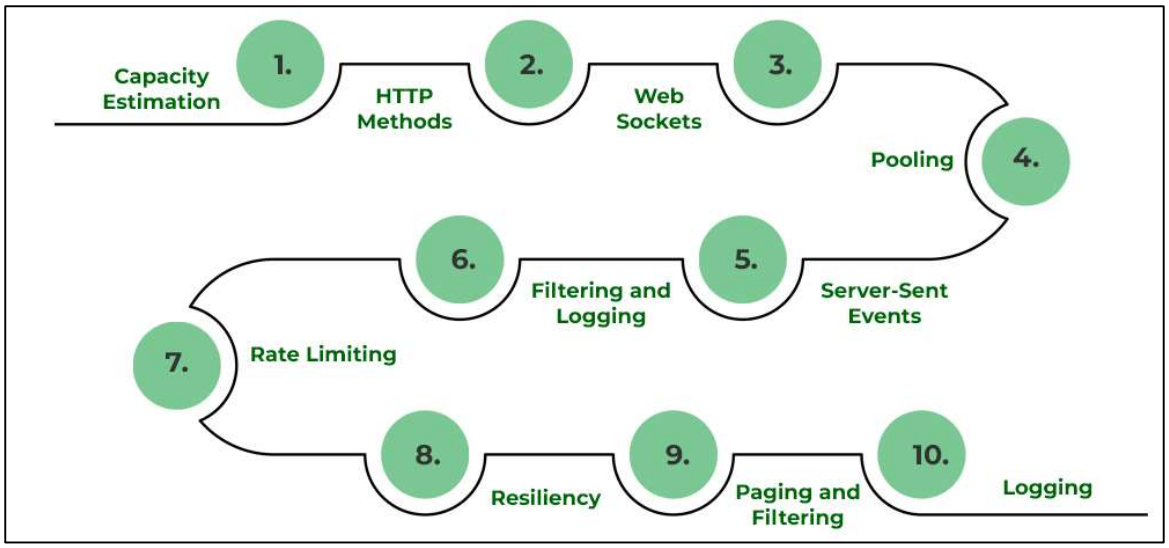


Fig 6.4 Checkpoints of HLD

Correct implementation & HLD designing of any application leads to the most user friendly and efficient application.

CHAPTER - 7

IMPLEMENTATION

Reference of Implementation. Projects & Documents : *https://internship-sem8.vercel.app/*

7.1 URL SHORTENER

7.1.1 Overview

URL Shortener API consists of two versions, V1 and V2. Version 1 generates 6-character short codes for original URLs, supports role-based authorization for users, allows public access for redirection, provides analytics with user-based authorization, and enables administrators to delete short URLs. Version 2 extends functionality by requiring a user ID in the request, generating 8-character short codes, introducing a super admin role for viewing analytics of all URLs, and maintaining role-based authorization for users and administrators. Both versions redirect short URLs publicly and implement secure role-based authorization for various user roles.

Github Repository:

https://github.com/RKITSoftware/Yash-Lathiya/tree/main/Demo/API%20Basic%20Demo/3_Web_Developement/3_Web_API/WebAPI/UrlShortner

Postman Documentation:

https://documenter.getpostman.com/view/30728898/2sA2r6Wirb

7.1.2 Technical Aspects

- .Net Framework
- Action Methods
- Response with Appropriate Status Code
- Security
- Versioning
- Role Based Authorization
- Token Based Authentication
- CORS implemented
- Delegates
- Validations
- Exception Filter
- Response Cache Filter
- Algorithm designing for generating unique id’s

7.1.3 Highlights

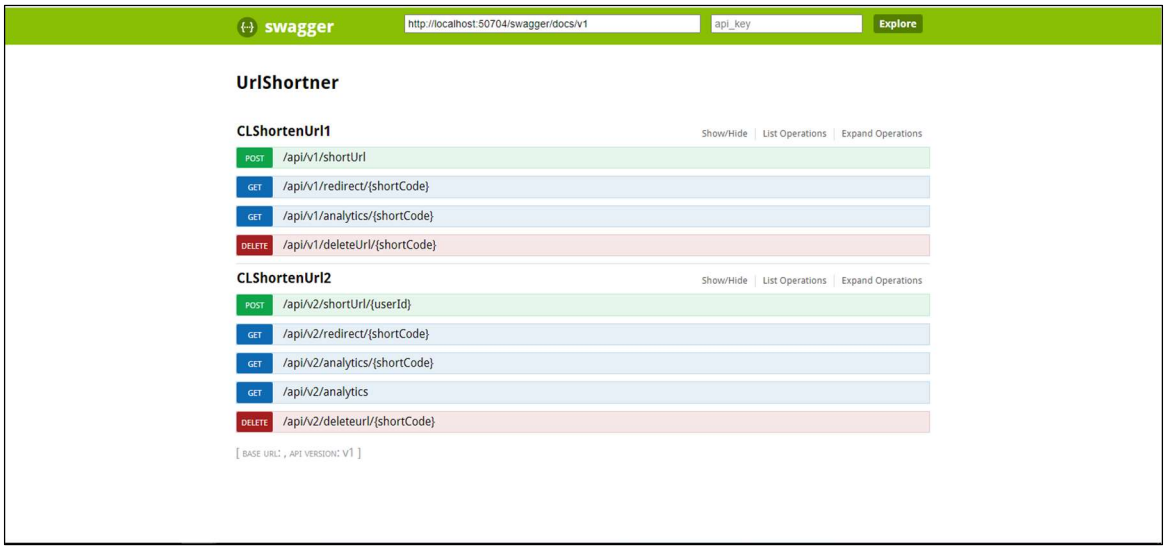


Fig 7.1 Highlights of URL Shortener

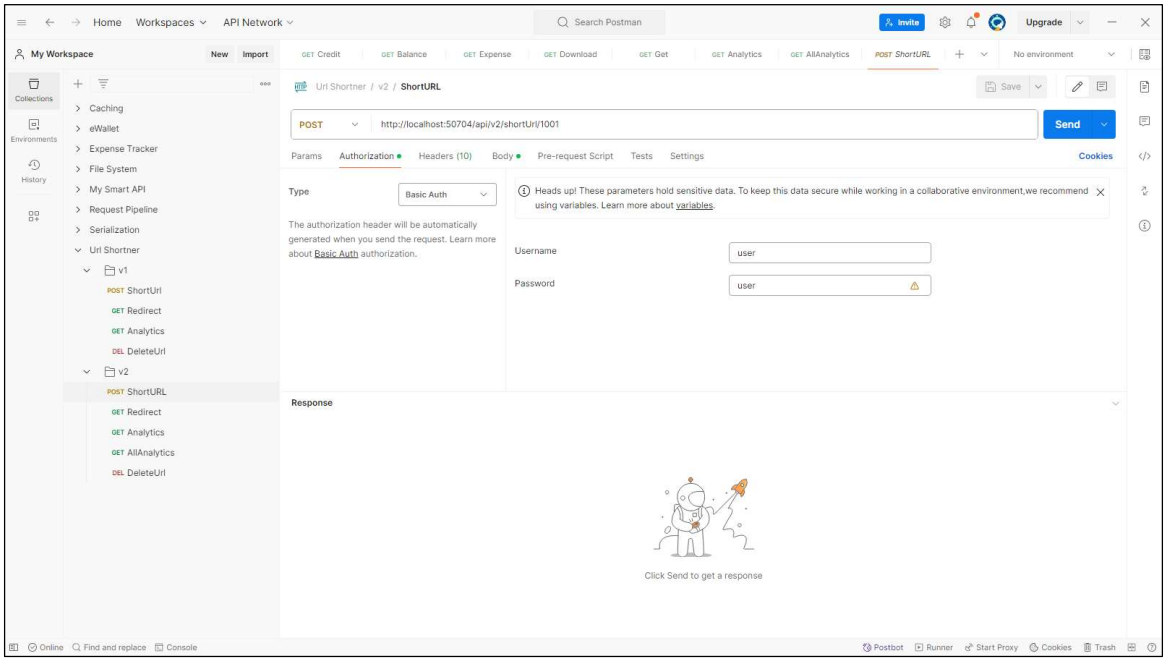


Fig 7.2 Highlights of URL Shortener

7.2 EXPENSE TRACKER

7.2.1 Overview

User can add expense & credit by crating profile This API also provides functionality to download report with encrypted security which will be stored in db. It is connected with *MySql* database to provides scalable solution. It has implemented by maintaining all industry level standard and workflow utilized by RKIT Software Pvt. Ltd.

Github Repository:

https://github.com/RKITSoftware/Yash-Lathiya/tree/main/Demo/API%20Basic%20Demo/5_C%23_Advance/Final%20Demo/ExpenseTracker/ExpenseTracker

Postman Documentation:

https://documenter.getpostman.com/view/30728898/2sA2rDy1wT

7.2.2 Technical Aspects

- .Net Framework
- Token Based Authentication
- Generics
- File System Integration
- Data Serialization
- Cryptography for Security
- Database Integration
- Supports ORM
- Email Integration
- Implements LLD Concepts

7.2.3 Highlights

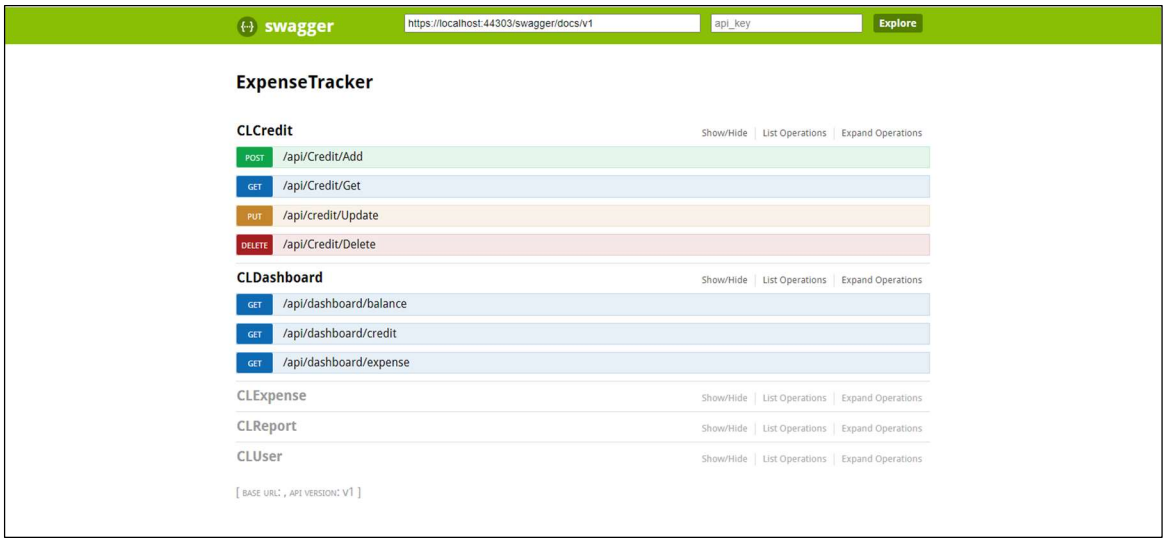


Fig 7.3 Highlights of Expense Tracker

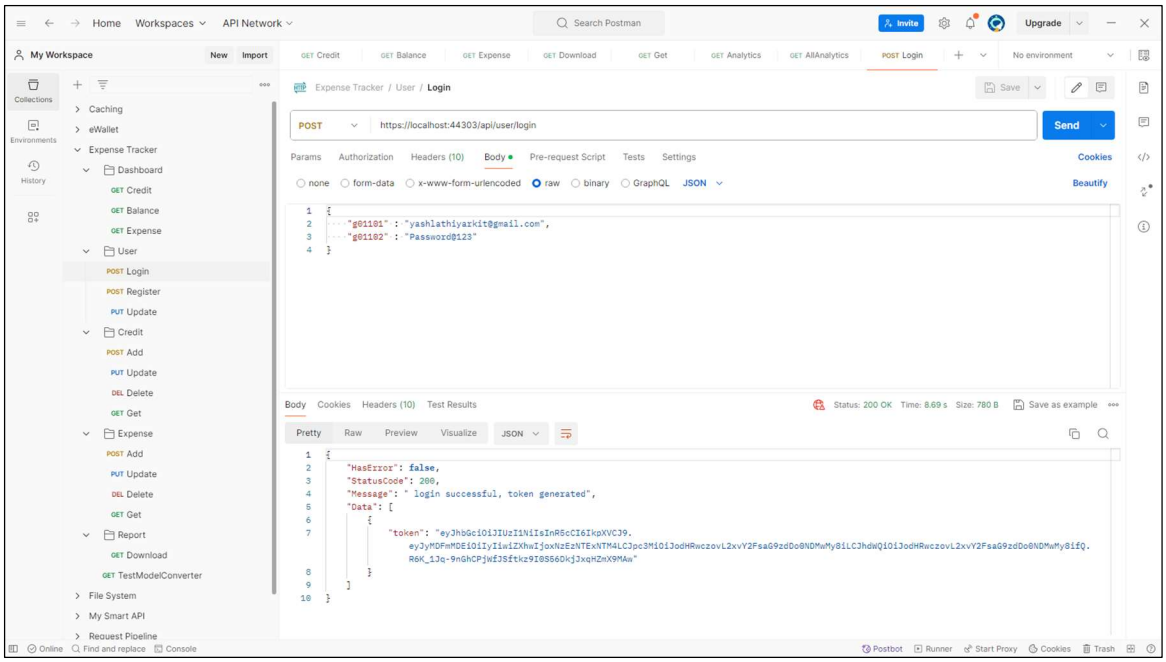


Fig 7.4 Highlights of Expense Tracker

CHAPTER - 8
CONCLUSION

8.1 OVERALL ANALYSIS OF INTERNSHIP

During my internship, I have learnt significant concepts of Web API designing along with database integration, system designing and C# concepts in .Net Framework and .Net Core Web API as industry level implementation.

8.2 DATES OF CONTINUOUS EVALUTION

8.2.1 Review 1 of Internship

17th February 2024

8.2.2 Review 2 of Internship

16th March 2024

8.2.3 Review 3 of Internship

20th April 2024

8.3 SUMMARY OF INTERNSHIP WORK

In January 2024, I embarked on my internship journey at RKIT, located in Rajkot, ON. Immersed in a vibrant work culture, I found myself surrounded by supportive colleagues who exemplified traits such as kindness, helpfulness, customer-centricity, empathy, and honesty. This environment provided the perfect backdrop for me to showcase my skills as a reliable team member and a committed learner. Throughout my internship, I embraced the opportunity to adapt and thrive in various technological domains, reflecting the versatility that is often required in corporate settings.

8.4 EXPERIENCE

I gained valuable experience in a professional setting and deepened my understanding of the skills mentioned earlier. As a result of my performance, I am offered a full-time position of Full Stack Developer with the company.

REFERENCES

Basic C#	*https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/*
Advance C#	*https://learn.microsoft.com/en-us/dotnet/csharp/advanced-topics/*
Web API	*https://www.tutorialsteacher.com/webapi*
.Net Framework	*https://learn.microsoft.com/en-us/dotnet/framework/*
Asp .Net Core	*https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio*
System Design	*https://www.youtube.com/@sudocode*
Authorization	*https://www.youtube.com/@SatyarthProgrammingHub*
System Design	*https://www.youtube.com/@KeertiPurswani*
Blogs	*https://medium.com/*
OrmLite	*https://docs.servicestack.net/why-servicestack*
Geeks for Geeks	*https://www.geeksforgeeks.org/system-design-tutorial/?ref=shm*
C# Corner	*https://www.c-sharpcorner.com/csharp-tutorials*
StackOverFlow	*https://stackoverflow.com/*
TutorialsPoint	*https://www.tutorialspoint.com/csharp/index.htm*
Redis	*https://redis.io/docs/latest/develop/data-types/*
NLog	*https://nlog-project.org/documentation/*
NLog	*https://github.com/NLog/NLog/wiki/Getting-started-with-ASP.NET-Core-6*
Postman	*https://learning.postman.com/docs/introduction/overview/*
Swagger	*https://swagger.io/solutions/api-development/*

APPENDIX I

WEEKLY DIARY

APPENDIX II

FEEDBACK FORM BY INDUSTRY EXPERT