Software Engineering Internship at Cefalo Bangladesh Ltd.

by

Md Tasin Tahmid
23141032

An internship report submitted to the Department of Computer Science and Engineering in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
February 2026

Declaration

It is hereby declared that

1. The thesis submitted is my own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Md Tasin Tahmid

23141032

Approval

The internship report titled "Software Engineering Internship at Cefalo Bangladesh Ltd." submitted by

1. Md Tasin Tahmid (23141032)

Of Fall, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science.

Examining Committee:

Supervisor:
(Member)

Dr. Muhammad Iqbal Hossain

Associate Professor
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi

Chairperson
Department of Computer Science and Engineering
Brac University

Abstract

This internship report provides an overview of the experiences, challenges, and learnings during a software engineering internship at Cefalo Bangladesh Ltd. During the six-month internship, which ran from September 4, 2023, to March 4, 2024, I actively participated in creating a web-based blog application. This was a kind of demonstration project for learning the fundamentals of web programming. More specifically, learning front-end and back-end development, unit testing, and database integration while adhering to industry standards and best practices were the main goals.
The report begins with an introduction I explained about the internship and the objective of the internship. The objective explains the whole plan for the six months time period and the learning progess accordingly. Then it continues with explaining the only project done in the time period. First of all, I explained what kind of project did I work on, what are the functionalities of that project and what tools and technologies I used to build that project. Then the report continues with explaining the implementation of the Blog Application. The project Implementation section is boken down further into three sections; they are the design of the Database, The design of fucntionalities of API and the API Endpoint testing using postman. Each section have their own diagrams or figure to explain their content properly. At last, the report concludes with a summary of key takeaways, highlighting the value gained from the internship experience and its contribution to the intern's academic and professional development.

Keywords: Software Engineering; Web Development; Frontend; Backend; Database; Unit Testing; Deployment; Content Negotiation; Representation; Endpoints; Software Development Lifecyle

Acknowledgement

Firstly, all praise to the Great Allah for whom my thesis have been completed without any major interruption.
Secondly, to my supervisor Dr. Muhammad Iqbal Hossain sir for his kind support and advice in our work. He helped me whenever I needed help.
And finally to our parents without their throughout sup-port it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

List of Figures

Chapter 1

Introduction

The field of software engineering is a dynamic and rapidly evolving domain, where theoretical knowledge meets the practical challenges of real-world application. This internship report encapsulates my experiences and learnings during the 6 months internship at Cefalo Bangladesh Ltd, where I had the opportunity to immerse myself in the difficulties of software development within a professional setting.

1.1 About Internship

As a student of BRAC University, one must complete a thesis, project or internship under the course CSE400 to get a bachelor's degree in Computer Science or Computer Science and Engineering. Those who want to complete their internship may choose any field of their interest as long as it is related to their degree. To become eligible to take an internship for CSE400 one must have a minimum of 72 university credits completed and get a six-month(minimum) internship with a reputable Software Comopany approved by BRAC University. In my case, it was Cefalo Bangladesh Ltd, a Norway-based software development and IT service provider company with its development center in Dhaka, Bangladesh and headquarters in Oslo, Norway. I was lucky to get a chance to complete my academic internship as a Trainee Software Engineer, where my job was to work on a demo project. I have worked on a Blog posting webstite, particularly both in Frontend and Banckend. It was a six months Internship and although I did not get the chance to work with client, the learnings were enough to start a career in web development in JavaScript stack.

1.2 Internship Objectives

I particularly chose an internship in software engineering to bridge the gap between theoretical knowledge acquired in academic settings and its practical application in a corporate environment. To be more specific, to broaden my skill set in the field of software engineering, understand the intricacies of the software development life cycle(planning, requirement analysis, design, coding, testing, deployment and maintenance), and immerse myself in a collaborative industry environment. Mainly, the following objectives guided the internship journey:

1. Full-Stack Proficiency: To develop proficiency in full-stack web development

by actively participating in the creation of web applications using Express.js for server-side development, React.js for client-side interfaces, and MySQL for database management.

Progress and Achievements: Successfully contributed to the development of a Blog Application by actively engaging in both front-end and back-end aspects, gaining hands-on experience in building complete web applications.

2. Database Integration and Management: Gaining expertise in integrating and managing MySQL databases within web applications.

Progress and Achievements: Successfully implemented and maintained MySQL databases into the Blog App where all the information of blogs and users are coming from the database, ensuring seamless integration with Express.js and React.js applications.

3. RESTful API Development: To acquire skills in designing and implementing RESTful APIs using Express.js to facilitate communication between the frontend and back-end components of web applications.

Progress and Achievements: Actively contributed to the creation of RESTful 2APIs for the Blog Application, following best practices in API design and ensuring smooth communication between the React.js client and Express.js server.

4. Front-End Development with React: To enhance proficiency in React.js for building dynamic and responsive user interfaces.

Progress and Achievements: Developed interactive and user-friendly interfaces for the Blog Application using React.js, incorporating state management, component-based architecture, and responsive design principles.

5. Deploying the Full-Stack Application: To deploy successfully the full-stack web application to a hosting environment, gaining practical experience in the deployment process.

Progress and Achievements: After completing building the full stack blog application, my task will be to deploy the application in any online hosting environment.

The objectives outlined above have served as a roadmap for my professional development, ensuring a comprehensive understanding of various aspects of software engineering. Most of the parts mentioned in objective are already covered since I am in my last phase of the Internship. Only objective which has not been covered yet is Deploying the full stack application which I will be covering very soon.

Chapter 2

Project Overview

In my six months Internship period I worked on a demo project which particularly is a Blog Posting Web Application. It is a single page application built using JavaScript. Since it was a demo project and the purpose of building the Blog posting website is to learn full stack Web Development, my main focus was on functional requirements only. Therefore, there was only one non-functional requirement mentioned below, along with some of the functional requirements:

2.1 Functional Requirements

1. User Registration and Authentication:
    - Users should be able to register for an account on the website.
    - Registered users should be able to log in securely using their credentials.
2. Create and Manage Blog Posts:
    - Authenticated users should be able to create, edit, and delete blog posts.
    - Users should be able to categorize their posts and add tags for easy navigation.
3. User Profile Management:
    - Users should have access to a profile dashboard where they can manage their account settings, password, and profile information.
4. Viewing Others Profile:
    - Users should be able to see other users profile.
    - Users should be able to see all the blogs of any particular user.

2.2 Non-functional Requirements

1. Responsive Design:

    - The website should be accessible and usable on various devices, including desktops, laptops, tablets, and smartphones.
    - The layout and content should adapt to different screen sizes and resolutions.

2.3 Used Technologies and Tools

2.3.1 Technologies used in Backend:

1. Node.Js:
   Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser. It allows developers to build scalable and high-performance network applications using JavaScript, which was traditionally confined to client-side scripting within web browsers. Node.js uses an event-driven, non-blocking I/O model, which makes it well-suited for handling concurrent operations and I/O-bound tasks. It employs asynchronous programming techniques, such as callbacks, Promises, and async/await, to handle I/O operations efficiently without blocking the execution thread.

2. Express:
   Express is a popular and minimalist web application framework for Node.js. It provides a robust set of features for building web applications and APIs quickly and efficiently. Its key features are Middlewares, Routing, Templating Engines, Error Handling, Static File Serving etc.

3. Seqelize:
   Sequelize is a promise-based Node.js ORM (Object-Relational Mapping) library for relational databases, primarily focused on SQL databases such as PostgreSQL, MySQL, MariaDB, SQLite, and Microsoft SQL Server. It provides a powerful set of features for managing database interactions, defining models, performing CRUD (Create, Read, Update, Delete) operations, and establishing associations between different data models.

4. JSON Web Tokens:
   JSON Web Tokens is a library based on Node which provides a flexible and secure mechanism for implementing authentication and authorization in modern web applications. JWTs are stateless, meaning the server does not need to store session state for authenticated users. This improves scalability and simplifies server-side architecture. It consists of three parts: a header, a payload containing claims, and a signature. After authentication, a server generates

a JWT, which the client stores and sends with subsequent requests to access protected resources.

5. JEST:
   Jest is a popular JavaScript testing framework developed by Facebook. It's widely used for testing JavaScript code, including React and Node.js applications. Jest provides an intuitive and feature-rich environment for writing and executing tests, including support for mocking, assertion libraries, code coverage, and parallel test execution. Its simplicity and powerful features make it a preferred choice for testing JavaScript applications across the development community.

2.3.2 Technologies used in Frontend:

1. React Js:
   React is an open-source JavaScript library for building user interfaces, primarily for web applications. Developed by Facebook, React allows developers to create interactive and reusable UI components that efficiently update in response to data changes. It uses a declarative approach to define UI components, making it easier to manage and maintain complex UIs. React's component-based architecture, virtual DOM, and state management capabilities make it well-suited for building modern, high-performance web applications. Additionally, React has a vibrant ecosystem with tools like React Router for routing, Redux for state management, and React Native for building mobile applications, making it a popular choice for frontend development.

2. Redux Tool-kit:
   Redux Took-kit a predictable state container for JavaScript applications, primarily used with frameworks like React, but it can be used with any other library or framework. It provides a centralized store to manage the application state and enables predictable state management by following a unidirectional data flow pattern. Major functionalities of redux toolkit are Store, Actions, Reducers, Dispatch, Subscriptions etc.

3. Jest:
   Like Backend I used Jest for unit testing. Jest provides simple and easy to use mock functions and mock component for React to write test cases for Unit testing.

4. Tailwind CSS:
   Tailwind CSS is a utility-first CSS framework that provides a set of pre-built utility classes to style HTML elements. Unlike traditional CSS frameworks that offer pre-designed components, Tailwind focuses on providing low-level utility classes that can be combined to create custom designs quickly and efficiently.

2.3.3 Tools used in Building the Blog Application:

1. Postman:
   Postman is a popular collaboration platform for API development that simplifies the process of building, testing, and documenting APIs. It offers a user-friendly interface and a wide range of features to streamline API development workflows. Postman allows developers to create and execute API requests using HTTP methods like GET, POST, PUT, DELETE, etc. Users can define request headers, query parameters, request body, and authentication credentials as needed. Postman supports various authentication methods, including Basic Auth, OAuth, and API keys.

2. Git and Github:
   Git is a distributed version control system (VCS) that allows developers to track changes to their codebase, collaborate with others, and manage multiple versions of their projects. It provides features such as branching, merging, and history tracking, making it easier to work on complex projects with multiple contributors.

   GitHub hosts Git repositories in the cloud, allowing developers to access their code from anywhere with an internet connection. Remote repositories on GitHub serve as a centralized location for collaborating on projects with team members.It provides features for collaboration, such as pull requests, issues, and project boards. Pull requests allow developers to propose changes, review code, and provide feedback before merging changes into the main codebase. Issues can be used to track bugs, feature requests, and other tasks, while project boards provide a visual overview of project progress.

Chapter 3

Project Implementation

3.1 Database Design

Since the Blog Application is a very simple application, I used only two entities for the database design. They are Blogs and Users.

3.1.1 Table-1: Blogs

The Blogs entity carries all the characteristics of a particular blog posted in the web application. The Blogs entity, representing a posted blog, has six fields and they are:

id CHAR(36),
title TEXT,
blogContent TEXT,
userId CHAR(36),
createdAt DATETIME and
updatedAt DATETIME.

Among all the fields, the field id CHAR(36) is used as a primary key for the table Blogs and the field userId CHAR(36) is used as a foreign key from the table Users. In other word, userId CHAR(36) field refers to the field id CHAR(36) in Users table which is the primary key of table Users.

3.1.2 Table-2: Users

The Users entity represents all the registered users from the Blog Application, who are authenticated to parform their operations. Apart from the primary key id CHAR(36) there are five more fields in the users table and they are:

username VARCHAR(255),
email VARCHAR(255),
password VARCHAR(255),
createdAt DATETIME and
updatedAt DATETIME.

3.1.3 Relationship: One to Many

From the diagram below we can see that these tables, Blogs and Users, are in a relationship using the Foreign Key userId CHAR(36) from the table Blogs and they are in One to Many relationship. That means Blog must have One and Only One User and any User can have multiple Blogs.

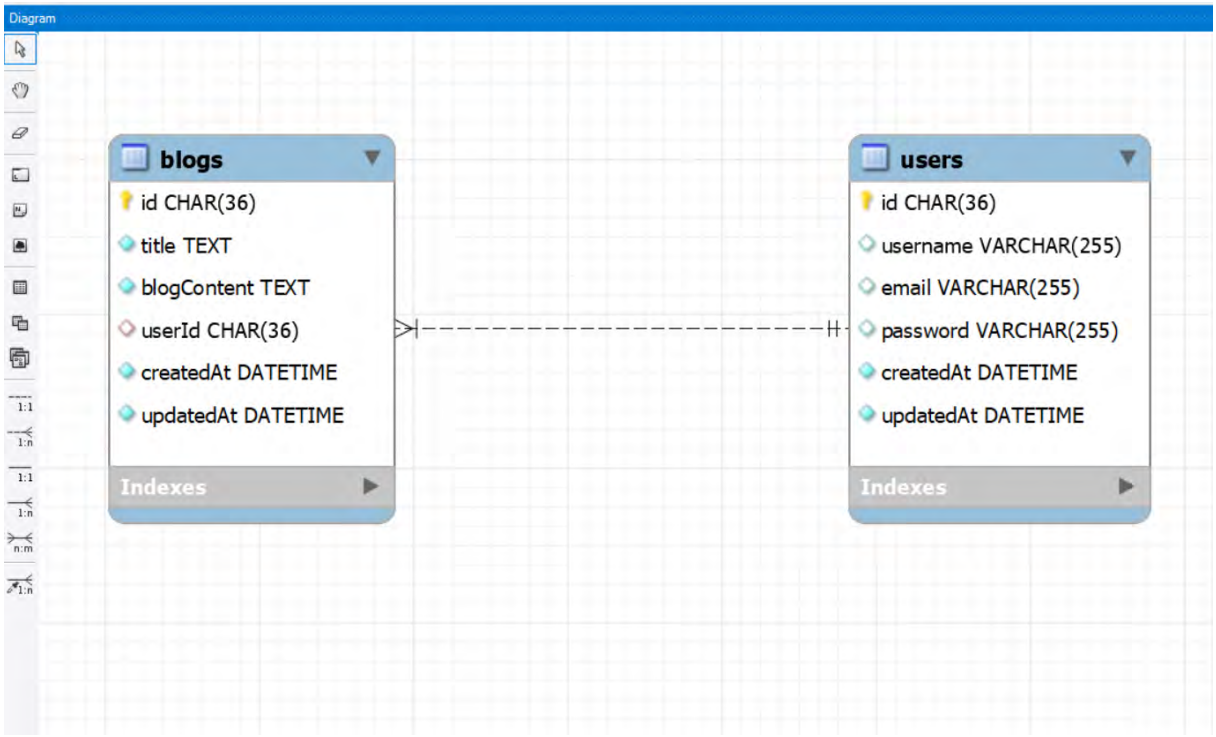

Figure 3.1: ER Diagram of Entity Users and Blogs.

3.2 API Design

In backend, I have used REST API to process all types of requests from Client (Browser, Postman or any types of Frontend) and send a response accordingly. This REST API analysizes the request from any client and retrieve data from Database before sending any response. Before diving deep into my API design, lets understand what an API is and why it is called REST API.

API: API or Application Programming Interface is kind of like a bridge or a middleman which establishes communication between two or more different Application, built maintaining different languages or different protocols.

REST API: Any API which follows the principles of REST Architecture is considered as REST API or RESTful Services.

In this project we also followed the rules and constraints of REST Architecture. Let's understand how I implemented the constraints of REST Architecture in the Blog Application.

1. Resources: In a REST API, everything is a resource. A resource can be anything that you can access via a URL. For example, a user profile, a cart item, or an image. In this project, we have two resources, Blogs and Users. We will retrieve or mutate any resources by using proper http method and resource URIs.

2. HTTP Methods: REST APIs use standard HTTP methods to perform actions on resources. We used the most common http methods in this project and they are:

   - GET: To retrieve any user Data or Blog data from an endpoint.
   - POST: To create a new Blogs or to register new user.
   - PUT: To update Blog details.
   - Patch: To update user password, partial update.
   - DELETE: To delete any Blog or any User account.

3. URLs: Each resource is identified here by a unique URL (Uniform Resource Locator). For example:

   - api/v1/users: Represents the collection of all User resources.
   - api/v1/blogs/123: Represents a specific Blog with ID 123.

   But depending upon the operation we want to perform on any particular resource, same URL can serve multiple purposes. In this case, HTTP methods will be different. Each combination of URL and HTTP methods can be considered as different Endpoints. Here, in this project we have 11 different endpoints. They are:

   (a) api/v1/users/register, with POST method: To register new Users.

   (b) api/v1/users/login, with POST method: To authenticate existing Users.

(c) api/v1/users/12, with PATCH method: To update User details partially, password update.

(d) api/v1/users/12, with DELETE method: To delete User account.

(e) api/v1/users/12, with GET method: To get all User Data of User with id 12.

(f) api/v1/users, with GET method: To get all Users Data as a List.

(g) api/v1/blogs, with GET method: To get all Blogs Data as a List.

(h) api/v1/blogs, with POST method: To create new Blog and post it.

(i) api/v1/blogs/12, with GET method: To get Blog details of Blog with id 12.

(j) api/v1/blogs/12, with PUT method: To update Blog details of Blog with id 12.

(k) api/v1/blogs/12, with DELETE method: To delete a Blog with id 12.

Here, c, d and e URLs are same but the methods are different(PATCH, DELETE and GET). Thus, they will serve as different endpoints; c updates partially, d deletes a user account, e gets a user information. Similarly, i, j and k have same URLs but their methods are different(GET, PUT and Delete). Therefore, they will serve as different endpoints; i gets blog details of blog id 12, j updates information of blog with id 12 and k deletes the blog with id 12.

4. Response Format: When a client makes a request to this REST API, the API first figures out that in which format the users want their data. This process is known as Content Negotiation. The Content Negotiation happens by using HTTP heads. There is an HTTP header called Accept which can contain different values like “application/json”, “application/xml”, “text/html” or “text/plain”. Depending upon the value of the header Accept the API will select the value format. Each value format is called Representation in the language of HTTP. In this project, my REST API can serve only four representations and they are JSON, XML, HTML and PLAIN TEXT.

By following all the above constraints, I have built the REST Api Which is Stateless. Stateless means this API dose not save the request information for any consecutive requests. That means each requests is considered as separated request to this API. That does not make this REST API session less. Because we can still preserve any session data by using HTTP Cookies or Tokens. In this project I have used Tokens, to be more specific, JWT authentication token. The problem with HTTP Cookies is that cookies are only supported by the browser. On the other hand, tokens can be accessible by any types of client agent. That is why I used Token based authentication in the Blog Application.

3.3 API Testing

In this project I tested all my API endpoints using Postman. Postman acts like a client which can send request to any API. As mentioned earlier, there are two types of resources in this application, Blogs and Users. Lets see some of the responses from our Blog API from Postman.

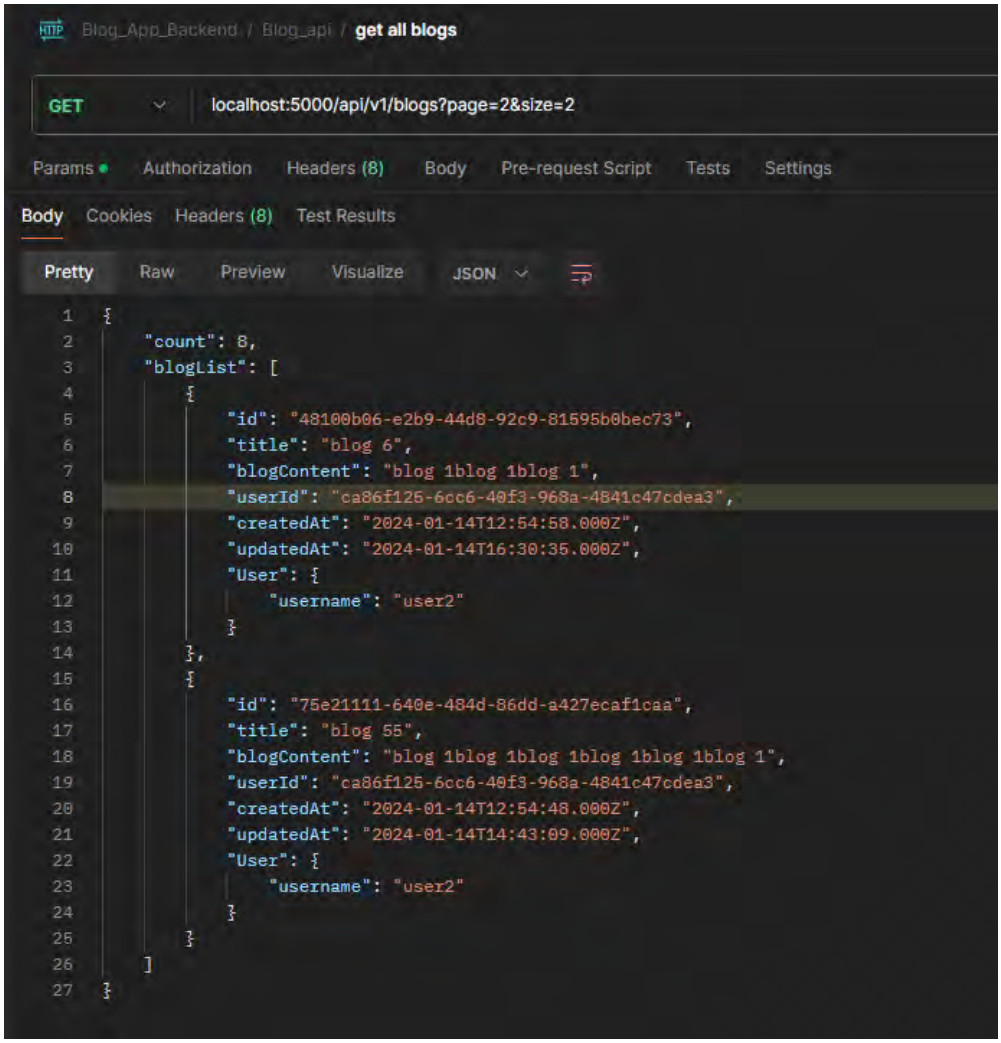

Figure 3.2: Get All Blogs.

In the Figure 3.2: Get All Blogs, we can see the URL is "localhost:5000/api/v1/blogs?page=2&size=2", where page=2&size=2 is used for pagination. Here, in this URL we requested using get method and as expected we got the list of Blogs of page number 2 which contains 2 blogs in it. Similarly, if we hit the same URL but without pagination part and using POST method. Then we would get the newly created Blog details.

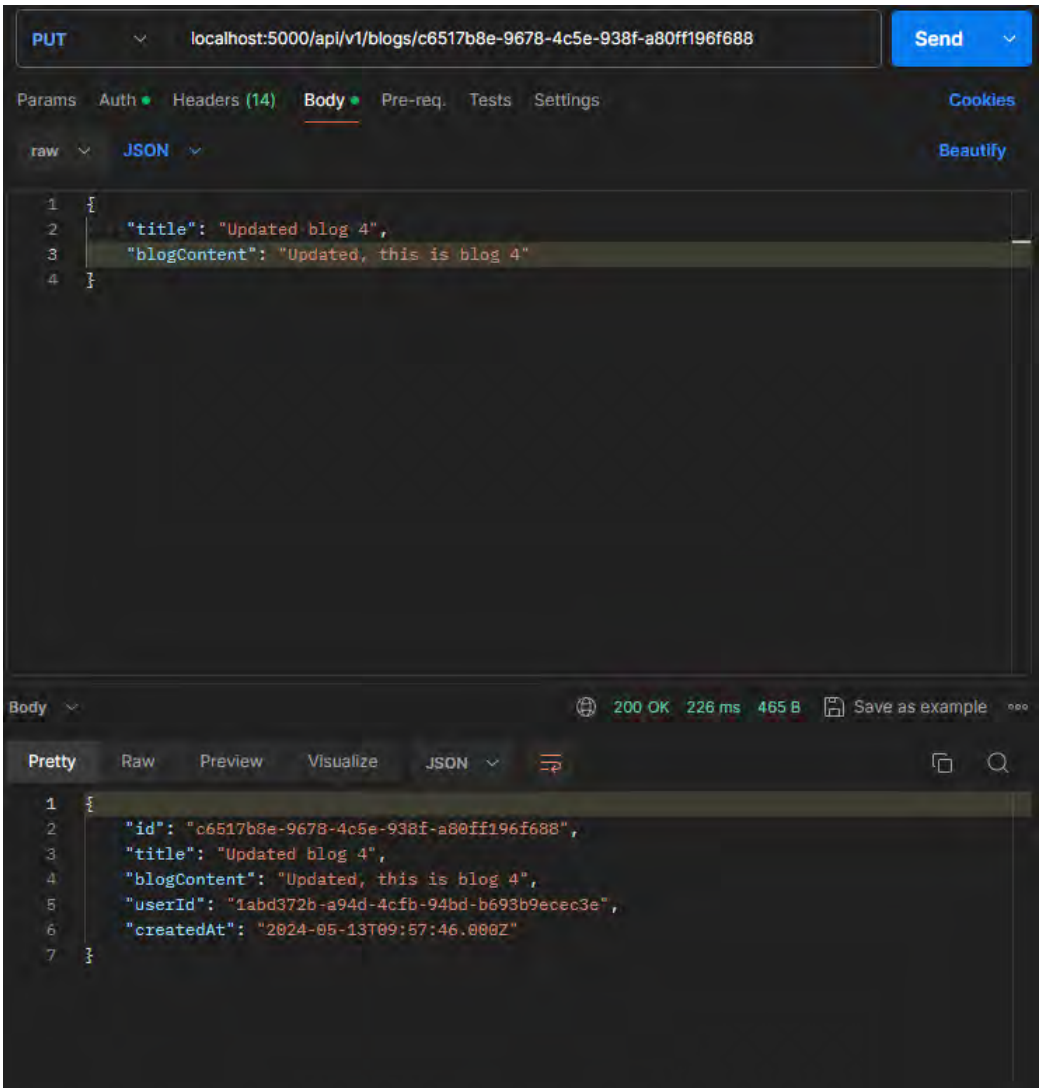

Figure 3.3: Update Blog By Id.

In the Figure 3.3: Update Blog By Id, we can see that we are hitting the url "localhost:5000/api/v1/blogs/c6517b8e-9678-4c5e-938f-a80ff196f688", by using the PUT method along with a request body where the updated body details for the Blog id c6517b8e-9678-4c5e-938f-a80ff196f688 is present. And in response, we are getting the Updated Blog details containing all the updated values.

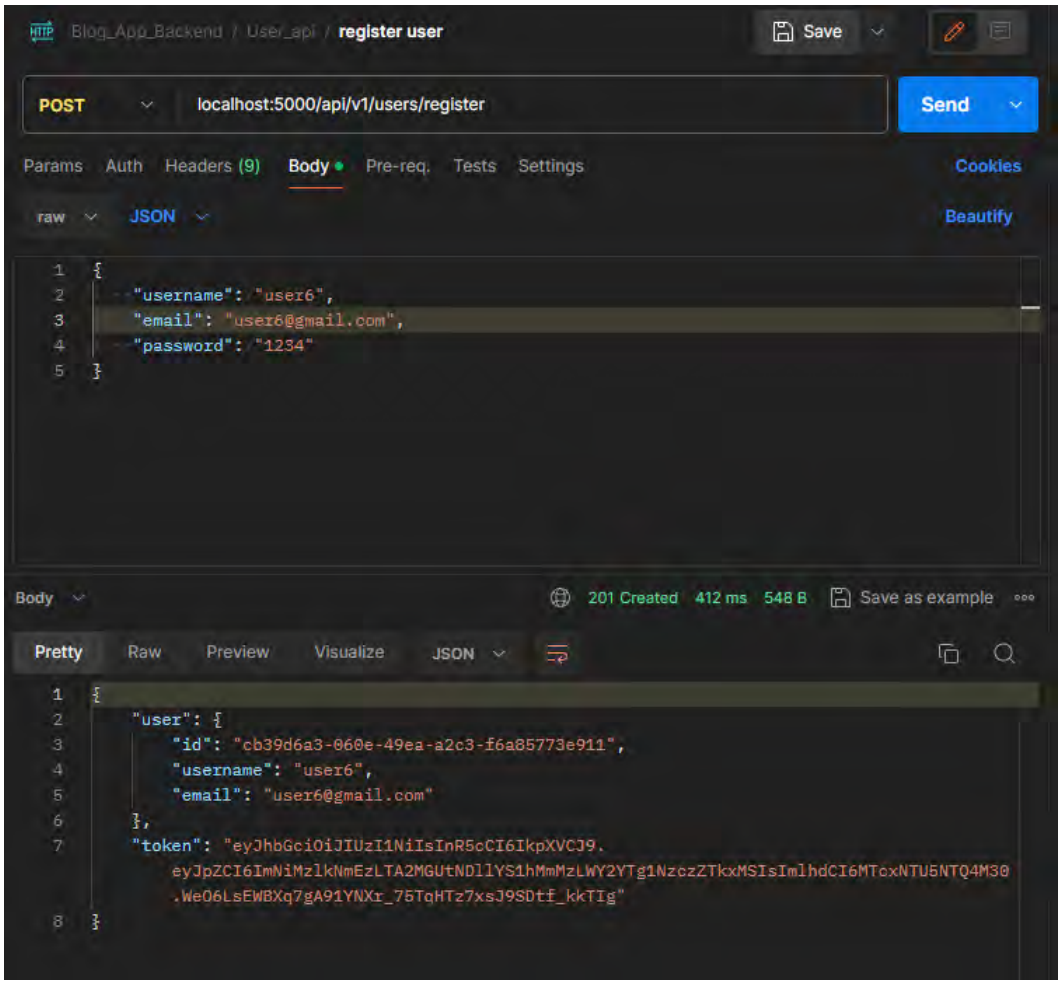

Figure 3.4: Register New User.

In the figure 3.4: Register New User, we can see, we are hitting the URL "localhost:5000/api/v1/users/register" with a request body which contains details of a new user using the HTTP method POST. That means we are requesting our API to create a new user for us using the given username, email and password. In response, we received a JSON object containing a User object and an authentication Token. This token will help us in future to continue our sessions. Thus, our API becomes stateless, but not session less.

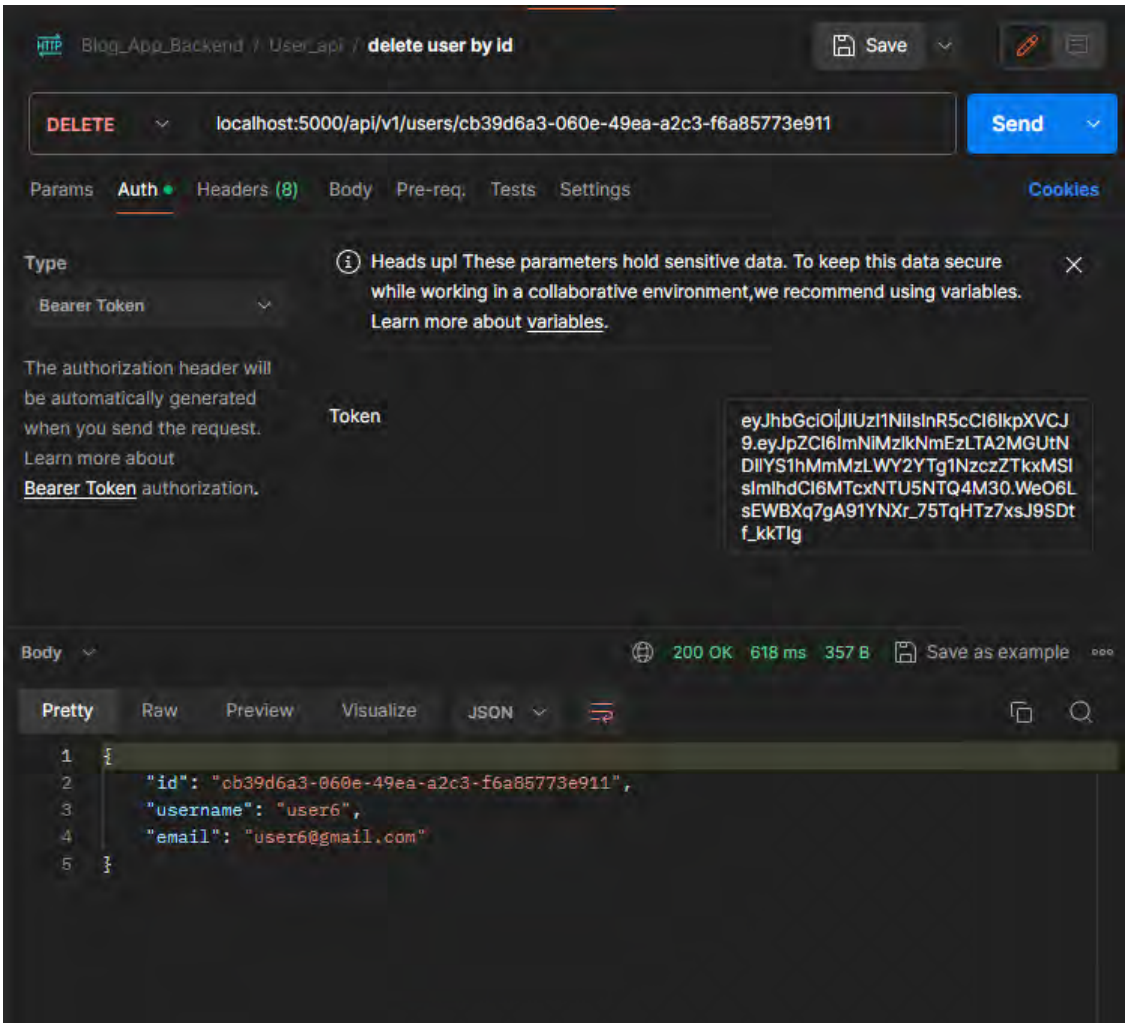

Figure 3.5: Delete User By Id.

In this figure 3.5 Delete User By Id, we are sending a Bearer Token with Authorization header. We got this token during Registration. In fact, we would have also got this similar token, if we logged in to the system instead of registering. Here, the URL is "localhost:5000/api/v1/users/cb39d6a3-060e-49ea-a2c3-f6a85773e911" and used method is DELETE. That means if token sent by the client through HTTP header is correct, the system will delete the user account with id cb39d6a3-060e-49ea-a2c3-f6a85773e911.

Chapter 4

System Design of Point Exchange Application

4.1 Project Description

This is a "Loyalty and Reward" Shopify app that lets customers earn reward points when they place orders and later redeem those points for discounts at checkout. Merchants can easily set up and manage rules that define how many points customers earn and how those points convert into discounts. The goal is to help stores increase repeat purchases by offering a simple and flexible loyalty system.

4.1.1 Shopify Platform Introduction

Shopify is a E-commerce platform that helps people set up and run an online store without worrying about the technical details. It gives merchants the tools to manage products, payments, orders, and customers in one place, so they can focus more on selling and growing their business. In a word, Shopify provides marchants ready made Ecommerce websites that are easily manageable.

4.1.2 Why I Chose a Shopify App Project

Before starting any project for my defense I wanted to choose one particular project where I can apply knowledge earned from my internship and also, which is closely connected to my current work and long-term learning goals. The reasons are as follows:

- It follows standard full-stack development practices I learnt in my internship(a React, Node project).
- It aligns closely with my current role and experience as a Shopify developer.
- It gives me practical, real-world experience working with GraphQL.
- It provides hands-on exposure to webhooks for handling real-time events.
- It also allows me to strengthen my skills by building something practical within the Shopify ecosystem.

4.1.3 Functional Requirements

1. Reward Rules:
    - Merchant should be able to Create Reward Rules.
    - Merchant should be able to Update Reward Rules.
    - Merchant should be able to Delete Reward Rules.
2. Points to discount conversion:
    - Customers should be able to see their points.
    - Customers should be able to convert their points into discounts.
3. Webhook Automation:
    - Application should be triggered when customers create orders using Orders/Fulfilled webhook to update customer points.
    - Application should be triggered when customers use discount using orders/create webhook to remove discount.

4.1.4 Non-functional Requirements

1. Responsive Design:
    - The application should be accessible and usable on various devices, including desktops, laptops, tablets, and smartphones.
    - The layout and content should adapt to different screen sizes and resolutions.

4.2 API Design

In the Point Exchange project, I have used three different types of communication technologies; they are REST API, GraphQL API and Webhooks. They serve different purposes in different parts of the application. For example:

- REST API is used to make communication between Point Exchange app's fronted and backend.
- GraphQL API is used to make communication between Point Exchange app's backend and Shopify server.
- Weebhook is used to trigger my application backend on some particular events happend on Shopify Store.

Depending on the action we want to perform on a specific resource, a single URL can have multiple uses. In such cases, the HTTP method distinguishes the operation. Each unique combination of a URL and an HTTP method can be treated as a separate endpoint. In this project, we have a total of 11 different endpoints and they are:

1. api/point-offset, with GET method: To fetch the value of Point Offset.
2. api/point-offset, with PUT method: To update the value of Point Offset.
3. api/discount-rules, with GET method: To get all discount rules.
4. api/discount-rules, with POST method: To create new discount rule.
5. api/discount-rules/:id, with PUT method: To update a particular discount-rule by id.
6. api/discount-rules/:id, with DELETE method: To delete a particular discount-rule by id.
7. api/discount, with POST method: To create discount code for any particular customer.
8. api/customers/:customerId/points, with GET method: To get total points of a customer by customer ID.
9. api/customers/:customerId/discounts, with GET method: To get all discounts of a customer by customer ID.
10. webhooks/app/orders_fulfilled: To trigger my app backend to update customer's total points.
11. webhooks/app/orders_create: To trigger my app backend to update customer's total discounts.

4.3 Uses Tools and Technologies

1. Frontend: Built with React and styled using Shopify Polaris to create a responsive, user-friendly interface that feels native to the Shopify admin.
2. Backend: Developed using Remix, handling server-side logic, routing, authentication, and communication with Shopify APIs.
3. Database: Uses PostgreSQL for data storage, with Prisma ORM to manage database models and queries in a clean and maintainable way.

4.4 Application Images

In the Figure 4.1: App Admin UI, we can see the Admin UI of the application point exchange. Here a store owner or merchant can create, update and delete the Discount rules. Additionally, a merchant can also update the point offset which helps to calculate total points for any particular customer on each order fulfillment (Total Points = Total Cost x Point Offset).

All of these data is coming from the application backend where those data is stored on the database. Any changes from this application admin UI make direct changes on the database through REST API.

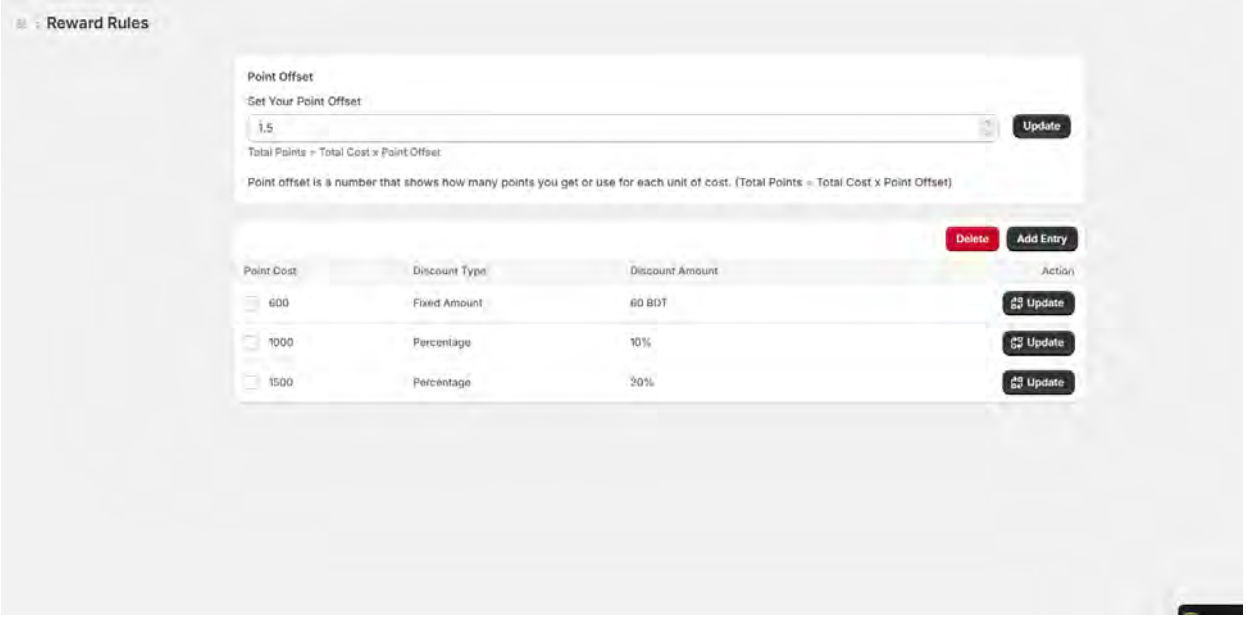

Figure 4.1: App Admin UI.

The Figure 4.1: Customer Profile shows the points and discounts of any particular customer on their profile. From here, customers can see their total collected points, the discounts they have achieved and the Discount Rules created by the merchant or the store owner. Additionally, customers can convert their points into discount according the the discount rules.

To create discount code, the customer has to choose an option from the Reward List and press the Redeem button. If the customers have available point bonus they will get discount code added to their profile and they can use theme as coupon code while making any order in future. When a customer presses on the Redeem button a request is sent on the application backend using REST API. After that, the application backend verifies the request and make another GraphQL request to Shopify server to create a new discount code for that particular Customer.

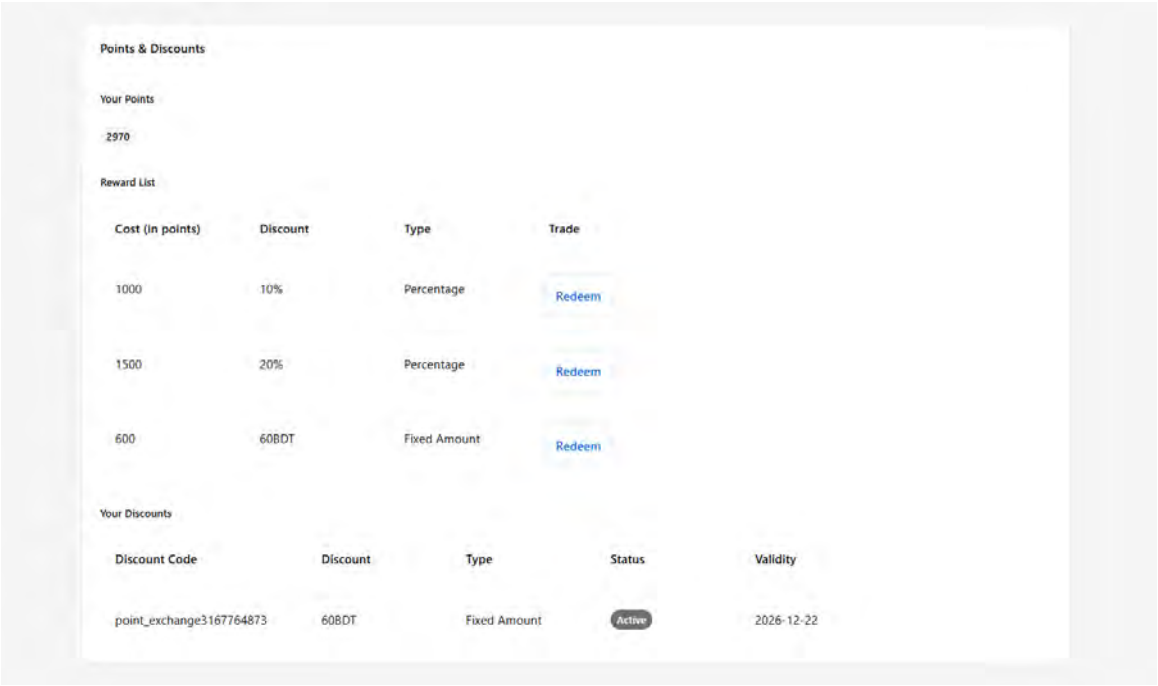

Figure 4.2: Customer Profile.

Chapter 5

Conclusion

In conclusion, the software engineering internship at Cefalo Bangladesh Ltd. has been a transformative and enriching experience. Over the course of the internship, I have had the privilege to work on a meaningful project from the scratch, learning from talented professionals, and gaining invaluable insights into the intricacies of software development in a corporate setting.

One of the key takeaways from this internship is the real-world application of the theoretical knowledge acquired during my academic journey. The hands-on experience in coding, testing, and deployment has deepened my understanding of the software development life cycle. Although my part did not cover all the steps of a software development life cycle, like requirement analysis and the system design, the rest of the parts were enough to motivate me to start a career in software development. The exposure to industry-standard tools, such as React Js, Express Js, Sequelize as ORM, Jest as a testing tool, Postman as api testing, gitHub as version control system etc, has broadened my skill set and equipped me with tools that are crucial for success in the field.

At last, I extend my gratitude to the entire trainee team at Cefalo Bangladesh Ltd and our mentors for their support, mentorship and the opportunity to contribute to an impactful project. This internship has been a stepping stone in my professional journey, and I am excited about the continued growth and learning that lies ahead.

Chapter 6

References

1. David Gourley and Brian Totty, (2002). *HTTP: The Definitive Guide.*
2. Subbu Allamaraju, (2010). *RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity.*
3. Robert C. Martin (Uncle Bob), (2008). *Clean Code: A Handbook of Agile Software Craftsmanship.*
4. David Flanagan, (2020). *JavaScript: The Definitive Guide.*
5. Mario Casciaro, (2014). *Node.js Design Patterns.*