

A Comprehensive SaaS Platform for Restaurant Digital Autonomy

*Project report submitted
in partial fulfilment of the requirement for the degree of*

Bachelor of Engineering in Information Technology

By

Harshnate Kumar Prasad (001911001004)

Ayush Sharma (002011001125)

Aditya Kumar Mishra (002011001128)

Under the guidance of

Prof. Bhaskar Sardar



Department of Information Technology

Faculty of Engineering and Technology,

Jadavpur University,

Salt Lake Campus 2020-2024

Department of Information Technology
Faculty of Engineering and Technology
Jadavpur University

BONAFIDE CERTIFICATE

This is to certify that this project report entitled “ *A Comprehensive SaaS Platform for Restaurant Digital Autonomy* ” submitted to Department of Information Technology, Jadavpur University, Salt Lake Campus, Kolkata, is a bonafide record of work done by “Harshnate Kumar Prasad (Registration No: 149580 of 2019-2020), Ayush Sharma(Registration No: 153827 of 2020-2021), Aditya Kumar Mishra(Registration No: of 2020-2021)” under my supervision from “July 27, 2023” to “ May 21, 2024”

Countersigned By:

Acknowledgments

We would like to acknowledge Prof. Bhaskar Sardar for actively conducting research in this area and allowing us to collaborate with him. He was quick to provide us with all the directions on how to achieve and demonstrated to us the value of the project and its societal impact, guiding us toward our goals. He was always willing to assist us and allay our worries about any obstacles this endeavour may have presented.

We would also like to thank Prof. Palash Kundu for helping us with his constant inputs.



JADAVPUR UNIVERSITY
Dept. of Information Technology

Vision:

To provide young undergraduate and postgraduate students a responsive research environment and quality education in Information Technology to contribute in education, industry and society at large.

Mission:

- M1:** To nurture and strengthen professional potential of undergraduate and postgraduate students to the highest level.
- M2:** To provide international standard infrastructure for quality teaching, research and development in Information Technology.
- M3:** To undertake research challenges to explore new vistas of Information and Communication Technology for sustainable development in a value-based society.
- M4:** To encourage teamwork for undertaking real life and global challenges.

Program Educational Objectives (PEOs):

Graduates should be able to:

- PEO1:** Demonstrate recognizable expertise to solve problems in the analysis, design, implementation and evaluation of smart, distributed, and secured software systems.
- PEO2:** Engage in the engineering profession globally, by contributing to the ethical, competent, and creative practice of theoretical and practical aspects of intelligent data engineering.
- PEO3:** Exhibit sustained learning capability and ability to adapt to a constantly changing field of Information Technology through professional development, and self-learning.
- PEO4:** Show leadership qualities and initiative to ethically advance professional and organizational goals through collaboration with others of diverse interdisciplinary backgrounds.

Mission - PEO matrix:

Ms/ PEOs	M1	M2	M3	M4
PEO1	3	2	2	1
PEO2	2	3	2	1
PEO3	2	2	3	1
PEO4	1	2	2	3

(3 – Strong, 2 – Moderate and 1 – Weak)

Program Specific Outcomes (PSOs):

At the end of the program a student will be able to:

- PSO1:** Apply the principles of theoretical and practical aspects of ever evolving Programming & Software Technology in solving real life problems efficiently.
- PSO2:** Develop secured software systems considering constantly changing paradigms of communication and computation of web enabled distributed Systems.
- PSO3:** Design ethical solutions of global challenges by applying intelligent data science & management techniques on suitable modern computational platforms through interdisciplinary collaboration.

Abstract

In the rapidly evolving restaurant industry, maintaining profitability while leveraging online food delivery platforms like Zomato has become increasingly challenging for restaurant owners. These platforms impose substantial listing and advertising fees, and pressure restaurants to reduce their price margins to stay competitive, leading to diminished profits despite higher sales volumes. To address these issues, we have developed a cloud-based SaaS solution tailored for medium to high-rated restaurants struggling with the technical and financial burdens of existing online food delivery services.

Our platform provides an alternative that empowers restaurant owners to establish an online presence without being overcharged by intermediary platforms. Key features include a cloud-based infrastructure, a multi-application solution catering to end-users, shopkeepers, and delivery agents, a subscription-based pricing model, automatic updates, scalability through microservices architecture, and comprehensive tools for order management, inventory tracking, menu updates, and real-time delivery tracking. Additionally, the platform incorporates advanced authentication mechanisms, efficient database management, real-time communication via WebSockets, and machine learning for personalised user experiences and operational efficiency.

This report details the development and implementation of our SaaS solution, covering the underlying technologies, design considerations, and benefits for restaurant owners. By leveraging cloud technology and a subscription model, our platform offers a cost-effective, scalable, and user-friendly alternative to traditional food delivery platforms, enabling restaurants to maintain profitability and enhance customer reach without compromising on quality or service.

Keywords : *Restaurant industry, SaaS solution, cloud-based platform, online food delivery, profitability, microservices architecture, real-time communication, WebSockets, machine learning, order management, subscription model.*

Index

1. Introduction.....	1
2. Authentication Module.....	3
2.1 Differentiation Between Session-Based and Token-Based Authentication.....	3
2.1.1 Scalability.....	3
2.1.2 Performance.....	4
2.1.3 Security.....	4
2.1.4 Flexibility and Modern Standards.....	5
2.2 JSON Web Tokens (JWT).....	5
2.2.1 Structure of JWT.....	5
2.3 OAuth 2.0 Authorization Framework.....	6
2.3.1 Core Components.....	6
Client.....	7
Resource Server.....	7
Authorization Server.....	7
2.4 OTP-Based Email Validation.....	7
2.5 Work Flow Of Authorisation and Authentication.....	8
3. Microservices Architecture for Enhanced Scalability.....	9
3.1 Problem Statement.....	9
3.2 Objectives.....	9
3.3 Design Patterns Under Microservices Architecture.....	10

3.3.1 Use of API Gateway.....	10
3.3.2 Service Discovery.....	10
3.4 Inter-Microservice Communication.....	10
4. Use of Particular Databases with Their Reason.....	12
4.1 SQL vs. NoSQL.....	12
4.1.1. SQL Databases.....	12
4.1.2. NoSQL Databases.....	13
4.2 Classification Based on Storage Mechanism: In-Memory and Disk Storage.....	14
Use Case.....	15
4.3. Categorization Based on Importance of Data and Data Access Pattern.....	15
4.4. About Redis (Location Data).....	16
5. Websockets : Flow Of Data From User to Admin And Admin To User.....	17
6. Functionality.....	21
6.1 End User Solution.....	21
6.2 Shopkeeper Solution.....	23
6.3 Delivery Boy Application.....	24
6.4 Payment Gateway Integration.....	25
7. Real Time Location Tracking.....	27
7.1. Application of RTLS.....	28
7.2. Failure of Normal RLTS Architecture.....	28
7.3 Why use SSE over HTTP/Web Sockets.....	28
7.3. Apache Kafka.....	29
7.4. Why is Apache Kafka Fast ?.....	32
7.5. Using Redis As A Central Data Store.....	33
8. Machine Learning Module.....	35
8.1. Introduction to Machine Learning in Food Ordering System:.....	35
8.2. Problem Statement and Objective:.....	36
8.3. Data Collection and Preprocessing:.....	36
8.3.1. Introduction to Amazon Fine Food Reviews Dataset.....	37

8.3.2. Exploratory Analysis.....	37
8.3.3. Data Preprocessing.....	37
8.4 Feature Engineering:.....	38
8.4.1. Features Extracted from Raw Data:.....	38
8.4.2. Techniques Used for Feature Selection or Dimensionality Reduction:.....	38
8.5. Model Selection:.....	39
8.5.1. Architecture of the RNN Model.....	39
8.5.2. Justification for RNN Architecture.....	40
8.6. Model Training.....	40
8.7. Model Evaluation.....	41
8.8. Rating Generation.....	41
8.8.1. Implementation Details:.....	42
8.8.2. Advantages Over Weighted Average:.....	42
8.9. Future Work and Improvements:.....	43
9. Future Scope: Advancing the Project.....	44
9.1. Product Recommendation.....	44
9.2. Automated Delivery Boy Assignment:.....	45
9.3. Analytics Improvement.....	45
9.4. Latency Optimization.....	45
9.5. Multi-Tenant System.....	45
10. Conclusion.....	47

List of Figures

Fig. 1 : Work Flow of authentication and authorisation when already signed in	8
Fig. 2 : Work Flow of authentication and authorisation when during signed in	8
Fig. 3 : Service Discovery	10
Fig. 4 : ER diagram	12
Fig. 5 : Web Socket Architecture	18
Fig. 6 : Order Management	20
Fig. 7 : Payment Gateway Architecture	27
Fig. 8 : Kafka Architecture	32
Fig. 9 : Exploratory Analysis	38
Fig. 10 : Evaluation Curves.....	42
Fig. 11 : Precision, Recall, F1 Score	43
Fig. 12 : ROC curve.....	43
Fig. 13 : High Level Design of Application.....	48

References

Paper Number	Paper Citation
[1]	Tsai, W., Bai, X., & Huang, Y. (2014). Software-as-a-service (SaaS): perspectives and challenges. <i>Science China Information Sciences</i> , 57, 1-15.
[2]	Cusumano, M. (2010). Cloud computing and SaaS as new computing platforms. <i>Communications of the ACM</i> , 53(4), 27-29.
[3]	Rahmatulloh, A., Gunawan, R., & Nursuwars, F. M. S. (2019). Performance comparison of signed algorithms on JSON Web Token. <i>IOP Conference Series: Materials Science and Engineering</i> , 550, 012023. Published under license by IOP Publishing Ltd. The 1st Siliwangi International Conference on Innovation in Research 2018 (SICIR), 14 August 2018, Bandung, Indonesia.
[4]	Solapurkar, P. (2017). Building secure healthcare services using OAuth 2.0 and JSON web token in IoT cloud scenario. <i>IEEE</i> .
[5]	Newman, S. (2021). Building microservices. " O'Reilly Media, Inc."
[6]	Vonheiden, B. (2019). Design and Implementation of a Service Discovery System with Dynamic Routing. Bachelor thesis, Kiel University, Kiel, 47 pp.

[7]	Li, Y., & Manoharan, S. (2013, August). A performance comparison of SQL and NoSQL databases. IEEE.
[8]	Cattell, R. (2011). Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), 12-27. DOI
[9]	Freitas, R. F., & Wilcke, W. W. (2008). Storage-class memory: The next storage system technology. IBM Journal of Research and Development, 52(4/5), 439-447.
[10]	Lee, H. H., Park, I. K., & Hong, K. S. (2008, September). Design and implementation of a mobile devices-based real-time location tracking. In 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (pp. 178-183). IEEE.
[11]	Ślodziak, W., & Nowak, Z. (2016). Performance analysis of web systems based on xmlhttprequest, server-sent events and websocket. In Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology–ISAT 2015–Part II (pp. 71-83). Springer International Publishing.
[12]	Garg, N. (2013). Apache kafka (pp. 30-31). Birmingham, UK: Packt Publishing.
[15]	Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena, 404, 132306.

Documentation Number	Documentation Citation
[13]	Apache Kafka Documentation. (n.d.). Retrieved from https://kafka.apache.org/documentation/
[14]	Redis Documentation. (n.d.). Retrieved from https://redis.io/docs/latest/

1. Introduction

In the competitive landscape of the restaurant industry, maintaining profitability while leveraging online food delivery platforms like Zomato has become increasingly challenging for restaurant owners. These platforms not only charge substantial fees for listing and advertising but also pressure restaurants to reduce their price margins to remain competitive. Consequently, restaurant owners often experience diminished profits despite higher sales volumes due to these additional costs and the heightened competition.

Recognizing this issue, we have developed a SaaS [1] solution specifically tailored for medium to high-rated restaurants facing difficulties in navigating the technical and financial burdens of online food delivery services. Our cloud-based platform offers an alternative that empowers restaurant owners to establish an online presence without being overcharged by intermediary platforms.

Our application provides a comprehensive solution with the following features

- **Cloud-Based Platform:** The application is hosted on the cloud[2], ensuring it is accessible over the internet without requiring users to manage their own servers or infrastructure. This accessibility allows restaurant owners, delivery agents, and end-users to interact with the platform from any location with an internet connection.
- **Multi-Application Solution:** Our platform includes three distinct applications designed to cater to the specific needs of end-users, shopkeepers, and delivery agents. Each application has its own interface and functionalities, ensuring a tailored experience for every user group. Despite sharing the same underlying infrastructure, the data and interactions of each group are securely separated to ensure privacy and security.
- **Subscription Model:** Our SaaS application employs a subscription-based pricing model. Restaurants pay a monthly fee to be listed on the platform, with options for additional fees for premium features and delivery services. This model provides a predictable and manageable cost structure for restaurant owners.
- **Automatic Updates:** With continuous deployment capabilities, the application is automatically updated with new features and security patches, ensuring all users benefit from the latest enhancements without manual intervention.

- **Scalability:** Utilising a microservices architecture and cloud instances, our application is designed to scale efficiently with demand. This architecture ensures optimal performance and resource allocation as the number of users grows.
- **Comprehensive Tools and Services:** The platform includes robust tools for order management, inventory tracking, menu updates, and delivery management. End-users can browse menus, place orders, and track deliveries in real-time, providing a seamless experience for all parties involved.

By offering these features, our SaaS solution addresses the critical pain points faced by restaurant owners in the current market. It provides a cost-effective, scalable, and user-friendly alternative to traditional food delivery platforms, enabling restaurants to maintain their profitability and enhance their customer reach without compromising on quality or service.

This report will delve into the development and implementation of our SaaS solution, detailing the underlying technologies, design considerations, and the benefits it offers to restaurant owners. Through this project, we aim to demonstrate how leveraging cloud technology and a subscription model can revolutionise the online food delivery ecosystem for medium and high-rated restaurants

2. Authentication Module

Authentication is a critical component in application security, ensuring that only authorised users can access sensitive data and functionalities. It verifies the identity of users, providing a secure way to protect resources from unauthorised access. Effective authentication mechanisms are essential to maintain user trust, comply with regulatory requirements, and prevent security breaches.

As applications become more complex and distributed, the choice of authentication method becomes increasingly important. Different approaches, such as session-based and token-based authentication, offer distinct advantages and challenges. Understanding these methods, along with their scalability and security implications, helps in selecting the most appropriate solution for an application's specific needs. This discussion will explore these authentication techniques, their underlying mechanisms, and their suitability in various scenarios.

2.1 Differentiation Between Session-Based and Token-Based Authentication

In designing secure and scalable applications, choosing the right authentication method is crucial. Two prevalent methods are session-based and token-based authentication, each with distinct mechanisms and implications. This section highlights why token-based authentication is preferable over session-based for modern applications, particularly in terms of scalability, performance, and security[3].

2.1.1 Scalability

- **Horizontal Scaling:** Token-based authentication is inherently more suited to horizontal scaling. Tokens are stateless and self-contained, which means they do not require server-side storage. Any server instance can verify a token, allowing the application to scale out effortlessly across multiple servers. This decentralisation improves load distribution and fault tolerance.

- **Server Independence:** Unlike session-based authentication, which relies on a central session store, token-based authentication enables server independence. This eliminates the need for complex session synchronisation across servers, reducing latency and potential bottlenecks. Hence it isn't required to build a state machine which therefore reduces the moving parts.

2.1.2 Performance

- **Reduced Server Load:** Token-based authentication reduces the server's workload as it does not need to maintain session state or frequently access a session store. Each request carries all necessary authentication information within the token, leading to more efficient request handling and improved response times.
- **Statelessness:** The stateless nature of tokens allows for more streamlined and faster processing of authentication, as there is no overhead associated with managing and validating sessions.

2.1.3 Security

- **Enhanced Security Features:** Token-based systems offer robust security features. For instance, JSON Web Tokens (JWTs) can include claims that ensure the token's integrity and authenticity. Tokens can also be encrypted to add an extra layer of security.
- **Mitigating CSRF:** Tokens, especially when stored securely (e.g., HttpOnly and Secure cookies), can mitigate Cross-Site Request Forgery (CSRF) attacks more effectively than session cookies, which are more vulnerable to such attacks.
- **Token Expiration and Revocation:** Tokens can be configured with short lifespans, reducing the risk of misuse if compromised. Furthermore, token revocation lists can be implemented to invalidate tokens when necessary, enhancing security control.
- **MAC-SHA256,** a widely used cryptographic hash function for signing JWTs, combines HMAC with SHA-256 to ensure message integrity and authenticity using a secret key for both hashing and verification. Hash functions guarantee data integrity by invalidating the token if any part of the JWT is altered, while the one-way nature of these functions ensures immutability and confidentiality, making it computationally infeasible to reverse the hash to its original input. Verifying the signature with the

secret or public key confirms that the token was issued by a trusted source, providing robust authentication.

2.1.4 Flexibility and Modern Standards

- **Cross-Domain and API Integration:** Tokens are versatile for use across different domains and with various APIs, making them ideal for modern, distributed applications. This flexibility is crucial for Single Page Applications (SPAs) and microservices architectures, where different services need to verify the same user identity.
- **OAuth 2.0 Adoption:** Token-based authentication, particularly through OAuth 2.0, provides a robust framework for managing authentication and authorization. OAuth 2.0's standardised approach enhances interoperability and security across different platforms and services.

In summary, token-based authentication is chosen for its superior scalability, performance, and security features. Its ability to seamlessly support horizontal scaling, reduce server load, and provide robust security measures makes it a better fit for modern applications compared to session-based authentication. This method aligns with the needs of distributed, scalable, and secure application environments, ensuring efficient and reliable user authentication.

2.2 JSON Web Tokens (JWT)

JSON Web Tokens (JWT) are a compact, URL-safe means of representing claims to be transferred between two parties. They are widely used for token-based authentication due to their simplicity and flexibility. A JWT is composed of three parts: a header, a payload, and a signature.

2.2.1 Structure of JWT

- **Header:** The header typically consists of two parts: the type of token (JWT) and the signing algorithm being used (e.g HMAC SHA256 or RSA).

```
{"alg": "HS256",  
  "typ": "JWT"}
```


- **Payload:** The payload contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private. Registered claims include predefined fields which are optional but recommended, such as `iss` (issuer), `exp` (expiration time), and `sub` (subject). Public claims can be defined at will, while private claims are custom claims agreed upon by parties using the JWT.

```
{ "sub": "1234567890",
  "name": "John Doe",
  "admin": true }
```

- **Signature:** To create the signature part, you take the encoded header, the encoded payload, a secret, and the algorithm specified in the header, and sign that. For example, if you want to use the HMAC SHA256 algorithm, the signature will be created in this way:

```
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
secret)
```

The output is three Base64-URL strings separated by dots that can be easily passed in URLs, POST parameters, or inside an HTTP header.

2.3 OAuth 2.0 Authorization Framework

OAuth 2.0[4] is an authorization framework that allows third-party services to exchange resources on behalf of a user without sharing the user's credentials. It defines four roles and several authorization flows to handle different use cases. This module explains the core components of OAuth 2.0: the Client, Resource Server, and Authorization Server.

2.3.1 Core Components

OAuth 2.0 facilitates secure, delegated access to resources. It is widely used in scenarios where users need to grant limited access to their data to third-party applications without exposing their credentials. OAuth 2.0 uses access tokens, which are short-lived and can be refreshed using refresh tokens.

Client

- **Definition:** The client is the application requesting access to resources on behalf of the user. It could be a web application, mobile app, or any other type of application that needs to access resources.
- **Responsibilities:** The client initiates the authorization request to the Authorization Server, handles the authorization response, and uses the access token to request resources from the Resource Server.

Resource Server

- **Definition:** The resource server is the server hosting the protected resources. It is responsible for handling requests from the client and validating the access tokens provided by the client.
- **Responsibilities:** The resource server validates the access token received in the request and serves the requested resources if the token is valid. It also ensures that the client has the necessary permissions to access the resources.

Authorization Server

- **Definition:** The authorization server is the server that issues the access tokens. It authenticates the user and obtains authorization from the user for the client to access the resources.
- **Responsibilities:** The authorization server authenticates the user, authorizes the client, and issues the access tokens. It also handles token refresh requests and revokes tokens if needed.

2.4 OTP-Based Email Validation

To enhance the security and reliability of user authentication, our application incorporates an OTP-based email validation system. Upon user registration or password recovery, an OTP (One-Time Password) is generated and sent to the user's email via the Gmail SMTP server, leveraging the SMTP protocol for secure communication. The OTP has a limited validation time, which is stored in the database to ensure timely usage. To maintain database hygiene and enhance security, a cron job is set up to automatically delete OTPs that have expired, ensuring that outdated tokens do not clutter the system or pose security risks. This mechanism ensures a robust, secure, and efficient way of verifying user email addresses.

2.5 Work Flow Of Authorisation and Authentication

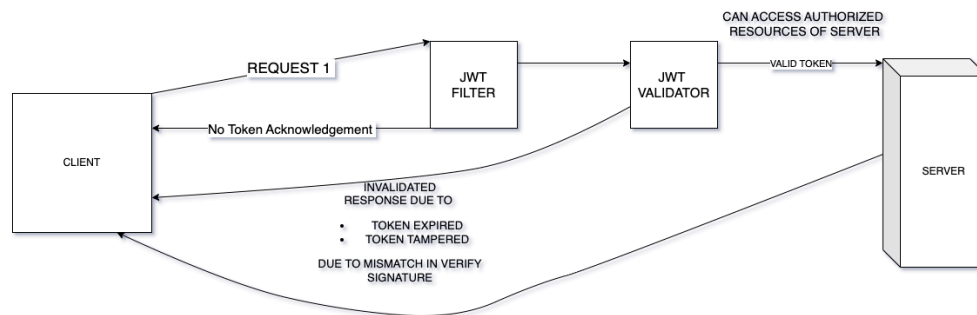


Fig 1

The first diagram shows the workflow when a client is already signed in. The client sends a request with a JWT to the server. The JWT filter intercepts the request to check for a token. If a token is present, it passes it to the JWT validator. The validator checks if the token is valid, including checking if it's expired or tampered with. If the token is valid, the client can access the server's resources. If the token is invalid, the client receives an error response.

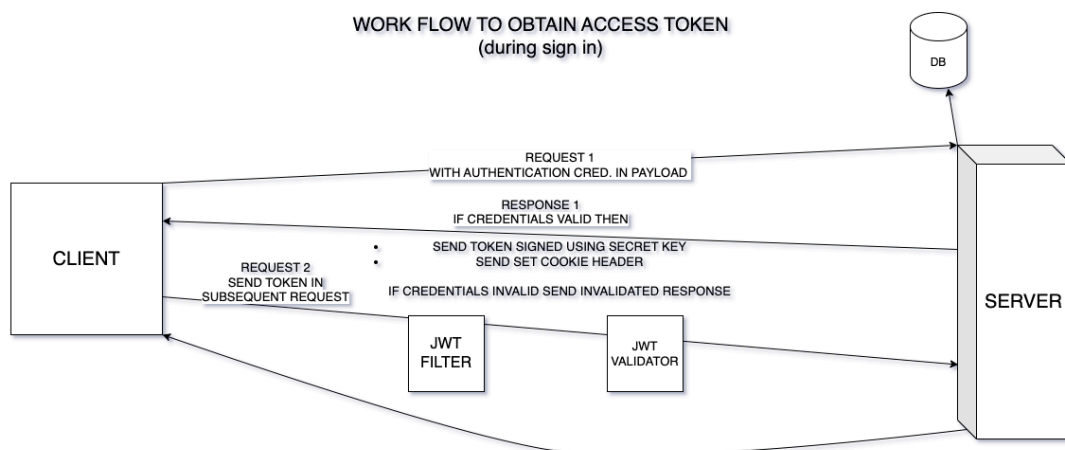


Fig 2

The second diagram outlines the process for obtaining an access token during sign-in. The client sends a request with authentication credentials to the server. The server checks the credentials against the database. If valid, the server generates a signed JWT and sends it back to the client, often setting a cookie header. The client uses this token in subsequent requests. If the credentials are invalid, the server responds with an error. The JWT filter and validator ensure that only requests with valid tokens are allowed to access protected resources.

3. Microservices Architecture for Enhanced Scalability

Microservices architecture is an architectural style that structures an application as a collection of loosely coupled services, each built around a specific business capability. This approach enhances scalability, flexibility, and maintainability, making it particularly suitable for complex, large-scale applications[5].

3.1 Problem Statement

As our application has grown, the large, monolithic codebase has become increasingly unmanageable. Adhering to SOLID principles within this monolith is challenging, leading to tight coupling and difficulties in implementing changes or new features. This complexity hampers our ability to rapidly develop, test, and deploy updates across the entire application. Additionally, scaling the entire application to meet increased demand is inefficient and resource-intensive.

3.2 Objectives

To address these challenges, we aim to transition to a microservices architecture. By adopting loose coupling, we ensure that services are independent, allowing for continuous integration and continuous deployment (CI/CD) practices to be applied efficiently. This independence enables project build, testing, end-to-end testing, and deployment to be managed on a per-service basis, rather than for the entire application. High cohesion will be maintained by grouping related functionalities within the same microservice, enhancing clarity and maintainability. For example, all user authentication functions will reside in a dedicated authentication service.


Microservices also allow for individual scalability and monitoring, enabling us to scale specific services according to demand, improving resource utilisation. Development teams can be organised around these services, fostering specialised expertise and more efficient workflows. This modular approach will significantly improve our application's scalability, flexibility, and overall performance.

3.3 Design Patterns Under Microservices Architecture

3.3.1 Use of API Gateway

To streamline our microservices architecture, we have integrated an API gateway. The API gateway serves as a single entry point for client requests, routing them to the appropriate microservices. When a client sends a request, it first goes to the API gateway, which then forwards it to the correct service. The service processes the request and sends the response back to the API gateway, which in turn sends it to the client. This setup simplifies client interactions with multiple services and centralises cross-cutting concerns such as security, logging, and rate limiting. For instance, authentication and authorization checks can be handled at the gateway level before a request reaches any microservice.(rate limiter)

3.3.2 Service Discovery



HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2024-05-22T02:46:52 +0530
Data center	default	Uptime	1 day 02:07
		Lease expiration enabled	true
		Renews threshold	10
		Renews (last min)	12

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - Ayush.api-gateway
DELIVERY-SERVICE	n/a (1)	(1)	UP (1) - Ayush.delivery-service:8000
ORDER-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.112:8081
TEST-GATEWAY	n/a (1)	(1)	UP (1) - DESKTOP-GBISURB-TEST-GATEWAY:9090
WORKER-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.112:8082

General Info

Name	Value
total-avail-memory	200mb
num-cores	4

Fig. 3

We employ Netflix Eureka[6] for service discovery, ensuring seamless communication between microservices. When a service starts, it registers itself with the Eureka server, making its location known to other services. When a service needs to communicate with another, it looks up the target service in the Eureka registry. All our microservices act as Eureka clients, automating the entire lookup process. This dynamic discovery mechanism eliminates the need for hardcoded service endpoints and allows for greater flexibility and

resilience in managing service instances.

To distribute traffic evenly across multiple instances of each service, load balancing is essential. Load balancers ensure that no single instance is overwhelmed, enhancing performance and availability. Combined with service discovery, load balancers dynamically adjust to changes in the number and status of service instances, providing robust fault tolerance.

3.4 Inter-Microservice Communication

Inter-microservice communication is handled through both synchronous and asynchronous methods. For synchronous communication, we use RESTful endpoints, enabling direct request-response interactions between services. For asynchronous communication, we utilise message brokers such as Apache Kafka, RabbitMQ, and Amazon SQS. These brokers facilitate message passing between services without requiring them to be online simultaneously, enhancing the decoupling and scalability of the system. This combination of communication strategies ensures that our microservices can interact efficiently, regardless of the timing and nature of their tasks.

By incorporating an API gateway, service discovery, and both synchronous and asynchronous communication methods, we have significantly enhanced the smoothness and efficiency of our microservices architecture. These improvements facilitate better service management, scalability, and resilience, aligning our system with modern best practices for microservices.

4. Use of Particular Databases with Their Reason

In our microservices architecture, choosing the right type of database for different services is crucial to ensure performance, scalability, and consistency. We have adopted a combination of SQL and NoSQL databases[7], each selected based on the specific needs of our application components. This section outlines our rationale for these choices and their respective use cases.

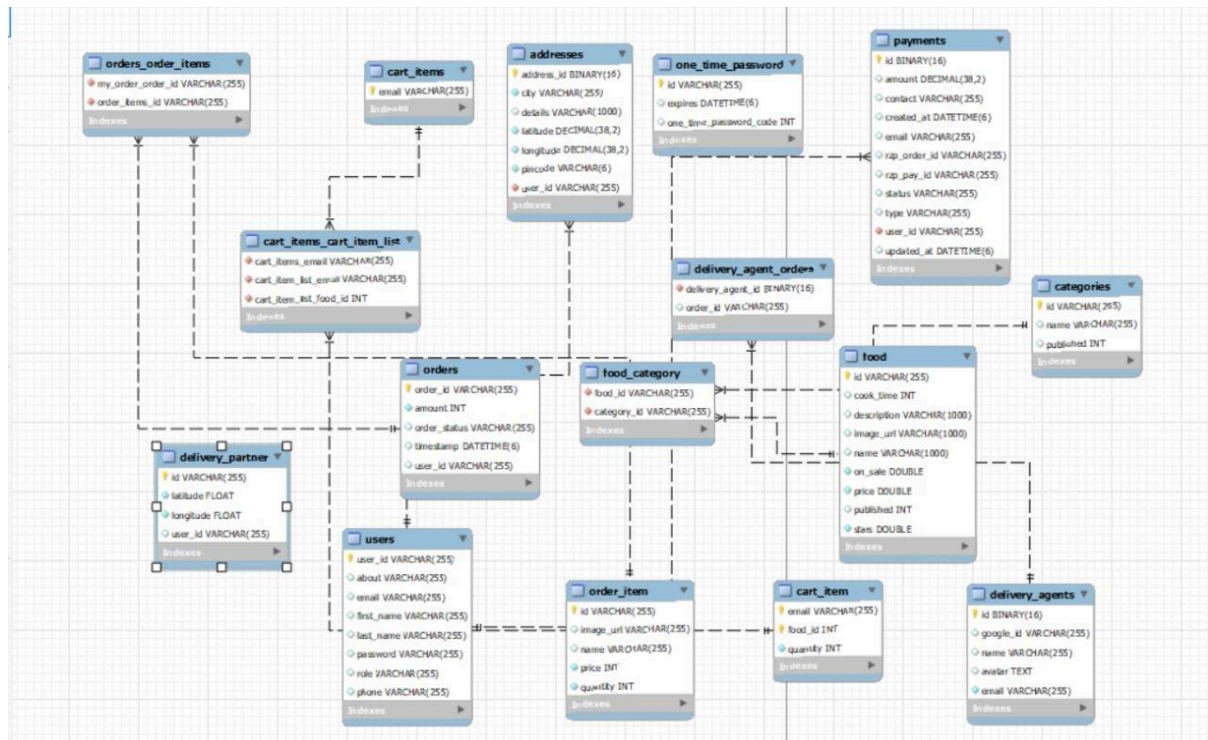


Fig. 4

4.1 SQL vs. NoSQL

4.1.1. SQL Databases

SQL databases are relational databases that offer robust transactional features and strong consistency, adhering to the principles of the CAP theorem—Consistency, Availability, and Partition tolerance (with a focus on C and P). They are well-suited for use cases requiring:

- **Transactional Integrity:** SQL databases are ideal for systems where data integrity and ACID (Atomicity, Consistency, Isolation, Durability) properties are paramount, such as payment systems where accuracy is critical.

- **Stable Schema:** When the data schema is relatively stable and not expected to change frequently. For example, user authentication details and order history in a food application benefit from the structured schema that SQL databases provide.
- **Complex Queries and Aggregation:** SQL databases excel at executing complex queries and performing aggregation functions, which are essential for generating reports and analysing data.

Cons:

- **Horizontal Scalability:** While SQL databases can be horizontally scalable, this often requires manual sharding and partitioning, involving complex algorithms and management overhead.
- **Vertical Scalability:** Scaling vertically is limited by the physical constraints of the server hardware.
- **Cross-Shard Joins:** Implementing joins across multiple shards is complex and can significantly impact performance.
- **Slower Read and Write:** Due to ACID (Atomicity, Consistency, Isolation, Durability) compliance, SQL databases can experience slower read and write operations in certain scenarios, impacting overall performance.

Use Cases:

- **User Accounts and Profiles:** Ensure robust, consistent management of user data with strong transactional integrity.
- **Orders and Transactions:** Maintain precise, reliable records of financial transactions and order histories.

4.1.2. NoSQL Databases

NoSQL databases like MongoDB are designed for flexibility and high scalability, following the principles of BASE (Basically Available, Soft state, Eventual consistency), with a focus on Availability and Partition tolerance (A and P). They are suitable for:

- **Schema Flexibility:** NoSQL databases allow for dynamic schema changes, making them ideal for scenarios where the data model evolves over time. For instance, in a food application, the categories and attributes of food items may change frequently.
- **High Scalability:** NoSQL databases[8] like MongoDB offer automatic sharding and partitioning, which simplifies horizontal scaling and improves read and write efficiency.

- High Throughput: They are optimised for handling large volumes of data and high throughput for read and write operations.

Cons:

- Aggregation Support: NoSQL databases typically offer less robust support for complex aggregations compared to SQL databases.
- Consistency: They provide eventual consistency rather than strong consistency, which may not be suitable for all use cases.

Use Cases:

- Real-time Notifications and Updates: Efficiently handle high-volume, low-latency data updates and notifications.
- Session Management : Store and manage user sessions with flexible schema and rapid read/write capabilities. Cart
- Location Data: Handle and query dynamic location-based information with scalability.
- Personalization and Recommendations: Manage and analyze personalized user data to generate recommendations quickly.
- Food Categories and Items: NoSQL databases efficiently manage dynamic data structures, allowing flexible categorization and seamless updates to food items and post-order processes.

4.2 Classification Based on Storage Mechanism: In-Memory and Disk Storage

In-memory databases store data in main memory rather than on disk, offering significantly faster read and write operations compared to disk-based storage. This speed is attributed to the direct access nature of main memory, which eliminates the need for disk I/O operations. However, in-memory databases suffer from the disadvantage of volatile data storage—data resides only in RAM and is lost if the database server is shut down unexpectedly.

In-memory databases leverage the high-speed access of RAM, allowing for rapid retrieval and modification of data without the latency associated with disk I/O operations[9]. This speed is crucial for applications requiring real-time data processing and low-latency response times. On the other hand, disk-based storage relies on physical disks, which have inherently

slower read and write speeds due to mechanical limitations. Disk I/O involves seeking, rotating, and accessing data on the disk platters, resulting in longer access times compared to in-memory fetches.

Cons:

1. **Data Volatility:** In-memory databases are volatile by nature, as data resides only in RAM and is lost if the database server is shut down unexpectedly. This poses a risk of data loss in the event of power outages or server failures.
2. **Limited Data Size:** The size of data that can be stored in memory is limited by the available RAM, which may restrict the scalability of in-memory databases for applications with large datasets.
3. **Cost:** In-memory databases may require more memory resources, potentially increasing hardware costs compared to disk-based storage solutions.
4. **Persistence Challenges:** Ensuring data persistence in in-memory databases requires additional mechanisms such as periodic snapshots or replication to disk, which can add complexity to the system.

Overall, while in-memory databases offer unparalleled performance for certain use cases, they come with trade-offs in terms of data volatility and scalability compared to disk-based storage solutions. It's essential to carefully evaluate the requirements of the application and the characteristics of each storage mechanism to determine the most suitable option.

Use Case

Location Data: In-memory databases like Redis are used to store and manage real-time location data of delivery agents, providing swift updates and seamless tracking capabilities while being able to tolerate occasional data loss due to their transient nature.

4.3. Categorization Based on Importance of Data and Data Access Pattern

Categorising databases based on the importance of data and access patterns helps optimise performance and resource allocation. In a food ordering app that tracks the location of

delivery persons, we can classify databases as follows:

1. Transactional Database (Importance: High, Access: Frequent):
 - Example: SQL database for storing order details, user accounts, and transactional data.
 - Explanation: This database ensures the integrity and consistency of critical data such as orders, payments, and user accounts. It handles frequent transactions, updates, and queries related to order processing, user authentication, and financial transactions.
2. Cache Database (Importance: Medium, Access: Very High):
 - Example: Redis for caching frequently accessed location data of delivery persons.
 - Explanation: Redis acts as an in-memory cache to store frequently accessed data, such as the real-time location of delivery persons. By caching this data, the app reduces latency and improves responsiveness for location-based services, enhancing the overall user experience.
3. Analytical Database (Importance: Low, Access: Infrequent):
 - Example: Data warehouse or SQL database for storing historical order data and analytics.
 - Explanation: This database serves as a repository for storing historical order data and facilitating advanced analytical processes to derive insights, trends, and reports essential for informed decision-making and strategic planning. While direct access to this data may be infrequent, its significance lies in its capacity to empower comprehensive business intelligence initiatives, driving organisational growth and competitiveness over the long term.

4.4. About Redis (Location Data)

Redis is a high-performance, in-memory data store known for its versatility and single-threaded, event-driven architecture, making it exceptional at fast caching and real-time data processing. In the context of a food ordering app:

- Usage: Redis can efficiently store and retrieve frequently accessed location data of delivery persons in real time.

- Advantages: Its in-memory nature allows for rapid read and write operations, making it ideal for caching frequently accessed data like location updates. Additionally, Redis offers data structures and features optimised for location-based services, such as geospatial indexing and sorted sets.
- Example: Storing delivery person locations with Redis' Geo Commands allows for efficient querying of nearby delivery persons based on customer location, enabling optimised order assignment and tracking.

Overall, Redis serves as a valuable component in the architecture of a food ordering app, enhancing performance and user experience by efficiently managing and serving location data in real time.

5. Websockets : Flow Of Data From User to Admin And Admin To User

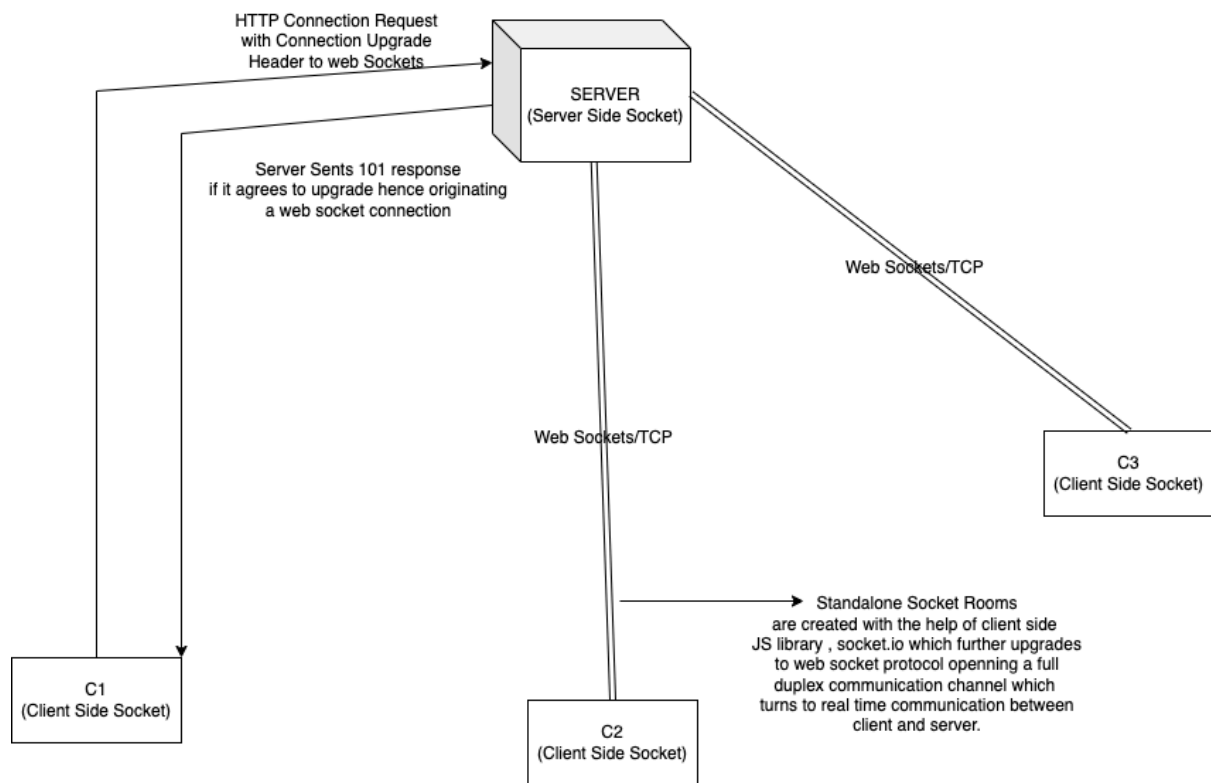


Fig 5

In our food ordering application, real-time communication is crucial for efficient and seamless user experiences, such as live order status updates and instant notifications. The provided diagram illustrates how WebSocket technology supports this functionality.

Establishing the WebSocket Connection

1. HTTP Connection Request: Clients (C1, C2, C3) send an HTTP request with a Connection: Upgrade header to the server, indicating a desire to switch to a WebSocket connection.
2. Server Response: The server responds with a 101 Switching Protocols status to confirm the upgrade, establishing a WebSocket connection.

WebSocket Communication

- Full-Duplex Channel: A WebSocket connection allows simultaneous two-way data flow between clients and the server, enabling real-time interactions.

- Client-Side Sockets: Clients (end-users, shopkeepers, delivery agents) maintain their own WebSocket connections for continuous data exchange.

Socket Rooms and Real-Time Communication

- Socket.io Library: Utilized on the client side to manage WebSocket connections and real-time events efficiently.
- Standalone Socket Rooms: Virtual channels are created on the server to group related communications, such as order tracking, restaurant management, and customer support.
- Real-Time Updates: These rooms ensure targeted real-time updates, like order status notifications to customers and order details to restaurants.

Benefits in Food Ordering Application

- Instant Updates: Customers receive real-time notifications about order status and delivery progress.
- Efficient Management: Restaurants can manage orders and inventory in real-time.
- Enhanced Communication: Real-time chat capabilities improve customer support and satisfaction.

Utilising WebSocket technology with tools like socket.io ensures our food ordering application delivers a responsive and interactive experience. It allows real-time data exchange, facilitating effective communication between customers, restaurant owners, and delivery agents.

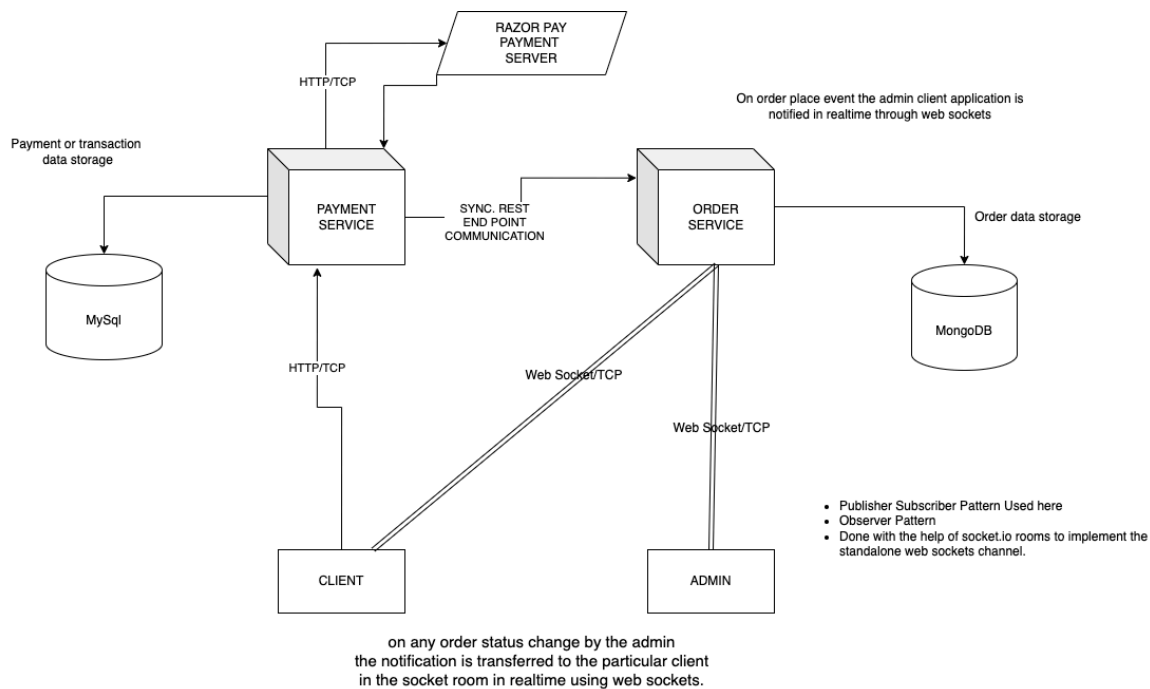


Fig 6

Our system leverages WebSockets to enable real-time data flow between the client application and the admin interface, ensuring that updates and notifications are instantaneous and synchronised. Here is a brief overview of the current setup as illustrated in the provided diagram:

1. **Client to Payment Service:** The client initiates a payment request, which is handled by the payment service via HTTP/TCP. The payment service communicates with the Razorpay payment server to process transactions.
2. **Payment Service to Data Storage:** Payment or transaction data is stored in a MySQL database, ensuring all transaction details are recorded for future reference and analysis.
3. **Payment Service to Order Service:** Upon successful transaction processing, the payment service communicates with the order service through synchronous REST endpoints. This step ensures that order details are updated in the system.
4. **Order Service Notifications:** The order service is responsible for updating order status and notifying relevant parties. It communicates with MongoDB for order data storage.

5. Real-Time Notifications via WebSockets: The order service uses WebSockets to notify both the client and admin applications of any changes in order status in real-time. This enables immediate updates and ensures that both the user and admin are always in sync with the current state of the orders.
6. WebSocket Communication: WebSockets facilitate direct, continuous, and bidirectional communication channels between the client, order service, and admin application. This setup follows the publisher-subscriber and observer patterns, implemented using socket.io rooms for isolated communication channels.

6. Functionality

In our comprehensive SaaS solution, we've meticulously crafted three distinct applications to cater to the diverse needs of our ecosystem: the End-User Solution, Shopkeeper Solution (Admin Application), and Delivery Solution. Each application serves a specific role in streamlining operations, enhancing user experiences, and ensuring seamless interactions across the platform.

6.1 End User Solution

At the heart of our ecosystem lies the User Application, a feature-rich platform designed to empower local clients with unparalleled convenience and choice in their food ordering endeavours. From intuitive order placement and real-time tracking to engaging blog access and seamless payment options, the User Application redefines the dining experience, placing control and satisfaction squarely in the hands of our valued customers.

1. **User Registration:** The user application facilitates seamless registration for clients, enabling them to create personalised accounts with minimal effort. By providing essential details such as name, email, and contact information, users can swiftly sign up and access the full range of features offered by the app.
2. **Food Ordering:** With an intuitive and user-friendly interface, clients can effortlessly explore a diverse array of food options curated from local restaurants. From appetisers to desserts, users can browse menus, view item details, and add their desired selections to the cart with just a few taps, streamlining the ordering process.
3. **Address Management:** Recognizing the importance of flexibility in delivery preferences, the user application empowers clients to manage multiple delivery addresses conveniently. Whether it's home, work, or a friend's place, users can effortlessly register and edit their addresses to ensure prompt and accurate food delivery wherever they may be.
4. **Review System:** In fostering a community-driven ecosystem, the app encourages users to share their dining experiences through a robust review and rating system.

Clients can express their satisfaction or provide constructive feedback on food quality, service, and overall restaurant experience, contributing to a transparent and informed dining community.

5. **Blog Access:** Beyond food ordering, the user application serves as a hub for culinary enthusiasts, offering access to a rich repository of engaging and informative blogs. From cooking tips and recipe ideas to restaurant spotlights and culinary trends, users can indulge their passion for food and expand their culinary knowledge through engaging content curated by industry experts.
6. **Coupon Code Application:** To enhance affordability and incentivize patronage, the app integrates a seamless coupon code application feature. Users can leverage exclusive discounts and promotional offers by simply applying valid coupon codes during the checkout process, unlocking savings on their orders and enhancing overall value.
7. **Order Tracking:** Keeping clients informed and empowered throughout the ordering journey, the user application provides real-time order tracking functionality. Users can monitor the status of their orders from confirmation to delivery, receiving timely updates and notifications to track their progress and anticipate arrival times accurately.
8. **Delivery Boy Tracking:** Ensuring transparency and peace of mind, the app enables users to track the real-time location of their assigned delivery person. By providing live location updates and estimated arrival times, clients can plan and prepare for the delivery, minimising wait times and maximising convenience.
9. **Payment Options:** With a focus on security and convenience, the user application supports a variety of secure payment options, including UPI and other approved methods. Clients can complete transactions with confidence, knowing that their financial information is protected, and enjoy a seamless and hassle-free payment experience.

6.2 Shopkeeper Solution

The Admin Application is designed to provide restaurant owners and administrators with comprehensive control and management capabilities. It offers a range of features to streamline operations and enhance service delivery

1. **Add and Remove Food Items:** Administrators can easily add new food items to the menu or remove existing ones. This allows for quick updates to offerings based on availability and customer preferences.
2. **Category Management:** Admins can create new categories or remove outdated ones, ensuring that the menu is well-organised and intuitive for users. This helps in categorising the food items effectively, making it easier for users to navigate.
3. **Order Tracking and Management:** The application provides real-time tracking of orders placed by users. Admins can monitor the status of each order, update statuses, and ensure timely processing and delivery. They can also approve or block orders as necessary, maintaining control over the transaction flow.
4. **User Management:** Administrators have the authority to view detailed user profiles, including order history and account information. They can also block users if needed, ensuring the platform remains secure and user behaviour is regulated.
5. **Analytics and Reporting:** The Admin Application offers robust analytics tools to monitor the overall performance of the app. This includes sales data, user activity, and order trends, providing valuable insights to inform business decisions and optimise operations. (DashBoard)
6. **Coupon Code Management:** Admins can create and manage coupon codes, allowing them to offer discounts and promotions. This feature helps in attracting new customers and retaining existing ones through strategic marketing efforts.

7. **Blog Posting:** The application allows administrators to post blogs about various restaurants, food trends, and other relevant topics. This helps in engaging with the community, providing valuable content, and promoting restaurants on the platform.
8. **Real-Time Updates with Web Sockets:** All these administrative actions and updates are done in real-time using web sockets. This ensures that any changes made by the admin are instantly reflected across the platform, providing a seamless and responsive user experience.
9. **Delivery Agent Assignment**

6.3 Delivery Boy Application

Rounding out our suite of applications is the Delivery Boy Application, a purpose-built solution designed to streamline delivery logistics and maximise efficiency on the ground. Equipped with real-time route optimization, order prioritisation, and seamless communication features, the Delivery Boy Application ensures prompt and reliable order fulfilment, delighting customers with timely deliveries and superior service.

1. **Order Assignment:** Once an order is placed and ready for dispatch, it is assigned to the chosen delivery agent. The assignment is done based on various factors such as proximity to the pickup location, availability, and workload balance. The delivery agent receives a notification with the details of the new order assignment, ensuring they are immediately aware of their next task.
2. **Order Status Check:** The delivery agent can check the current status of the order within the application. This includes stages such as "Order Received," "Picked Up," "In Transit," and "Delivered." Real-time updates allow the delivery agent to keep track of the order progress and take necessary actions to move the order to the next stage.
3. **Map Functionalities:**
 - Location of Source and Destination:** The application provides the delivery agent with the exact locations of the pickup point (source) and the delivery point (destination) on an integrated map.
 - Route Navigation:** The map functionality

includes route planning and navigation, offering the most efficient path between the source and destination. This feature helps in reducing delivery time and optimizing fuel usage. Live Traffic Updates: The map may also include real-time traffic information, allowing the delivery agent to avoid congested routes and make timely deliveries.

4. User Details: The application provides the delivery agent with essential details about the customer, including their name, phone number, and address. This information ensures that the delivery agent can contact the customer if needed, verify the delivery details, and provide personalised service.

6.4. Payment Gateway Integration

The provided diagram outlines the integration flow of a payment gateway within our system, detailing the steps from initiating a payment request to notifying the client of the payment status. Here's a brief explanation of each step involved:

1. Initiate Payment Request: The client begins the process by initiating a payment request through the API Gateway. This step involves submitting the order details and the amount to be paid.
2. Redirect to Payment Service: The API Gateway redirects this payment request to the Payment Service. This redirection ensures that the payment processing is handled securely and efficiently.
3. Create Razorpay Order Object: The Payment Service sends a request to the Razorpay Payment Server to create an order object. This request includes the order details and the specified amount, generating a unique order receipt.
4. Receive Razorpay Order Object: The Razorpay Payment Server processes the request and returns the order object to the Payment Service. Concurrently, a billing receipt is created on Razorpay's internal server for tracking purposes.

5. **Open Payment Window:** The Payment Service returns the Razorpay order object to the client through the API Gateway. The client then opens the payment window, allowing the user to complete the payment within a stipulated time period.
6. **Complete Payment:** The user pays the stipulated amount using various payment methods such as Card or UPI. The Razorpay Payment Server processes the transaction accordingly.
7. **Webhook Notification:** Upon any payment event (success, failure, or hold), an integrated webhook is triggered by Razorpay. This webhook notifies the Payment Service of the transaction status in real-time.
8. **Data Storage:** The Payment Service stores the necessary transaction details, including the transaction ID, payment status, and amount, in a MySQL database. This ensures that all payment information is recorded and accessible for future reference and reconciliation.
9. **Notify Client:** Finally, the Payment Service notifies the client about the payment status directly after the request is processed. This immediate feedback ensures that the user is informed of the transaction outcome without delay.

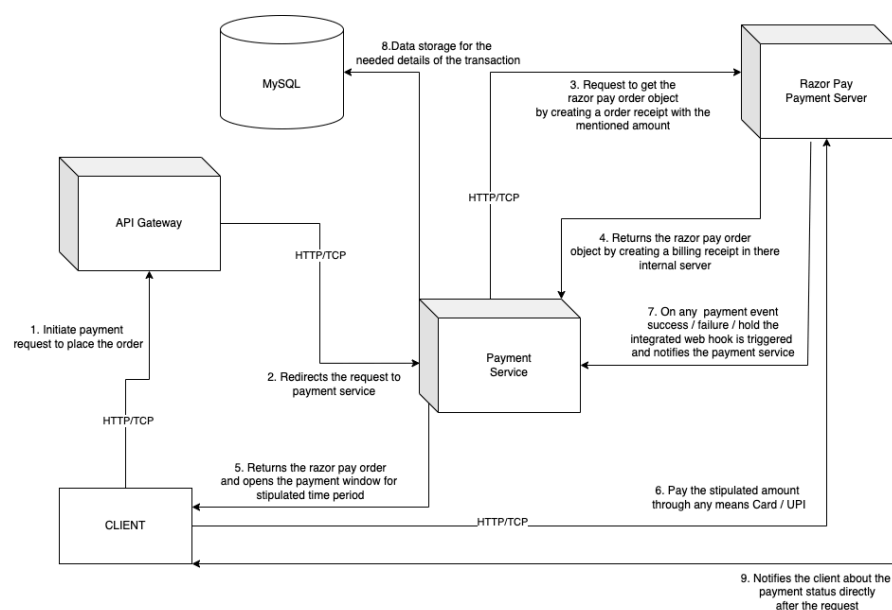


Fig 7

This payment gateway integration flow ensures a secure, efficient, and real-time payment processing system. By leveraging the strengths of Razorpay's API, our Payment Service, and real-time notifications, we provide users with a seamless and reliable payment experience. The system's design ensures that all critical steps are handled efficiently, from initiating a payment request to final client notification.

7. Real Time Location Tracking

The tracking method that makes use of GPS as well as logistics databases to determine the current location of a person, vehicle, or object at any moment in time is known as Real-time Tracking. The GPS-equipped device communicates its location data to a central server or a cloud-based platform using various communication technologies such as cellular networks, Wi-Fi, or satellite communication. The server or cloud platform receives the location data from multiple devices and processes it to determine the real-time location of each device. This processing may involve algorithms to filter out inaccurate data and improve the accuracy of location tracking.

7.1. Application of RTLS

In our SaaS solution, we leverage Real-Time Location Tracking System (RTLS)[10] technology to enable the continuous streaming of location updates to users at scale. Our primary objective is to provide real-time visibility into the location of delivery agents and seamlessly display it on a map interface within our application. By implementing RTLS, we ensure that users can track the precise location of their delivery agents in real-time, enhancing transparency and enabling efficient coordination of deliveries. This feature not only improves operational efficiency but also enhances customer experience by providing accurate and timely updates on delivery statuses. Through the continuous streaming of location updates, we empower users to make informed decisions and optimize their delivery processes effectively.

7.2. Failure of Normal RLTS Architecture

Traditional databases are ill-suited for Real-Time Location Tracking Systems (RLTS) at scale due to several key limitations. Firstly, databases prioritise data integrity over real-time updates, which is crucial for time-sensitive tracking in RLTS. Additionally, they often lack the necessary throughput to handle the continuous stream of location updates efficiently. This inability to handle a high volume of writes is further exacerbated by the heavy emphasis on write operations inherent in real-time tracking systems. Even with master-slave architecture, maintaining consistency poses significant challenges. Consequently, these limitations render traditional databases inadequate for supporting RLTS architectures effectively at scale.

7.3 Why use SSE over HTTP/Web Sockets

Server-Sent Events (SSE) are often preferred over HTTP/WebSockets for real-time location updates due to several key reasons. SSE is simpler to implement, operating over standard HTTP/HTTPS protocols without requiring additional protocol negotiation[11]. This simplicity makes it easier to set up and maintain, ensuring that location updates are seamlessly delivered to the user. Given that the communication in this scenario is predominantly unidirectional—from the server to the client—SSE is a perfect fit. The server sends continuous latitude and longitude updates of the delivery agent to the user's app, aligning well with SSE's design for one-way data streaming.

Moreover, SSE offers built-in support for automatic reconnection. If the connection drops, the client's app will automatically attempt to reconnect to the server, ensuring that the user continues to receive real-time updates without interruption. This feature is crucial for maintaining a stable and reliable connection for tracking the delivery agent's movements. SSE is also lightweight, utilizing simple text-based data streams, which can be more efficient than the binary framing used in WebSockets, especially when frequent updates are needed.

Additionally, since SSE operates over HTTP/HTTPS, it is compatible with existing network infrastructure, such as proxies, firewalls, and load balancers, making it easier to integrate into the current system without significant changes. While WebSockets enable full-duplex communication and can be useful for scenarios requiring bidirectional communication, the simplicity, reliability, and HTTP compatibility of SSE make it an ideal choice for the food delivery app, where real-time updates on the delivery agent's location are critical and the communication is primarily from the server to the user.

7.3. Apache Kafka

Apache Kafka is an open-source distributed event streaming platform originally developed by LinkedIn and later donated to the Apache Software Foundation. It is designed to handle high-throughput, fault-tolerant, and real-time data streams[12].

Here are some key features of Apache Kafka:

1. **Scalability:** Apache Kafka is designed to scale horizontally, allowing you to add more brokers to your cluster as your data throughput increases. This scalability ensures that your system can handle growing volumes of data without compromising performance or reliability.
2. **High Throughput:** Kafka is optimized for high throughput, enabling it to process and transmit large volumes of data efficiently. Its distributed architecture and use of partitioning allow for parallel processing of data across multiple nodes, further enhancing throughput.
3. **Data Durability:** Kafka provides strong data durability guarantees by persisting data to disk and replicating it across multiple brokers in a cluster. This ensures that even in the event of hardware failures or network issues, data remains available and can be recovered without loss.
4. **Fault Tolerance:** Kafka is inherently fault-tolerant, with built-in mechanisms for data replication, leader election, and automatic failover. If a broker or node fails, Kafka can seamlessly continue processing data without interruption, ensuring high availability and reliability.
5. **Real-time Stream Processing:** Kafka supports real-time stream processing through its integration with frameworks like Kafka Streams, Apache Flink, and Apache Spark. This allows you to perform real-time analytics, data transformations, and complex event processing on data streams as they flow through the Kafka cluster.
6. **Distributed Architecture:** Kafka's distributed architecture enables it to distribute data across multiple nodes in a cluster, providing scalability, fault tolerance, and high availability. Each node in the cluster can act as a producer, consumer, or broker, allowing for efficient data exchange and processing across the entire system[13].

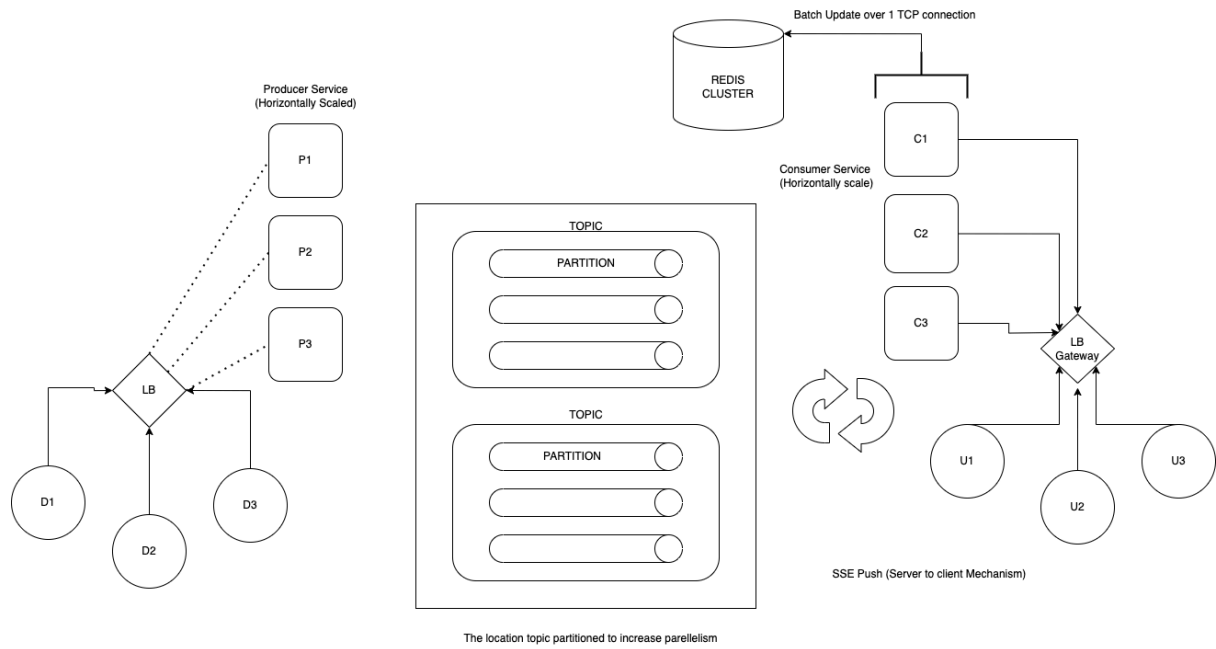


Fig. 8

In the depicted Apache Kafka architecture for handling real-time location updates, batch updates play a crucial role in efficiently managing and processing large volumes of data. After the producer services (P1, P2, P3) collect location data from delivery agents (D1, D2, D3) and send it to Kafka topics, the data is organized into partitions within those topics to enhance parallel processing capabilities.

Once the data is in Kafka, it is not immediately sent to the consumers. Instead, it is collected and batched. This means that multiple messages are grouped together into a single batch before being sent over a TCP connection to the Redis cluster. Batch processing is beneficial because it reduces the overhead associated with frequent, smaller data transfers, thereby improving network efficiency and reducing latency. The Redis cluster serves as a high-speed in-memory database that temporarily holds the batched location data, making it quickly accessible for subsequent processing stages.

On the consumer side, services (C1, C2, C3) read these batched updates from Redis. These consumer services are horizontally scaled to handle the high throughput of incoming data efficiently. By processing data in batches, the consumers can more effectively manage the load, perform necessary computations, and prepare the data for delivery to end-users.

The final step involves pushing the processed location data to the clients (U1, U2, U3) via Server-Sent Events (SSE). SSE streams the data from the server to the client in real-time,

ensuring that users of the food delivery app receive timely updates on the delivery agents' locations. The load balancer (LB Gateway) helps distribute these updates evenly among clients to maintain a consistent and reliable user experience

.

Overall, batch updates in this architecture optimize the use of network resources, enhance processing efficiency, and contribute to the system's ability to handle real-time data at scale, ensuring users receive accurate and timely location information.

Overall, Apache Kafka's combination of scalability, high throughput, data durability, fault tolerance, real-time stream processing, and distributed architecture makes it a powerful and versatile platform for building robust and scalable event streaming data pipelines, making it an ideal choice for implementing Real-Time Location Tracking Systems (RLTS) and similar use cases.

7.4. Why is Apache Kafka Fast ?

Apache Kafka's remarkable speed and efficiency are foundational to its widespread adoption in real-time data processing and event streaming applications. This is attributed to several key design principles and optimizations that enable Kafka to handle high-throughput data streams with minimal latency. Among these, the Zero Copy Principle and SequenceIO stand out as critical contributors to Kafka's exceptional performance. By leveraging these principles, along with features like partitioning, batching, and its distributed architecture, Kafka achieves unparalleled speed and scalability. Let's delve into how these elements work together to make Kafka one of the fastest and most efficient event streaming platforms available today.

1. **Zero Copy Principle:** Kafka utilizes the Zero Copy Principle, which minimizes data copying during data transmission. Instead of copying data between kernel space and user space, Kafka leverages operating system features to directly transfer data from producers to network sockets, and from network sockets to consumers. This reduces CPU overhead and memory consumption, resulting in higher throughput and lower latency.
2. **Sequence I/O:** Sequence I/O is a key component of Kafka's design that contributes to its speed and efficiency. It involves storing data in sequential order on disk, allowing

for fast reads and writes. Kafka's segmented commit log structure ensures that data is written sequentially to disk, making it efficient for both producers and consumers to append and retrieve data. Sequential I/O operations are highly optimized by operating systems and hardware, resulting in improved performance compared to random I/O.

3. **Partitioning:** Kafka partitions data across multiple brokers in a cluster, enabling parallel processing and distribution of data. Each partition can be hosted on a separate broker, allowing for horizontal scaling and improved throughput. Partitioning ensures that data is evenly distributed and processed in parallel, maximizing Kafka's speed and scalability.
4. **Batching:** Kafka optimizes data transmission by batching multiple messages into larger payloads before sending them over the network. This reduces network overhead and improves efficiency by amortizing the cost of network communication over multiple messages. Batching also allows Kafka to achieve higher throughput by reducing the number of individual I/O operations required for data transmission.
5. **Distributed Architecture:** Kafka's distributed architecture enables it to scale horizontally across multiple nodes in a cluster. This allows Kafka to distribute data processing and storage across multiple brokers, ensuring that workloads are evenly balanced and resources are effectively utilized. The distributed nature of Kafka contributes to its speed and scalability by allowing it to handle large volumes of data across multiple nodes in parallel.

Kafka's speed and efficiency are the result of a combination of factors, including the Zero Copy Principle, SequenceIO, partitioning, batching, and its distributed architecture. These design principles and optimizations enable Kafka to handle high-throughput data streams efficiently, making it a popular choice for building real-time data pipelines and stream processing applications.

7.5. Using Redis As A Central Data Store

In the realm of data storage solutions, Redis stands out as a powerful and versatile option, particularly for scenarios requiring high performance, scalability, and real-time data

processing. Its unique combination of in-memory storage, support for various data structures, persistence options, and pub/sub messaging capabilities makes Redis a popular choice for a wide range of applications. Let's explore how Redis serves as a central data store and the key features that contribute to its effectiveness[14].

- **In-Memory Data Storage:** Redis operates primarily as an in-memory data store, storing data in RAM rather than on disk. This allows for exceptionally fast read and write operations, making Redis ideal for use cases that demand low-latency data access. By keeping data in memory, Redis delivers high performance, enabling rapid data retrieval and manipulation.
- **High Performance:** Due to its in-memory architecture and efficient data structures, Redis offers exceptional performance for read and write operations. Its single-threaded design and non-blocking I/O ensure minimal overhead and fast response times, even under heavy loads. This high performance makes Redis well-suited for use cases where real-time data processing and responsiveness are critical.
- **Support of Huge Library of Data Structures:** Redis provides a rich set of data structures, including strings, lists, sets, sorted sets, hashes, and more. These data structures are optimized for specific use cases, allowing developers to choose the most appropriate structure for their application needs. Whether storing simple key-value pairs or implementing complex data models, Redis offers a versatile toolkit to handle diverse data requirements.
- **Persistence Options:** While Redis primarily stores data in memory for optimal performance, it also offers persistence options to ensure data durability. Redis supports different persistence mechanisms, including snapshotting and append-only file (AOF) persistence. These options allow data to be periodically saved to disk or logged to ensure that it is not lost in the event of a system failure or restart.
- **Pub/Sub Messaging:** Redis includes built-in support for publish/subscribe messaging, enabling real-time communication between components of an application. Publishers can broadcast messages to multiple subscribers, facilitating event-driven architectures.

and enabling seamless integration between different parts of an application. This pub/sub functionality enhances Redis's utility as a central data store for event-driven and real-time applications.

- **Scalability:** Redis is designed to scale horizontally to handle increasing data volumes and user loads. It supports clustering, sharding, and replication, allowing data to be distributed across multiple nodes in a cluster. This distributed architecture enables Redis to scale seamlessly, ensuring that it can accommodate growing data requirements while maintaining high performance and availability.

Redis's role as a central data store is bolstered by its in-memory storage, high performance, support for diverse data structures, persistence options, pub/sub messaging, and scalability features. Whether powering real-time applications, caching frequently accessed data, or enabling pub/sub messaging, Redis offers a robust and flexible solution for storing and processing data in a wide range of use cases.

8. Machine Learning Module

8.1. Introduction to Machine Learning in Food Ordering System:

In recent years, the integration of machine learning techniques into various industries has revolutionised the way businesses operate, and the food industry is no exception. Machine learning offers powerful tools to analyse data and extract valuable insights, making it a natural fit for enhancing food ordering systems.

Predicting product ratings based on user reviews is a crucial aspect of modern food ordering platforms. By leveraging machine learning algorithms, these systems can automatically analyse user feedback to predict the rating a particular product is likely to receive. This predictive capability is invaluable for enhancing user experience and decision-making processes.

Understanding the importance of predicting product ratings can significantly impact the success of a food ordering system. Positive ratings can attract more customers, increase sales, and improve overall user satisfaction. Conversely, negative ratings can signal areas for improvement, allowing businesses to address issues proactively and maintain high-quality service.

8.2. Problem Statement and Objective:

In traditional food ordering systems, user ratings are often subjective and not always concise, leading to potential misinterpretations of the true quality of the food. Additionally, misclassifications based solely on numerical ratings can further exacerbate this issue. The main focus of this project is to address these challenges by implementing a machine learning solution to convert user-written reviews into a more objective and reliable rating system.

The objective is to develop a model that can accurately predict the sentiment of user reviews, thereby assigning a mathematical value to each review based on the probability of it being negative or positive. By transforming qualitative user feedback into quantitative data, the aim

is to provide a more accurate representation of the food quality, reducing the likelihood of misleading users.

Ultimately, the goal is to enhance user experience and decision-making by providing more transparent and trustworthy information about the food offered through the ordering system.

8.3. Data Collection and Preprocessing:

The foundation of any machine learning project lies in the quality and relevance of the dataset used for training and evaluation. For this project, the Amazon Fine Food Reviews dataset from Kaggle was selected as the primary source of data. This dataset contains a rich collection of reviews for various food products, along with associated ratings provided by users.

8.3.1. Introduction to Amazon Fine Food Reviews Dataset

The Amazon Fine Food Reviews dataset is a comprehensive repository of user-generated reviews and ratings for food products available on the Amazon platform. It encompasses a wide range of food items, including snacks, beverages, condiments, and more, providing a diverse and representative sample of consumer preferences and opinions.

8.3.2. Exploratory Analysis

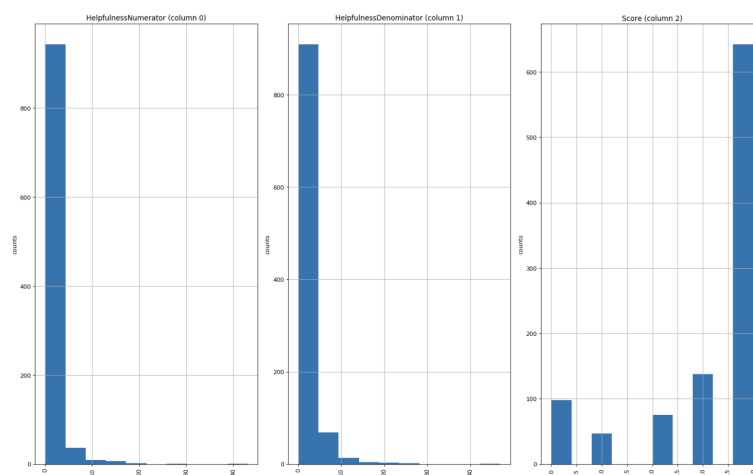


Fig. 9

Initial exploration of the Amazon Fine Food Reviews dataset reveals a notable positive bias, with a predominance of reviews rated 5 compared to lower ratings (4, 3, 2, 1). Despite this skewness, the dataset is considered a valuable resource due to its diverse collection of user-generated reviews across various food categories. Understanding the distribution and characteristics of the dataset is crucial for developing effective machine learning models aimed at sentiment analysis, particularly in addressing imbalanced class distributions and enhancing decision-making in food ordering systems.

8.3.3. Data Preprocessing

After gaining insights from the exploratory analysis, the next step involves preprocessing the data to prepare it for model training. This typically includes steps such as:

- Text cleaning: Removing irrelevant characters, punctuation, and special symbols from the text.
- Tokenization: Breaking down the text into individual words or tokens for further analysis.
- Stopword removal: Eliminating common words that do not contribute to the overall sentiment analysis.
- Lemmatization or stemming: Reducing words to their base or root forms to normalize the text.

By meticulously preparing the data through preprocessing steps, we ensure that the machine learning model can effectively learn from the input features and accurately predict the sentiment of user reviews.

8.4 Feature Engineering:

8.4.1. Features Extracted from Raw Data:

- Tokenization: The raw text data is tokenized using a tokenizer object. Tokenization involves breaking down the text into individual tokens or words, which serves as the basis for feature extraction in natural language processing tasks.

- **Padding:** After tokenization, the sequences of tokens are padded to ensure uniform length using the `pad_sequences` function. Padding is necessary for maintaining consistent input dimensions across samples and facilitating batch processing in the neural network model.
- **Embedding:** An embedding layer is added to the model, which learns dense vector representations (embeddings) of the input tokens. These embeddings capture semantic relationships between words and enhance the model's ability to understand and generalize from the textual data.

8.4.2. Techniques Used for Feature Selection or Dimensionality Reduction:

- **Vocabulary Size Limitation:** The vocabulary size is limited to a maximum number of words (in this case, 5000) using the `num_words` parameter in the tokenizer object. This technique restricts the vocabulary size and helps control the dimensionality of the input data, potentially reducing computational complexity and overfitting.
- **Embedding Dimension:** The output dimension of the embedding layer is specified as 50. This parameter controls the dimensionality of the learned embeddings. Choosing an appropriate embedding dimension balances the model's ability to capture intricate relationships between words with the risk of overfitting due to high dimensionality.

The feature engineering process in this model involves tokenizing the raw text data, padding sequences to ensure uniform length, learning dense vector representations of tokens using an embedding layer, and processing sequences of tokens with an LSTM layer. Additionally, techniques such as limiting vocabulary size and controlling embedding dimensionality are employed to manage feature space and enhance model performance.

8.5. Model Selection:

The RNN (Recurrent Neural Network) approach was chosen over the other approach due to its suitability for processing sequential data like text. RNNs excel at capturing long-range dependencies and contextual information, making them well-suited for sentiment analysis tasks where understanding the entire context of a review is crucial. In contrast, others may struggle to capture the complex relationships present in text data, particularly when dealing

with high-dimensional and sequential input.

8.5.1. Architecture of the RNN Model

The RNN model architecture comprises three main layers:

1. **Embedding Layer:** Converts tokenized input sequences into dense vector representations (embeddings). This layer learns to represent words in a continuous vector space, capturing semantic relationships between words.
2. **LSTM (Long Short-Term Memory) Layer:** Processes the embedded sequences of tokens to capture temporal dependencies and extract relevant features from the text data. LSTMs are effective at retaining information over long sequences, making them ideal for sequential data processing tasks.
3. **Dense Output Layer:** Produces final output predictions using a softmax activation function. In this binary classification task, the output layer consists of two units corresponding to the positive and negative sentiment classes.

8.5.2. Justification for RNN Architecture

The RNN architecture was considered suitable for sentiment analysis in text data due to its ability to capture sequential dependencies and contextual information. By leveraging the sequential nature of the input data, the RNN model can effectively learn the nuanced relationships between words and accurately predict sentiment based on the entire context of the review. The achieved test accuracy of approximately 92.8% demonstrates the effectiveness of the RNN model for rating prediction in the food ordering system[15].

8.6. Model Training

The model training process spans over ten epochs, during which the loss consistently decreases while accuracy steadily improves. Initially, the loss stands at 0.2163 with an accuracy of 91.44%, gradually decreasing to 0.0865 with an accuracy of 96.74% by the tenth epoch. This indicates that the model is effectively learning and adapting to the dataset over successive training iterations, achieving higher levels of accuracy with reduced loss over time.

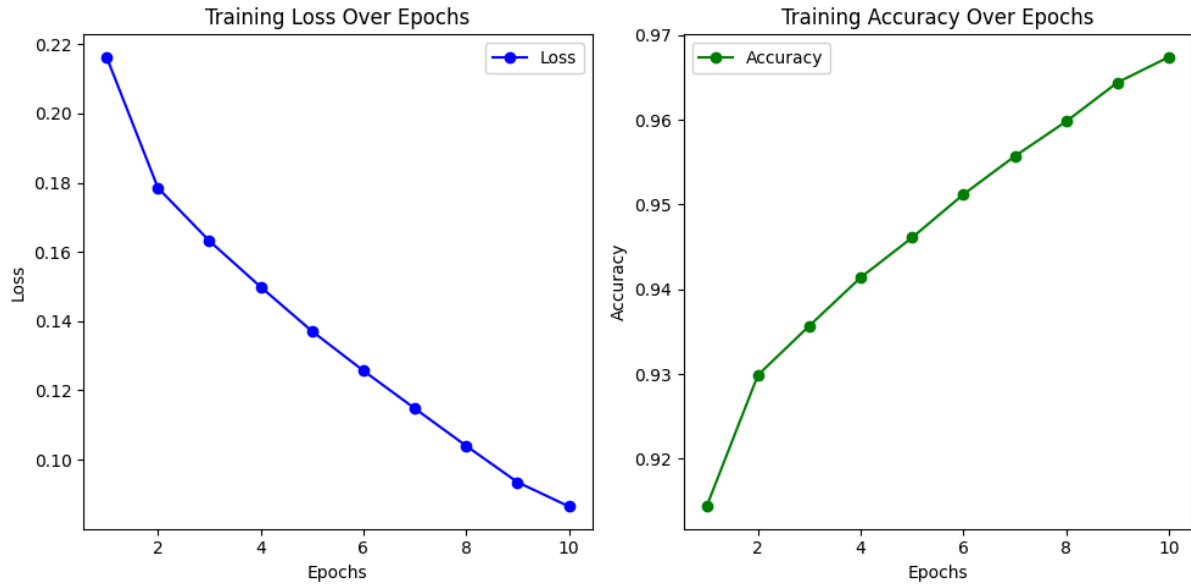


Fig 10

8.7. Model Evaluation

Key performance metrics such as Macro F1 Score, Micro F1 Score, and Hamming Loss are computed to assess the model's effectiveness in classification tasks. The Macro F1 Score, indicating the harmonic mean of precision and recall across all classes, stands at 0.8513, while the Micro F1 Score, considering the overall accuracy of the model, is notably higher at 0.9248. The Hamming Loss, representing the fraction of labels that are incorrectly predicted, is relatively low at 0.0752, indicating the model's capability to make accurate predictions across multiple classes.

Further analysis is provided through the Precision-Recall Report, offering insights into the model's performance on individual classes. With precision and recall scores ranging from 0.79 to 0.95 and 0.71 to 0.96 respectively, the model demonstrates robustness in distinguishing between different classes. The weighted average F1 Score of 0.92 signifies overall high performance, while the samples average F1 Score reaffirms consistency in classification accuracy across the dataset.

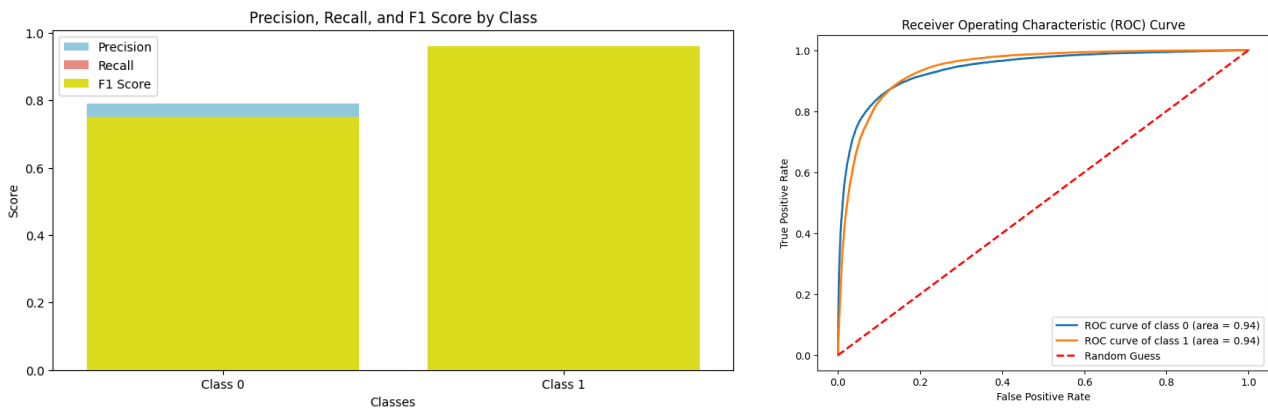


Fig.11, 12

8.8. Rating Generation

The Rating Generation sub-module utilises machine learning techniques to predict ratings for food items based on user reviews. This module is crucial for providing users with valuable insights into the quality of food items available on the platform, aiding them in making informed decisions while ordering.

8.8.1. Implementation Details:

1. Sigmoid Function Mapping:

- The probability scores of positive and negative sentiment obtained from the sentiment analysis model are mapped onto the range $[1, 5]$ using the sigmoid function.
- The sigmoid function ensures a non-linear mapping of probability scores onto the rating scale, capturing subtle differences in sentiment and providing a more nuanced representation of user feedback.

2. Rating Calculation:

- Each user review is processed individually, and the sigmoid function is applied to calculate a rating between 1 to 5 based on the positive and negative probability scores.
- This approach ensures that even small differences in the probability scores lead to significant changes in the resulting rating, enhancing the accuracy of the rating predictions.

3. Overall Rating Aggregation:

- The individual ratings generated for each review are aggregated to calculate the overall rating for each food item.
- The aggregation is performed by taking the average of all the ratings associated with a particular food item, providing a comprehensive measure of its overall quality.

8.8.2. Advantages Over Weighted Average:

- **Non-linear Mapping:** The sigmoid function offers a non-linear mapping of probability scores onto the rating scale, capturing the uncertainty in sentiment more effectively compared to a linear weighted average.
- **Smooth Transition:** The sigmoid function ensures a smooth transition between extreme ratings, reflecting subtle nuances in sentiment and providing a seamless user experience.
- **Constrained Ratings:** Ratings generated using the sigmoid function are naturally constrained within the valid range of 1 to 5, eliminating the need for additional post-processing steps.

The Rating Generation sub-module plays a pivotal role in enhancing the user experience by providing accurate and informative ratings for food items based on user reviews. By leveraging the sigmoid function for rating calculation, the module ensures a nuanced and reliable representation of user sentiment, contributing to the overall success of the food ordering platform.

8.9. Future Work and Improvements:

Enhancements for Rating Prediction Module:

- **Integration of Transformers or Pretrained Models:** Implementing advanced architectures such as transformers or pretrained models like BERT or GPT could significantly improve the accuracy and robustness of the rating prediction module. These models are adept at capturing complex contextual relationships in text data and

may provide more nuanced sentiment analysis compared to traditional RNN or LSTM architectures.

- **Addressing Positive Bias in Dataset:** Addressing the positive bias in the dataset by augmenting it with more balanced or diverse reviews could help improve the generalisation and fairness of the rating prediction model. Strategies such as data augmentation, synthetic data generation, or collecting additional user reviews from diverse sources may help mitigate the bias and improve the model's performance across different scenarios.

Overcoming Limitations:

- **Algorithmic Improvements:** Exploring alternative algorithms or approaches for aggregating individual product ratings to compute the final rating could enhance the accuracy and robustness of the rating prediction module. Simple averaging may not always capture the true sentiment of user reviews accurately, especially in cases where certain reviews hold more weight or significance than others. Techniques such as weighted averaging, sentiment analysis-based aggregation, or collaborative filtering methods could be explored as alternatives to improve the final rating estimation.

By exploring these avenues for future work and improvements, the rating prediction module can evolve to become more accurate, adaptive, and user-centric, thereby enhancing the overall user experience and satisfaction with the food ordering system. Continued research and development efforts in these areas will contribute to advancing the state-of-the-art in food recommendation and rating prediction systems.

9. Future Scope: Advancing the Project

As we look ahead, several avenues emerge for enhancing the capabilities and efficiency of our food ordering application, paving the way for a more seamless and user-centric experience.

9.1. Product Recommendation

Implementing robust product recommendation systems can significantly enhance user engagement and satisfaction by providing personalised suggestions tailored to individual preferences and browsing behaviour. Leveraging machine learning algorithms and collaborative filtering techniques, we aim to deliver relevant and enticing product recommendations that foster increased user interaction and order conversion rates.

9.2. Automated Delivery Boy Assignment:

Transitioning from manual to automated delivery boy assignment processes holds the potential to streamline operations and optimise resource allocation. By integrating intelligent algorithms and geo-sharding methodologies, we envision a dynamic system capable of efficiently assigning delivery personnel to orders based on proximity, workload, and real-time traffic conditions, thereby reducing delivery times and enhancing overall service efficiency.

9.3. Analytics Improvement

Empowering shopkeepers with advanced analytics tools and insights enables data-driven decision-making and performance optimization. By enhancing analytics capabilities, we seek to provide comprehensive visibility into key metrics such as sales trends, customer behaviour patterns, and inventory management, enabling shopkeepers to identify opportunities for growth, refine operational strategies, and drive business success.

9.4. Latency Optimization

Addressing latency concerns across the application infrastructure is paramount for ensuring prompt and reliable service delivery to users. Through performance tuning, optimization of network architecture, and leveraging cutting-edge technologies such as content delivery

networks (CDNs) and edge computing, we aim to minimise latency and enhance responsiveness, thereby delivering an unparalleled user experience characterised by swift order processing and seamless interactions.

9.5. Multi-Tenant System

Introducing a multi-tenant system empowers shopkeepers to manage their inventory and operations independently within a shared platform, catering to diverse business models and operational requirements. By implementing customizable management interfaces and access controls, we envision a flexible and scalable solution that accommodates varying business needs while maintaining data integrity and security across different tenant environments.

In essence, the future scope of our project is characterised by a commitment to innovation, efficiency, and user-centricity. By embracing emerging technologies, refining operational processes, and prioritising customer satisfaction, we aspire to continually elevate the standard of service delivery in the realm of food ordering and delivery, driving sustainable growth and success for all stakeholders involved.

10. Conclusion

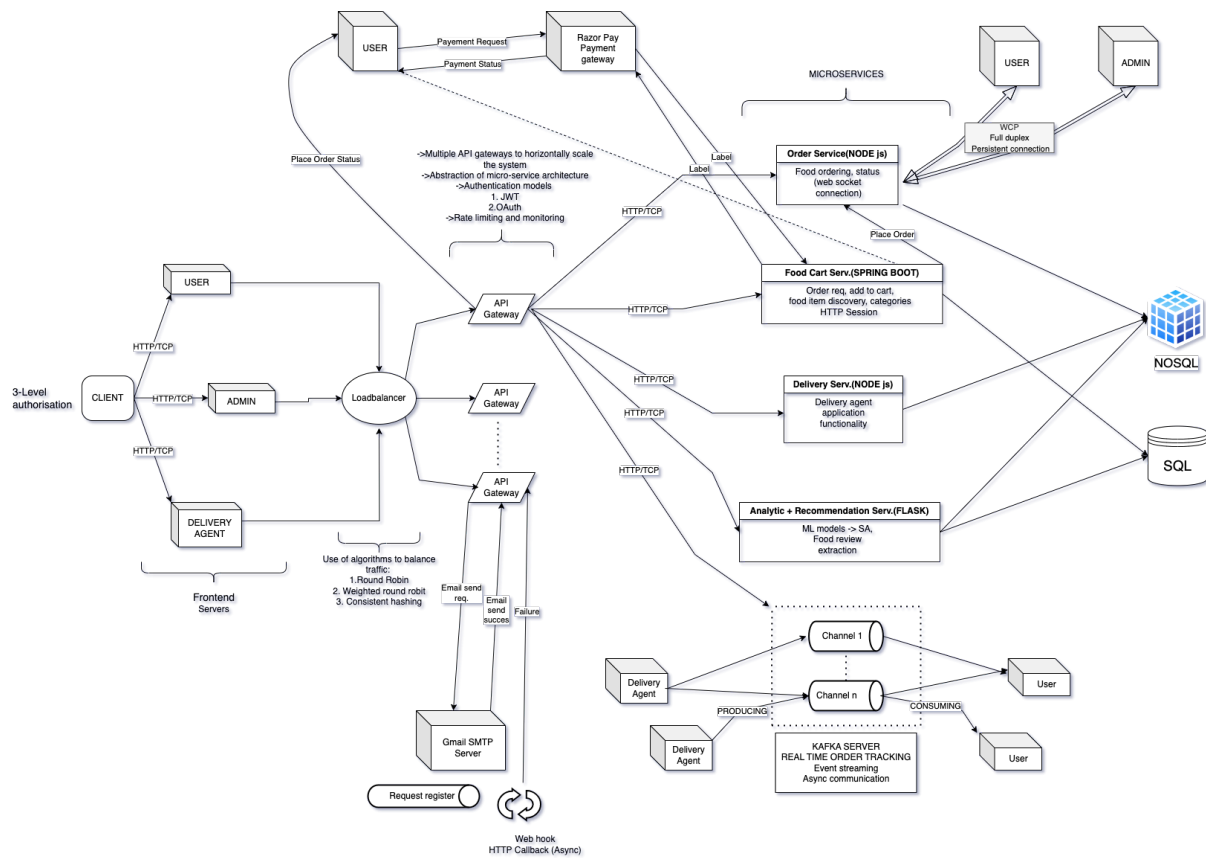


Fig. 13

In this report, we have outlined the development and implementation of our SaaS solution tailored for medium to high-rated restaurants navigating the complexities of online food delivery services. Our cloud-based platform offers a transformative alternative to traditional intermediary platforms, enabling restaurant owners to establish a robust online presence without incurring exorbitant fees or sacrificing profit margins.

We began by identifying the critical challenges faced by restaurant owners in the current competitive landscape, particularly the financial and technical burdens imposed by existing online food delivery platforms. Our solution directly addresses these challenges through a comprehensive suite of features designed to enhance efficiency, scalability, and user experience.

Key features of our platform include:

- **Cloud-Based Infrastructure:** Ensuring global accessibility and eliminating the need for restaurants to manage their own servers.
- **Multi-Application Solution:** Providing tailored experiences for end-users, shopkeepers, and delivery agents through distinct applications.
- **Subscription-Based Pricing Model:** Offering predictable costs and additional premium features for enhanced service options.
- **Automatic Updates:** Maintaining system security and feature enhancements through continuous deployment.
- **Scalability:** Utilising microservices architecture to ensure optimal performance and resource allocation as user demand grows.
- **Comprehensive Tools and Services:** Enabling efficient order management, inventory tracking, menu updates, and real-time delivery tracking.

Throughout the report, we delved into the technical aspects underpinning our solution, including authentication mechanisms, microservices architecture, database selection, real-time communication protocols, and machine learning modules for enhanced customer experience and operational efficiency.

Our authentication module, leveraging JSON Web Tokens (JWT) and OAuth 2.0, ensures secure and scalable access control. The adoption of microservices architecture, supported by robust inter-service communication and service discovery patterns, facilitates seamless scalability and maintainability. The choice of databases, balancing between SQL and NoSQL, addresses the diverse data storage and retrieval needs, while Redis serves as an efficient central data store for location tracking.

Additionally, we explored the integration of machine learning to personalise user experiences and optimise restaurant operations. By analysing customer reviews and implementing advanced rating generation techniques, our platform offers nuanced insights that surpass traditional weighted averages.

In conclusion, our SaaS solution represents a significant advancement in the online food delivery ecosystem for medium and high-rated restaurants. It empowers restaurant owners to maintain profitability, streamline operations, and enhance customer reach without compromising on quality or service. As we look to the future, we anticipate further enhancements in product recommendation, automated delivery management, analytics, and system optimization, solidifying our commitment to revolutionising the food delivery industry.