A Project Report

On

Microservices & Event Driven Architecture

BY

Vibhum Raj Tripathi (ID: 2020A7PS0247H)

Animesh AV (ID: 2020A7PS0193H)

Varun Vaddiraju (ID: 2020A7PS0173H)

Under the supervision of

Dr. Narasimha Bolloju

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

CS F491 SPECIAL PROJECT / CS F366-67 LABORATORY PROJECT /

CS F376-77 DESIGN PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(October 2023)

ACKNOWLEDGMENTS

We are greatly thankful to our mentor for this project, Dr. Narasimha Bolloju for allowing us the rare opportunity to work on a project which entailed learning new tech stacks and developmental methodologies and was not limited by any other objectives except exploration of and experimentation with Microservices based architecture, which would undoubtedly prove to be abundantly useful in our professional lives as well.

We also thank the institute as a whole for allowing project based curriculums to exist enabling us to take on projects like these to sharpen our skills differently than normal information based courses, and rely on a more hands on approach to get through the project and share our findings with others in need.

Lastly, we would also like to thank the other academic researchers and developers which developed the foundational technology stacks we used, and provided us with the foundational information on how to proceed with a new developmental pipeline.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Microservices & Event Driven Architecture**" submitted by Mr. Vibhum Raj Tripathi (ID: 2020A7PS0247H), Mr. Animesh AV (ID: 2020A7PS0193H), Mr. Varun Vaddiraju (ID: 2020A7PS0173H) in partial fulfillment of the requirements of the course CS F491 SPECIAL PROJECT, CS F366-67 LABORATORY PROJECT, CS F376-77 DESIGN PROJECT embodies the work done by them under my supervision and guidance.

Date: 22 Oct 2023 **(Dr. Narasimha Bolloju)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

In the digital age characterized by a relentless pursuit of convenience and adaptability, the RentWave project is poised to revolutionize the rental services industry by seamlessly integrating Microservices (MS) and Event-Driven Architecture (EDA). This visionary approach involves breaking down complex monolithic applications into agile, independently deployable services while enhancing real-time responsiveness and system flexibility.

RentWave adeptly tackles the challenges of development intricacies, real-time interactions, scalability, and operational reliability, weaving a highly agile, responsive, and scalable platform for users seeking a wide range of rentals, from books to vehicles and guest rooms. The project's comprehensive full-stack web application employs unique technology stacks for each microservice, implements EDA for seamless communication, leverages cloud solutions for hosting, and establishes a robust deployment pipeline through DevOps technologies.

As RentWave unfolds, it is poised to further enrich offerings and redefine the boundaries of convenience and adaptability in the world of rental services.

CONTENTS

INTRODUCTION

I. What are MicroServices

Microservices is an architecture pattern used in designing and developing software applications in which an application is built as small, loosely coupled and independently deployable services. This is an alternative to the monolithic pattern where all services are tightly coupled. A microservices based system is broken down into a set of individual services where each service is responsible for their specific and well-defined functions. These microservices generally expose APIs which are used to establish communication between them while also remaining loosely coupled.

II. Approaches To Benefit From MicroServices

Microservices can prove to be extremely beneficial in many aspects like development time, scalability, reliability to name a few of them. A bad design however, can be equally devastating to scalability and costs of an application. There are many approaches to design a good microservices based system.

Some of the most important ones are:

- **Clear Service Boundaries**: Define clear boundaries for each microservice, ensuring that they have specific, well-defined responsibilities. Avoid overlapping functionality between services.
- **Independent Development and Deployment**: Develop and deploy each microservice independently. This allows for faster development cycles and reduces downtime during updates.

III. Scope And Objectives

This project aims to explore the various methodologies available to design and develop a Micro-services based application called **RentWave** while simultaneously evaluating the pros and cons in various aspects of design and development which serves to emulate a real-world software application which needs a high level of scalability, security, reliability and cost-effectiveness.

RentWave aims to have the following objectives accomplished:

- Designing and Development of various microservices
- Seamless interaction between the various services through APIs
- Use event driven architecture to further decouple services
- Deploy the project online and ensure scalability

- Develop a deployment pipeline using DevOps Technologies like Docker and GitHub Actions.

These objectives will be used as the evaluation metric during the design process of the application.

BACKGROUND

I. Literature Review

Google defines a microservices architecture as an application that is developed as individual services. Each individual service is designed to handle a specific functionality of the application that is as independent as possible from other parts of the application. In case a service needs to communicate with other services, an interface can be exposed by each service so that each request can be parsed by a service as per its own requirements. It also suggests to containerize all the microservices as it simplifies the development by allowing a developer to focus on the functionalities rather than fixing various dependencies.[1]

Scalability works differently in a microservices based design than that of a monolithic application. Scaling a monolithic application is a comparatively simpler process where allocating more resources to the existing system can work when the traffic on the application increases. However, scaling in a monolithic application is a rigid process where independent scaling of services is not feasible since the entire application needs to be scaled even if the objective is to scale up a single service in the system. This is solved by using the microservices pattern where each service can be scaled independently of each other which results in a more cost-effective scaling pipeline.[2]

Microservices however also introduce new problems like "dependency hell" which arise in cases where multiple microservices share common dependencies like a library or a package. If a change in the library is needed to introduce a new feature or modify an existing functionality this might cause breaking changes in the other services. Not upgrading the library in the other services might work for a while but eventually the two different versions of the same library go out of sync which causes any further changes even remotely dependent on the library to become impossible. There also can arise circular dependencies where say a microservice X relies on Y and Y relies on Z but Z relies on X which makes any change in these services to require a lot of effort so as to not introduce bugs or worse, downtime which can cost billions in large corporations. [3]

Event-Driven Architecture (EDA) is a design concept that helps in decoupling services by emitting events which can then be listened to by the other services. This means that the service emitting the service need not know anything about the service that would be consuming the event which would result in a more reliable system even if one of the instances of a microservice fails, since the event that was to be handled by that instance can now be handled by the other instances or even delay handling the event until the instance comes back online. EDA also enables easier scaling since it simplifies the addition of new event producers and the load balancing of the event consumers is automated where the events are redistributed to the working nodes in case of a failure in one of them. [4]

II. Successful Implementations

Amazon

Amazon is one of the first of the large corporations today to have successfully implemented a microservices pattern with over a few hundred microservices that are currently in use. Amazon started off with a monolithic system in the early 2000's but as the number of developers and traffic on their applications increased development in their system proved to be a challenge as well as scaling up their monolith. As it adopted the microservices way of developing their applications, they soon realized massive improvements in performance and ease in scalability.[3]

During the migration each individual functionality was identified and the highly interdependent code was separated into small chunks of code that provided a small and specific function. Once all the individual chunks were identified and separated, they were put together into sections where dependent parts were put together in one section and the ones that are independent were put in other sections. This meant that the code in one section was independent of code in the other section so that the developers can work parallely in different sections without introducing breaking changes to the system. In cases where some amount of dependency was unavoidable, the sections introduced interfaces which are the only means through which the services could communicate. This was the beginning of the Service Oriented Architecture that we are familiar with today.[4]

Amazon also started developing the various solutions in regards with the microservices pattern of design like Amazon Web Services (AWS) which boasts over 200 fully working services in regards to various domains of development like data storage, logging services, cloud applications, etc.

Netflix

Netflix is another of the instances where migrating to microservices impacted its success today in a major way. Netflix originally started off in 2007 as a DVD distributor when online streaming was not a thing yet. By 2008, it was struggling to keep its services up and a major turning point came when its services went down for three days straight due to a database corruption. This led to them opting to a horizontally scalable database system rather than their existing vertically scaled database where they had no backup and an outage in it could have disastrous outcome.[5]

In 2009, Netflix began designing its microservices architecture and chose AWS as its cloud provider. This migration to cloud also enabled it to extend its services to a wider audience as well as improving its reliability and uptime. Netflix claims that the migration to cloud improved its uptime to **four nines**. Netflix hit two billion API requests per day to its API Gateway in 2013 and by 2017 it had over 700 microservices functioning on the cloud. Today, Netflix boasts over a 1000 microservices and over 40 million users worldwide yet functions smoothly due to its transition to microservices.[6]

III. Failed Implementations

Prime Video Quality Analysis [7]

Amazon's Prime Video is a massive streaming platform worldwide with over 200 million users. The Video Quality Analysis (VQA) team at prime video designed a monitoring tool for liverstreams but was not originally designed for high scalability and was aimed to monitor a few thousand concurrent livestreams and scale it up gradually over time. However, it realized that the system was not performing well when it started adding more streams to its platform and also noticed the high costs of scaling up its infrastructure to the required standards.

They initially came up with a distributed microservice architecture that mainly consisted of three components i.e, Media converter to convert the stream into a buffer that is then to a Defect Detector to identify any issues with the stream and finally an orchestration service to handle the orchestration and control the flow. This design was theoretically scalable but they hit a hard limit of 5% of the scale they needed to achieve and the cost of scaling was not good either. The reason for this was mainly due to the costs of passing the video data around the microservices which they tried to resolve by storing them temporarily in a S3 bucket but the calls to the bucket were getting expensive.

They realized that the microservices pattern was not giving them much benefits over the design they had earlier and switched to a monolith pattern. To resolve the scalability issue they parametrized their service so that they could specify each service to handle a different subset of the data. This migration to a monolith architecture from microservices reduced their costs by 90% as well as increasing scalability of their monitoring tool.

Twitter And Wix

Most organizations do massively benefit from a microservice architecture but this also over complicates their system by adding dependencies that are not essential to the functioning of their applications.

Twitter is a massive social media platform with millions of users worldwide which also provides a lot of advertising services as well as provides APIs to developers to develop applications making use of it. Twitter's CEO Elon Musk claims that only a mere 20% of microservices in twitter are essential for its functioning while the rest provide little to no functionality or performance improvements.[8]

Wix is another such instance where migrating to microservices proved to be difficult. Wix began its migration to microservices from its monolithic system when they discovered that bugs in one part of the system had the potential to bring down the entire application. They encountered new issues like addressing failures and debugging which were not the case in their monolith system. They had created an entire framework to work with microservices. The employees at Wix also had to face challenges to migrate to microservices from monolith which they were used to which resulted in a culture shift. [9]

IV. Summary Of Challenges

Scalability

Most organizations do migrate to microservices in their objective to become scalable but do not have a clear estimate of how high they might need to go. This results in either of the two cases:

- **Underestimating Scale**: The upper limit of the scaling of the application is massively underestimated which though seems to work fine would result in scaling hard limits with the existing architecture. This can only be resolved by overhauling the entire design process which would take a considerable amount of time as well as performance issues during that time due to the high load on the under powered system.
- **Overestimating Scale**: The converse is also troublesome where the scale is severely overestimated and resources are allocated to the system which remain unused resulting in un-reasonable costs of infrastructure while providing absolutely no benefit.

Communication Between Microservices

Microservices often need to communicate in some form or the other for various needs like authentication, fetching dependent data etc. This would mean that data needs to travel through long chains just to accomplish some basic functionality. This would also result in increased latency, lower throughput and higher network costs.

The long chains of communication also can introduce multiple points of failure which would mean that debugging and fixing issues in critical times would be time consuming and can result in massive losses in revenue. In case a particular service is down, the entire application has the potential to go down with it if the communication between microservices are not designed properly.

Infrastructure and Design

The microservices need to be designed with careful thought into each and every aspect to avoid any unforeseen troubles and the infrastructure must be scoped out to ensure that the suitable infrastructure is available and within the budgets as well as how reliable the infrastructure is. In cases where the infrastructure itself fails, the application would go down and would be difficult to get it back up without finding suitable replacements for it which generally takes a considerable amount of time.

RentWave OVERVIEW

In the current age where convenience and adaptability are highly sought after, the RentWave project presents an innovative platform that aims to redefine the rental services landscape. RentWave acts as a comprehensive hub for a wide range of commodities, providing a versatile solution for individuals looking to rent various items, including books, vehicles, and guest rooms. The project's primary goal revolves around developing a robust, full-stack web application built on microservices principles, each utilizing distinct technology stacks to mirror real-world application development.

The central aspect of this endeavor is the adoption of an event-driven architecture, enhancing dynamism and responsiveness. By combining this architectural approach with cutting-edge cloud solutions and the implementation of DevOps best practices, RentWave emerges as a frontrunner in modern software design. This integration of advanced technologies and forward-thinking methodologies positions RentWave as an industry influencer, setting new standards for the seamless and adaptable rental experience that today's consumers expect. In an era where convenience and adaptability are key, RentWave signifies the innovative progress towards a future where renting various items becomes more accessible, efficient, and hassle-free.

I. Key Challenges Addressed

Data Ownership In Microservices

Firstly, dealing with the complexity of building the platform is a big challenge. Making a platform that can handle many types of rentals and user features can be really hard. To tackle this, RentWave uses a microservices approach. Each microservice focuses on a specific part of the platform, which makes it easier to develop and maintain, making the whole process more straightforward and understandable.

Event Driven Architecture

In addition, the imperative need for real-time interaction demands a strategically engineered solution. Users now anticipate instantaneous engagement and up-to-the-minute updates. In response, RentWave strategically employs an event-driven architecture. This architectural decision substantially addresses the task of establishing smooth communication among the diverse components of the platform, thereby ensuring users remain seamlessly connected and continuously informed.

Event-driven architecture bestows several technical advantages upon RentWave. It fosters an environment where components of the platform can respond to events, such as user actions or external triggers, in real-time. This dynamic responsiveness enables immediate updates and feedback to users, enhancing their experience. Moreover, it facilitates the decoupling of various system elements, allowing them to function independently, which not only simplifies development and maintenance but also enhances system resilience.

By utilizing event-driven architecture, RentWave can also scale horizontally to accommodate growing user demands, as each microservice can handle events and workloads independently. This scalability ensures that the platform can seamlessly expand its capacity to meet the demands of a growing user base, without compromising the quality of real-time interactions.

Development And Deployment Pipeline

The aspect of Deployment and Operations within the RentWave project is a critical element in ensuring the platform's continuous functionality and improvement. To meet this imperative requirement, we harnessed the power of DevOps technologies, specifically leveraging tools such as Docker and GitHub Actions. This strategic adoption of DevOps principles serves not only to streamline the deployment process but also to orchestrate a seamless and efficient operational environment for the platform.

A pivotal facet of this endeavor involves the establishment of a robust Continuous Integration and Continuous Deployment (CI/CD) pipeline, which is indispensable in the modern software development landscape. This CI/CD pipeline plays a multifaceted role in RentWave's development ecosystem. It automates the integration of code changes, ensuring that new features, bug fixes, and enhancements are smoothly incorporated into the platform without disruptions or errors.

DESIGN REVIEW

I. Requirements

RentWave is a web-application that serves to simplify the renting process of various commodities like unused books and vehicles. To serve this purpose the following are the requirements that are planned to be accomplished by the application:

Functional Requirements

- Users can register and login to the application to access the core functionalities.
- Users can post an Advertisement for their unused books and vehicles with a rental price they are happy with.
- Users can view and search for other users' advertisements so that they can rent items that they like.
- Users can rent the items they like for a duration that is permissible by the owner of the advertisement.
- Users must be charged from their wallet automatically if they exceed the agreed upon duration.
- Users must be notified when their item has been bought for rent.

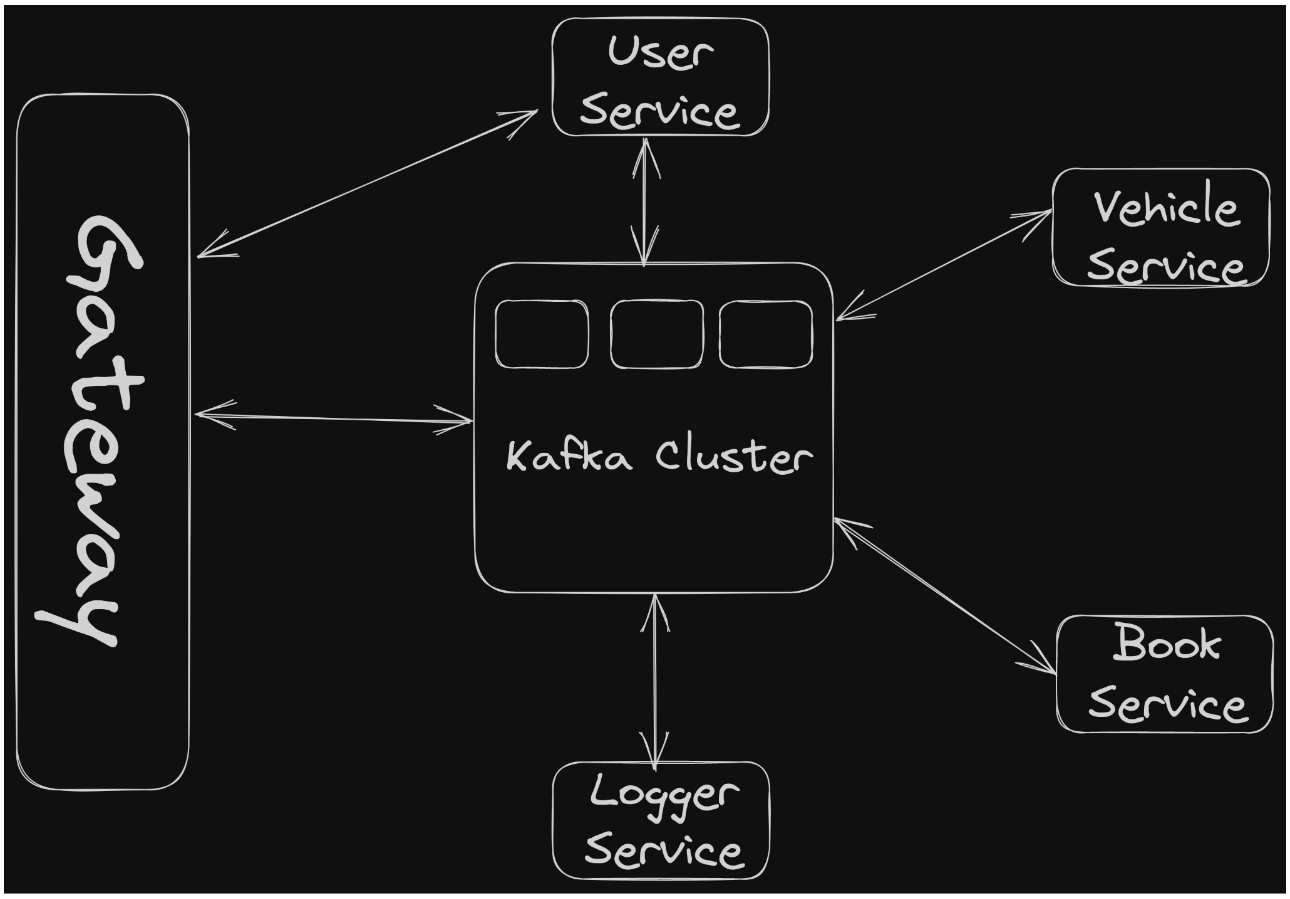
Non-Functional Requirements

- Application must be highly scalable.
- Application must be secure and support for future external APIs.
- Application must be reliable and resistant to independent failures of microservices.
- Deployment must be quick and feasible.
- Maintain logs for each event for quick and easy resolution of issues.

II. Overview Of Microservices

RentWave makes use of several microservices in order to fulfill its requirements of authentication, notifications, scalability, etc.

The brief overview of the core microservices used is shown below:



Microservices Overview

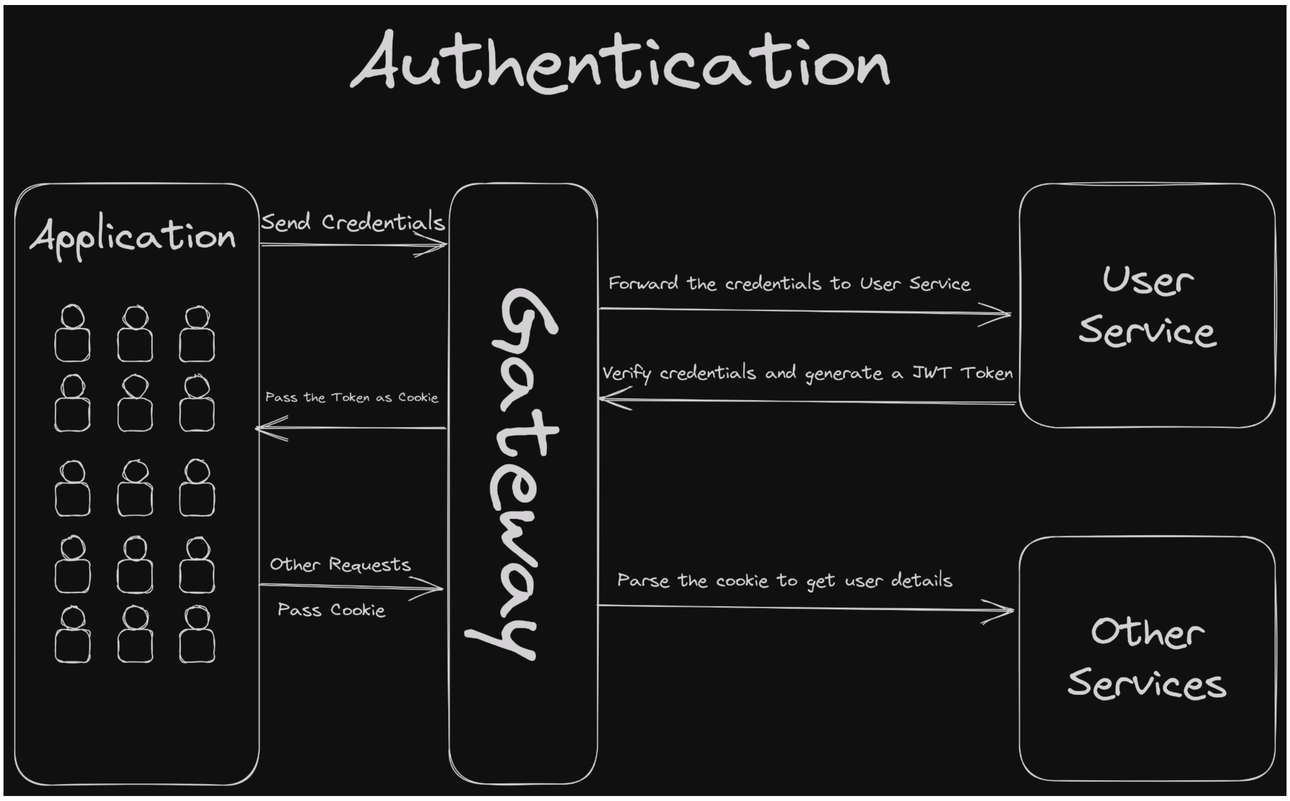
The functionalities offered by each of the above mentioned services will be described in detail in the following section.

III. Detailed Design

User Service

This service stores the data related to the user like his address, name, etc. as well as acts as the authentication service by handling the registration and login of users. Upon login, the user is granted an access token that is then used to authenticate the user in the future requests. If the token turns out to be invalid or does not exist then the request fails and the user is denied access to any of the core functionalities of the application.

Below is a brief overview of how the authentication works:



Authentication Flow

Book Service

This service handles the users that add new books for rent. This service also handles toggling the availability of books for instances where the owner of the book does not want the book to be rented out. Each book is identified by a unique isbn number. If a user is adding a book whose isbn already exists then that advertisement is linked to that book to make it easier to query multiple advertisements of the same book since the isbn is the same.

This service emits events like book.add, book.update etc., which are used by other services like logger service to log when the book was added/modified or the notification service to notify users that a new book is available for rent.

Vehicle Service

This service handles the users that add new vehicles for rent. This service also handles toggling the availability of vehicles for instances where the owner of the vehicle does not want the vehicle to be rented out. Each vehicle is tagged with a type of the vehicle so that the users can query a vehicle specific to their needs like size, seating, boot space, etc.

This service emits events like vehicle.add, vehicle.update etc., which are used by other services like logger service to log when the vehicle was added/modified or the notification service to notify users that a new vehicle is available for rent.

Rental Service

The rental service manages the renting process by users who wish to rent out items from the available pool advertisements. This service is used to display the current status of rented items of a user as well as it enforces a penalty on a user if they are late on returning the item.

The rental service will wait for a rent.complete event and if the event does not arrive before the agreed upon period of rent expires then it emits a rent.late event which will be used to deduct balance automatically from the wallet of the user.

Kafka Cluster

Kafka Cluster is used as the event broker and serves a central role in the event driven system of RentWave. All the other microservices subscribe to events in the kafka cluster which then sends the events as soon as it receives them to the subscribers. There are multiple broker nodes in the cluster to ensure reliability of the events and to make sure that events are not lost completely when a broker node goes down.

Logger Service

The logger service serves only a single purpose i.e, to log every event that occurs in the application. The logger service is important as this provides timestamped events and logs which help in not only debugging but also to resolve issues that users face while using the application which is an important aspect of user experience in any application.

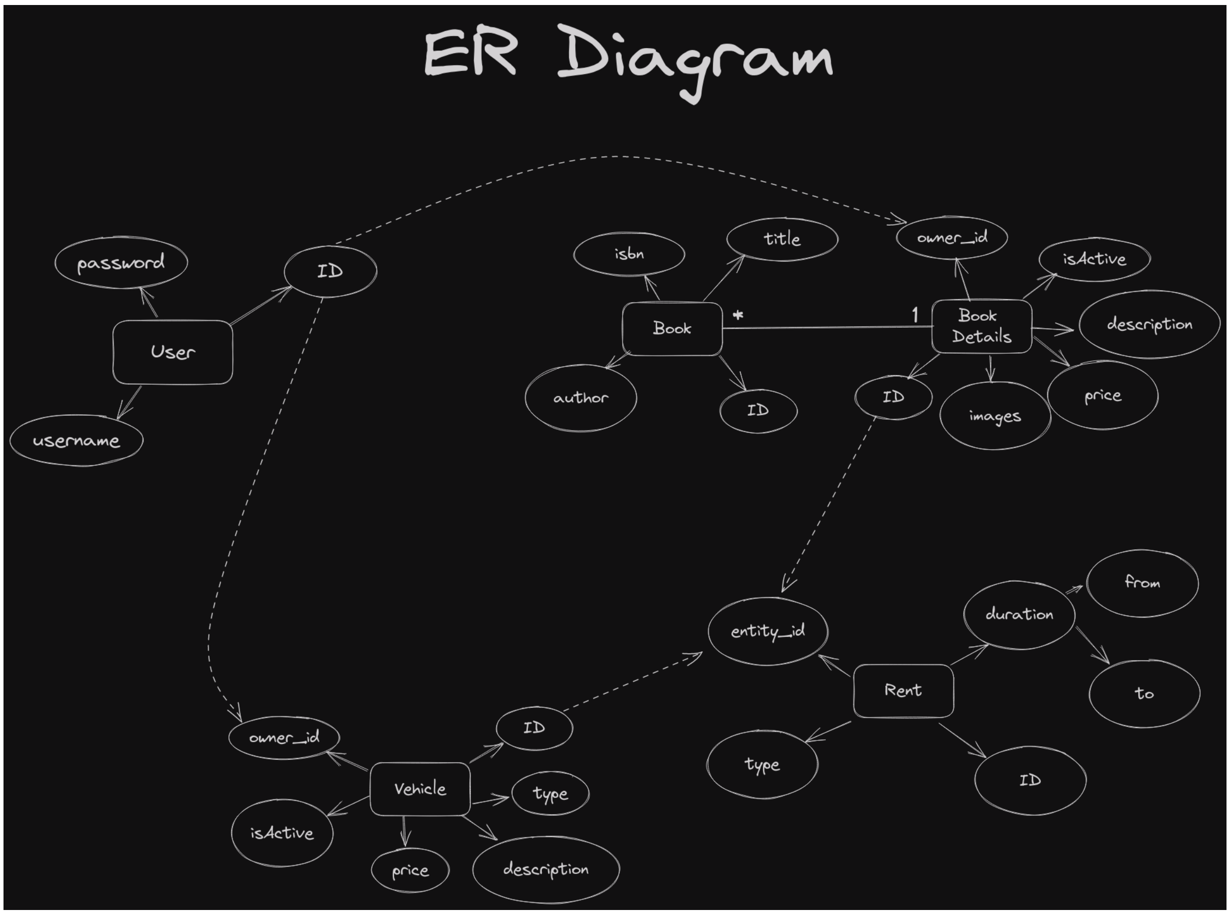
API Gateway

The API Gateway is an important aspect of RentWave since it establishes communication between the various microservices in the application. The presence of gateway eliminates the need of every service to know the location of every other service since the gateway is the only service that knows the locations of the other services and all communication is channeled through the gateway.

RentWave IMPLEMENTATION

RentWave is designed such that the data need not move around services too much which reduces latency, network costs as well as improves security by not exposing data that is not required to accomplish a particular task. This is enabled by an efficient database design.

The ER Diagram below shows the data model in brief:



ER Diagram

I. User Service and Book Service

The User Service and Book Service is implemented using golang[10] which is an extremely performant language and is widely used in various backend related services in the software development industry due to its wide support for various tools and libraries.

Performance is vital in the user service since this is responsible for authentication which becomes extremely important and has high probability to have increased traffic where time of execution becomes the most important factor and also helps in reducing costs.

Golang is also a typesafe language which improves the security of the authentication service and prevents misuse of the APIs. The emerging popularity of the language also ensures that the service will remain maintainable in the future which is a serious problem for companies that use legacy code. The new developers do not have the expertise in the tech stack and this results in an unmaintainable codebase where no issues can be fixed.

Golang has the Gin[16] framework which is used to build the server of these services and maps the handler function to the requests. Golang also provides the

GO-ORM[17] which serves as the ORM(Object Relations Mapper) for the server and the PostgreSQL database.

A PostgreSQL database is used since the schema in these services is relatively rigid and does not need to change throughout the iterations of the development process. This also helps in vertical scalability of these services since SQL databases are highly vertically scalable as compared to its NoSQL counterparts like MongoDB which are horizontally scalable.

Authentication

The authentication in RentWave is an access token based implementation. Once a user enters the username and password to register, the password is then hashed using SHA-256[12] which is a popular asymmetric hashing algorithm[11] and then sent over to the gateway so that the gateway never receives the actual password itself as well as the password is never sent over the network to prevent MITM[13] attacks. The database stores the username and the hash of the password.

During login, after the user enters the username and password and presses the login button, the password is hashed and then sent to the gateway which then forwards the request to the user service. The user service fetches the hash of the password that it stored in the database during registration and compares that hash to the one that is provided by the user. If the hashes match then a JWT[14] is generated where the payload has the user id and timestamp. This token is returned to the gateway which then converts the access token to a cookie by adding an expiration time and sends it to the client.

This cookie is then sent along with every other request and is verified before the access to the actual API is given.

II. API Gateway And Vehicle Service

The API Gateway and Vehicle Service are written in Typescript[15] which is a typesafe version of javascript and helps improve development experience as well as time. This is in conjunction with NodeJS which is a runtime environment for Javascript and enables running javascript outside of the context of a browser.

Typescript is chosen as the language here because of its compatibility with JSON. Most of the communication between the frontend and even between the microservices happens through HTTP APIs having JSON body as the payload. This enables the gateway to handle the requests from the frontend more efficiently. Since the gateway itself does not handle any business logic, the performance tradeoff is useful in favor of reliability using JSON.

Though the API Gateway itself is stateless and thus does not need a database, the vehicle service uses MongoDB for its storage needs as opposed to an SQL database since the schema for the service is likely to change due to several factors like change in license policy or even general renting policies. A NoSQL database supports a changing schema more readily than the SQL variants.

III. Frontend

The frontend for the web application is written using SvelteKit[18] which is written on top of Svelte[19], a UI Framework. This is chosen as opposed to the more popular frameworks like NextJS which is built on top of ReactJS and AngularJS due to the following reasons:

- **Performance**: SvelteKit does not work on a virtual DOM like ReactJS or AngularJS and then update the actual DOM which is significantly slower than working on actual DOM.
- **SEO**: ReactJS works by sending an empty HTML document in its initial request along with some JS which eventually populates the HTML with the content, this is generally fine but for a search engine which indexes pages based on their content. This is disastrous for ReactJS applications since the search engine indexes an empty HTML page with no content as it does not wait for the javascript to do its work.
- **Shifting Trend**: Though ReactJS is still hugely popular, it has been coming under attack due to various reasons which is causing a shift to other frameworks with Svelte being at the forefront of them.

CONCLUSION

In a world where convenience and adaptability are highly valued, the RentWave project aspires to bring significant changes to the rental services industry. This project combines Microservices (MS) and Event-Driven Architecture (EDA) to improve the way rentals work.

Microservices simplify the development and maintenance of applications by breaking them into smaller, manageable services. Event-Driven Architecture enhances real-time responsiveness and flexibility. However, adopting these technologies comes with challenges, including development complexities, real-time interactions, scalability, and operational reliability.

RentWave addresses these challenges by using microservices to streamline development. Each microservice focuses on specific aspects of the platform. Event-Driven Architecture ensures real-time interaction, improving communication between platform components. This approach offers technical benefits, such as real-time responsiveness, independent component operation, and scalability.

RentWave also benefited from DevOps technologies like Docker and GitHub Actions for efficient deployment and operation. A Continuous Integration and Continuous Deployment (CI/CD) pipeline automates code integration, ensuring smooth updates.

There is a lot more scope in leveraging the strengths of event-driven architecture in the existing system to enable a more robust system and seamless handling of requests.

In summary, the RentWave project seeks to enhance the rental services landscape by combining these advanced technologies. Users can enjoy a variety of rentals, from books to vehicles and guest rooms. As the project progresses, it is expected to improve its offerings and expand the boundaries of convenience and adaptability in the rental services sector.

REFERENCES

1. What is Microservices Architecture?- Google Cloud
2. Microservice Scalability Challenges and How to Overcome Them - developer.com
3. Why and How Netflix, Amazon, and Uber Migrated to Microservices: Learn from Their Experience
4. What Led Amazon to its Own Microservices Architecture
5. Completing the Netflix Cloud Migration
6. Optimizing the Netflix API
7. Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%
8. Elon Musk on Twitter
9. Wix.com Migration to Microservices
10. Golang
11. Asymmetric Hashing Algorithms
12. SHA-256
13. MITM Attack
14. JWT
15. Typescript
16. Gin Framework
17. Gorm
18. SvelteKit
19. Svelte