CSCI 5902 - ADVANCED CLOUD ARCHITECTING

Fall 2023

TERM PROJECT

"Architecting Applications on AWS"

Submitted by

Emayan Vadivel - B00934556

Under the Guidance of

Dr. LU YANG
Faculty of Computer Science

December 04, 2023

Contents

1 Project Introduction

1.1 Summary

PizzaMan is a **single-page react application (SPA)** that allows users to explore different pizzas available in the store and order their favourite ones smoothly. In architecting the PizzaMan application's hosting on AWS, I meticulously crafted a solution that aligns with best practices and addresses key considerations of cost, performance, security, and scalability [1]. My choices reflect a thoughtful approach to creating a robust and efficient infrastructure. I initiated the process by implementing a CI/CD pipeline using AWS CodeCommit, CodePipeline, and CodeBuild. This streamlined the development workflow, ensuring that changes to the codebase trigger an **automated deployment**, fostering **efficiency** and reducing manual interventions [8].

For hosting the static website, I opted for Amazon S3. This choice not only provides a reliable hosting solution but also allows for the *separation of static and dynamic content*, optimizing **scalability** and **performance**. To handle dynamic data storage, I integrated Amazon DynamoDB. Its serverless architecture, coupled with **low-latency** performance, makes it an ideal choice, aligning seamlessly with the overall serverless approach of the application. In bolstering the application's security, I employed Amazon CloudFront, implementing a Web Application Firewall (WAF) and leveraging ACM for SSL certificates. This combination ensures global content delivery with robust **protection** against common web exploits [4,5,6].

To enhance user authentication, I integrated Lambda@Edge, allowing for authentication with AWS Cognito at the CloudFront edge locations [7]. This not only secures the application but also reduces the load on the backend, optimizing performance. Placing Application Load Balancers within a Virtual Private Cloud (VPC) across multiple availability zones enhances **fault tolerance** and **availability**. This strategic decision ensures that the application remains resilient, even in the face of potential failures in one availability zone. The use of a Lambda function (PizzaLambda) as the core application logic ensures a serverless, **event-driven** architecture. This function performs specific tasks, such as fetching content from the S3 bucket, providing a scalable and cost-effective solution [5].

Throughout the design process, I adhered to key architecting principles. The architecture is scalable, handling variable workloads efficiently, secure with multiple layers of protection, and reliable with fault-tolerant measures in place. My choices reflect a commitment to **cost optimization**. Leveraging serverless components and pay-as-you-go models contributes to an infrastructure that scales economically, aligning expenses with actual resource usage. In conclusion, the PizzaMan application's hosting on AWS embodies a holistic approach, considering every aspect from development workflows to user authentication, security, and infrastructure scalability. The architecture reflects a nuanced understanding of AWS services, resulting in a resilient and cost-effective solution.

1.2 Features

So far, I have implemented the following list of features using various AWS Services in my web application:

Table 1.1: **Feature Descriptions**

Features	**Description**
User Authentication	Users can Register and Sign In to the application securely to order their favourite pizza.
Home Screen	A Visually appealing and welcoming home screen for the enhanced user experience. Also, acts as an entry point for the application.
Product Screen	Allows users to view all the available pizza and food items in the store. Besides, users can also add their preferred pizzas to the cart.
Cart	Allows users to see their favourite pizzas in the cart along with the quantities and total price.
Orders	Allows users to view their past orders.

1.3 Project's Tech Stack

```
# Tech Stack
- Web Technologies: HTML, CSS, JavaScript
- Framework: React.js
- Cloud Services: AWS (Lambda, S3, DynamoDB, Cognito, CodeCommit, CodeBuild, CodePipeline, Route53,
    CloudFront, VPC, ALB, WAF, ACM, CloudFormation)
```

1.4 Tech Stack Description

The following list of items will describe my project tech stack briefly:

- **Web Technologies:** Utilizing fundamental web technologies such as HTML, CSS, and JavaScript to create a dynamic and interactive user interface for the PizzaMan application.
- **Framework - React.js:** Employing React.js as the front-end framework, enabling the development of a single-page application with modular and reusable components for an efficient and responsive user experience.
- **Lambda:** Leveraging AWS Lambda for serverless computing, allowing the execution of code in response to events without the need for traditional infrastructure.
- **S3 (Simple Storage Service):** Employing S3 for scalable and secure storage of static website content and images, ensuring reliable and efficient content delivery [5].
- **DynamoDB:** Utilizing DynamoDB as a fully managed NoSQL database service to store and retrieve data with low-latency performance [6].
- **Cognito:** Implementing AWS Cognito for user authentication, managing user identities securely, and providing seamless sign-up and sign-in experiences [7].
- **CodeCommit, CodeBuild, CodePipeline:** Employing AWS Code services for version control, continuous integration and deployment, and streamlining the software development and release process [8].
- **Route53:** Utilizing Route53 for domain registration and DNS routing, ensuring users can access the PizzaMan application seamlessly [10].
- **CloudFront:** Leveraging CloudFront as a content delivery network (CDN) to distribute content globally with low latency and high data transfer speeds [12].
- **VPC (Virtual Private Cloud):** Implementing VPC for a logically isolated section of the AWS Cloud, providing enhanced network security and control [2].
- **ALB (Application Load Balancer):** Utilizing ALB for distributing incoming application traffic across multiple targets, enhancing fault tolerance and availability [13].
- **WAF (Web Application Firewall):** Strengthening security with WAF to protect web applications from common web exploits and filter malicious traffic [14].
- **ACM (AWS Certificate Manager):** Enhancing security by using ACM to manage SSL/TLS certificates for secure communication over the web [3].
- **CloudFormation:** Implementing AWS CloudFormation for infrastructure as code, allowing the definition and provisioning of AWS infrastructure securely and efficiently.

2 AWS Services

2.1 Satisfying the requirements mentioned in the project description

As per the requirements mentioned in the term project description, I have used the following list of AWS services in 8 different categories for various features of my PizzaMan application:

Table 2.1: AWS Services Usage

AWS Service Category	**AWS Service Used**	**Count**
Compute	AWS Lambda, Application Load Balancer	2
Storage	Amazon S3	1
Database	Amazon DynamoDB	1
Networking & Content Delivery	Amazon Route 53, Amazon CloudFront, VPC	3
Developer Tools	AWS CodeCommit, AWS CodePipeline, AWS CodeBuild	3
Security	AWS WAF (Web Application Firewall), AWS Certificate Manager (ACM)	2
Identity & Compliance	Amazon Cognito	1
Management & Governance	AWS CloudFormation	1

In the assignment description, it is mentioned to use any 1 storage-related AWS Service. So, according to my PizzaMan project requirements, I have used services in both the storage and database sections namely **S3** and **DynamoDB**. Because, due to the space constraints for storing images as items in DynamoDB (*DynamoDB has a maximum item size of 400KB per item*) [3], I have used S3 buckets for storing all my high-resolution static pizza images as objects. Then I used all the object URLs as items in my DynamoDB tables along with other product data. I will explain more about my data storage in the following sections.

2.1.1 AWS Service mappings for implemented features

Table 2.2: Feature Implementation based on the used AWS Services

Features	**Description**
User Authentication	**AWS Cognito** was used for user authentication. It provides features such as user registration and sign-in.
Cart	**Amazon DynamoDB** was used to store user data, order details and product details in separate tables.
Product Images	**Amazon S3** was used to store high-resolution product images, and **DynamoDB** was used to store product details, including the URLs of the images stored in the S3 bucket.

So, Table 2.1 describes the requirements satisfaction of all the required AWS Services in my PizzaMan project and Table 2.2 describes the features in which those AWS Services are used.

2.1.2 Overview

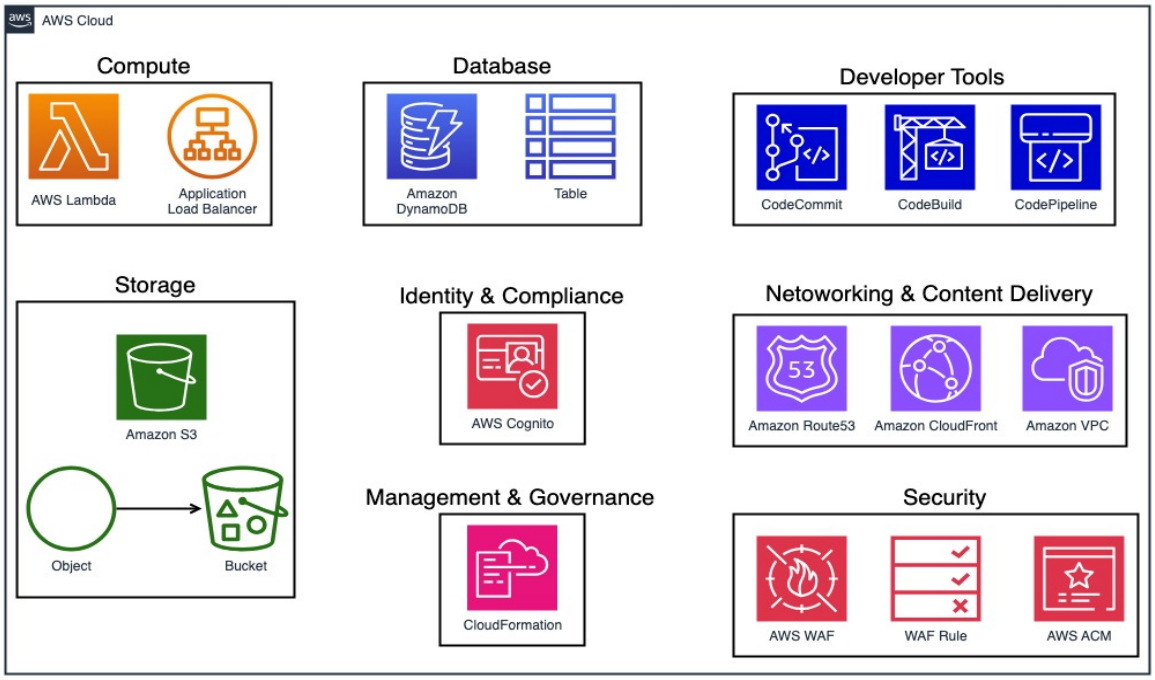

Figure 2.1: **List and Categories of used AWS Services**

2.2 Comparison with Alternatives

Table 2.3: Analysis with Alternative services

Services	**Alternatives**	**Description**
AWS Cognito	AWS IAM	AWS IAM (Identity and Access Management) is an alternative to Cognito for controlling access to AWS services. However, it doesn't support features like user registration or password reset.
AWS Lambda	AWS EC2	AWS EC2 (Elastic Compute Cloud) provides resizable compute capacity in the cloud and can be used as an alternative to Lambda for more complex and longer-running applications [16].
AWS S3	AWS EFS	AWS EFS (Elastic File System) provides simple, scalable file storage for use with Amazon EC2 instances. It could be used as an alternative to S3 for certain use cases [13].
AWS DynamoDB	AWS RDS	AWS RDS (Relational Database Service) provides an alternative to DynamoDB for applications that require a relational database structure [14].
AWS Application Load Balancer	AWS Network Load Balancer	AWS Network Load Balancer is an alternative to Application Load Balancer, providing high-throughput, low-latency load balancing for applications [8].
AWS Route 53	AWS CloudMap	AWS CloudMap can be an alternative to Route 53 for service discovery and DNS management in a microservices architecture.
AWS CloudFront	AWS Global Accelerator	AWS Global Accelerator is an alternative to CloudFront for improving the availability and performance of applications by using static IP addresses [6].
AWS ACM (Certificate Manager)	AWS Key Management Service (KMS)	AWS KMS can be used in conjunction with ACM for more control over key management and encryption of SSL/TLS certificates [7].
AWS WAF (Web Application Firewall)	AWS Shield	AWS Shield is an alternative to WAF, providing managed DDoS protection for web applications running on AWS [8].

2.2.1 Reason for using mentioned services over alternatives

In architecting the PizzaMan application on AWS, my choices of services were deliberate and tailored to the specific needs and characteristics of the application. Each selection aligns with the unique requirements, ensuring optimal performance, security, and scalability. My detailed justification for opting for the chosen AWS services over their alternatives is as follows:

- **AWS Cognito vs. AWS IAM:** I chose AWS Cognito over AWS IAM because Cognito is specifically designed for **user authentication and management** in web and mobile applications. It provides out-of-the-box functionalities for user registration, sign-in, and password reset, streamlining the implementation of secure user identity features [7]. While AWS IAM is robust for controlling access to AWS services, it lacks the user-centric features essential for PizzaMan's user authentication.

- **AWS Lambda vs. AWS EC2:** I chose AWS Lambda due to its serverless architecture and **on-demand** nature, allowing me to execute code in response to events without the need for managing servers. PizzaMan being a web application with varying workloads benefits from Lambda's automatic scaling and cost efficiency [4]. AWS EC2, while versatile, requires more management overhead and may not be as well-suited for the event-driven, scalable nature of PizzaMan.

- **AWS S3 vs. AWS EFS:** I selected Amazon S3 for its simplicity, scalability, and suitability for storing **static website content and images**. S3 excels in providing high **durability** and **availability** for objects, making it ideal for PizzaMan's static assets [5]. AWS EFS, while offering file storage for EC2 instances, is better suited for scenarios requiring shared file access among multiple instances, which is unnecessary for PizzaMan's use case.

- **AWS DynamoDB vs. AWS RDS:** I certainly believe that DynamoDB's **NoSQL, fully managed** database service perfectly aligns with PizzaMan's requirements for a **flexible, high-performance** database. Its seamless **scalability, low-latency** access, and automatic backups make it a superior choice for the application's data needs compared to the more structured AWS RDS, which is designed for relational databases.

- **AWS Application Load Balancer vs. AWS Network Load Balancer:** I chose AWS Application Load Balancer for its ability to distribute incoming application traffic across multiple targets and support for **content-based routing**. This is crucial for maintaining **high availability** and **fault tolerance** for PizzaMan [14]. AWS Network Load Balancer, while excelling in high-throughput scenarios, lacks the application-aware features that Application Load Balancer provides.

- **AWS Route 53 vs. AWS CloudMap:** I chose AWS Route 53 for its **domain registration** and **DNS routing** capabilities. Route 53 seamlessly integrates with other AWS services and provides the necessary functionality to ensure users can access PizzaMan **without disruptions** [10]. AWS CloudMap, while suitable for service discovery, doesn't offer the same breadth of domain management features as Route 53.

- **AWS CloudFront vs. AWS Global Accelerator:** I chose AWS CloudFront's content delivery network (CDN) capabilities to enhance the **availability** and **performance** of PizzaMan **globally**. Its ability to **cache and distribute content** from edge locations ensures **low-latency** access for users worldwide [12]. AWS Global Accelerator, although providing static IP addresses for improving availability, lacks the content delivery features essential for a responsive user experience in a web application.

- **AWS ACM vs. AWS KMS:** I selected AWS ACM for its simplicity and seamless integration with other AWS services. ACM automates the management of **SSL/TLS certificates**, ensuring secure communication [3]. While AWS KMS provides more control over key management, ACM aligns with the ease-of-use and automation principles crucial for PizzaMan's SSL certificate requirements.

- **AWS WAF vs. AWS Shield:** I chose AWS WAF to secure PizzaMan against **web exploits** and **malicious traffic**. WAF provides a specific application layer **protection** and **customization**, making it suitable for the unique security needs of a web application [13]. AWS Shield, while effective for DDoS protection, lacks the granular control over web application security offered by WAF.

3 Architecture

3.1 Final application architecture

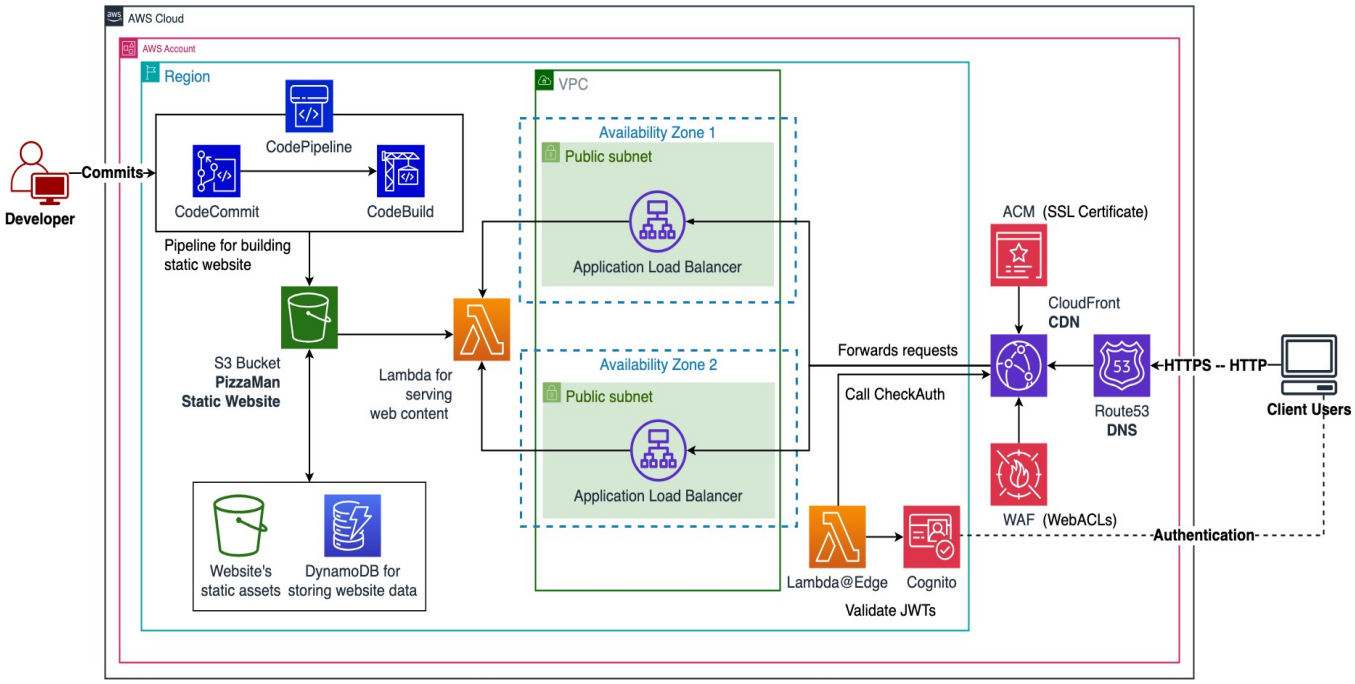

Figure 3.1: **Cloud Architecture Diagram for PizzaMan application**

This section elaborates on the architecture of the PizzaMan application, explaining why each chosen service is the best fit, taking into account factors such as cost, performance, security, and scalability.

Source Code Management with AWS CodeCommit: I opted for AWS CodeCommit as my source code repository because it seamlessly integrates with the AWS ecosystem. This choice aligns with my goal of building a **scalable** and **secure development pipeline** for the PizzaMan application. CodeCommit provides a fully managed, secure, and cost-effective version control system, ensuring smooth collaboration among developers [9]. So, the developer commits his new code to this CodeCommit repository. This new commit will trigger the pipeline.

Continuous Integration and Deployment with AWS CodePipeline and CodeBuild: AWS CodePipeline and CodeBuild form the core of my CI/CD pipeline, **automating the deployment process** whenever changes are committed to CodeCommit. CodePipeline orchestrates the workflow, and CodeBuild efficiently builds and packages the static website and stores it in an S3 Bucket [9]. This setup ensures rapid and reliable delivery of new features and updates to PizzaMan while minimizing manual interventions.

Static Website Hosting and Static Image Storage with Amazon S3: Amazon S3 emerged as the storage solution for static website content and images due to its **simplicity, cost-effectiveness, and scalability**. S3 provides **low-latency** access to objects, optimizing the retrieval of static content. Leveraging separate S3 buckets for the website and images allows for efficient management and scaling, catering to the diverse content needs of PizzaMan. The **pay-as-you-go** model ensures cost efficiency while guaranteeing reliable and high-performance content delivery [2]. So, I used an S3 bucket for static website hosting for this PizzaMan Application. I used a separate S3 bucket for static image storage allows for optimized **performance** and better organization. This decision enhances the efficiency of content delivery to end-users, providing a seamless and visually appealing experience [2].

Data Storage with AWS DynamoDB: I chose DynamoDB as the database service for its seamless scalability, low-latency performance, and fully managed nature. The NoSQL data model suits PizzaMan's requirements, allowing for flexible and rapid development. The serverless nature of DynamoDB ensures **optimal cost management**, as I only pay for the resources consumed during database operations [6]. This approach aligns with the application's need for a dynamic and responsive data storage solution. I used DynamoDB tables for storing the PizzaMan application user data, pizza (product) data and past orders data.

Domain Management with AWS Route 53: Route 53 serves as the domain registration and DNS management service, ensuring a reliable and scalable method for users to access PizzaMan. Its global *anycast routing* optimizes DNS queries, enhancing the application's **availability**. The integration with other AWS services, such as CloudFront, facilitates seamless DNS updates [10]. Route 53's pay-as-you-go pricing model aligns with cost-effectiveness, making it the ideal choice for domain and DNS management. This route 53 endpoint will be the entry request point for my PizzaMan application.

Content Delivery and Security with AWS CloudFront, WAF, and ACM: CloudFront, as a **content delivery network (CDN)**, accelerates the delivery of PizzaMan's content to users worldwide. Leveraging **edge locations reduces latency**, ensuring a responsive user experience. Enabling WAF (Web Application Firewall) enhances security by **protecting** against common **web exploits**. The integration with ACM (Certificate Manager) ensures SSL/TLS certificates for secure and **encrypted communication** [11]. CloudFront's scalability, performance, and security features make it the optimal choice for content delivery.

Authentication with Cognito and Lambda@Edge: Lambda@Edge plays a pivotal role in user authentication at the edge locations of CloudFront. This serverless compute service executes code in response to CloudFront events, allowing for seamless and secure **user authentication**. Integrating with AWS Cognito ensures robust user identity management. The serverless architecture eliminates the need for additional infrastructure management, contributing to **cost efficiency** and **optimal performance** [7]. AWS Cognito provides a comprehensive solution for user authentication and authorization. Its user-friendly features, such as user registration, login, and password recovery, align perfectly with PizzaMan's requirements. The integration with Lambda@Edge at CloudFront enhances security by ensuring user **authentication at the edge locations**. Cognito's scalability and managed services reduce the operational overhead, making it an ideal choice for user identity management [7].

Load Balancing and Serverless Compute with AWS Application Load Balancer and Lambda: ALBs, deployed in a VPC with two availability zones, efficiently route user requests to the *PizzaLambda* function. The two-zone deployment enhances **fault tolerance and availability**. ALBs intelligently distribute incoming traffic, ensuring optimal performance and **scalability** [14]. This setup aligns with PizzaMan's need for a robust and scalable routing mechanism, catering to varying user loads.

Lambda Function (PizzaLambda) for Serving Web Content: The PizzaLambda function serves as the backend logic for handling user requests. Triggered by ALBs, it performs operations like fetching objects from the S3 bucket hosting the static website [4, 14]. The serverless nature of Lambda ensures **cost-effective execution**, scaling based **on demand**. Its integration with other AWS services makes it a suitable choice for executing backend functions in response to user requests.

VPC for Networking: Utilizing a Virtual Private Cloud (VPC) with two availability zones ensures **high availability** and **fault tolerance** for PizzaMan. This network isolation enhances security, and the **multi-AZ deployment** mitigates the risk of infrastructure failures, contributing to a **resilient** architecture [2].

3.2 Assessing my architecture against several factors

The detailed considerations for each factor are cost, performance, security, and scalability against my PizzaMan Application architecture are as follows;

Cost Consideration:

- **AWS CodeCommit, CodePipeline, and CodeBuild:** These services offer a pay-as-you-go model, ensuring that I only incur costs when actively using the pipeline, making it a **cost-effective choice** for my CI/CD needs [9].

- **Amazon S3:** With a **tiered storage pricing model**, S3 optimizes costs by charging based on actual usage. This aligns perfectly with my goal of **minimizing expenses** while hosting the static website and images [5].
- **Amazon DynamoDB:** DynamoDB's serverless model means I only pay for the read and write capacity I consume. This ensures that I'm **not over-provisioning** resources, leading to cost efficiency [6].
- **AWS Lambda:** Lambda's serverless nature allows me to execute code without the need to provision or manage servers actively. I only pay for the compute time consumed during execution, contributing to significant cost savings.

Performance Consideration:

- **Amazon CloudFront:** With its global edge locations, CloudFront significantly **improves content delivery speed** by caching PizzaMan's static assets closer to end-users worldwide [12].
- **Amazon DynamoDB:** DynamoDB's fast and predictable performance ensures quick access to data, supporting responsive and **low-latency** interactions within PizzaMan.
- **AWS Lambda:** As PizzaMan's traffic grows, Lambda's **auto-scaling** capabilities provide optimal performance, effortlessly handling varying workloads.

Security Consideration:

- **AWS WAF:** Integrated with CloudFront, WAF protects PizzaMan from **web-based attacks** like DDoS, enhancing the overall security posture.
- **AWS ACM:** ACM ensures secure communication between users and PizzaMan by providing SSL/TLS certificates, **encrypting data** in transit and bolstering user trust.
- **AWS Cognito:** Cognito's robust authentication capabilities contribute to PizzaMan's security by managing user identities securely.

Scalability Consideration:

- **Amazon DynamoDB:** DynamoDB scales effortlessly with the growing demands of PizzaMan. Its auto-scaling feature dynamically adjusts capacity to handle increased workloads.
- **AWS Lambda:** Being a serverless compute service, Lambda scales automatically in response to the number of incoming requests, ensuring seamless scalability.
- **AWS Application Load Balancer:** ALB enables **horizontal scalability** by distributing incoming traffic across multiple Lambda instances, enhancing PizzaMan's overall capacity.

In summary, my PizzaMan Application architecture strikes a balance between cost efficiency, high performance, robust security measures, and seamless scalability. By leveraging the strengths of each chosen AWS service, I've tailored the architecture to meet the specific needs of PizzaMan, ensuring a reliable, secure, and cost-effective solution.

4 Architecting Principles and Best Practices

4.1 AWS Well-Architected Framework

This section breaks down the elaborate explanation pillar-wise, focusing on how the PizzaMan Application architecture aligns with the AWS Well-Architected Framework principles.

Operational Excellence: In line with this pillar, I designed my architecture for **frequent updates**. Also, I can make *frequent, small and reversible changes.*

- **Automated Source Code Management:** Utilizing AWS CodeCommit for source code management aligns with operational excellence by providing a fully managed and **scalable** solution. This choice ensures that version control is **automated**, fostering collaboration among developers [15].
- **Continuous Integration and Deployment:** AWS CodePipeline and CodeBuild contribute to operational excellence by automating the continuous integration and deployment process. This **minimizes manual intervention** [15], reduces deployment errors, and **accelerates the release cycle.**
- **Monitoring and Logging:** The architecture can further enhance operational excellence by incorporating AWS services like CloudWatch for **monitoring** and **logging** [15]. This would provide real-time insights into application performance, facilitating **proactive issue resolution**.
- **CloudFormation:** I also used AWS CloudFormation for **automated resource provisioning** to reduce the deployment lead time. I used CloudFormation for the creation of some of the services in my architecture.

Security: When it comes to security, I've used IAM policies to ensure the **principle of least privilege** to grant the least access to resources like S3 Buckets. Also, I tried to apply **security at all layers** of my application using services like WAF, ACM, Cognito, and VPC.

- **Isolated Storage for Images:** I decided to store static images in a separate S3 bucket because it will enhance security. It follows the principle of least privilege, minimizing the attack surface and potential risks. This segregation ensures that even if one part of the system is compromised, the others remain secure.
- **DNS Management with Route 53:** Leveraging AWS Route 53 for domain management ensures **secure and reliable DNS**. This aligns with the security pillar by providing a resilient and secure mechanism for routing users to the application.
- **Edge Authentication with Cognito and Lambda@Edge:** The architecture incorporates security best practices by leveraging Cognito for **user authentication**. Lambda@Edge ensures that **authentication** is performed at the **edge locations**, reducing latency and enhancing security [15].
- **Content Delivery and Protection:** Combining **CloudFront, WAF, and ACM** demonstrates a security-centric approach to content delivery. CloudFront's secure and fast content delivery, coupled with WAF's **protection against web exploits**, contributes to a robust security posture.

Reliability: To stay in line with the reliability pillar, I've used services like application load balancer **(ALB)** in two availability zones in a VPC to ensure **high availability** of the application [15]. Moreover, I've used DynamoDB and S3 buckets for improved durability in terms of storing application data.

- **Durable Static Website Hosting:** Amazon S3 is chosen for static website hosting due to its **durability**. This aligns with the reliability pillar, ensuring that the PizzaMan Application's web content is stored in a **highly available** and **resilient** manner.
- **Scalable Data Storage with DynamoDB:** Utilizing DynamoDB for data storage enhances reliability by providing a serverless, scalable, and highly available database solution. This ensures **consistent performance** even during periods of increased demand.

- **Serverless Compute with Lambda:** The decision to use Lambda for serverless compute aligns with reliability principles [15]. It ensures that compute resources are efficiently **scaled based on demand**, contributing to a reliable and responsive application.

Performance Efficiency: My architecture for the PizzaMan Application demonstrates strong alignment with the performance efficiency pillar of the AWS Well-Architected Framework by leveraging **caching, scalability**, **global reach**, and cost optimization to deliver a high-performance and cost-effective user experience.

- **Optimized Content Delivery:** CloudFront, in conjunction with WAF and ACM, contributes to performance efficiency. It optimizes content delivery, reduces latency, and ensures a fast and responsive user experience.
- **Efficient Load Balancing with ALB:** AWS Application Load Balancer optimizes load distribution, enhancing performance efficiency. It ensures that incoming requests are efficiently distributed across multiple targets, improving overall application responsiveness [15].
- **Global Reach:** CloudFront's global network of edge locations ensures that content is delivered with low latency to users around the world, improving the application's performance and user experience.

Cost Optimization: The architecture of my PizzaMan Application demonstrates strong alignment with the cost optimization pillar of the AWS Well-Architected Framework by leveraging serverless computing, managed services, optimized data storage, global content delivery, and granular cost attribution and analysis to achieve cost-effective and efficient operation.

- **Serverless Compute Model with Lambda:** Embracing the serverless compute model with Lambda aligns with cost optimization principles. It eliminates the need for provisioning and managing servers, ensuring cost efficiency based on actual usage.
- **Pay-as-You-Go Hosting with S3:** AWS S3's pay-as-you-go model for static website hosting contributes to cost optimization [15]. There are no upfront costs, and you only pay for the storage and data transfer used, aligning with efficient cost management.

Sustainability: The PizzaMan Application Architecture exhibits a commitment to sustainability by leveraging serverless computing, embracing a pay-as-you-go model, optimizing global content delivery, and reducing operational overhead through managed services. These decisions collectively contribute to a more sustainable and environmentally conscious hosting solution on AWS.

- **Serverless and Autoscaling:** The PizzaMan Application embraces a serverless architecture, particularly with AWS Lambda. This decision aligns with sustainability principles as serverless computing eliminates the need for maintaining and operating servers continuously. Resources are provisioned and scaled automatically based on demand, leading to optimal resource utilization and reduced energy consumption.
- **Pay-as-You-Go Model:** Leveraging AWS services like S3, Lambda, and DynamoDB with a pay-as-you-go model contributes to sustainability. The application only consumes resources based on actual usage, avoiding the inefficiencies associated with traditional infrastructure where resources might remain idle [15].
- **Global Content Delivery with CloudFront:** The use of CloudFront for global content delivery aligns with sustainability principles by optimizing the delivery of content to users. By utilizing edge locations worldwide, CloudFront minimizes latency and reduces the need for users to access centralized servers, resulting in a more energy-efficient content delivery process [15].
- **Reduced Overhead with Managed Services:** Preferring managed services like AWS CodeCommit, CodePipeline, CodeBuild, S3, DynamoDB, and others reduces operational overhead. Managed services handle routine tasks such as database management, code deployment, and storage maintenance. This not only enhances operational efficiency but also contributes to sustainability by minimizing the human effort required for routine operational tasks.

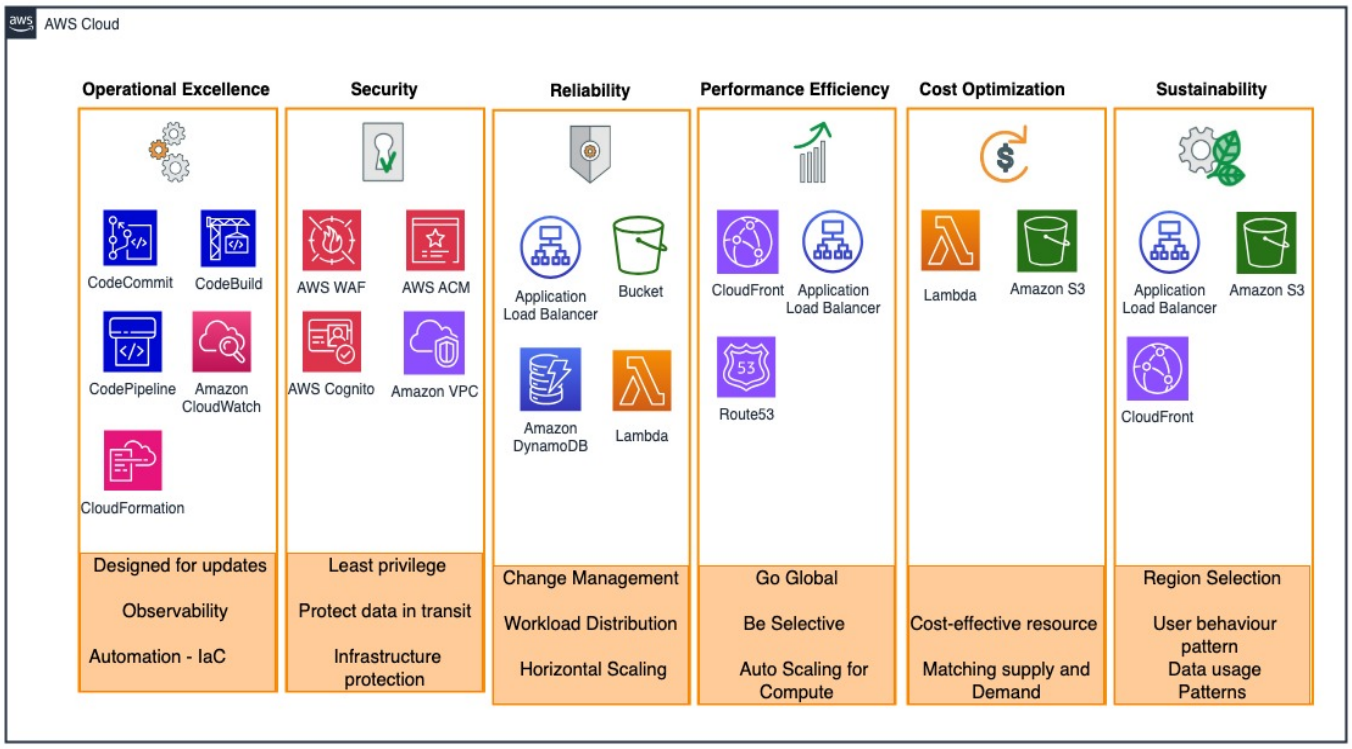

Figure 4.1: **My Architecture alignment with the AWS Well-Architected Principles**

Figure 4.1 shows how the AWS services I used in my architecture align with the AWS Well-architected framework. In **operational excellence** pillar, I included services like CodeCommit, CodeBuild, CodePipeline, CloudWatch and CloudFormation. I designed my system to **adapt** any number of **frequent and reversible updates.** Also, my architecture showcases improved observability (Monitoring and Logging) [15]. By using CloudFormation, I leverage the power of automating the infrastructure provisioning process **(Infrastructure-as-Code)** [15].

In the Security pillar, I have included services like AWS WAF, ACM, Cognito and VPC. By using these services, I am ensuring the application of **security at all layers**. Also, I followed the **principle of least privilege** and tried to protect **data in transit** [15] using an SSL certificate from ACM (HTTPS enabled). Finally, by using VPC, I am protecting my infrastructure in an isolated network.

Considering the reliability pillar, I have designed my architecture in a way that it can adapt to any range of workloads **(workload distribution)**, dynamic data storage class changes **(change management)** and **horizontal scaling** to adapt to the workload changes. In the performance efficiency pillar, by using CloudFront I leveraged the advantages of global infrastructure (edge locations to reduce the latency and improve the performance). I was very selective with the usage of the custom domain for my Route53 hosted zone for user convenience.

In the cost optimization pillar, I am using the most **cost-effective** [15] resources like lambda functions and S3 buckets. Also, I am following the **pay-as-you-go** model to stay in line with my budget. In the sustainability pillar, I am very cautious about preventing the wastage of resources or unnecessary utilization of resources. To maintain and integrate environmental sustainability with my architecture, I select regions that are close to my end users (to avoid latency issues) and analyze user behaviour and data usage patterns to effectively provision the appropriate resources at the right time to avoid underutilization or overutilization of resources.

In summary, the PizzaMan Application architecture is intricately aligned with the AWS Well-Architected Framework across all pillars. From operational excellence to sustainability, each architectural decision is driven by well-established AWS best practices and principles. This ensures that the PizzaMan Application not only meets its functional requirements but also exhibits a robust, secure, and cost-effective hosting solution on AWS.

5 Demonstration

5.1 Application Screenshots

- Figure 5.1 shows the main entry page of the application i.e.: The home Page of the PizzaMan Application. Figure 5.2 shows the menu page of the application that contains all the pizzas present in the store.
- Figure 5.3 shows the login page of the application that will allow the user to securely log into the application. Figure 5.4 shows the register page of the application that will allow the user to securely register the application.
- Figure 5.5 shows the user entering new registration credentials like email and password. Figure 5.6 shows the received verification email to allow the user to verify the email for a successful registration process.
- Figure 5.7 shows the confirmed response indicating that the user email is verified successfully. Figure 5.8 shows the user entering the login credentials to log into the application.
- Figure 5.9 shows the home screen after a successful user login. Figure 5.10 shows the user is adding several items to the cart.
- Figure 5.11 shows the total price of the cart items. Figure 5.12 shows the option to proceed to checkout to complete the order.
- Figure 5.13 shows that the address is not found and allows the user to add a new one. Figure 5.14 shows the user is adding a new address by filling out all the fields.
- Figure 5.15 shows that the user-entered address is added successfully. Figure 5.16 shows the user choosing the mode of payment as Cash on Delivery, the default option. Figure 5.17 shows that the order was placed successfully.

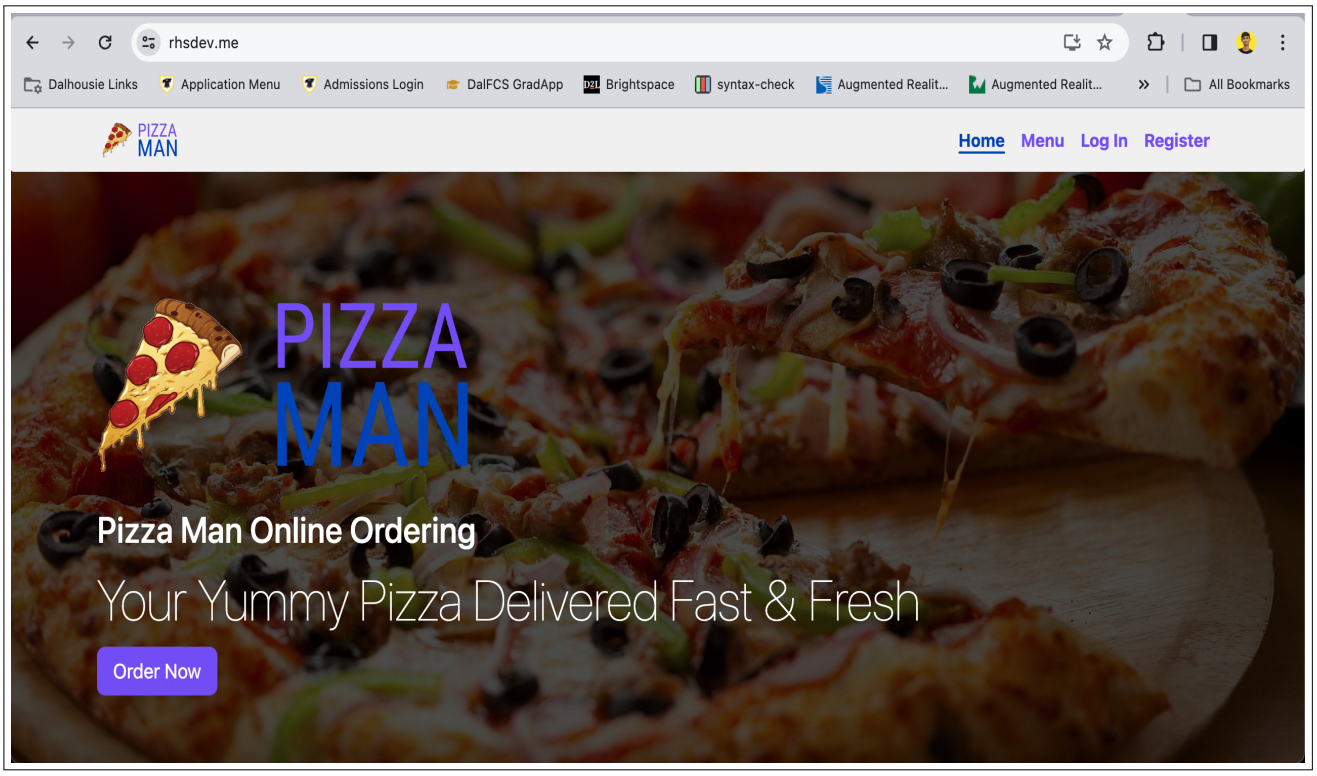

Figure 5.1: **Home Page of the PizzaMan Application**

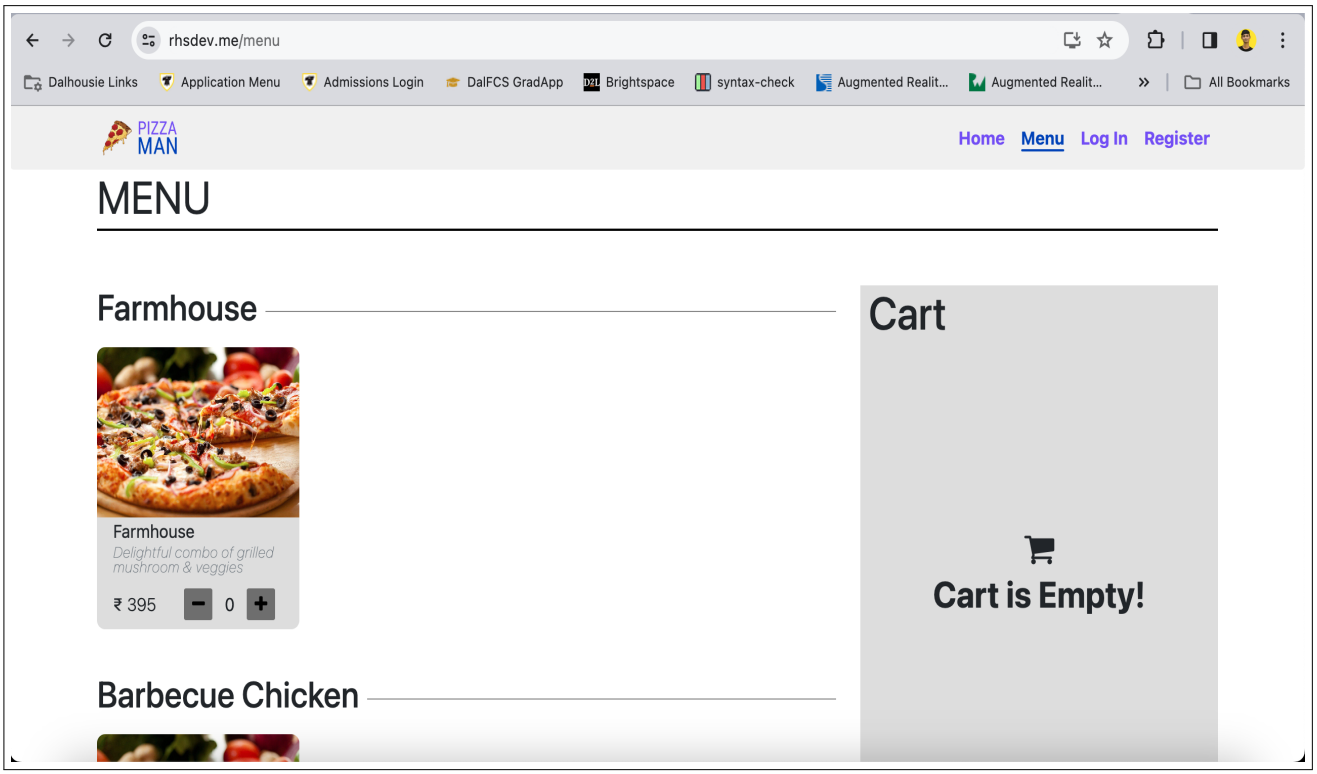

Figure 5.2: **Menu Page of the PizzaMan Application**

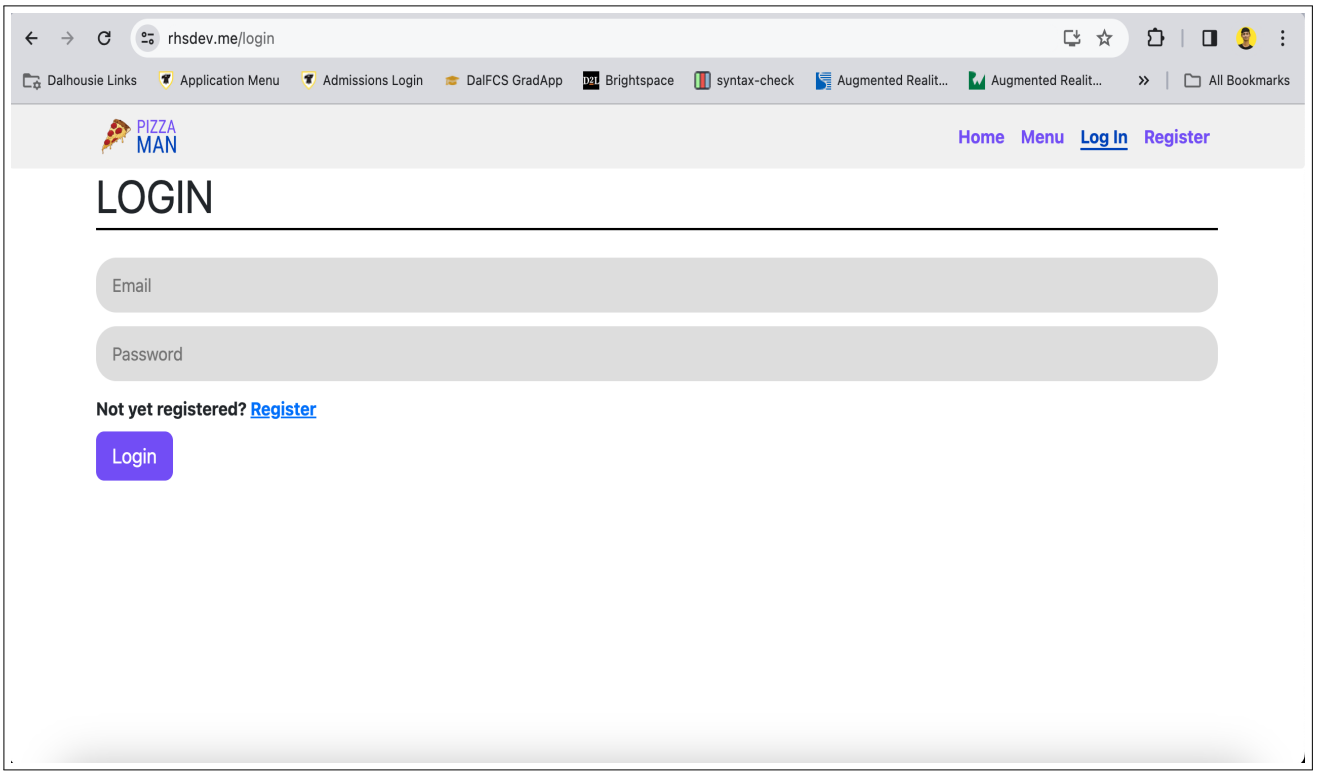

Figure 5.3: **Login Page of the PizzaMan Application**

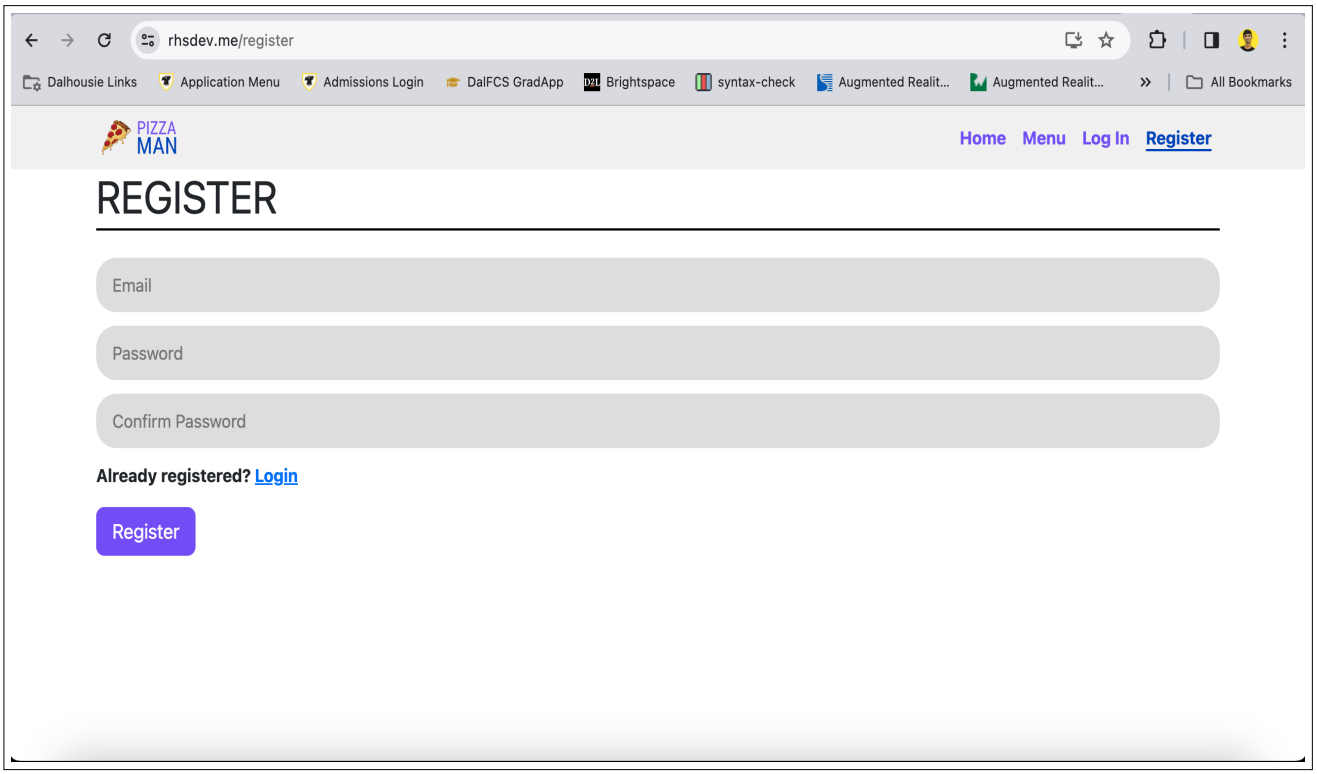

Figure 5.4: **Register Page of the PizzaMan Application**

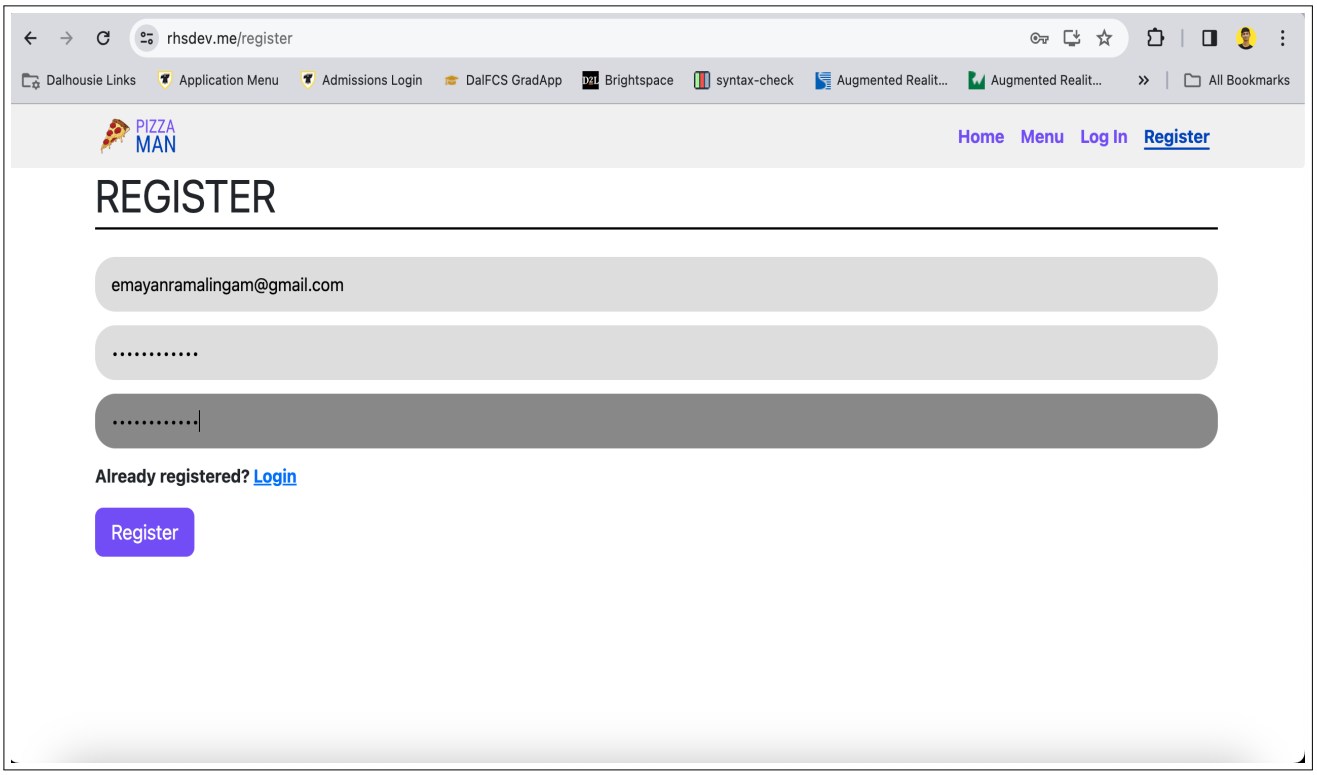

Figure 5.5: **New User Registration**

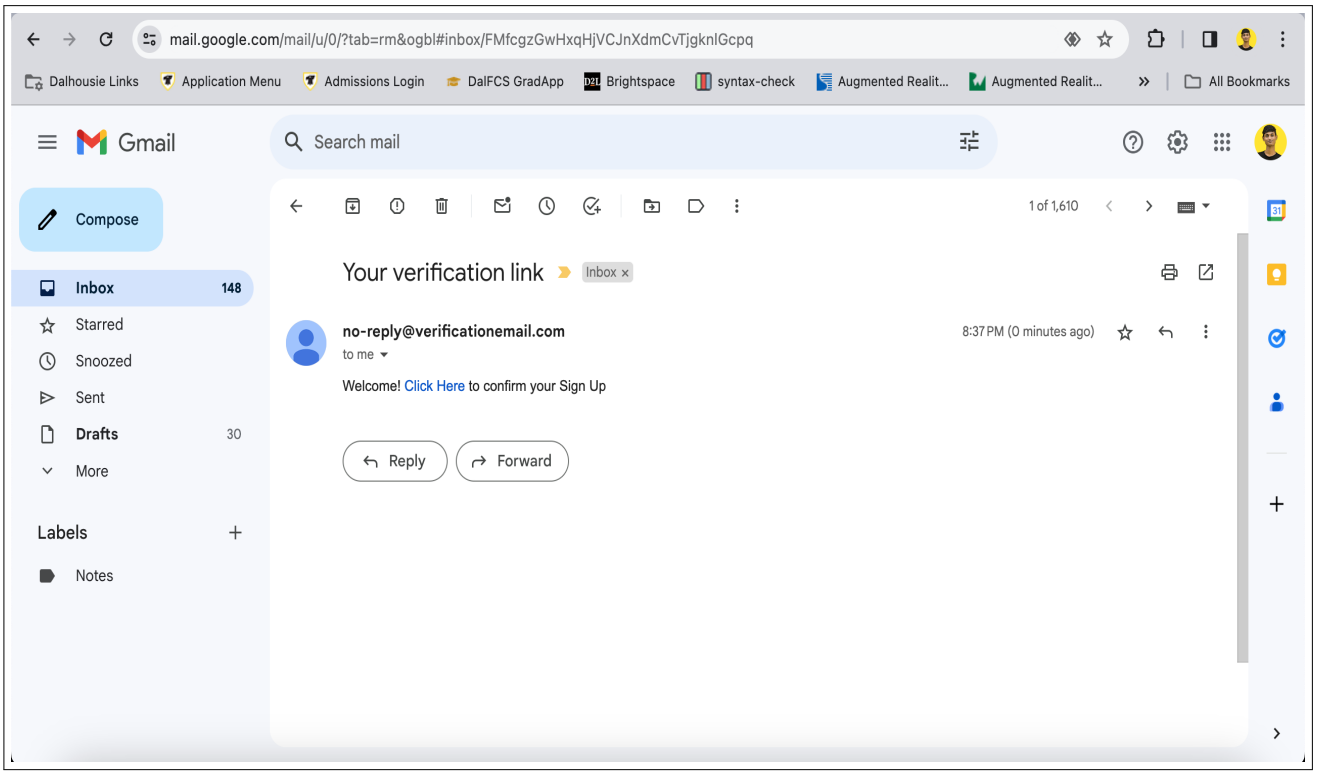

Figure 5.6: **Email Verification Mail**

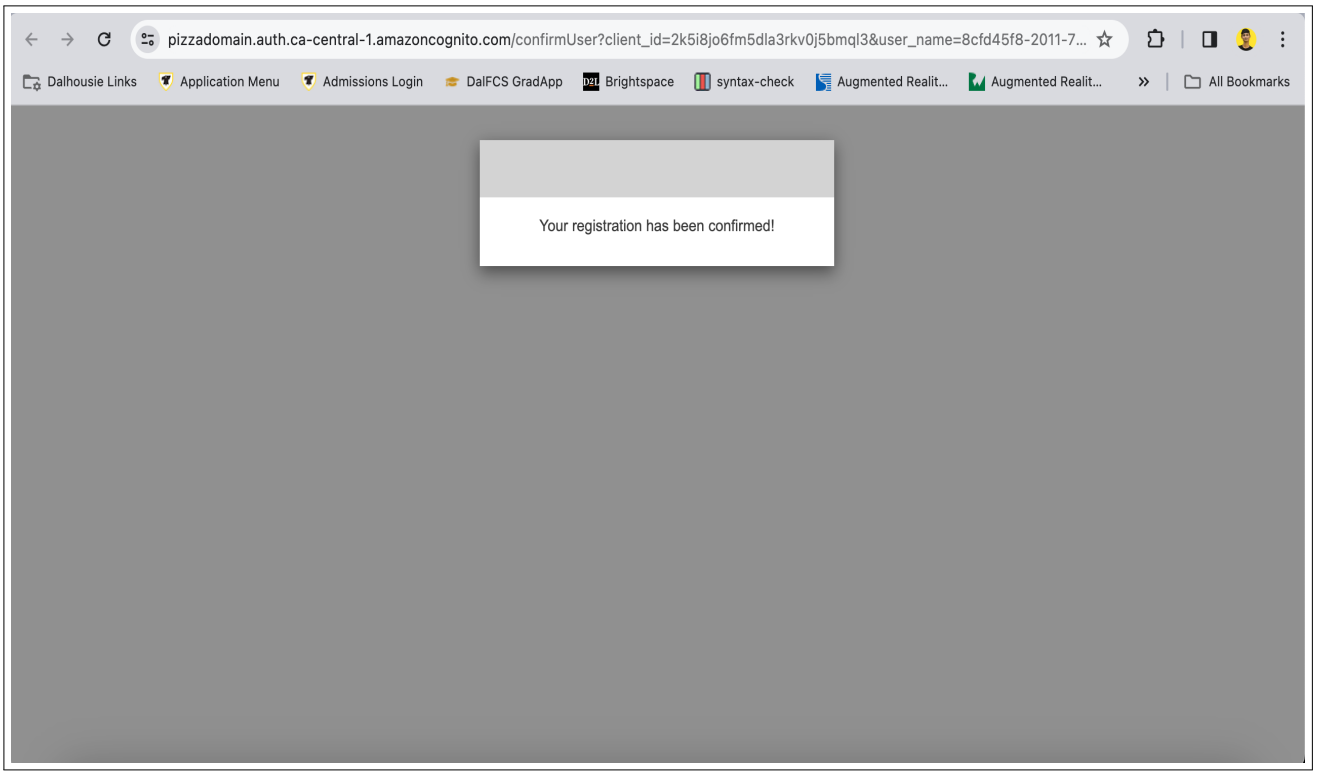

Figure 5.7: **Registration Confirmed**

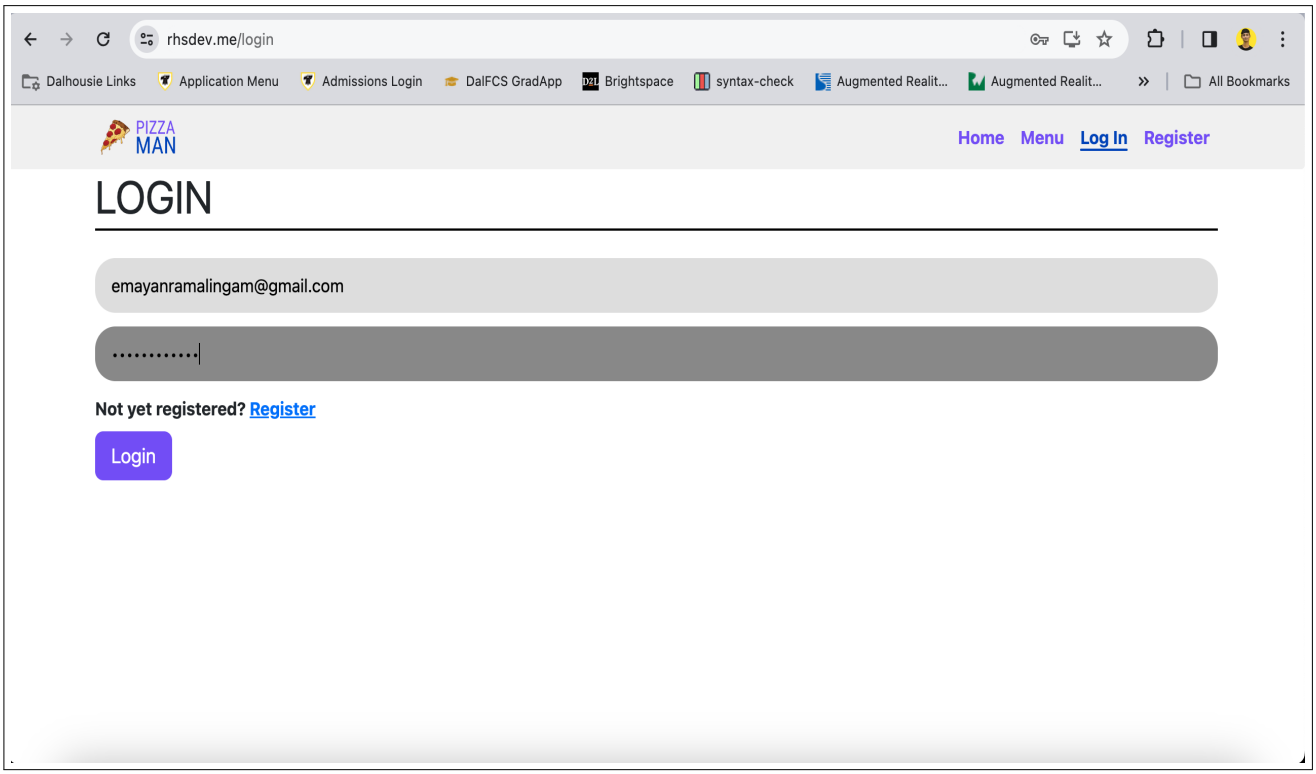

Figure 5.8: **User Login**

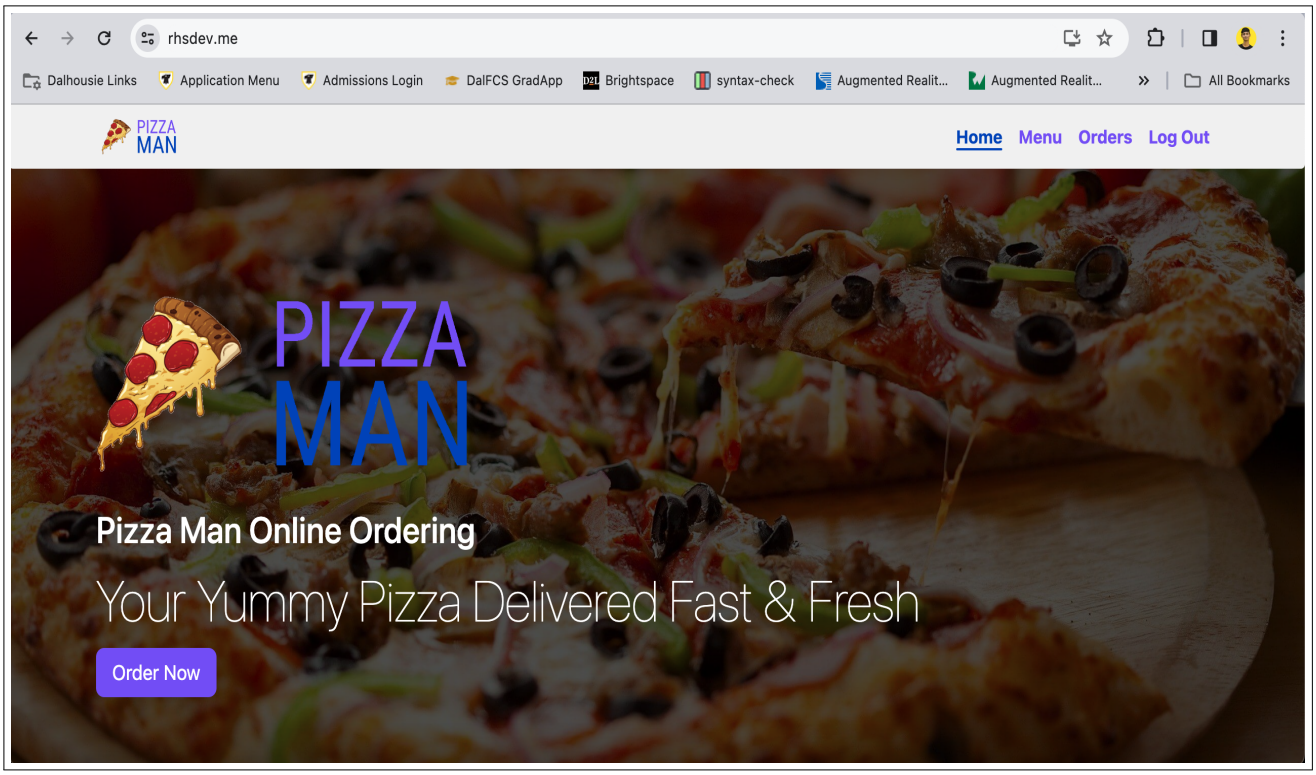

Figure 5.9: **Logged In Successfully**

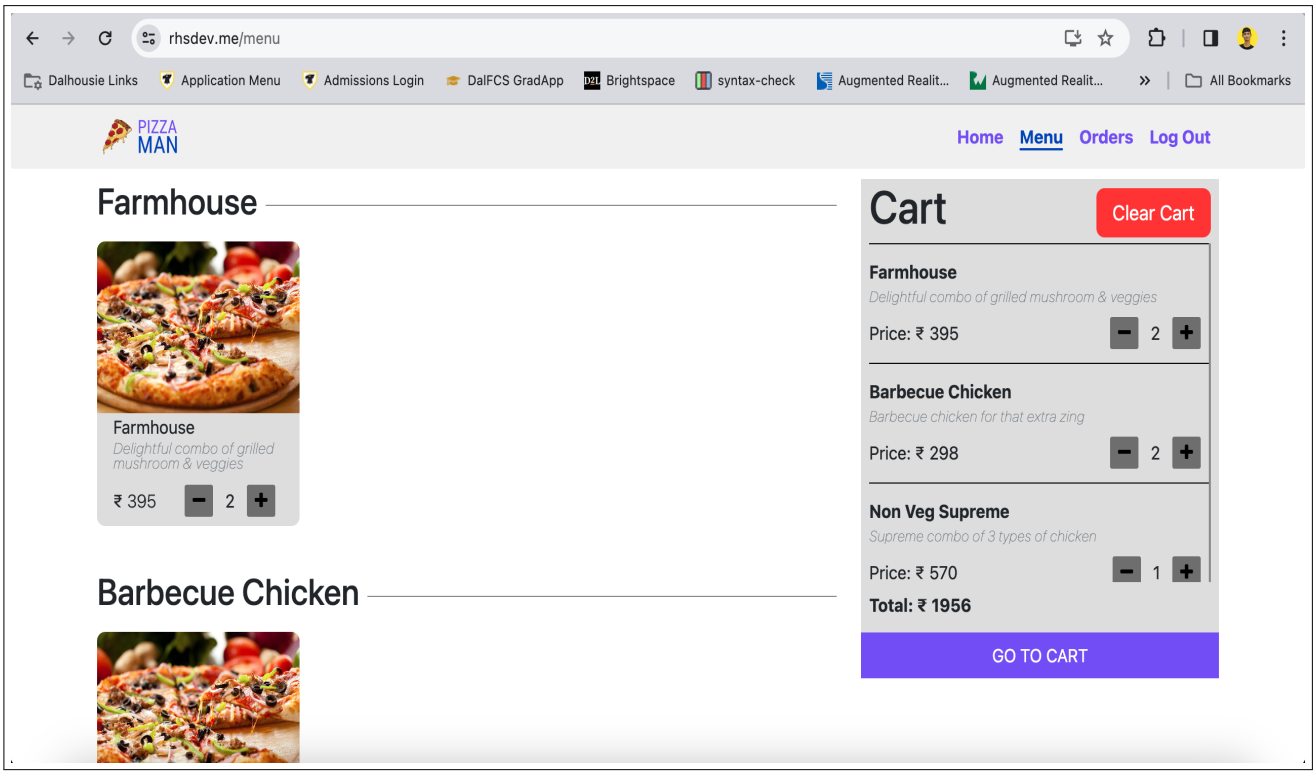

Figure 5.10: **Adding Items into the Cart**

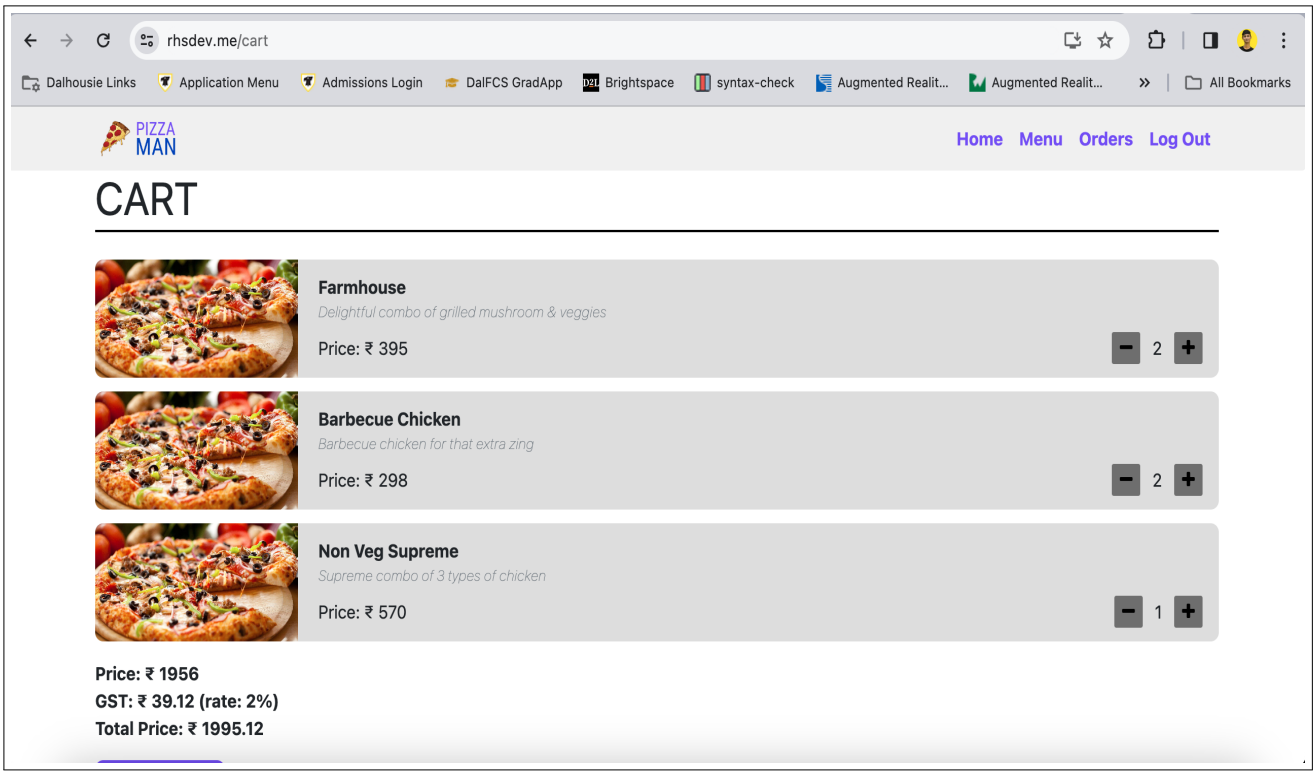

Figure 5.11: **Total amount for the final order plus tax**

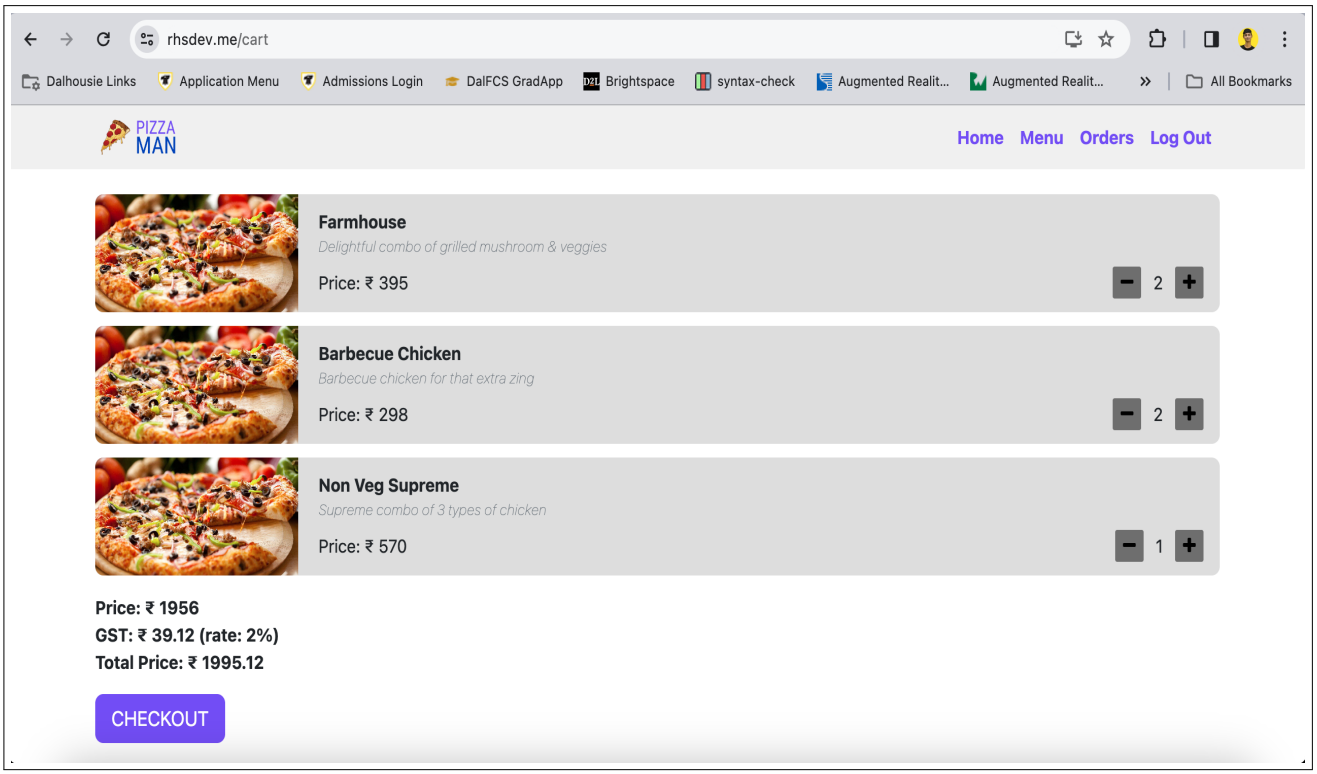

Figure 5.12: **Proceeding to Checkout**

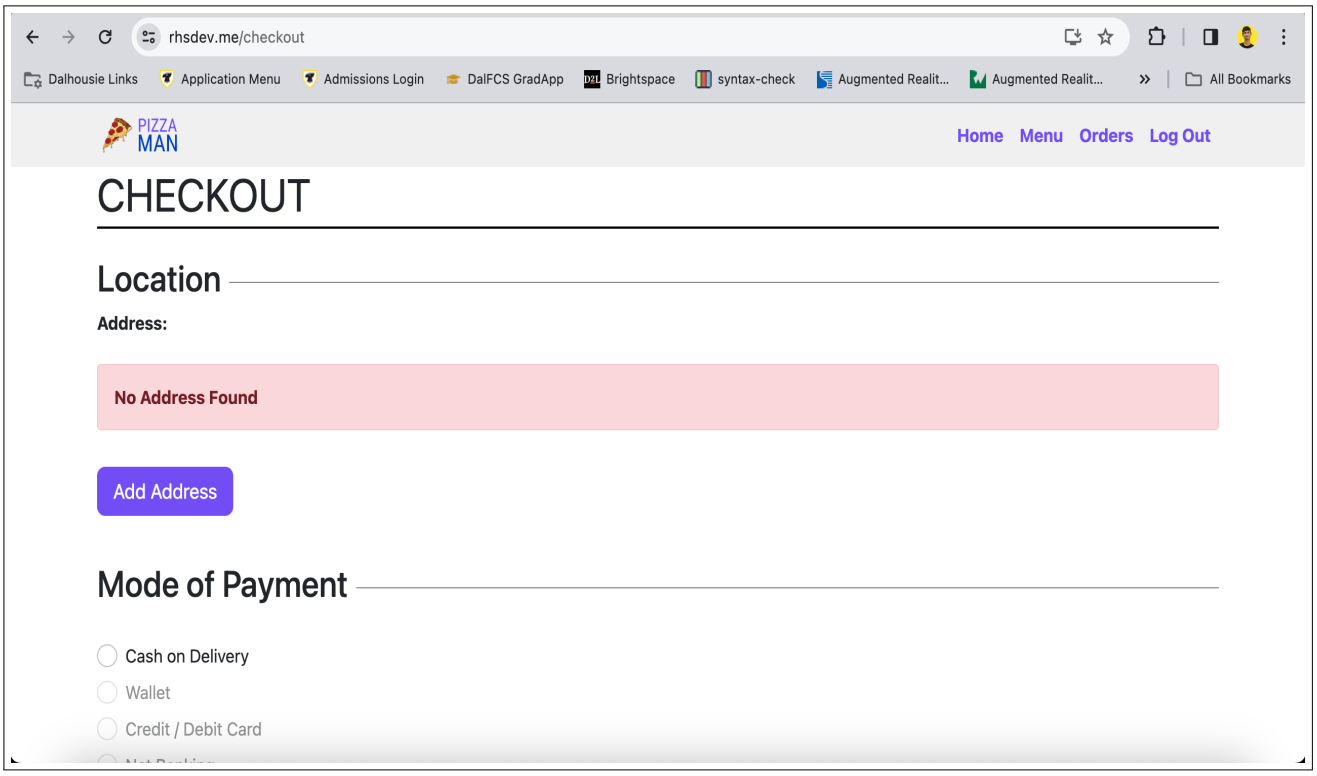

Figure 5.13: **Checkout Page - No Address found for this user**

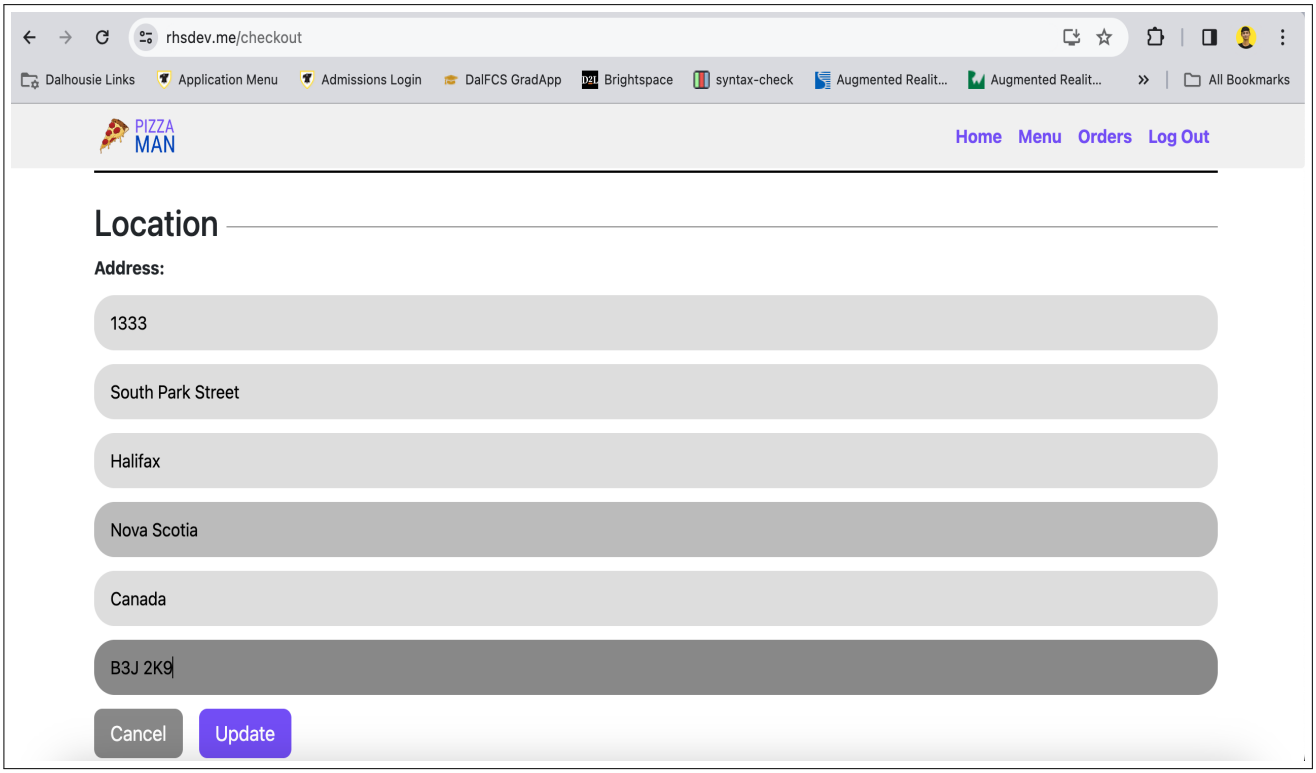

Figure 5.14: **Adding new address for the user**

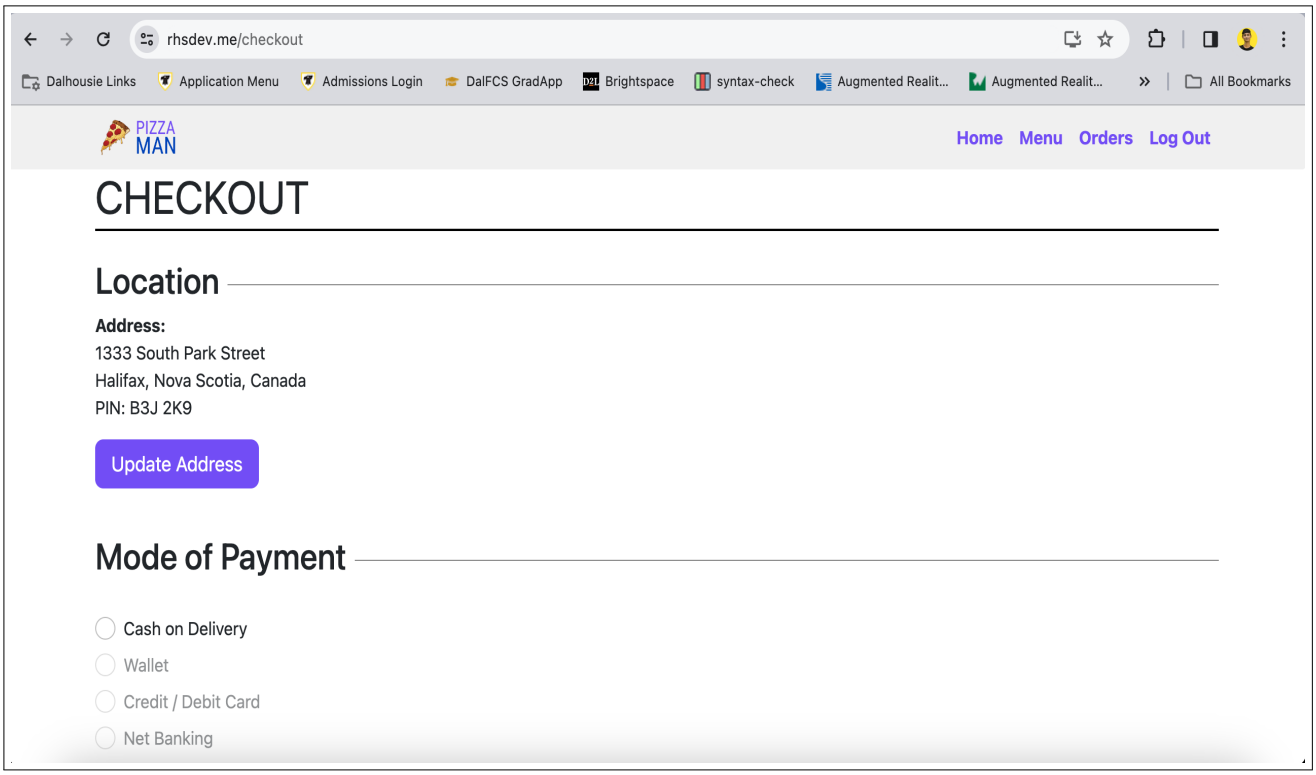

Figure 5.15: **Added address successfully**

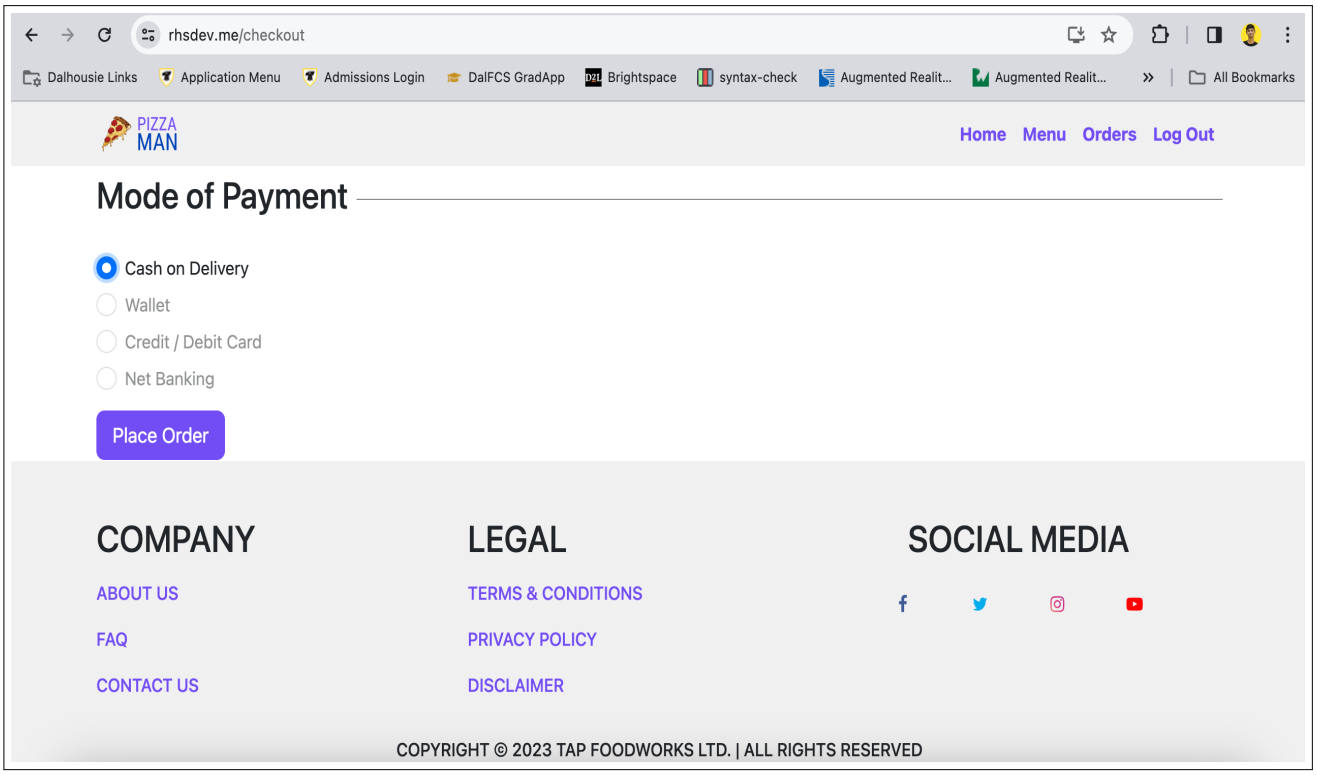

Figure 5.16: **Choosing mode of payment**

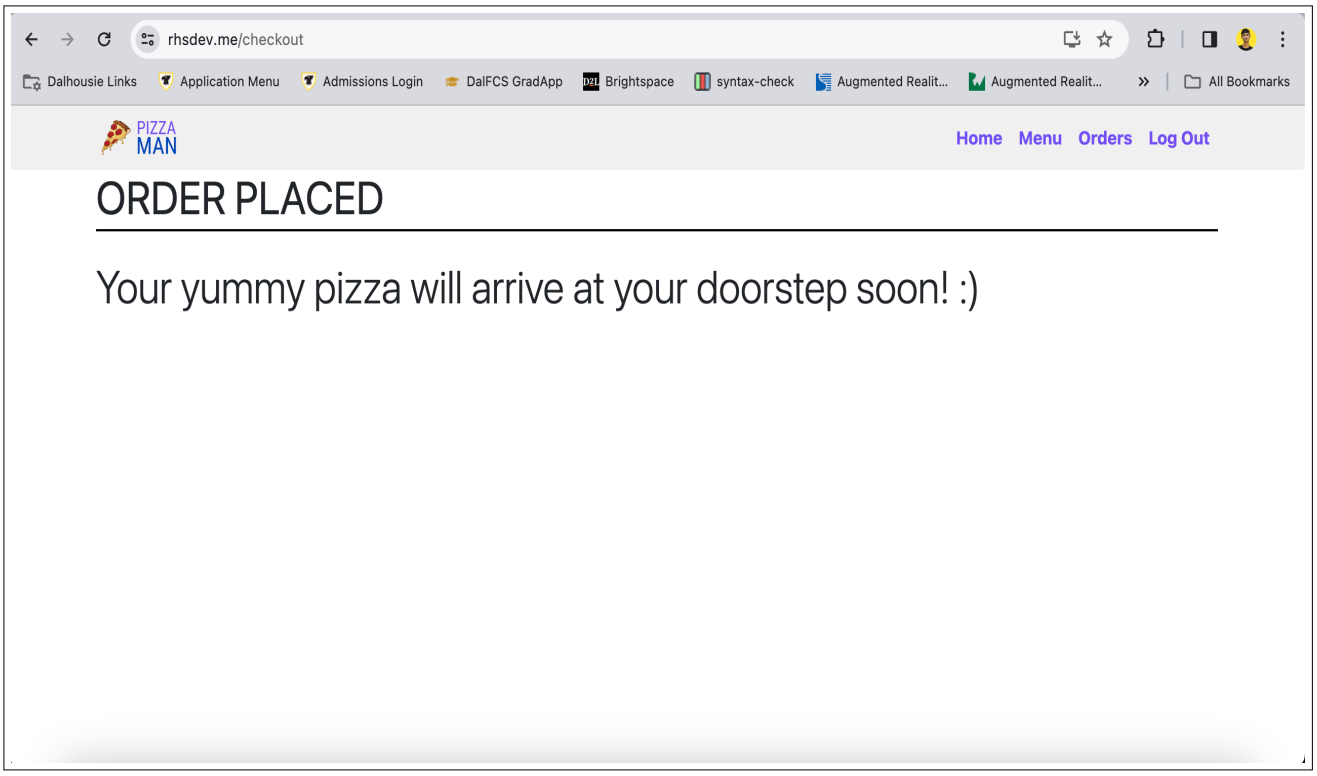

Figure 5.17: **Order placed successfully**

References

[1] Amazon Web Services, "An AWS Cloud architecture for web hosting," AWS Whitepaper, Available: https://docs.aws.amazon.com/whitepapers/latest/web-application-hosting-best-practices/an-aws-cloud-architecture-for-web-hosting.html. [Accessed: December 03, 2023].

[2] Amazon Web Services, "Serve static content in an Amazon S3 bucket through a VPC by using Amazon CloudFront," AWS Patterns, Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/serve-static-content-in-an-amazon-s3-bucket-through-a-vpc-by-using-amazon-cloudfront.html. [Accessed: December 03, 2023].

[3] Amazon Web Services, "AWS CertificateManager (ACM)," User Guide, Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-certificatemanager-certificate.html. [Accessed: December 03, 2023].

[4] Amazon Web Services, "What is AWS Lambda?," AWS Developer Guide, Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. [Accessed: December 03, 2023].

[5] Amazon Web Services, "What is Amazon S3?," AWS Developer Guide, Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. [Accessed: December 03, 2023].

[6] Amazon Web Services, "What is Amazon DynamoDB?," AWS Developer Guide, Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html. [Accessed: December 03, 2023].

[7] Amazon Web Services, "What is Amazon Cognito?," AWS Developer Guide, Available: https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html. [Accessed: December 03, 2023].

[8] Amazon Web Services, "Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider," AWS User guide, Available: https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-s3deploy.html. [Accessed: December 03, 2023].

[9] Amazon Web Services, "Complete CI/CD with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline," Amazon VPC, Available: https://aws.amazon.com/blogs/devops/complete-ci-cd-with-aws-codecommit-aws-codebuild-aws-codedeploy-and-aws-codepipeline/. [Accessed: December 03, 2023].

[10] Amazon Web Services, "What is Amazon Route 53?," AWS Developer Guide, Available: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/Welcome.html. [Accessed: December 03, 2023].

[11] Amazon Web Services, "Deploy a React-based single-page application to Amazon S3 and CloudFront," AWS Patterns, Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/deploy-a-react-based-single-page-application-to-amazon-s3-and-cloudfront.html. [Accessed: December 03, 2023].

[12] Amazon Web Services, "Use an Amazon CloudFront distribution to serve a static website," AWS Developer Guide, Available: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/getting-started-cloudfront-overview.html. [Accessed: December 03, 2023].

[13] Amazon Web Services, "AWS WAF," AWS Developer Guide, Available: https://docs.aws.amazon.com/waf/latest/developerguide/waf-chapter.html. [Accessed: December 03, 2023].

[14] Amazon Web Services, "Application Load Balancers," AWS Application Load Balancers, Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/application-load-balancers.html. [Accessed: December 03, 2023].

[15] Amazon Web Services, "AWS Well-Architected Framework," User Guide, Available: https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html. [Accessed: December 03, 2023].