A project submitted in partial fulfilment of the award of the degree of BSc (Hons) Digital Technology and Solutions (Software Engineering Specialism).

A Synoptic Project

Alpaca Cafe

A Hospitality Ordering System for a Multi-Store Coffee Shop

S240258 | University of Suffolk | June 2025

Project Url: coffee.leon-skinner.dev
GitHub Url: https://github.com/TidalCub/Alpaca-Cafe
Word Count: 8,772

Acknowledgements

I would like to express my deepest gratitude to my dissertation supervisor, Dr Kakia Chatsiou, for their ongoing support, guidance, and feedback during both the write-up and the synoptic project. Their encouragement and insights have been essential to my progress.

I also wish to thank my amazing partner, Sarah, for her continuous support during this time and her time proofreading this dissertation.

I also wish to thank the teaching staff at the University of Suffolk for their dedication and for providing a solid academic foundation during my studies.

Special thanks to BT for their support during my studies towards my apprenticeship and for providing me with a great technical foundation to build upon.

I am deeply grateful to my friends and colleagues for their moral support, discussions, advice, and feedback that have helped shape this project.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

"Ordering food has not changed for 30 or 40 years."
(Steve Easterbrook, 2011), President of McDonald's Europe.

In the rapidly evolving technological landscape, digital solutions are becoming increasingly integrated into everyday tasks. One area that has undergone significant transformation is the way food is ordered. An increasing number of fast-food chains, traditional sit-down restaurants, coffee shops, and pubs are shifting away from the traditional cashier or waiter-based ordering systems in favour of online platforms and self-service kiosks (SSKS).

Many businesses now replace staff with online ordering systems or self-service kiosks, improving efficiency and increasing upselling opportunities (Xavier, 2025). Furthermore, research indicates that self-service ordering solutions contribute to overall sales growth (El-Said & Tall, 2019a) and improve operational efficiency (Ishak et al., 2021).

This dissertation does not seek to evaluate the effectiveness of SSKs or analyse customer perceptions of these systems, as existing literature extensively documents customer behaviour and the advantages SSKs offer businesses. Instead, the focus will be on the design, development, and implementation of a lightweight, drop-in hospitality ordering system with integrated mobile ordering capabilities.

Current hospitality ordering systems used by major fast-food chains such as McDonald's, KFC, and Burger King rely on custom-built infrastructure, often requiring substantial investment in on-site networking equipment. In contrast, smaller establishments, such as independent restaurants and pubs, frequently depend on software-as-a-service (SaaS) solutions that offer limited customisation to suit specific business needs. However, there is a noticeable gap in the market for mid-sized restaurant chains that seek a solution that balances flexibility and affordability, one that does not need extensive networking infrastructure but still allows for customisation and seamless integration with existing operations.

In this dissertation, section 2, Literature Review, aims to analyse and review existing literature and information around existing hospitality systems, along with the relevant technologies, methodologies, accessibility standards and relevant legislation.

Section 3, Planning, will cover the planning aspect of the system, reviewing competing systems and the requirements of the system. Section 4, Development, will focus on the development of the system, fulfilling the requirements set out in Section 3 and discussing notable points of the development process.

Section 5, Deployment and Production, will explore how the project was deployed into a production environment and the security considerations. In addition, the management of a production application, such as monitoring. This report will then be evaluated in section 6.

discussing requirements that have or haven't been achieved and if the application is fit for purpose.

2 Literature Review

This literature review explores existing research and technology implementations in hospitality ordering systems, focusing on self-service kiosks (SSKs), web development methodologies, and the technologies chosen for this project, including Ruby on Rails and MQTT. Additionally, it reviews standards related to web accessibility and regulatory compliance relating to the development of the project and the industry.

2.1 Self-Service Kiosks (SSKs)

Self-service kiosks (SSKs) have become increasingly prevalent within the hospitality sector, driven by the introduction of both free-standing kiosks and mobile ordering solutions. This shift is largely motivated by the desire to improve operational efficiency, increase order throughput, and reduce staffing costs (Shiwen et al., 2021).

While not directly focused on food service, a study conducted by Oracle Hospitality in partnership with Skift titled “Hospitality in 2025: Automated, Intelligent… and More Personal” found that 73% of travellers are more likely to choose hotels offering self-service technologies (SSTs), demonstrating a growing consumer preference for automation in hospitality environments.

Further supporting this trend, Tillster’s Self-Service Kiosk Index (2019) reported that 30% of customers prefer using a self-service kiosk even when the queue length is the same as that of a traditional cashier. This highlights the importance of perceived control, speed, and convenience in shaping consumer choices.

The evidence suggests that customer sentiment is shifting significantly towards self-service solutions, with kiosks and mobile ordering systems gaining traction across hospitality domains. In response to this shift, businesses that fail to adopt such technologies risk falling behind competitors. This project aims to equip Alpaca Cafe with a modern ordering system that supports both kiosk-based ordering and mobile app pre-orders, aligning with these evolving customer expectations.

These trends informed the development and inclusion of both kiosk and mobile ordering options, which are implemented in Section 4.6.

2.2 Software Development Methodologies

Agile methodologies have been widely adopted in modern software engineering due to their flexible, iterative nature and the "release early, release often" ideology, which enables continuous feedback and faster feature delivery. According to the 17th State of Agile Report (2023), approximately 71% of businesses report adopting Agile practices. Within these, Scrum

is the most commonly used framework, implemented by 81% of Agile teams, often alongside hybrid models like Scrumban.

This widespread adoption underscores Agile's effectiveness in improving team collaboration, adaptability to change, and faster time-to-market. For this project, a Scrumban model was initially utilised to manage tasks and iterations. However, due to the solo nature of development, this approach was later replaced with a DevOps methodology incorporating Continuous Integration and Continuous Deployment (CI/CD) pipelines.

CI/CD enables rapid iteration, automated testing, and reliable deployments, which are critical for customer-facing, high-availability systems like Alpaca Cafe. These practices reduce the risk of human error, allow frequent, small updates, and support continuous improvement (Fortinet, 2025).

The shift from Scrumban to DevOps, discussed in Section 4.2, was guided by these methodological insights.

2.3 Key Technologies – Ruby On Rails

Ruby on Rails is a web application framework built on top of Ruby that emphasises convention over configuration, facilitating rapid development (Hansson, 2005). And utilises the Model-View-Controller (MVC) architecture that promotes organised code structures and maintainability.

The MVC architectural pattern, which separates concerns within an application into three layers: the Model (data and business logic), the View (presentation layer), and the Controller (application logic and request handling). This structure not only enhances code maintainability and testability but also aligns with best practices in modern software engineering (Fowler, 2003). By enforcing this separation, Rails enables developers to build modular applications that are easy to scale and debug.

Furthermore, Rails places a strong emphasis on developer happiness and clean, readable code, making it particularly suitable for agile environments that require frequent iteration and refactoring. This philosophy aligns well with test-driven development (TDD), which Rails natively supports through frameworks like RSpec and Minitest.

2.4 Web Standards

There is a range of agreed standards around developing applications for the web, covering a range of different areas such as inclusiveness, security and maintainability. These standards have become common and an expected element on the internet, and falling short of these standards can see declining traffic, declining customer satisfaction and, in some instances, be punishable under the original Public Sector Bodies (Websites and Mobile Applications) Accessibility Regulations 2018.

Standards defined by the World Wide Web Consortium (W3C), such as HTML5 and CSS3, help ensure consistent behaviour across browsers, enhance usability, and support accessibility for users with disabilities.

A key framework for accessibility is the Web Content Accessibility Guidelines (WCAG), currently at version 2.1, which outlines success criteria under four principles: Perceivable, Operable, Understandable, and Robust (WCAG 2.1, 2025). Compliance with WCAG 2.1 is not only best practice but also a legal obligation under regulations like the UK Equality Act 2010 and the Public Sector Bodies (Websites and Mobile Applications) Accessibility Regulations 2018.

For customer-facing platforms like Alpaca Cafe, accessibility is particularly important, as the interface must accommodate a wide range of users. This includes support for keyboard navigation, screen readers, high contrast modes, and semantic markup, all of which are addressed in the project's frontend implementation. By aligning with these standards, the system promotes digital inclusivity and reduces the risk of alienating potential customers.

Furthermore, semantic HTML enhances search engine optimisation (SEO), improves performance, and simplifies testing and automation, critical concerns for scalable systems. Web standards also support progressive enhancement, ensuring that the application remains functional even in degraded environments or on older devices.

2.5 Compliance

In any business, there is a range of different legislation and regulations that have to be abided by. While there are many that can apply to the project in question, two will be the focus of this compliance section of the literature review. These are the Payment Services Regulations and the UK's Data Protection Act 2018.

The following discussions on compliance legislation guiding the implementation of secure payments using Stripe is detailed in Section 4.5.1, while data protection considerations are described in Section 5.5.

2.5.1 Payment Service Regulations 2017

The Payment Services Regulations 2017 govern the use of accepting payment online and in person, how the information collected during these transactions should be processed, stored and shared, and the duty businesses have when they accept payments. Due to the nature of the proposed system, accepting online payments, the payment service regulations underpin much of the logic and functionalities of the system.

The implications of the regulations for the mobile aspect of the system are ensuring encrypted communication with a payment provider. This is covered by ensuring secure HTTP (HTTPS) is used during the payment part of the process, so details are encrypted. The current plan for the

system is to utilise a third-party payment processor, Stripe, that is already compliant with the regulations. This keeps payment information safe and secure and removes obligations from the business. However, some parts of the regulations still apply, such as needing to be transparent with fees, prices and transaction details.

2.5.2 Data Protection Act 2018

The Data Protection Act 2018 governs how information can be collected, stored, used and shared within a system and application. Any business must adhere to the legislation; failing to do so can and has previously led to large fines, as seen where Advanced Computer Software Group was fined £3 million for breaching their regulatory duties (Gourlay, 2025).

To ensure compliance, systems must implement appropriate technical and organisational measures, such as data minimisation, secure access control, encryption, and breach response procedures. For customer-facing platforms like Alpaca Cafe, compliance is critical not only for legal reasons but also to maintain user trust and service integrity.

3 Planning

This project to develop and launch a mobile order system for a multi-location coffee shop is a continuation of a previous assignment, Multi-Media Mobile and Internet, based on the PHP version of the same idea from Software Design, Development and Engineering. Some of the planning has already taken place, with this being a continuation. Although most aspects have been changed to meet changing requirements and production needs, some aspects remain from the original prototype.

3.1 Systems Review

With the advent of COVID-19, the rise of online ordering skyrocketed (Morell, 2022), with wider place adoption of online ordering for in-person pickup and delivery options to minimise human contact. These services and software are still prevalent within the hospitality sector. With businesses such as Uber Eats, Just Eat, and Deliveroo offering in-person pickup and delivery, businesses can easily offer these options to their customers. The downside of these companies is that they can only offer online ordering and don't integrate with other aspects of a system, requiring separate inventory management (Goyal, 2025).

Square, on the other hand, can offer the order-ahead aspects and other functionality a growing business needs (Herrera, 2025). Square provides order-ahead functionality in addition to in-person options like POSsk, kiosks and card readers.

However, Square has not been used for this project; despite the comprehensive tools and full integration, it imposes limitations that constrain the required flexibility and scalability needed for this system. Squares' ecosystem is largely closed and tailored towards general retail and smaller food outlets, which doesn't support complex product customisation ordering and product-ingredient relationships. It also does not allow for deep integration with other infrastructure, such as MQTT-based messaging for real-time communication between POSs, recipe printers, order screens and recipe printers.

While Square may be perfect for small businesses, multi-location businesses require more flexibility and control, which allows for efficient inventory management between multiple locations and order synchronisation. Finally, a custom-built solution ensures that the system can evolve to support future requirements and unique operational workflows that may not be accommodated by an off-the-shelf platform like Square (Hanenko, 2025).

3.2 Requirements

Table 1 shows the high-level requirements with a MoSCoW rating.

Table 1, High-Level Requirements MoSCoW Ratings

Requirement	Description	MoSCoW Rating
Secure Payment integration with Stripe	Integrate with Stripe to process payments from customers for online and in-person orders. Supporting multiple payment methods and types. While also complying with FC.	Must
Google Recommendation AI integration	Integrate with Google recommendations. AI to provide customers with tailored recommendations, increase upsells by offering higher-priced items, and encourage additional purchases with sides, cakes, and pastries.	Must
Selecting between stores to order	Support multiple store locations where customers can order from any location.	Must
Product Customization	Allow for products to be customised and ingredients to be substituted with other, like ingredients. Encourage upselling to add-ons.	Should
Availability of products and ingredients	Implement product and ingredient availability on a store level, and prevent customers from ordering unavailable options; upsell and suggest other products instead.	Must
Automated Emails	Implement automated email sending for order confirmations, account sign-ups, and password recovery, as well as a newsletter.	Could
Admin Management	The admin management dashboard and controls over stores and the application allow management to manage the store, stock levels, orders, and availability and add and remove products. Along with statistics dashboards. Including management for accounts. Creating a segregated and secure area for management.	Should
User Roles and Permissions	Implement policy-based authentication to secure the application, follow the principle of least privilege, and create appropriate roles and suitable levels of access.	Must
Connected Devices	Set up a network of connected devices for kiosks, point of service terminals (POS), and receipt printers using a suitable protocol and the software to connect to the overall system.	Could

Table 2, Low-Level Requirements MoSCoW Ratings

Requirement	Description	MoSCoW Rating
Code Coverage	There must be a suitable level of testing in the application to	Must

	Meet a minimum 95% line coverage for testing.	
Availability	The application, when deployed, must have a 99.9% annual availability.	Must
Data storage	The chosen technology and configuration for data storage must have a high level of redundancy, be kept secure and be backed up.	Must
Compliance	The application must comply with all relevant legislation and standards, such as GDPR and the Payment Services Regulations 2017	Must
Accessibility	The front end of the application must adhere to all W3C accessibility standards and be accessible to all users.	Must

3.3 Methodologies

Agile is a broad term covering various flexible approaches to software development, all focused on iterative progress and adapting to changing requirements. Unlike the traditional waterfall method, where development follows a fixed, sequential path, Agile allows for faster, more responsive workflows (Mishra & Alzoubi, 2023).

This project experimented with two Agile frameworks, both incorporating elements of Kanban. Originating in manufacturing (Wakode et al., 2015), Kanban uses a physical or digital board, where tasks move through columns representing stages of a process. As work progresses, tasks shift across the board, reflecting their current status.

Figure 1 shows the Kanban board used in this project, with the column breakdown detailed in Table 3.

Table 3, Kanban Columns

Column	Description
Backlog	Work that has not yet been prioritized
Todo	Work that has been prioritised and is waiting to be worked on and completed
In Progress	Work from the Todo column that is actively being worked on
Done	Work that has been finished, reviewed and merged into the code base

One of the trialled frameworks is Scrumban, a hybrid of Scrum and Kanban, combining sprints, a set length of time in a development cycle, with Kanban methodology (Alqudah & Razali, 2018). Figure 1 contains an additional column called “Sprint” that represents the current cards that have been planned for the current sprint. Cards would be prioritised from the to-do column in a sprint planning session and moved into the sprint column in a priority order.

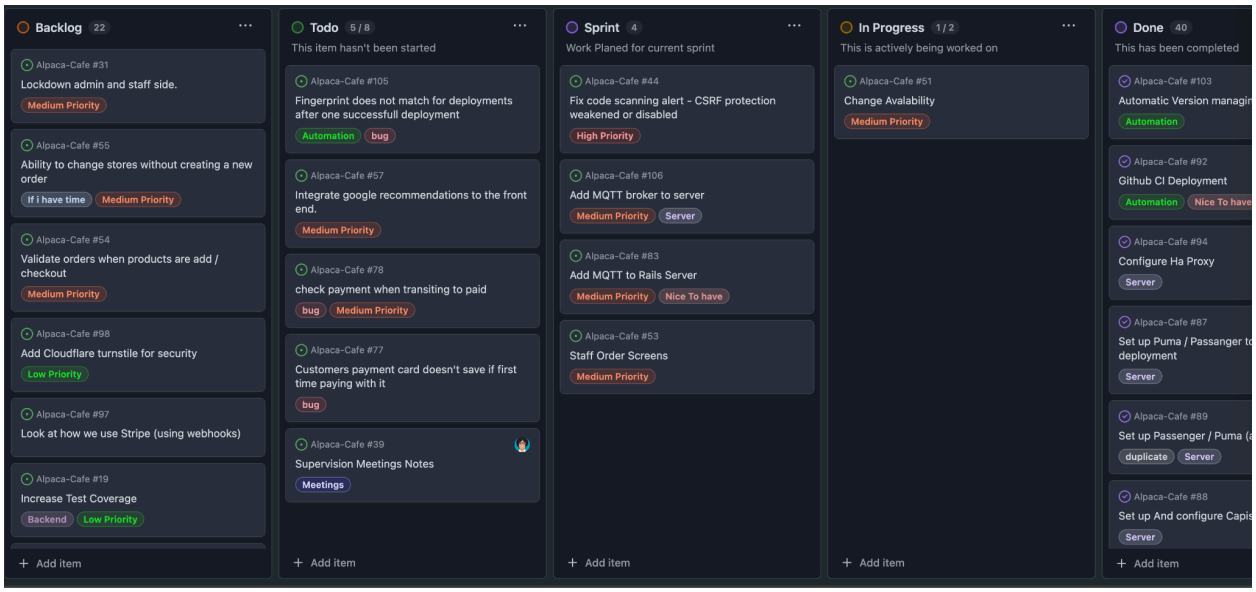

Figure 1: A Scrumban kanban board

During the development, the philosophy of clean code was followed. Clean Code by Martin (2009) is the idea that clean code can be read and enhanced by a developer other than its original author; it's a set of guidelines and rules that help developers in writing good, clean code. One of these principles that is apparent is the lack of comments in the code.

Overreliance on comments in code leads to messier functions and code, in which the comments explain what it does, instead of letting the code speak for itself. While not using comments, developers have to write functions that explain themselves and learn other forms of explanation, like documentation and tests.

3.4 Front-end designs

This project is a continuation of previous work undertaken as a prototype. This work included the design and implementation of branding and front-end design decisions. Due to this work previously being undertaken, it is not included in this report. Examples of design decisions can be seen in Appendix 1. Figure 2 is included to provide content and an overview of the design of the application.

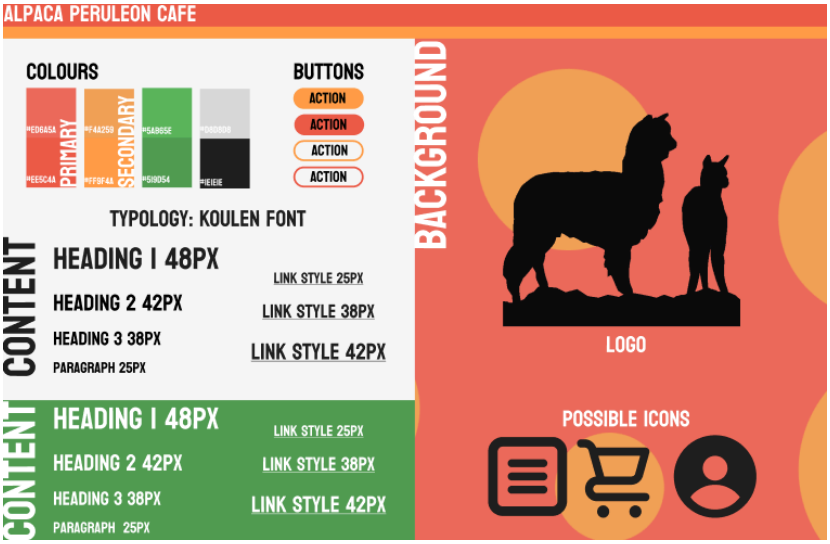

Figure 2, Design Tile

Some design choices were made during this development, the most important of which is the change of the index/home page. The new designs have taken a split approach, with different layouts and contents depending on whether a customer is logged in or not.

Figure 3, User Home Page

Figure 3 displays the updated version for logged-in customers. The reasoning behind having two separate views is to tailor the experience based on the user's status.

The main focus of signed-in customers is to encourage ordering. To make it easy for customers to order, the “order now” button is made prominent and large and placed in an easy-to-access place on the screen. This design choice aims to increase conversion rates and drive more orders. In addition, personalised recommendations for products are also included in a prominent place on the screen.

The original design focused on encouraging customers to make an account or to sign in, as accounts are required to place orders. Making the main call-to-action dependent on the current state of the customer, this reduces unhelpful design choices, like showing an order now button when the user can't order, and vice versa.

4 Development

Initially, the ScrumBan methodology was implemented but subsequently abandoned in favour of DevOps and its Continuous Integration and Continuous Delivery (CI/CD). While sprint-based frameworks offer substantial benefits for larger development teams, they demonstrate little utility in smaller teams or solo development (Tamburri et al., 2020). When used by individual developers, sprint-based approaches converge toward CI/CD practices yet lack the continuous deployment benefits that CI/CD provides.

CI/CD maximises the development cycle through automation and frequent deployments, following the agile principle of “release early, release often. This circular workflow enables faster, more reliable code delivery and continuous feedback, improving quality and reducing time-to-market. By automating testing and deployment, defects are caught earlier, collaboration improves, and the system adapts more efficiently to change. Implementation details are covered in section 4.2.

As mentioned, kanban boards played a critical part in the development process. Figure 4 shows the kanban board layout used for the remainder of the project. Figure 5 shows a burn-up graph for the project, highlighting the rate of change on the kanban board, with the number of open and closed cards showing the ongoing development of the project.

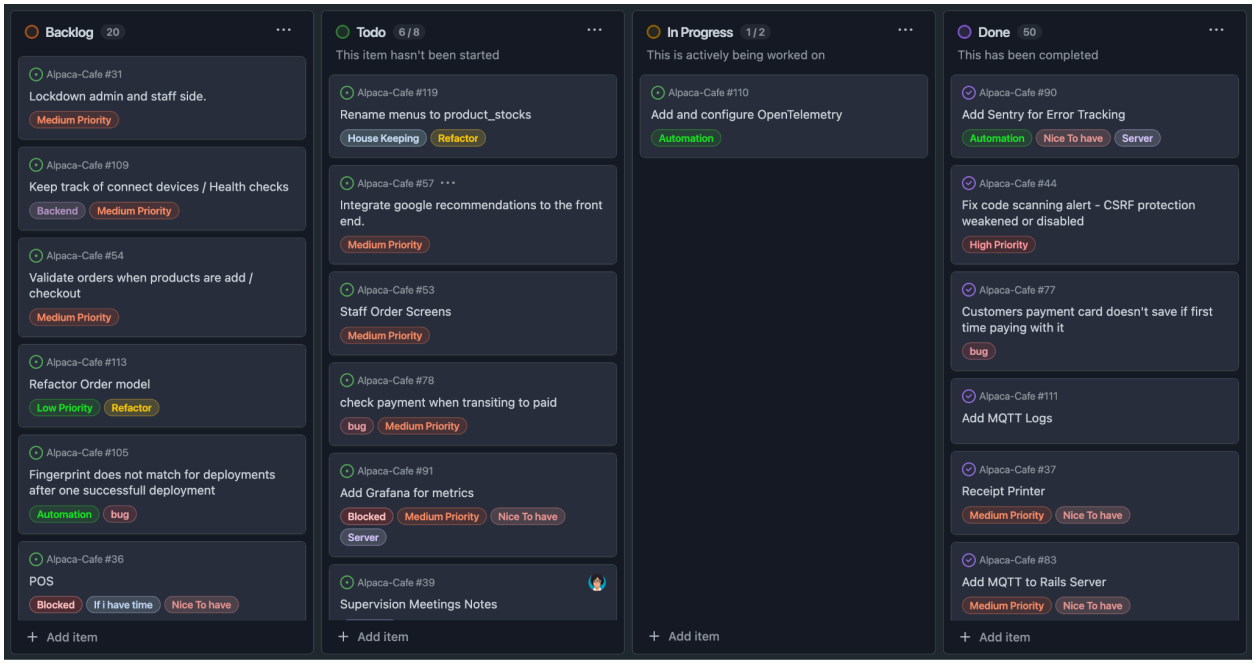

Figure 4, Continuous Integration Kanban board.

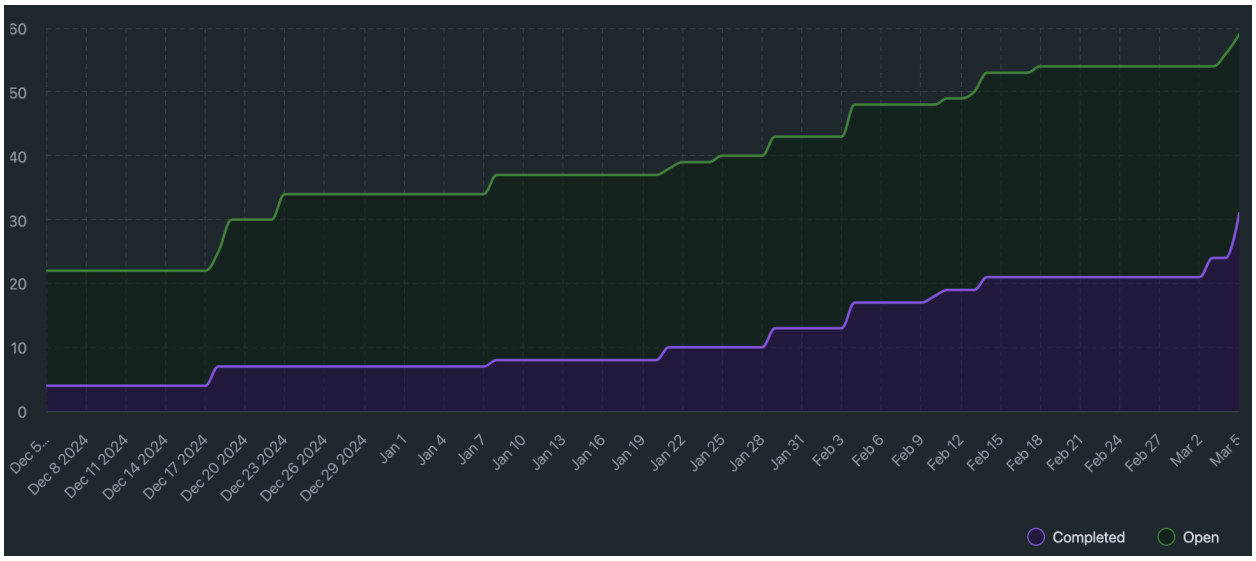

Figure 5, a burn-up graph showing open and closed cards over time.

Version control is a long-standing practice in software development. It is a way of tracking and managing changes to a codebase, maintaining a complete history, and allowing developers to collaborate together. This fundamental practice serves as the backbone of modern software engineering, underpinning collaborative development workflows and enabling continuous integration and deployment processes (Devineni, 2020).

GitHub has been used in this project, providing version control capabilities and additional functionality that will be discussed in 4.2.

Additionally, SemVer versioning has also been incorporated with version control, tracking released versions of the application. SemVer versioning is a way of tracking and identifying versions of code through numbers, generally in the format x.x.x, with a major, minor, and patch number highlighting the level of work and changes made to the project.

4.2 Workflows, Automation, and CI/CD Pipelines

As mentioned in 3.3, DevOps and CI/CD focus on automation to improve development (Devineni, 2020). Following this, a number of automations and workflows have been created and implemented.

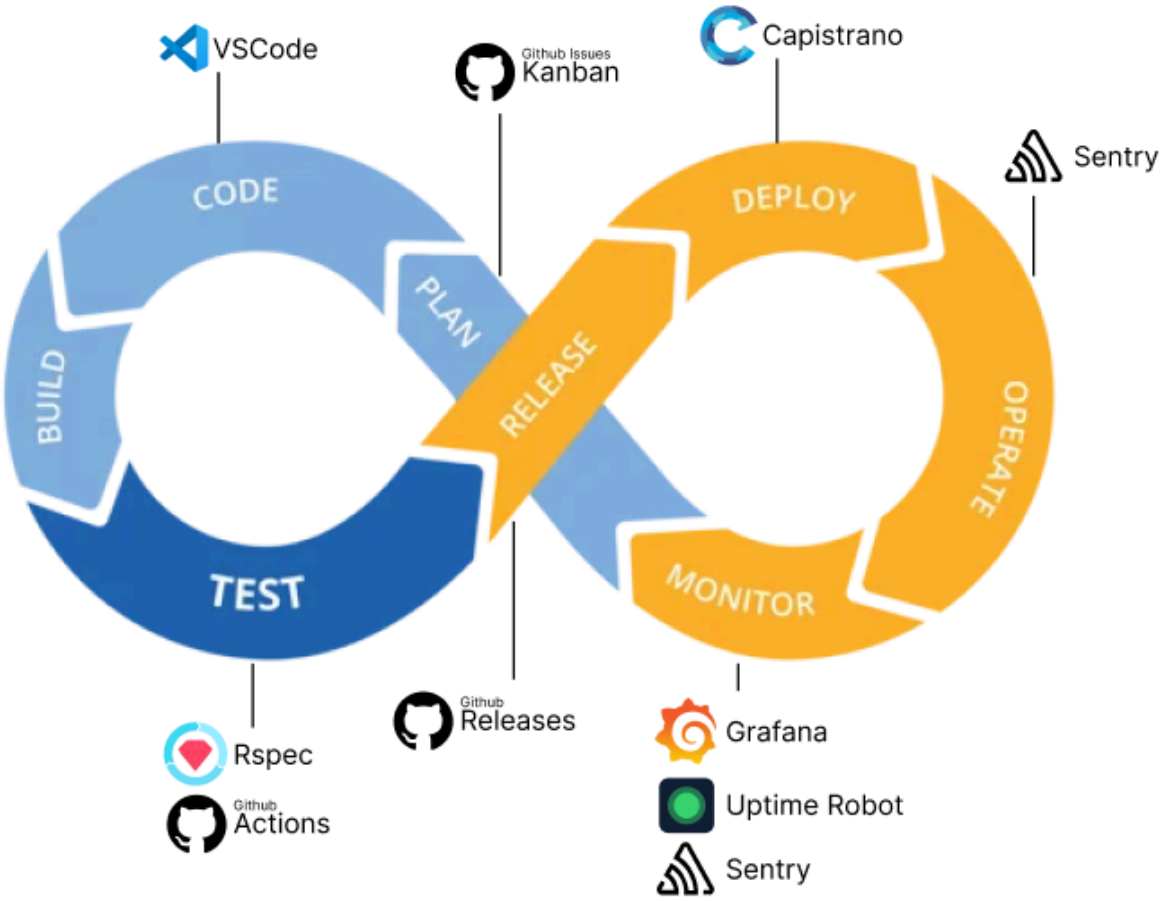

Figure 6, The tools used in DevOps

One of these automations is the integration with a GitHub bot, Dependabot, where it scans code looking for security issues relating to dependencies, and if one is identified, it will attempt to patch the project automatically. This improves the visibility of possible vulnerabilities and fixes them without the need for a developer. These pull requests can be seen in Figure 7.

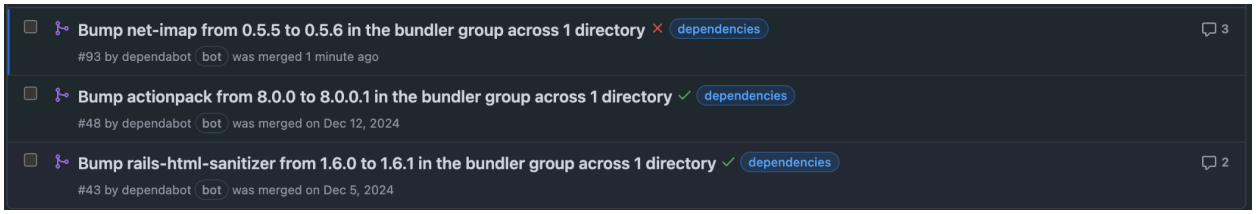

Figure 7, Dependabot's Pull Requests

GitHub Actions were also implemented into the project, where it runs feature and unit tests, linting and code quality checks. These all alert to any issues in the code base before it is deployed to production. This, paired with automatic deployment. All this comes together to make a robust and valuable pipeline; this pipeline is illustrated in Figure 8.

Figure 8, Illustrated Pipeline

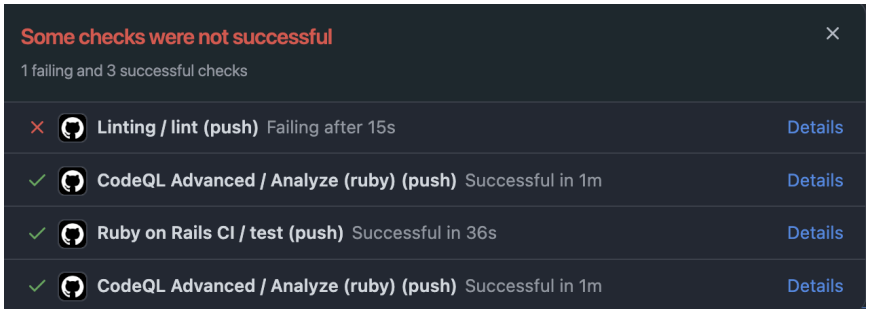

Figure 9, GitHub Actions

Figure 9 shows the GitHub actions running on a push to the main branch. In this situation, linting failed, and the user was alerted to the failure for correction.

4.3 Technologies

Technologies play a vital role in the development of an application. Different technologies offer different benefits, drawbacks, and abilities to their counterparts. For this application, Ruby was chosen, in the latest version 3.2.3, along with Rails 8. Ruby on Rails was a prime candidate for

this project, as it provides a robust framework for rapid application development with its convention-over-configuration philosophy. The framework uses the CRUD approach, providing a great convention in Object Relationship Management development, which this project heavily utilises with the relationship between models. More on the relationships in 4.4.3.

Ruby on Rails may not be the industry go-to nowadays, with JavaScript frameworks like React or Svelte being popular choices (Stack Overflow, 2024); however, these frameworks focus on the front-end of an application but often require additional backend solutions, creating a more complex tech stack (Ramsingh et al., 2022). Rails' full-stack nature eliminates this fragmentation, providing a cohesive development experience from database to front end, following the View Model Controller approach. Furthermore, Rails' mature ecosystem offers battle-tested gems for common functionalities like authentication, authorisation, and payment processing, reducing development time and potential security vulnerabilities that might arise from implementing these critical components from scratch. Ruby's emphasis on testing also ensures higher quality code with fewer bugs, a crucial consideration for this application's reliability requirements.

Some of the aforementioned gems that have been included for extending functionality and ease of development are Devise, Bootstrap and Stripe. While this is not an extensive list, each one of these gems provides essential functionality. Devise is a popular choice that extends Rails' built-in warden, which handles user authentication. More on gems and their use cases can be found in Appendix 2.

Other technologies used in this application include Python 3. This is used in connected devices to provide functionality locally and to connect to the system. Python was used instead of Ruby in this instance, as it runs well on multiple platforms and comes pre-installed on Linux environments. The syntax of Python is also similar to Ruby's, easing development and reducing context switching. Python also has an extensive amount of libraries that can extend the functionality, similar to Ruby's Gems, and as such, can be used in conjunction with MQTT for intersystem communication. More on MQTT in 4.6.1 MQTT vs HTTPS.

4.4 Testing

During the development of this project, automatic and continuous testing were paramount. Automated testing allows for the whole application to be tested each time a change is made, catching unintended behaviour changes while speeding up development, as an entire test suite can be processed quicker and more accurately than manual testing. Some manual testing was performed, as seen in Appendix 3.

4.4.1 Test Driven Development

The main philosophy followed during development was test-driven development. Test-driven development (TTD) is the process of writing a test before writing the code and then making the code satisfy the test. This differs from other approaches where code is written first, then a test is

made to ensure it doesn't break in the future. However, there is a downside to the latter approach, as it promotes a reactive test approach that may leave bugs and artefacts in the code and may not account for edge cases while leading to more code than what is needed (Fucci et al., 2016).

TDD has several benefits and is considered a better approach for agile working (Agha et al., 2023). One of the benefits of TDD is improved code quality, as the code is written to pass specific tests, leading to more reliable and bug-free software. Since tests are written before the code, developers naturally write cleaner, more modular, and well-structured code that adheres to best practices (Ramzan et al., 2024).

TTD also improved requirement validation, as it forces developers to think deeply about requirements before implementation and ensures that all expected behaviours and edge cases are considered upfront (Alexei, 2024).

4.4.2 Testing Technologies

RSpec, the primary test framework, was chosen for its test-driven development (TDD) approach, which promotes readable and maintainable tests. To support comprehensive testing, gems like Faker, FactoryBot, and VCR were also used.

Faker generates realistic fake data, names, emails, and addresses, helping test varied input scenarios and reducing hardcoded values. FactoryBot simplifies test setup by dynamically creating test objects, keeping tests clean and scalable.

VCR was used to handle external API calls, particularly for Stripe, by recording and replaying responses. This avoids repeated external requests during tests and reduces costs.

4.4.3 Code Coverage

Code coverage refers to the amount of code that is tested using unit tests. It's a good indicator of how stable the overall application is. It also highlights branch coverage, catching edge cases within the code. It is considered great practice to have a high code coverage to ensure the code is tested, and it guards against any unintended changes with code changes as well as technology upgrades (Bhuyan, 2024).

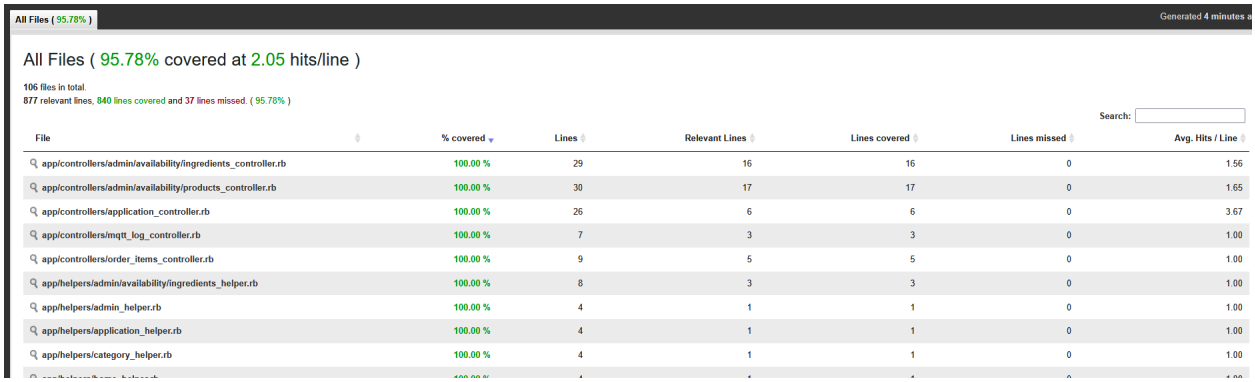

Figure 10, Code Coverage Report

Using a gem called SimpleCov, which analyses the code that's tested when the tests are run. Figure 10 shows the report that is generated. In this application, 95% code coverage is the minimum amount, with 100% being the desired coverage.

Individual file code coverage can be found inside the project, `coverage/index.html`.

4.5 Notable Points and Milestones

4.5.1 Payment Processing – Stripe

A third-party payment processor handles transactions between customers and businesses, securely managing payments without exposing the business to sensitive financial data. These services act as intermediaries, simplifying payment acceptance, ensuring compliance, and offering access to various payment methods without direct bank agreements.

Stripe is a widely used processor for online payments, subscriptions, marketplaces, and in-person transactions. Its flexibility makes it the preferred choice for companies like OpenAI, Google, Shopify, and Airbnb.

Stripe was chosen for this project due to its developer-friendly tools, clear documentation, and strong industry reputation. Crucially, it ensures compliance with the FCA and Payment Services Regulations 2017 by handling all payment data and personal information.

Alternatives like PayPal, Adyen, and Braintree were considered. While PayPal is popular and offers strong buyer protection, it scored lower for developer experience and fee structure (Smplfy, 2025).

4.5.1.1 Implementation

Some challenges were faced during development around implementing and using Stripe. The desired implementation of Stripe followed a more complex process than most implementations.

A standard implementation of Stripe is where the customer enters payment details and authorises the payment on checkout, where funds are immediately captured by Stripe. This is illustrated in Figure 11.

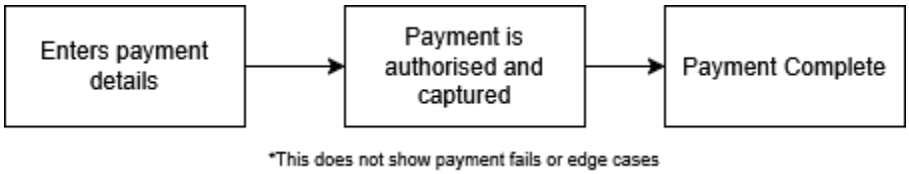

Figure 11, A Standard Stripe Implementation.

However, the desired implementation, illustrated in Figure 12, includes an additional step where customer payments are authorised and the funds are held by Stripe but not yet captured. When the customer checked into the store to collect their order, they were then charged.

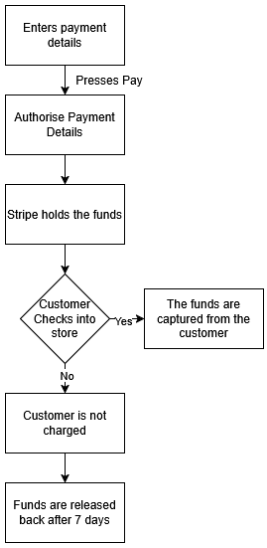

Figure 12, The Desired Stripe Implementation

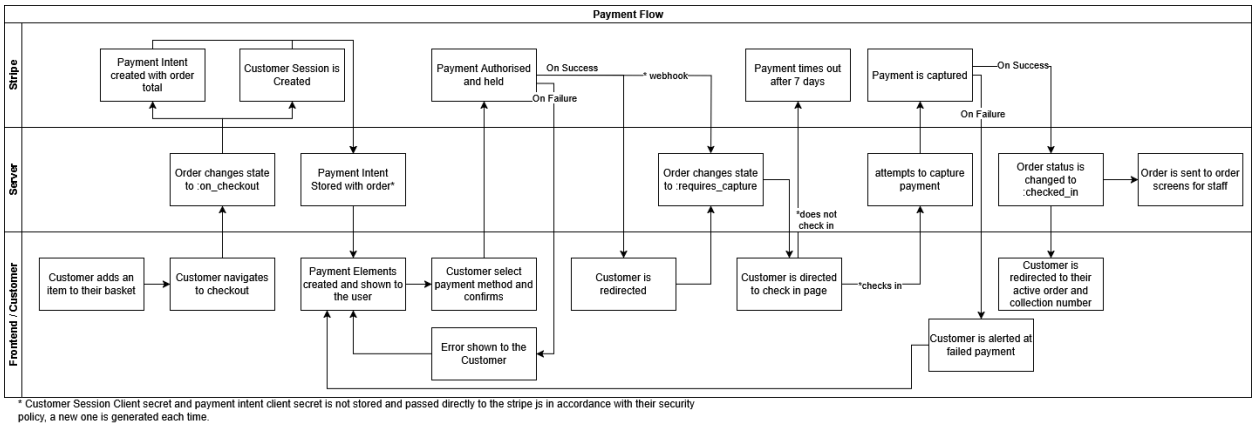

Figure 13, Customer Payment Flow

Figure 13 shows a swimlane diagram of the entire customer flow and how Stripe is leveraged into the application. The payment processing implementation uses both client-side JavaScript and server-side code to complete the journey. A customer session is created through Stripe, which ties everything that follows to a particular customer and provides the saved payment methods. Then, a payment intent is created, which includes information on the order and the total amount. This code has been extracted into a service module, following Ruby on Rails best practices, and can be seen in Appendix 5.

When the customer navigates to the checkout page, the payment elements are shown using Stripe's JavaScript, where they can choose a saved payment method or enter a new one. A snippet of the JavaScript code can be seen in Figure 14.

Uni-Rails-1 [WSL: Ubuntu] - stripe_elements_controller.js

```
6   this.stripe = Stripe(this.element.dataset.publishedKey);
7
8   this.elements = this.stripe.elements({
9     clientSecret: this.element.dataset.stripeElementsClientSecret,
10    customerSessionClientSecret: this.element.dataset.customerSessionClientSecret,
11
12  })
13
14  let paymentElement = this.elements.create('payment', {
15    layout: {
16      type: 'accordion',
17      defaultCollapsed: false
18    }
19  })
20
21  paymentElement.mount('#payment-element')
22 }
23
24 confirmPaymentIntent() {
25   const elements = this.elements;
26   this.stripe.confirmPayment({
27     elements,
28     confirmParams: {
29       return_url: 'https://coffee.leon-skinner.dev/payments',
30       payment_method_data: {
31         allow_redisplay: 'always',
32       },
33     },
34   })
35 }
```

Figure 14, The Stripes JavaScript

When the customer clicks the pay button, Stripe attempts to authorise the payment and, if it succeeds, places a hold on the funds. When the user checks into the store, the payment is then captured.

In this process, the customer only sees and needs to complete two actions: entering the payment details and clicking check-in. This can be seen in Figure 15, showing the views of checkout and check-in.

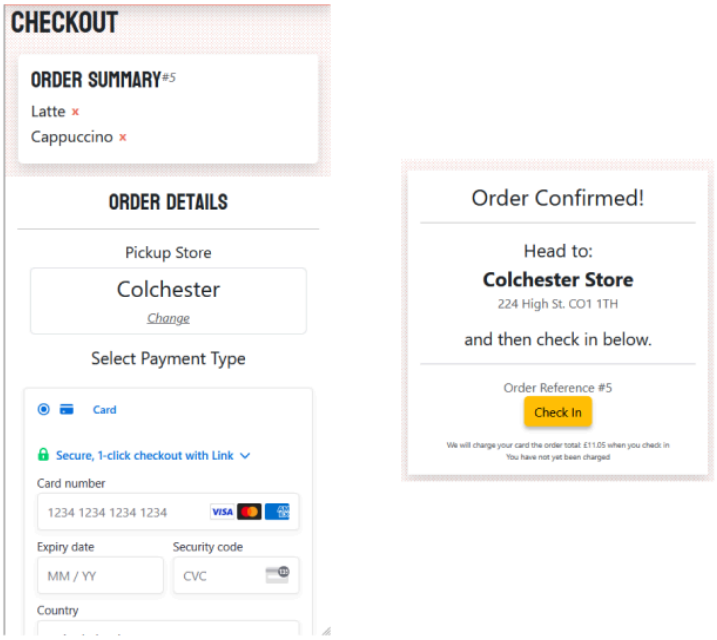

Figure 15, Views of the Payment Flow

4.5.2 Google Recommendations AI

Google Recommendations AI, currently known as Vertex AI, is a large language model by Google that takes in events, such as page views, customer baskets, and purchases, and then provides tailored product recommendations to the customer.

Product recommendations are an important part of e-commerce, helping businesses personalise the shopping experience, increase conversions, and drive customer engagement. By analysing user behaviour, such as browsing history, cart additions, and past purchases, recommendation engines like Vertex AI can suggest relevant products, improving both customer satisfaction and sales.

Many large retailers and marketplaces rely on AI-driven recommendations to enhance their platforms. According to Google, businesses using Vertex AI for recommendations have seen increased revenue per session and improved customer retention. Studies show that personalised product suggestions can account for up to 35% of e-commerce revenue (Lindecrantz et al., 2020). The ability to deliver relevant recommendations in real time makes AI-powered solutions like Vertex AI a valuable tool for modern online retail. Vertex AI has been chosen for this project for these reasons.

The implementation of Vertex AI is straightforward. Firstly, there were some criteria that the application had to meet to start getting recommendations.

- 100 or more products
- 10,000 Detailed page views in 90 days
- 10,000 Home-page views

A product catalogue needs to be uploaded to the large language model, which contains the product, the unique ID, categories, and other optional information. The product catalogue created for this application can be found in Appendix 7. Once uploaded, user events can be sent to Google to start creating recommendations.

User events can be captured in different ways; in this case, a server-side approach has been implemented. When a user either navigates to the home page, navigates to a product page, adds something to their basket, or purchases an item, a service is triggered on the server, which sends off the event to Google. Figure 16 shows an extract of this service; the entire service can be seen in Appendix 6.

Uni-Rails-1 [WSL: Ubuntu] - google_retail_tag_service.rb

```ruby
30  def send_request(request_body)
31      uri = URI(GOOGLE_API_URL)
32      request = Net::HTTP::Post.new(uri)
33      request['Authorization'] = "Bearer #{@access_token}"
34      request['Content-Type'] = 'application/json; charset=utf-8'
35      request.body = request_body
36
37      response = Net::HTTP.start(uri.hostname, uri.port, use_ssl: true) do |http|
38        http.request(request)
39      end
40
41      log_response(response)
42    end
```

Figure 16, An Extract of Google Tag Service

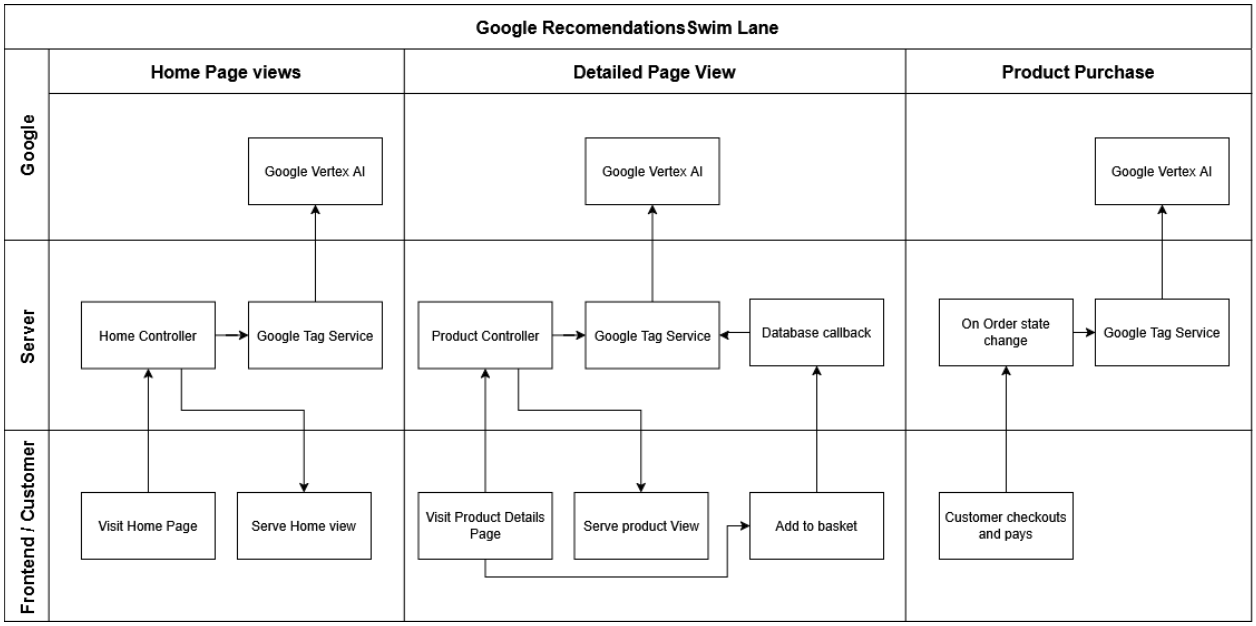

Figure 17, Google Recommendations Tagging Swimlane

Figure 17 shows how the tagging is triggered and sent to Google to be used by Vertex AI in producing recommended products.

The server-side approach was chosen over a front-end approach, perhaps using Google Analytics, to be able to enrich the events and provide additional information securely and without the risk of tampering. This approach also removes the need for tracking cookies, benefiting users by preserving their privacy while still allowing event generation for product recommendations, even when cookies are blocked.

Implementing Vertex AI came with challenges, particularly due to fragmented and unclear documentation spread across multiple services. The lack of detailed examples led to trial and error, and compatibility issues with the existing infrastructure required extra configuration and testing to ensure smooth data flow.

Despite these difficulties, thorough research and experimentation allowed for a successful integration.

4.5.3 Database and Table Relations

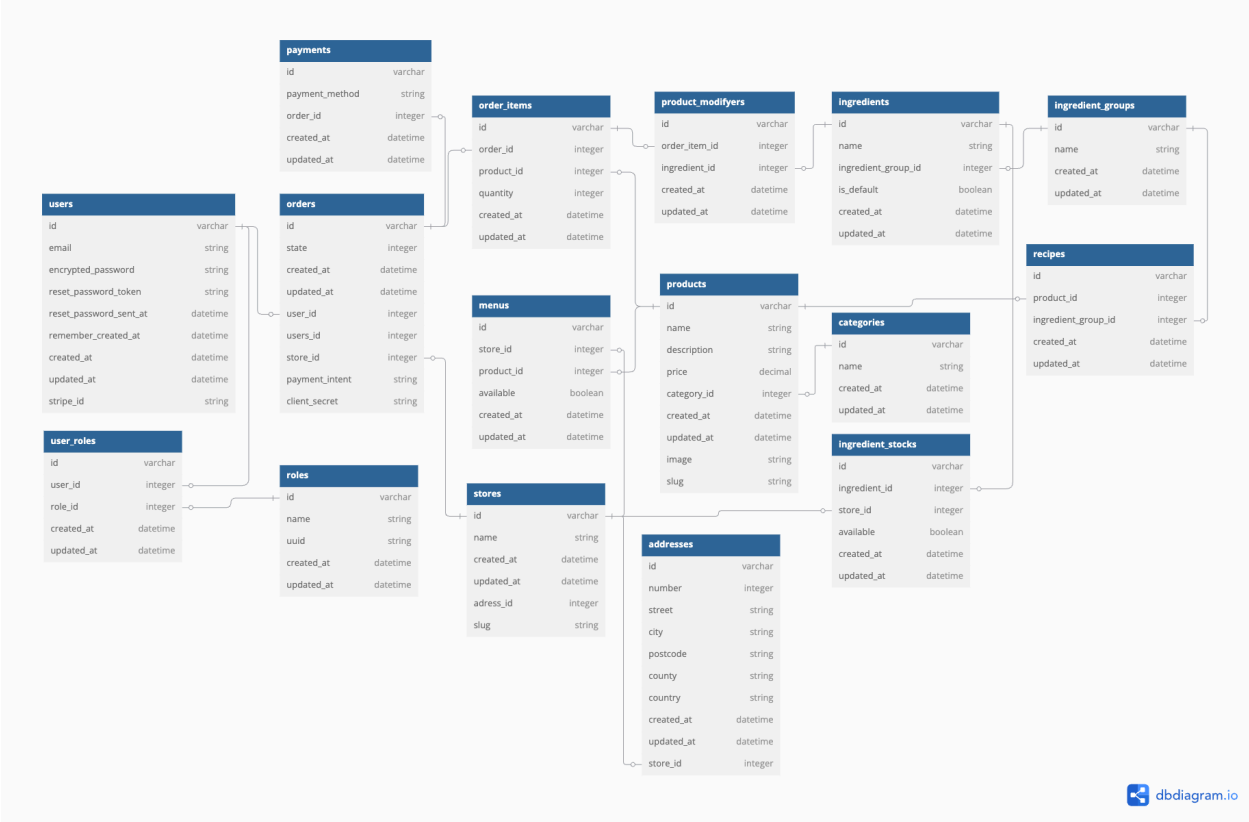

Figure 18, Complete Database ERD

At first glance of Figure 18, this project contains a rather large and complex database, which is essential for maintaining the integrity and functionality of the application. The database has been designed following the principles of normalisation to ensure data redundancy is minimised while maintaining efficiency in data retrieval and management. This structured relational database forms the backbone of the application, supporting key functionalities such as user authentication, order processing, and inventory management.

The database consists of several interrelated tables, each serving a distinct purpose within the system. Without this, the complex relationship between products and their ingredients, the relationship between which ingredients can be substituted with each other, and the availability of ingredients and products on a store level wouldn't work.

Figure 19, Orders Users Relationship ERD

This figure 19 shows a simplified overview of an order. The “orders” table links to the “users” table, which shows the customer of the order. The “orders” table is a representation of the overall order and has child records called “order_items” that show the individual items in the order. This relation is many to one. “Order_items” is a representation of “products”, and this is a one-to-one relationship.

However, what makes this more complicated is when the customer wants to modify the product. In this case, the database now looks like figure 20.

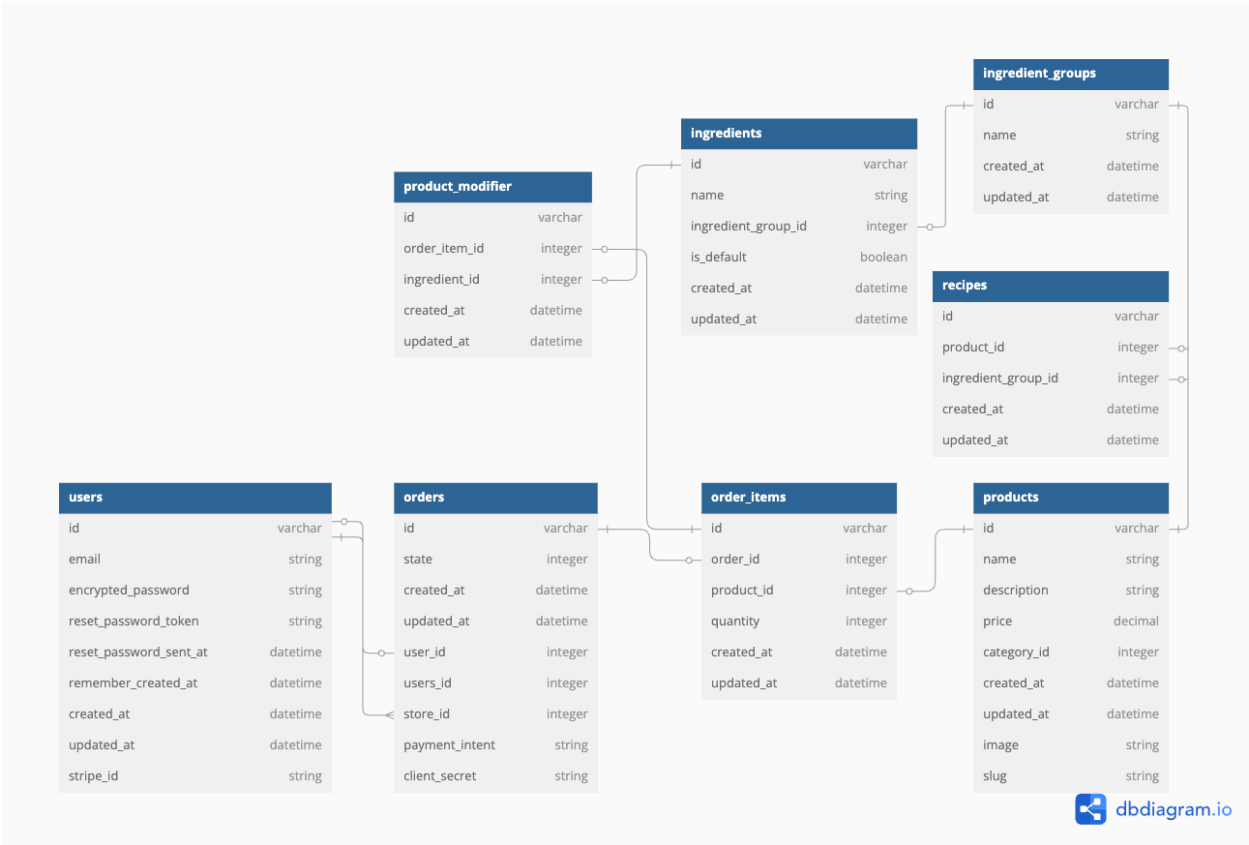

Figure 20, Customizable Products ERD

This now has the inclusion of the “product modifier” table, which represents a single modification to the product, a many-to-one relationship. This links to the “ingredients” table, representing the substituted ingredients. To know what ingredients can be substituted, ingredients belong to the “Ingredient_groups” table. Ingredients in the same group can be swapped. And to know what ingredient groups go with what product, so the milk option doesn't show for an Americano or a croissant, for example, a “recipe” link table exists to link the “ingredient_group” table to the “product” record. This is a one-to-many relationship.

It does get more complex than that, with the introduction of store-specific availability, where there is a link table between stores and the products called “menus”, which represents availability, and stores and ingredient groups called “Ingredient_stock” that do the same for ingredients. This relationship can be seen in figure 21.

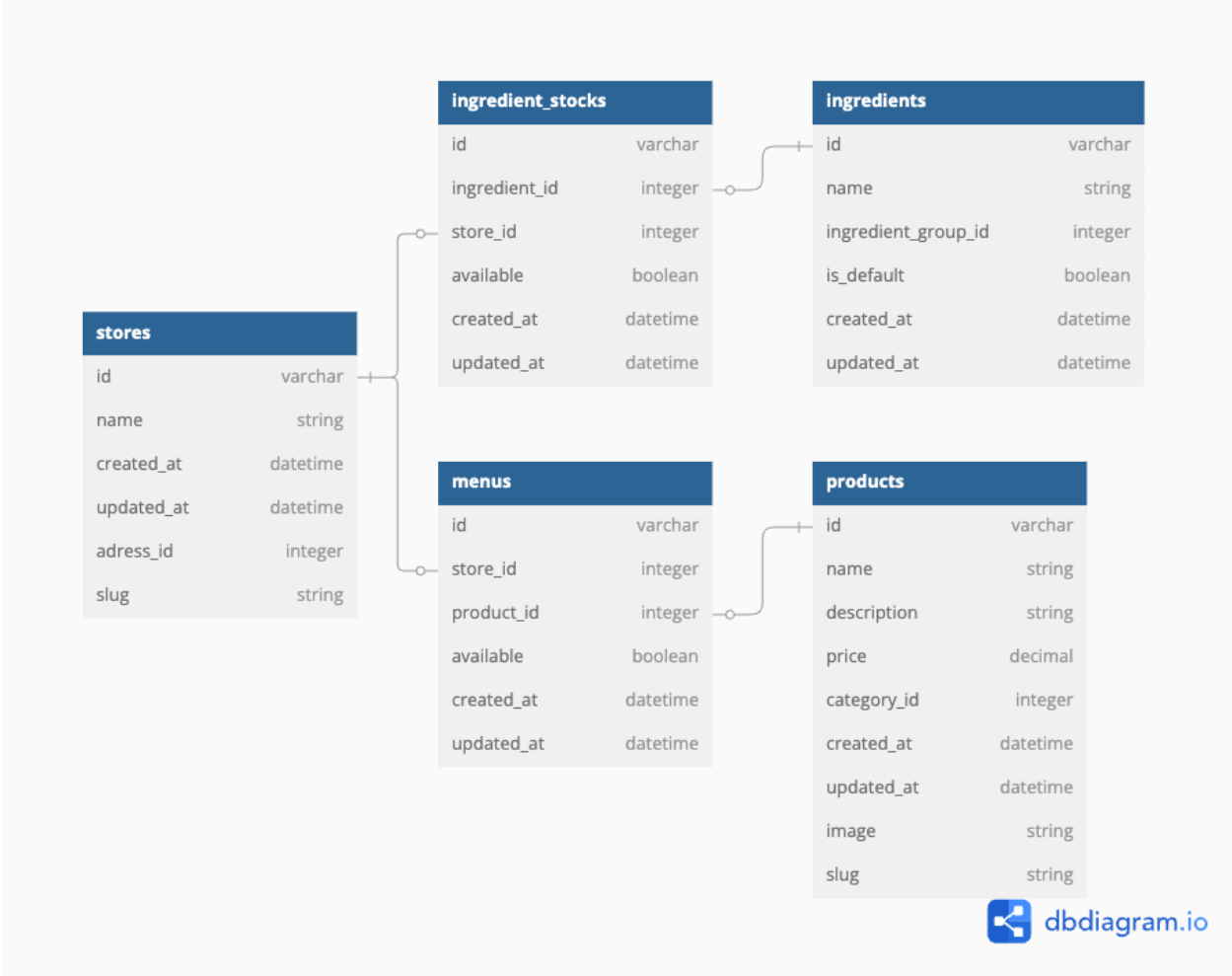

Figure 21, Ingredient Stock ERD

The database does then have other tables, such as “users” and “user roles” that are used for policy-based authentication and security, “addresses” for the store's address and some other tables that contain information useful in the project.

4.6 Connected Devices

Connected devices is the unique term given to the system and ability to connect edge devices into the overall system to add additional physical functionality, such as integrated kiosks, point-of-sale terminals, receipt printers and order screens.

4.6.1 MQTT vs HTTPS

To allow devices to interact and connect to the system, a suitable protocol needed to be chosen, prioritising reliability and security. MQTT and HTTP/S were both strongly considered for which protocol to use in production to communicate with the proposed connected devices. While both have merits, MQTT was selected due to its lightweight, publish-subscribe model and suitability for real-time IoT communication.

Unlike HTTPS, which uses a request-response model and requires client polling, MQTT supports efficient bi-directional communication with lower bandwidth usage. This makes it ideal for event-driven updates such as order notifications. Although HTTPS offers simpler authentication and broader support, its resource demands made it less suitable in this context.

Other protocols such as AMQP, HTTP, and DDS were briefly evaluated but not pursued further.

4.6.2 Consuming and Posting to MQTT

Some connected devices are designed to consume and display data sent via MQTT. This includes tasks such as printing receipts, showing orders to staff, or displaying order numbers for collection. An example of the architecture of MQTT is illustrated in Figure 22. The others publish messages to a topic for the main app to process. To handle incoming and outgoing MQTT messages in a programmable way, Python is used on these devices.

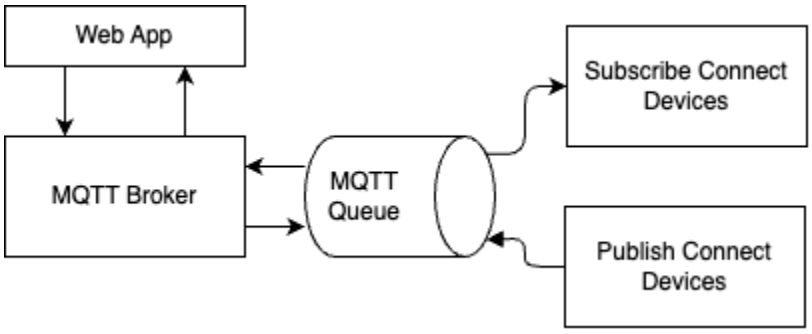

Figure 22, MQTT Illustrated

The paho-mqtt Python library is used to connect to a specified MQTT topic, allowing devices to receive relevant messages. This process is illustrated in Figure 22. Figure 23 shows an extract of the printer.py file that handles printing order receipts when customers check into the store.

AlpacaPoS - printer.py

```python
def load_config():
    cm = Config_Manager()
    cm.load_config()
    return cm.host, int(cm.port), "printer", cm.username, cm.password

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected successfully to MQTT broker")
        client.subscribe(MQTT_TOPIC)
    else:
        print(f"Failed to connect, return code {rc}")

def on_message(client, userdata, msg):
    message = msg.payload.decode()
    print(f"Message received: {message}")
    try:
        receipt = format_receipt(message)
        print_receipt(receipt)
    except Exception as e:
        print(f"Error writing to printer: {e}")

if __name__ == "__main__":
    MQTT_BROKER, MQTT_PORT, MQTT_TOPIC, MQTT_USERNAME, MQTT_PASSWORD = load_config()
    client = mqtt.Client()
    client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)
    client.on_connect = on_connect
    client.on_message = on_message

    try:
        print(f"Connecting to MQTT broker at {MQTT_BROKER}:{MQTT_PORT}...")
        client.connect(MQTT_BROKER, MQTT_PORT, 60)
        client.loop_forever()
    except Exception as e:
        print(f"Connection failed: {e}")
```

Figure 23, Extract of Printer.py

To securely manage MQTT credentials, a custom Python library called config_manager (see Appendix 8) was developed. It encrypts the host, port, username and password into a config.json.enc file, ensuring sensitive data is not stored in plain text. This aligns with best practice for encryption at rest, protecting credentials from unauthorised access. A master key handles encryption, reducing the risk of data leaks from accidental access, unauthorised users or version control exposure, while still allowing secure decryption when needed.

Additionally, a custom setup wizard, seen in Appendix 9, has been created for both graphical and command-line interfaces, streamlining the device setup process.

4.6.3 Types of Devices

The current implementation of connecting devices to the application can support a range of different use cases. Table 4 shows some common use cases that might be found within a coffee shop that can utilise the connectivity of connected devices.

Table 4, Types of Connected Devices

Type of Device	Description
Receipt Printer	Allows the printing of receipts.
Kiosk	Used to allow customers to order on a self-service kiosk, increasing order throughput.
Staff Order Screens	Allows staff to see current pending orders that need to be prepared. Allows for a more efficient workflow and cuts down on traditional paper tickets used in kitchen environments.
Collection Screens	Displays order numbers and statuses, showing which orders are ready to collect at the collect counter. It allows for a higher throughput of orders and a designated wait area.
Point of Sale Terminal	Allows staff to input orders into the system.

5 Deployment and Production

5.1 Infrastructure

There are several ways to deploy the application and its infrastructure. For smaller projects, a common choice is a platform as a service (PaaS) like Heroku. PaaS simplifies deployment by removing the need to manage servers; developers just upload code and configure settings through a simple interface. However, this convenience comes at a higher cost.

While ideal for smaller apps, PaaS can become expensive and limiting as the system grows and needs more services (Perry, 2024).

In such cases, self-hosting can offer a more flexible and cost-effective alternative. Though setup requires time to configure servers, services, and security, it often results in lower long-term costs. This project uses Linode, a cloud hosting provider offering virtual machines, removing the burden of physical server maintenance while keeping full control over the environment.

An infrastructure diagram is shown in Figure 24, highlighting the system's complexity and range of services.

5.1.1 Overview of Infrastructure

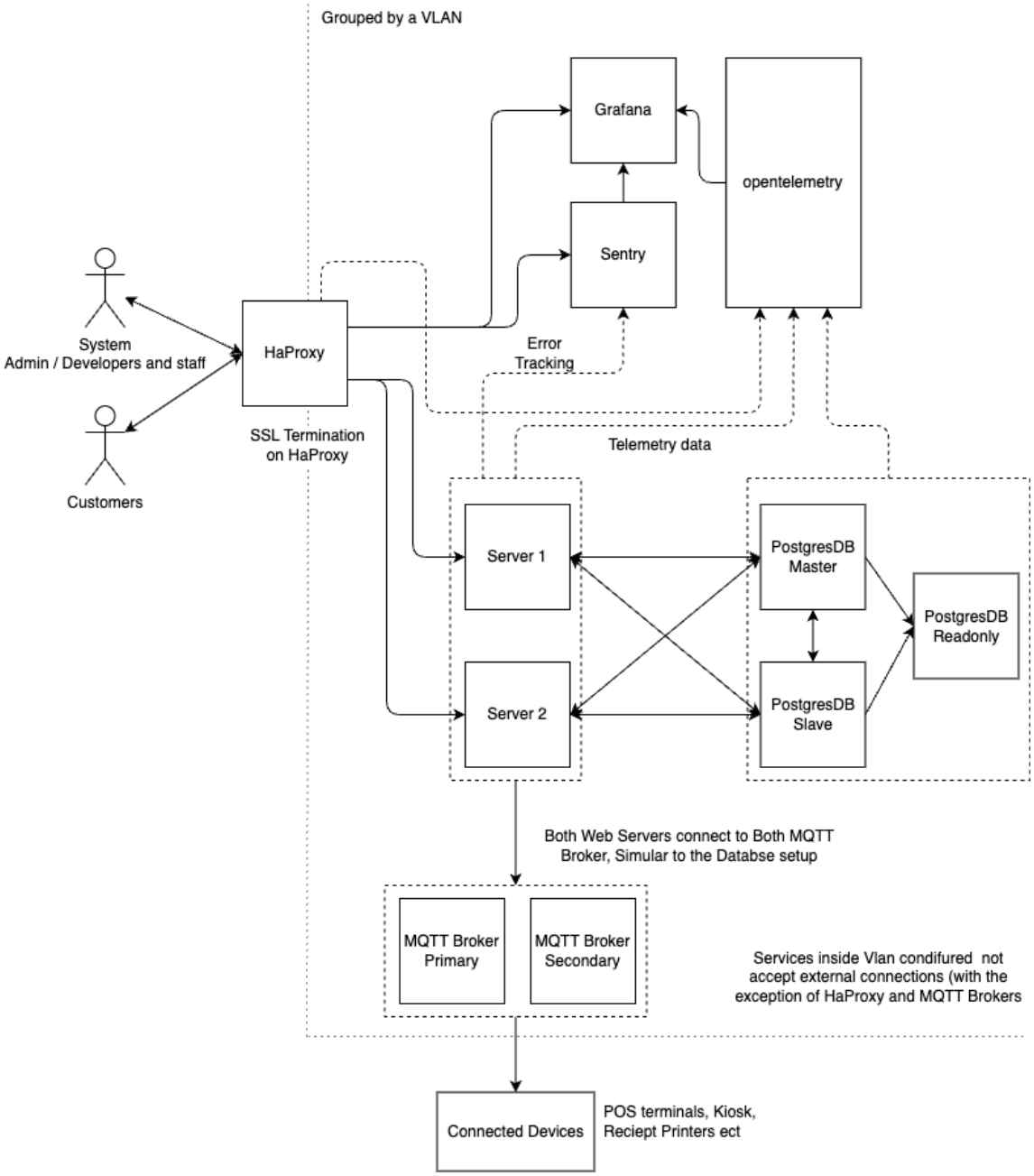

Figure 24. Infrastructure Illustrated

5.1.2 Hosting and Cost

Cost can be hard to calculate depending on the hosting provider, PaaS, the level of redundancy in the system, availability, and the load the system can handle. To analyse and evaluate the cost associated with the infrastructure, Linode and Heroku will be compared based on similar attributes.

Both Linode and Heroku show pricing in USD, and as both do this, there will be no conversion to GBP. The following prices show the total cost per month.

Linodes offer different types of “Linode” (a server), with either dedicated or shared CPU resources and the amounts of RAM shown in Table 5.

Table 5, Linode's Pricing List for Dedicated Linodes (*Linode Pricing*, n.d.)

Server Name (Linode)	Price per month	Price Per Hour	Ram	CPUs	Storage	Transfer	Network In / Out
Dedicated 4 GB	$36	$0.05	4 GB	2	80 GB	4 TB	40 Gbps / 4 Gbps
Dedicated 8 GB	$72	$0.11	8 GB	4	160 GB	5 TB	40 Gbps / 5 Gbps
Dedicated 16 GB	$144	$0.22	16 GB	8	320 GB	6 TB	40 Gbps / 6 Gbps
Dedicated 32 GB	$288	$0.43	32 GB	16	640 GB	7 TB	40 Gbps / 7 Gbps
Dedicated 64 GB	$576	$0.86	64 GB	32	1280 GB	8 TB	40 Gbps / 8 Gbps
Dedicated 96 GB	$864	$1.30	96 GB	48	1920 GB	9 TB	40 Gbps / 9 Gbps
Dedicated 128 GB	$1,152	$1.73	128 GB	50	2500 GB	10 TB	40 Gbps / 10 Gbps
Dedicated 256 GB	$2,304	$3.46	256 GB	56	5000 GB	11 TB	40 Gbps / 11 Gbps

Table 6, Current Infrastructure Cost Breakdown on Linode

Service	Server Choice	Price Per Unit	Qty	Total
Web Server	Dedicated 16 GB	$144.00	2	$288.00
HaProxy	Dedicated 4 GB	$36.00	1	$36.00
Postgresql Database	Dedicated 4 GB	$36.00	2	$72.00
Read-Only Postgresql Database	Linode 2 GB	$12.00	1	$12.00
MQTT Broker	Linode 2 GB	$12.00	2	$24.00
Grafana	Linode 2 GB	$12.00	1	$12.00

Total	$444.00

The primary cost is the web servers, as these need to handle multiple concurrent requests quickly with little downtime. The databases also need to have high availability but don't need as many resources as the web servers. All other supporting services don't handle user requests and, as such, don't need as many resources, which makes them cheaper to run.

Heroku's servers are called "Dynos" and offer add-ons such as databases and other services. This differs from Linode, where everything runs on a server.

Table 7, Current Infrastructure Cost Breakdown on Heroku

Service	Server Choice	Price Per Unit	Qty	Total
Web Server	Performance-M	$250.00	2	$500.00
HaProxy	N/A			$0.00
Postgresql Database	Standard 0	$50.00	2	$100.00
Read-Only Postgresql Database	N/A			$0.00
MQTT Broker	Stackhero	$19.00	1	$19.00
Grafana	Graphite	$19.00	1	$19.00
Total				$638.00

HAProxy is not needed on Heroku, as it hosts the load balancers that the application uses. With Heroku, a read-only database isn't used, as the primary purpose of this was to preserve the data and act as a backup, as Heroku provides backups as standard.

The downside with Heroku is the tier system for databases. Standard 0 has a max storage of 64 GB, although this would take a while to reach. Once it hits 64 GB, the next tier with more storage is Standard 2 with 256 GB, and that comes with a cost of $200 a month, taking the total up to $788 a month.

Table 6 and Table 7 do not include Sentry or OpenTelemetry despite being shown in Figure x. This is because OpenTelemetry is a standard to facilitate the transport of metrics and doesn't need a server/service to run, and Sentry is not self-hosted in this application. Both of which also do not have associated costs.

Despite the added benefits that come with Heroku, it is $194 more expensive than Linode. Part of this is the developer simplicity, where everything is configured and works out of the box.

Whereas with Linode, everything has to be set up and configured. Ultimately, the choice was to use Linode for hosting, as it was cheaper and provided more flexibility.

5.1.3 Load balancing

As seen in figure 24, this application has HAProxy acting as an entry point into the application. HAProxy is a high-availability load balancer and proxy that handles incoming requests to a group of servers and then balances the requests and proxies them onto the correct server (Islam, 2024). Balancing the requests is when a collection of servers performing the same function, such as web server 1 and web server 2, get sent a portion of the overall requests. In this system, two servers handle all the requests.

5.1.4 Redundancy

Redundancy is an important part of a high-traffic, high-availability application, allowing an application to stay available, even when a part of the infrastructure fails and becomes unavailable. The infrastructure, illustrated in figure 25 and figure 26, has been designed with redundancy and availability as the highest priority.

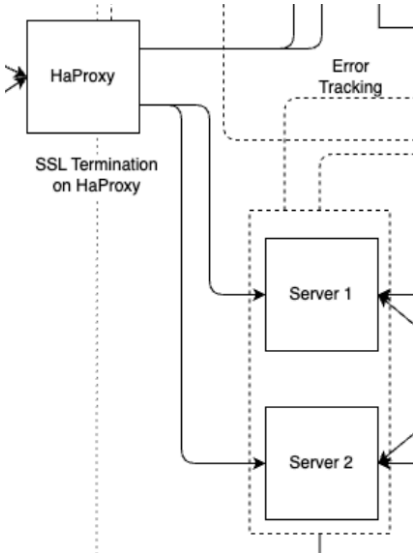

Figure 25, Two Server Redundancy Illustration

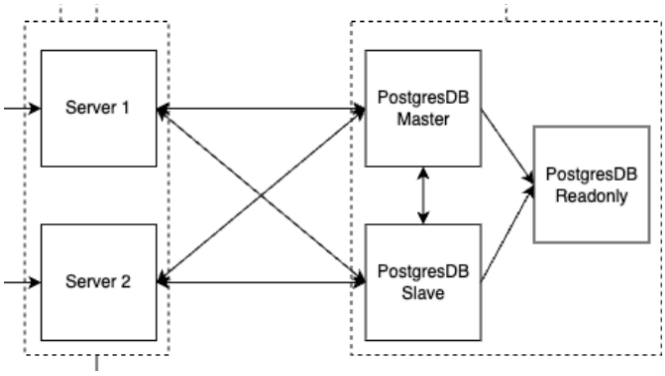

Figure 26, Database Redundancy Illustration

Alpaca_Peruleon_Cafe - database.yml

```yaml
production:
  primary:
    database: primary_database
    username: root
    password: <%= ENV['ROOT_PASSWORD'] %>
    adapter: postgresql
    encoding: unicode
  primary_replica:
    database: primary_database
    username: root
    password: <%= ENV['ROOT_READONLY'] %>
    adapter: postgresql
    replica: true
    encoding: unicode
  secondary_replica:
    database: secondary_database
    username: readonly_user
    password: <%= ENV['SECONDARY_READONLY_PASSWORD'] %>
    adapter: postgresql
    replica: true
    encoding: unicode
```

Figure 27, Database Configuration

The database cluster is set up to provide a suitable level of redundancy for the system. Each web server has connectivity to all three database instances, which are all held in different data centres in Linode. PostgreSQL provides support for database clusters, failover, and replication (*PostgreSQL Documentation, High Availability, Load Balancing, and Replication*, 2025), which is utilised in this system. If one database goes down, the web servers will automatically change connection to secondary, and PostgreSQL will promote the secondary to primary. Figure 27 shows the configuration in database.yml within the application.

If both primary and secondary database servers become unavailable, the application will switch over to a read-only mode. While this does stop customers and staff from changing anything on the system, it preserves the data and allows it to be accessed.

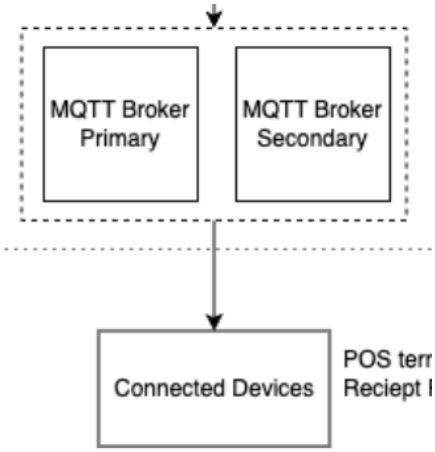

Figure 28, MQTT Redundancy Illustration

As MQTT extends the functionality of the system to connect connected devices, this is also designed and implemented with redundancy. There are two MQTT brokers running in parallel, located in different data centres on Linode. Both brokers have connectivity to both web servers, splitting the workload. If one broker becomes unavailable, traffic will start using the remaining MQTT broker exclusively. This setup mimics the database configuration. An example of the setup can be seen in Figure 28.

5.2 Monitoring and Reporting

Monitoring an application when in production is critical for ensuring its reliability, performance, and availability. It allows key metrics to be tracked, such as response times, error rates, and system resource utilisation. This enables proactive identification of performance issues and errors, leading to an improved experience for the user and a higher availability application, as well as quicker rectification of errors. Figure 29 illustrates the monitoring infrastructure of the system.

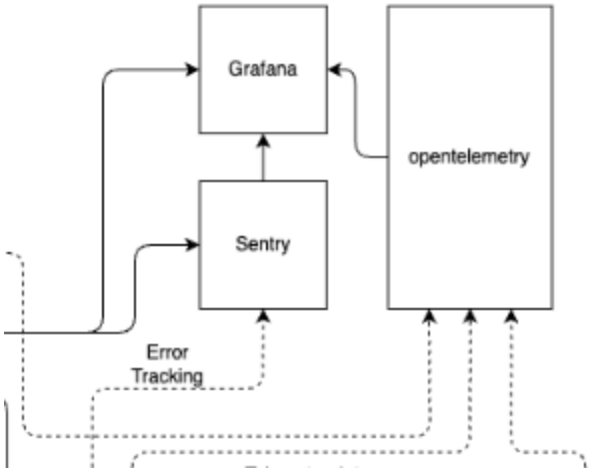

Figure 29, Application Monitoring illustration

5.2.1 Sentry

To adequately monitor the application and the other systems while in production, several tools have been deployed. One of these tools is Sentry. Sentry monitors an application and reports on a lot of different aspects of a Rails application, such as warnings, errors, logs, database transactions, load times, user misery, and a few more. Figure 30 shows the Sentry dashboard for current errors.

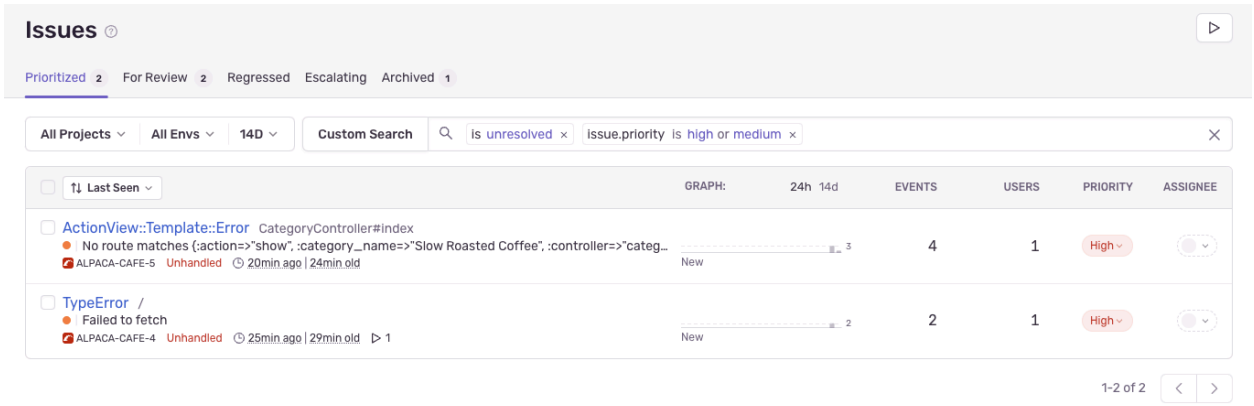

Figure 30, Sentry Error Dashboard

Sentry is used within this project for ongoing monitoring, error tracking and alerting. Successfully identifying a missing template (view), which is seen in figure 30.

5.2.2 OpenTelemetry

OpenTelemetry is an open-source tracing and monitoring framework used to provide visibility of an application or system. It uses a set of standardised protocols and tools for collecting and

routing telemetry data towards a vendor-agnostic background, such as Jaeger, Zipkin, or Prometheus.

OpenTelemetry has been integrated within the Rails application, HAProxy, database, MQTT broker, and certain connected devices. Essentially providing a system-wide visibility to all the inner workings.

The benefits of using OpenTelemetry are that it provides a unified, standardised approach to collecting and analysing telemetry data across all components of a system. This enables deeper visibility into system performance, simplifies debugging and root cause analysis, and enhances reliability by identifying potential issues before they impact users. Additionally, its vendor-agnostic nature ensures flexibility in choosing backend observability tools without being locked into a specific ecosystem. Figure 31 shows an example of OpenTelemetry Prometheus metrics on https request durations.

```
ruby_collector_working 1


# HELP ruby_collector_rss total memory used by collector process
# TYPE ruby_collector_rss gauge
ruby_collector_rss 31002624


# HELP ruby_collector_metrics_total Total metrics processed by exporter web.
# TYPE ruby_collector_metrics_total counter
ruby_collector_metrics_total 36


# HELP ruby_collector_sessions_total Total send_metric sessions processed by exporter web.
# TYPE ruby_collector_sessions_total counter
ruby_collector_sessions_total 7


# HELP ruby_collector_bad_metrics_total Total mis-handled metrics by collector.
# TYPE ruby_collector_bad_metrics_total counter
ruby_collector_bad_metrics_total 0

# HELP ruby_http_requests_total Total HTTP requests from web app.
# TYPE ruby_http_requests_total counter
ruby_http_requests_total{action="index",controller="home",status="200"} 1

# HELP ruby_http_request_duration_seconds Time spent in HTTP reqs in seconds.
# TYPE ruby_http_request_duration_seconds summary
ruby_http_request_duration_seconds{action="index",controller="home",quantile="0.99"} 0.14123324118554592
ruby_http_request_duration_seconds{action="index",controller="home",quantile="0.9"} 0.14123324118554592
ruby_http_request_duration_seconds{action="index",controller="home",quantile="0.5"} 0.14123324118554592
ruby_http_request_duration_seconds{action="index",controller="home",quantile="0.1"} 0.14123324118554592
ruby_http_request_duration_seconds{action="index",controller="home",quantile="0.01"} 0.14123324118554592
ruby_http_request_duration_seconds_sum{action="index",controller="home"} 0.14123324118554592
ruby_http_request_duration_seconds_count{action="index",controller="home"} 1

# HELP ruby_http_request_redis_duration_seconds Time spent in HTTP reqs in Redis, in seconds.
# TYPE ruby_http_request_redis_duration_seconds summary
```

Figure 31, OpenTelemetry Prometheus Metrics

Traces represent the complete journey of a request as it moves through a system, while spans are individual units of work within a trace, capturing specific operations and their dependencies. Together, they provide a detailed view of request flow and performance across distributed services.

5.2.3 Grafana

To provide more value to the information collected by Sentry and OpenTelemetry, Grafana has been used to display the metrics in visual representations and graphs to provide an overall idea of the system and application and to provide a more user-friendly approach. Grafana is an open-source analytics and interactive visualisation web application. It can produce charts, graphs, and alerts for an application or system.

This allows aspects of a system to be visualised, identifying areas for concern and improvement, as well as representing the overall success of an implemented system and application.

Figure 32, Dashboard Illustrating Orders and Stores

In addition to visualising an overall system performance through graphs, which can be seen in figure 33, it can also be used to show business performance and analytics, illustrating trends, user behaviour and revenue. Figure 32 shows a Grafana dashboard representing total revenue, the revenue split between stores, the trend of orders and the order split by stores. Grafana can take data from multiple sources; in this instance, this is direct for the Postgres database, allowing for several different aspects of the database and records to be visualised.

Figure 33, Dashboard Illustrating Infrastructure Metrics

5.2.4 Up-Time Monitoring

While the other technologies in the monitoring stack revolve around reporting, metrics and logs, which are useful for long-term evaluation of the system and quick error fixes, monitoring the uptime of the application and getting alerted instantly when systems go down is even more important. Uptime Robot is used to monitor the uptime and availability of the system.

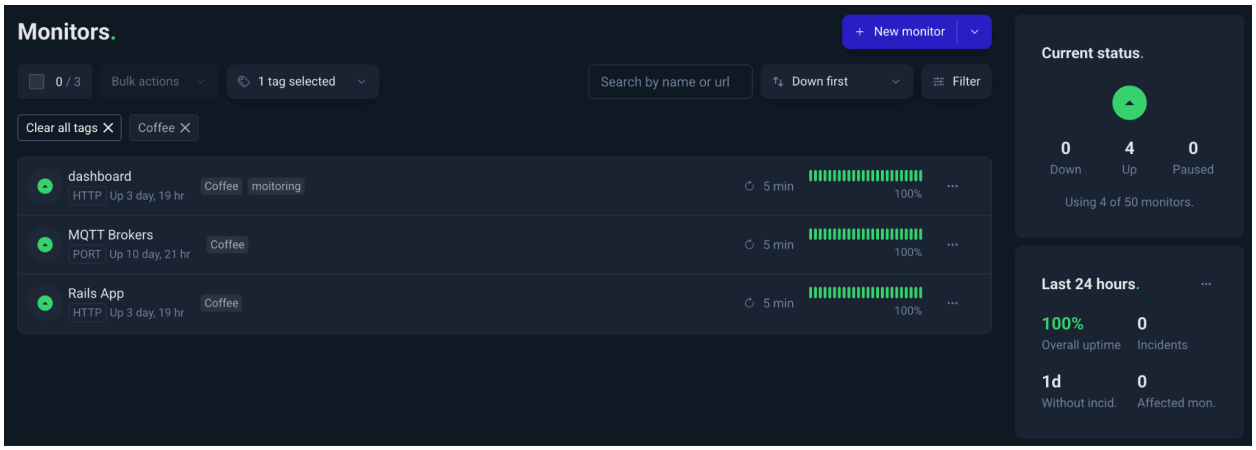

Figure 34, Uptime Monitoring Dashboard

Figure 34 shows the uptime monitoring dashboard on Uptime Robot for the Rails app, the MQTT broker and the Grafana dashboard.

5.3 Performance

Performance is an important part of a system intended to be the main point of sale for a business. To meet the performance targets of the system, multiple different solutions have been implemented to reduce load and compute time.

One of these solutions is well-structured SQL queries within the application. SQL is the backbone of all applications, especially a market app like Alpaca Cafe. Being mindful and structuring SQL queries can reduce load times. Improperly or poorly written SQL queries can increase load time (Nuriev et al., 2024).

Many performance-related improvements come down to how logic is structured and implemented within the code, with no one-size-fits-all approach. Being mindful of how the code is written can significantly impact performance (Tornhill & Borg, 2022). This includes minimising large data queries, reducing unnecessary operations, and optimising loops and conditions. Small inefficiencies, when repeated at scale, can lead to noticeable slowdowns. Taking the time to structure logic carefully can result in better load times and a smoother overall experience.

5.4 Scalability

While scalability was not originally planned or in the original requirements, it is still an important aspect of the overall system to consider. Scalability is the ability to dynamically scale to meet increasing demands quickly. This may be an increase in traffic to the application, where the originally provisioned two web servers become overloaded, and additional servers are needed to handle demand.

There are multiple ways scalability can be achieved within an information system, normally utilising container technologies like Docker or Kubernetes, with an orchestrator that can dynamically launch additional instances of the application when needed. A high-level example of Kubernetes being implemented can be seen in Figure 35.

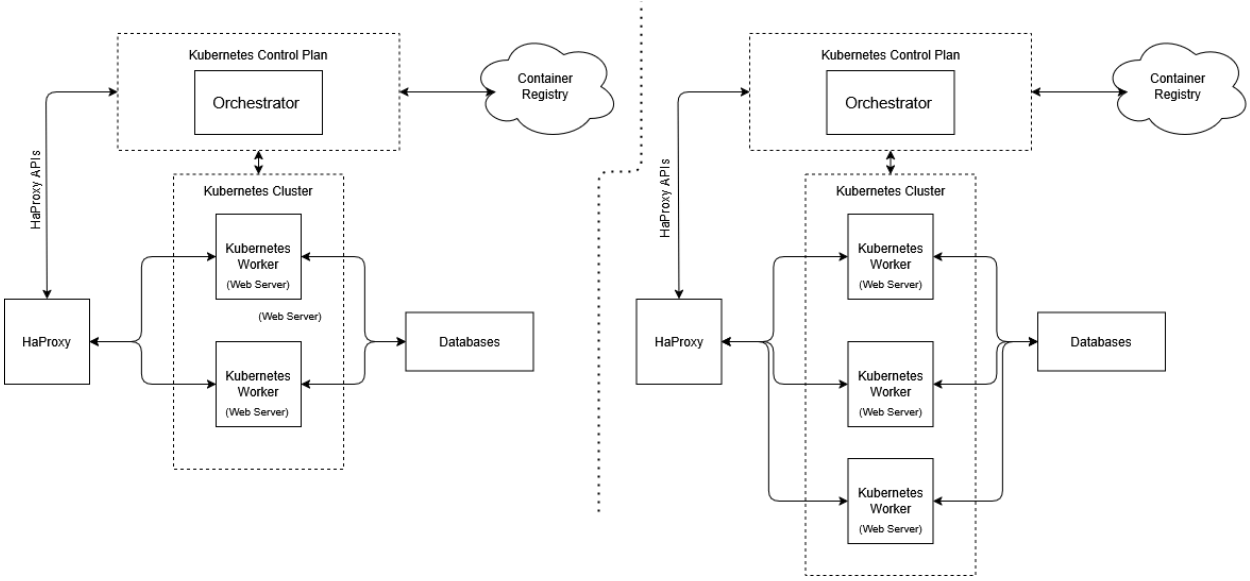

Figure 35, High-Level Kubernetes Example

PostgreSQL is a high-availability database, being able to scale horizontally with additional instances (Ahmed, 2024). That also comes configured with data replication that handles the syncing of data across multiple database servers.

5.5 Security

5.5.1 VLANS

Virtual Local Area Networks (VLANs) are logical groupings of devices that allow a single physical network to be segmented into multiple isolated networks. VLANs have been utilised in this project to enhance security by limiting access and isolating parts of the system from each other to prevent unauthorised communication.

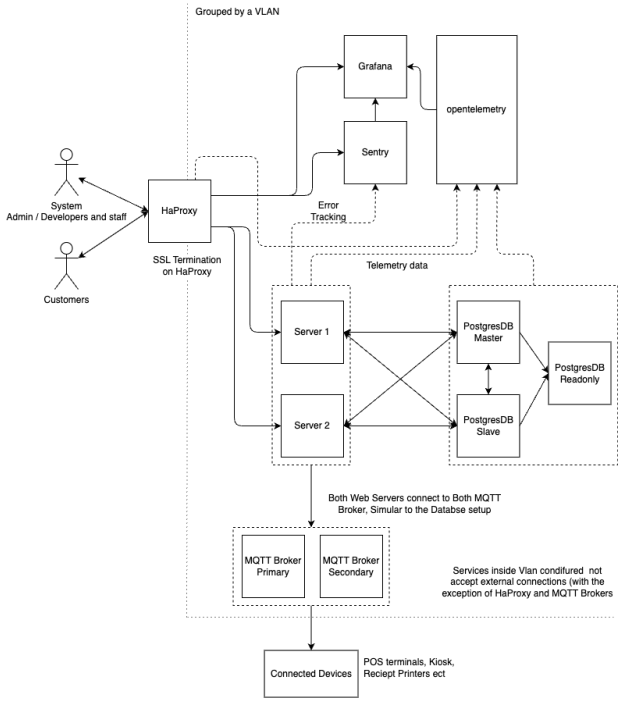

Figure 36, Infrastructure Overview Re-shown

As outlined in section 5.1 and shown again in Figure 36, all incoming traffic is routed through the HAProxy load balancer, which proxies requests to the web application instances. No external traffic should reach any other service directly. To enforce this, the application is hosted within a VLAN, with internal networking rules ensuring only services inside the VLAN can communicate.

This setup centralises traffic control through HAProxy, improving visibility and reducing the attack surface by blocking any traffic that bypasses the load balancer.

The only exception, detailed in section 5.5.2 (Firewall Policies and Proxies), is the MQTT broker, which is strictly limited to secure traffic on port 8883.

5.5.2 Firewalls, Policies, and Proxies

Table 8 shows the firewall policies for the networking side of the project, representing the different components in the system and which traffic they can accept.

Table 8, Networking Firewall Policies

Component	Traffic Allowed	Purpose
General VLAN	Internal traffic within Va LAN	Allows necessary system communication while blocking external access
SSH Access	SSH requests using SSH keys only (password login disabled)	Prevents brute-force attacks while enabling developer access for maintenance
HaProxy	Accepts all requests on port 443 (HTTPS SSL)	Serves as the entry point for the entire system
Web Servers	Traffic to/from Stripe (whitelisted domains, per Appendix X)	Allows direct transaction status checks without routing through HAProxy
MQTT Brokers	Traffic from authenticated devices with verified Stripe transactions	Ensures only genuine, paid orders can be processed
Cloudflare	Legitimate traffic (filters bots, analyses suspicious activity)	Protects against known bad IP addresses and DDoS attacks

These firewall policies follow a block-all approach, only allowing necessary connections to mitigate brute force attacks, man-in-the-middle attacks, and any unauthorised traffic that could be malicious.

5.5.3 User Permissions

User permissions within the application follow role-based access control (RBAC), where a user account is assigned different roles based on their access needs, which control the different aspects of the system that the user is entitled to access. Policies attached to resources, records and controllers dictate which users are allowed to access and what actions they can perform. More in Appendix 10.

6 Evaluation

6.1 Accessibility

During the development and planning of the application, a big emphasis was placed on accessibility. Efforts were made to ensure that the interface is usable for as many people as possible, including those with visual, motor, or cognitive impairments. This included the use of high-contrast colour schemes, scalable text, and clear visual hierarchy. Semantic HTML and ARIA attributes were implemented to support screen readers and other assistive technologies (MDN, 2025). Keyboard navigation was tested to ensure that all interactive elements are accessible without a mouse. These steps help to meet accessibility standards and promote inclusive design.

6.2 Fit for purpose

6.2.1 Performance

Performance is a key aspect of any system and can be measured in many different ways, each focusing on different metrics. On the facing side of the application, load speed and responsiveness are objectively the most important (Welland, 2025). Figure 37 shows the load speed of the mobile version of the application home screen utilising Google Chrome's developer tools.

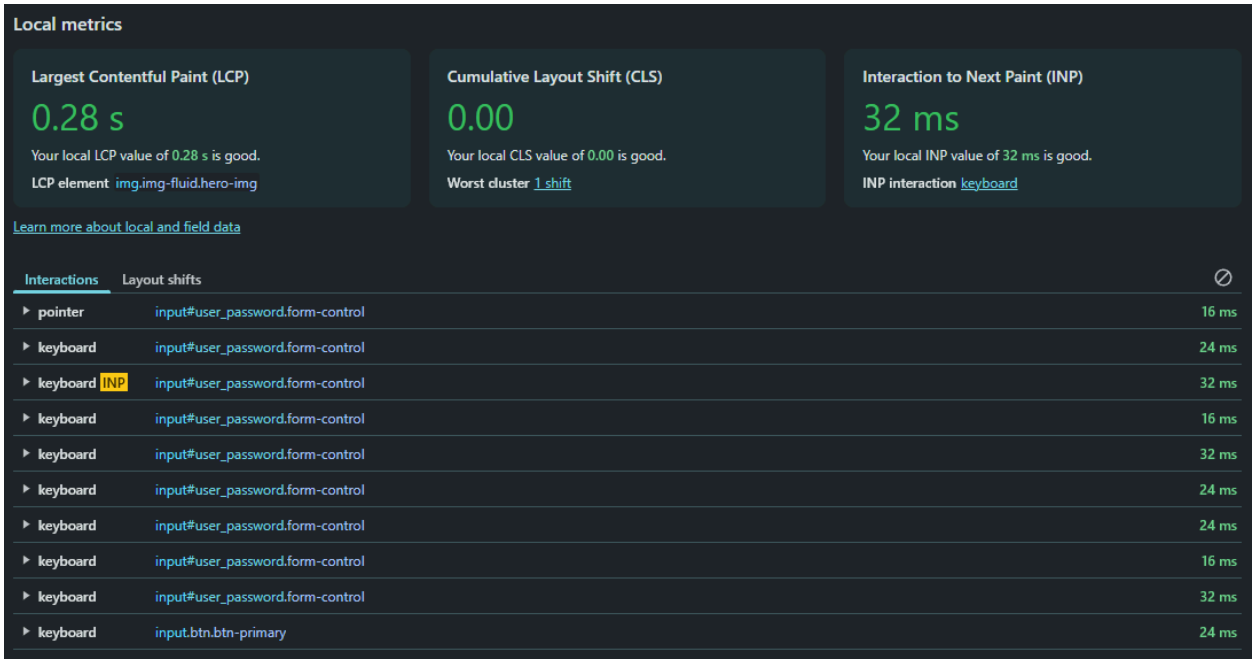

Figure 37, Performance Report from Google's Developer Tools

This report shows the responsiveness of the application to users inputs and actions. All responses are fast, and there is little delay.

6.2.2 Requirements

All requirements set out in section 3.2 have been achieved. Table 9, seen below, Which shows the requirements, the description, the MoSCoW rating and if the particular requirement has been achieved.

Table 9, Achieved Requirements

Requirement	Description	MoSCoW Rating	Achieved?
Secure Payment integration with Stripe	Integrate with Stripe to process payments from customers for online and in-person orders. Supporting multiple payment methods and types. While also complying with FC.	Must	Yes
Google Recommendation AI integration	Integrate with Google recommendations. AI to provide customers with tailored recommendations, increase upsells by offering higher-priced items, and encourage additional purchases with sides, cakes and pastries.	Must	Yes
Selecting between stores to order	Support multiple store locations where customers can order from any location.	Must	Yes
Product Customization	Allow for products to be customised and ingredients to be substituted with other, like ingredients. Encourage upselling to add-ons.	Should	Yes
Availability of products and ingredients	Implement product and ingredient availability on a store level, and prevent customers from ordering unavailable options; upsell and suggest other products instead.	Must	Yes
Automated Emails	Implement automated email sending for order confirmations, account sign-ups and password recovery, as well as a newsletter.	Could	Yes
Admin Management	Admin management dashboard and controls over stores and the application allow management to manage the store, stock levels, orders, availability, and add and remove products. Along with statistics dashboards. Including admin management for accounts. Creating a segregated and secure area for management.	Should	Yes
User Roles and Permissions	Implement policy-based authentication to Secure the application, follow the principle of	Must	Yes

	least privilege and create appropriate roles and suitable levels of access.		
Connected Devices	Set up a network of connected devices for kiosks, point of service terminals (POS), and receipt printers using a suitable protocol and the software to connect to the overall system.	Could	Yes

6.2.3 Future Work

Not all proposed or nice-to-have features are able to be developed and implemented within a project timeline. While all the requirements are set out in section 3.2 and have been met, there are still areas that can be expanded, improved and added to.

Connected devices is one area which can be expanded to include new connectivity and connected devices such as kiosks, point-of-sale terminals and order collection screens. While receipt printers were introduced, this was to demonstrate the ability of connected devices and how the system can be expanded.

Other work that could improve the application is the ability to change drink size and charge for additional add-ons. Currently the application only allows for one size drink, and add-ons do not incur extra charges.

7 Conclusion

The development of the Alpaca Cafe hospitality ordering system has demonstrated a practical and scalable solution for mid-sized, multi-location coffee shops seeking greater control and flexibility than typical SaaS platforms provide. By leveraging modern development methodologies, robust technologies, and a modular architecture, the project successfully delivers a feature-rich platform tailored to real-world operational needs.

The system meets its primary objectives: supporting multiple store locations, enabling rich product customisation, and integrating secure payment processing through Stripe. Additionally, the implementation of connected devices using MQTT allows seamless interaction between the central system and physical devices such as kiosks, receipt printers, and staff order screens. Integration with Google Recommendations AI provides intelligent product suggestions, further enhancing the customer experience and driving potential upsell opportunities.

From a technical standpoint, the application follows best practices in security, accessibility, and compliance. Full CI/CD pipelines, automated testing with high code coverage, and proactive monitoring tools like Sentry and Grafana ensure that the system is production-ready and maintainable. The adoption of open standards, adherence to accessibility requirements, and compliance with regulations like the Data Protection Act 2018 and the Payment Services Regulations 2017 reinforce the robustness of the platform.

The infrastructure design, featuring load balancing, redundancy, and monitoring, ensures high availability and scalability, which are critical for any business aiming to grow. Self-hosting on Linode offers a cost-effective yet powerful alternative to platform-as-a-service offerings, balancing operational control with financial sustainability.

In summary, Alpaca Cafe is a comprehensive and well-engineered solution that bridges the gap between rigid SaaS products and expensive enterprise-grade systems. It lays a strong foundation for future enhancements and real-world deployment, providing a clear example of how thoughtful software engineering can directly address specific industry needs.

8 References

3. Protect data at rest and in transit. (n.d.). Retrieved June 24, 2025, from https://www.ncsc.gov.uk/collection/device-security-principles-for-manufacturers/protect-data-at-rest-and-in-transit

17th State of Agile Report | Analyst Reports | Digital.ai. (n.d.). Digital.ai. Retrieved June 24, 2025, from https://digital.ai/resource-center/analyst-reports/state-of-agile-report/

Abrams, L. (2025, June 10). Massive Heroku outage impacts web platforms worldwide. *BleepingComputer*. https://www.bleepingcomputer.com/news/technology/massive-heroku-outage-impacts-web-platforms-worldwide/

Agha, D., Sohail, R., Meghji, A. F., Qaboolio, R., & Bhatti, S. (2023). Test driven development and its impact on program design and software quality: A Systematic literature review. *VAWKUM Transactions on Computer Sciences*, *11*(1), 268–280. https://doi.org/10.21015/vtcs.v11i1.1494

Ahmed, I. (2024, June 11). *Postgres Horizontal Scalability and Vertical Scalability Pathways using High Availability Clusters*. Retrieved June 23, 2025, from https://www.pgedge.com/blog/scaling-postgresql-navigating-horizontal-and-vertical-scalability-pathways

Alexei, A. (2024, November 15). The subtle art of test driven development - Arkan Alexei - medium. *Medium*. https://medium.com/%40arkanalexei/the-subtle-art-of-test-driven-development-e369b86dee39

Alqudah, M., & Razali, R. (2018). An Empirical Study of Scrumban Formation based on the Selection of Scrum and Kanban Practices. *International Journal on Advanced Science Engineering and Information Technology*, *8*(6), 2315–2322. https://doi.org/10.18517/ijaseit.8.6.6566

ARIA - Accessibility | MDN. (2025, June 7). MDN Web Docs. Retrieved June 24, 2025, from https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

Beyond, G. (2024, November 28). *How many companies use agile in 2025? (Key Usage Statistics) - Go beyond*. Which Proxies. Retrieved June 24, 2025, from https://www.go-beyond.biz/data-statistics/how-many-companies-use-agile

Bhuyan, A. P. (2024, October 15). *The importance of code coverage in software testing: benefits, challenges, and best practices*. DEV Community. Retrieved March 17, 2025, from https://dev.to/adityabhuyan/the-importance-of-code-coverage-in-software-testing-benefits-challenges-and-best-practices-fp7

Bigelow, S. J., & Gillis, A. S. (2024, May 10). *RESTful API*. Search App Architecture. Retrieved March 10, 2025, from https://www.techtarget.com/searchapparchitecture/definition/RESTful-API

Chapter 26. High Availability, Load Balancing, and Replication. (2025, May 8). PostgreSQL Documentation. Retrieved June 24, 2025, from https://www.postgresql.org/docs/current/high-availability.html

CI/CD - what is it and why do you need it? | Fortinet. (n.d.). Fortinet. Retrieved June 24, 2025, from https://www.fortinet.com/resources/cyberglossary/ci-cd

Collins, E. (2021, May 18). LaMDA: our breakthrough conversation technology. *Google*. https://blog.google/technology/ai/lamda/

Devineni, S. K. (2020). Version Control Systems (VCS) The pillars of modern software development: Analyzing the past, present, and anticipating future trends. *International Journal of Science and Research (IJSR)*, *9*(12), 1816–1829. https://doi.org/10.21275/sr24127210817

Easterbrook, S. (2011). *McDonald's to shake up food ordering system*. Financial Times. Retrieved March 16, 2025, from https://www.ft.com/content/e28f6864-7f1e-11e0-b239-00144feabdc0?ftcamp=rss#axzz1MXwsDn7i

Edwards, T. (2024, September 28). *Transport for London (TfL) cyber attack: What you need to know*. BBC News. Retrieved March 10, 2025, from https://www.bbc.co.uk/news/articles/ceqn7xng7lpo

El-Said, O. A., & Tall, T. A. (2019a). Studying the factors influencing customers' intention to use self-service kiosks in fast food restaurants. In *Information and Communication Technologies in Tourism* (pp. 206–217). https://doi.org/10.1007/978-3-030-36737-4_17

El-Said, O. A., & Tall, T. A. (Eds.). (2019b). *Studying the factors influencing customers' intention to use self-service kiosks in fast food restaurants*. https://link.springer.com/chapter/10.1007/978-3-030-36737-4_17

Equality Act 2010. (C.15). London: The Stationery Office. (2010).

Firebase services. (n.d.). Google for Developers. Retrieved March 10, 2025, from https://developers.google.com/assistant/console/firebase-services#:~:text=Cloud%20Firestore%20offers%20seamless%20integration,visiting%20the%20Cloud%20Firestore%20docs.

Fowler, M. (2003). *Patterns of enterprise Application Architecture*. Addison-Wesley Professional.

Fucci, D., Erdogmus, H., Turhan, B., Oivo, M., & Juristo, N. (2016). A dissection of the Test-Driven development process: Does it really matter to Test-First or to Test-Last? *IEEE Transactions on Software Engineering*, *43*(7), 597–614. https://doi.org/10.1109/tse.2016.2616877

Gourlay, D. (2025, May 16). *ICO Fines Data Processor £3M: A Wake-Up call*. Website. Retrieved June 15, 2025, from https://www.mfmac.com/insights/data-protection/a-wake-up-call-for-data-processors-ico-issues-3m-fine

Goyal, A. (2025, April 22). How Delivery Apps Are Transforming the Restaurant Industry. *The Evolution of Delivery Apps and Their Impact on Restaurants*. Retrieved April 23, 2025, from https://www.restroworks.com/blog/the-evolution-of-delivery-apps-and-their-impact-on-restaurants

Hanenko, K. (2025, March 17). *Custom PoS systems: key features and benefits | Altamira*. Altamira. https://www.altamira.ai/blog/custom-pos-systems-key-features-and-benefits/

Hansson, D. H. H. (2005). *Ruby on Rails: Compress the complexity of modern web apps*. Ruby on Rails: Compress the Complexity of Modern Web Apps. Retrieved June 15, 2025, from https://rubyonrails.org/doctrine

Herrera, A. (2025, March 3). *Clover vs Square: Best POS Systems Compared*. TechnologyAdvice. https://technologyadvice.com/blog/sales/clover-vs-square

IBM. (2024, December 10). Redis. *What is Redis?* Retrieved January 7, 2025, from https://www.ibm.com/think/topics/redis#:~:text=Redis%20(REmote%20DIctionary%20Server)%20is,disk%20persistence%2C%20and%20high%20availability.

Introduction to Amazon ElastiCache Serverless (1:30). (n.d.). [Video]. Amazon Web Services, Inc. https://aws.amazon.com/pm/elasticache/?trk=b762e965-94e6-4b29-9f6e-9bd706318229&sc_channel=ps&ef_id=Cj0KCQiAvbm7BhC5ARIsAFjwNHuYDefKSKN2KT61nCz2FbRy9EigOcmgDMo10q3phfD55L1Xv7D8OkkaAqJjEALw_wcB:G:s&s_kwcid=AL!4422!3!548413717917!e!!g!!aws%20elasticache!14790609304!129387858124&gclid=Cj0KCQiAvbm7BhC5ARIsAFjwNHuYDefKSKN2KT61nCz2FbRy9EigOcmgDMo10q3phfD55L1Xv7D8OkkaAqJjEALw_wcB

Introduction to AWS Lambda - Serverless compute on Amazon Web Services (3:01). (n.d.). [Video]. Amazon Web Services, Inc. https://aws.amazon.com/pm/lambda/?trk=27324d1f-ee08-40b9-8e7b-5ac228e2fecc&sc_channel=ps&ef_id=Cj0KCQiAvvO7BhC-ARIsAGFyToVGFMDE0fv2eioi5oFTAXLTArass8maiMWkURoKonXDjzgn5Jbgp1waAq2_EALw_wcB:G:s&s_kwcid=AL!4422!3!651612449951!e!!g!!aws%20lambda!19836376234!148728884764&gclid=Cj0KCQiAvvO7BhC-ARIsAGFyToVGFMDE0fv2eioi5oFTAXLTArass8maiMWkURoKonXDjzgn5Jbgp1waAq2_EALw_wcB

Ishak, F. a. C., Lah, N. a. C., Samengon, H., Mohamad, S. F., & Bakar, A. Z. A. (2021). The Implementation of Self-Ordering Kiosks (SOKs): Investigating the challenges in fast food restaurants. *International Journal of Academic Research in Business and Social Sciences*, *11*(10). https://doi.org/10.6007/ijarbss/v11-i10/11491

Islam, M. A. (2024, December 5). *HAProxy: A comprehensive guide to load balancing and proxying*. https://www.linkedin.com/pulse/haproxy-comprehensive-guide-load-balancing-proxying-md-areful-islam-dldxc#:~:text=HAProxy%20(High%20Availability%20Proxy)%20is,those%20hosted%20on%20cloud%20platforms.

Jamesevans. (2024, September 2). *[29 Nov]: Issues with Data Feeds*. TfL Tech Forum. Retrieved December 10, 2024, from https://techforum.tfl.gov.uk/t/updated-29-nov-issues-with-data-feeds/3609

Lindecrantz, E., Gi, M. T. P., & Zerbi, S. (2020, April 28). *Personalizing the customer experience: Driving differentiation in retail*. McKinsey & Company. Retrieved March 10, 2025, from https://www.mckinsey.com/industries/retail/our-insights/personalizing-the-customer-experience-driving-differentiation-in-retail

Linode pricing. (n.d.). Linode. Retrieved March 17, 2025, from https://www.linode.com/pricing/

Lumigo. (2024, August 30). *AWS Lambda Cost Factors, Cost Comparisons and Optimization [2024]*. Retrieved March 10, 2025, from https://lumigo.io/learn/aws-lambda-cost-guide/#:~:text=OpenFaaS%20takes%20a%20m

arkedly%20different%20approach%20to,Lambda%2C%20making%20it%20difficult%20to%20compare%20costs.

Martin, R. C. (2009). Clean code: A Handbook of Agile Software Craftsmanship. Pearson Education.

Mishra, A., & Alzoubi, Y. I. (2023). Structured software development versus agile software development: a comparative analysis. *International Journal of Systems Assurance Engineering and Management*, *14*(4), 1504–1522. https://doi.org/10.1007/s13198-023-01958-5

Morell, L. (2022, December 1). *The impact of the COVID-19 pandemic on online food delivery apps*. Digital Scholarship@UNLV. Retrieved April 23, 2025, from https://digitalscholarship.unlv.edu/thesesdissertations/4549/

Nuriev, M., Zaripova, R., Sinicin, A., Chupaev, A., & Shkinderov, M. (2024). Enhancing database performance through SQL optimization, parallel processing and GPU integration. *BIO Web of Conferences*, *113*, 04010. https://doi.org/10.1051/bioconf/202411304010

Perry, M. (2024, July 14). *The hidden costs of PaaS: everything you should know - Qovery*. Retrieved June 24, 2025, from https://www.qovery.com/blog/the-hidden-costs-of-paas-everything-you-should-know/

Point of sale API and POS integration tools. (n.d.). Square Developer. https://developer.squareup.com/docs/pos-api/what-it-does?#requirements-and-limitations

Power your entire business | Square. (n.d.). Square. https://squareup.com/

Ramsingh, A., Singer, J., & Trinder, P. (2022). Do fewer tiers mean fewer tears? Eliminating web stack components to improve interoperability. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2207.08019

Ramzan, H. A., Ramzan, S., & Kalsum, T. (2024). Test-Driven Development (TDD) in small software development teams: Advantages and challenges. *Researchgate*, 1–5. https://doi.org/10.1109/icacs60934.2024.10473291

Recommendations AI. (n.d.). Google Cloud. Retrieved March 10, 2025, from https://cloud.google.com/use-cases/recommendations?hl=en

Self-hosting serverless with OpenFaaS. (n.d.). Mskog. Retrieved March 10, 2025, from https://www.mskog.com/posts/self-hosting-serverless-with-openfaas

Serverless Function, FaaS Serverless - AWS Lambda - AWS. (n.d.). Amazon Web Services, Inc. Retrieved March 10, 2025, from https://aws.amazon.com/lambda/

Shiwen, L., Kwon, J., & Ahn, J. (2021). Self-Service technology in the hospitality and tourism Settings: A Critical Review of the literature. *Journal of Hospitality & Tourism Research*, *46*(6), 1220–1236. https://doi.org/10.1177/1096348020987633

SKIFT + ORACLE HOSPITALITY. (n.d.). *Hospitality in 2025: Automated, intelligent. . . and more personal*. https://www.oracle.com/a/ocom/docs/industries/hospitality/hospitality-industry-trends-for-2025.pdf

Smplfy. (2025, May 17). PayPal vs Adyen: 8 Critical Reasons One Platform Dominates Global Payments - Smplfy. *Smplfy*. https://smplfy.co/paypal-vs-adyen-platform-offers-global-reach/

Stack Overflow. (2024). Https://survey.stackoverflow.co/2024/technology. *Stack Overflow*. Retrieved June 17, 2025, from https://survey.stackoverflow.co/2024/technology

Tamburri, D. A., Kazman, R., & Fahimi, H. (2020). Organisational structure patterns in agile Teams: An Industrial Empirical study. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2004.07509

The Public Sector bodies (Websites and Mobile Applications) (No. 2) Accessibility Regulations 2018 (SI 2018/852) (Revoked). London: The Stationery Office. (n.d.).

Tillster. (2024, November 26). *Consumer demand for restaurant kiosks on the rise — Tillster*. Tillster. Retrieved June 6, 2025, from https://www.tillster.com/blog/2020/2/20/consumer-demand-for-restaurant-kiosks-on-the-rise

Tornhill, A., & Borg, M. (2022). Code Red: The Business Impact of code Quality -- A quantitative study of 39 proprietary production codebases. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2203.04374

Wakode, R. B., Raut, L. P., & Talmale, P. (2015). Overview on Kanban Methodology and its Implementation. *IJSRD : International Journal for Scientific Research and Development*, *3*(2), 2518–2521. http://www.ijsrd.com/articles/IJSRDV3I2771.pdf

Web Content Accessibility Guidelines (WCAG) 2.1. (2025, May 6). https://www.w3.org/TR/WCAG21/

Welland, M. (2025, June 3). *Your slow page load speed could be losing customers and money*. Niteco. Retrieved June 15, 2025, from https://niteco.com/articles/why-website-performance-load-speed-important

What is Flutter? - Flutter App Explained - AWS. (n.d.). Amazon Web Services, Inc. Retrieved March 10, 2025, from https://aws.amazon.com/what-is/flutter/

Wiggers, K. (2023, April 18). Reddit will begin charging for access to its API. *TechCrunch*. https://techcrunch.com/2023/04/18/reddit-will-begin-charging-for-access-to-its-ap

Xavier. (2025, February 26). *The Economic Impact of Self-Service Kiosks: Unleash your restaurant's profit potential*. GAAP Point-of-Sale. Retrieved March 16, 2025, from https://www.gaap.co.za/the-economic-impact-of-self-service-kiosks-unleash-your-restaurants-profit-potential/

9 Appendices

Appendix 1, Design Wireframes

Appendix 2, Gems Used

The project has utilised many Ruby on Rails gems to provide additional functionality with minimal developer time spent. Theses gems, which is not a complete list, are as follows:

Gem	Description
Stripe	Gem developed by Stripe to provide payment functionality within a web application, wrapping API calls into nice Rails-friendly methods.
Devise	Extends Rails warden functionality and provides Ruby's user authentication, such as signing up, logging in, two-factor authentication, password resets and more.
Bootstrap	A Sass gem that provides prebuilt CSS and JavaScript components to speed up styling.
Sentry Rails	Provides a hook into sentry, the error monitoring where it extends rails error catch functionality and sends errors and stack traces to the sentry dashboard
Capistrano	Used to deploy the application onto different hosts, such as production, from the command line.
Opentelemetry	Used to track and store telemetry data in rails and send it to the dashboard.

Appendix 3, Manual Testing

As mentioned in section 4.4 Testing, the project closely followed the practice of test-driven development, TDD, where test conditions are written before the code to ensure code is written to satisfy the test. As TDD was adopted early in the development, the code was continuously tested and evaluated to meet the requirements and to catch bugs and edge cases early. The TDD approach, testing and test coverage have already been discussed in 4.4.1 and 4.4.3, respectively, and will not be discussed further in this section.

In addition to utilising TDD, a small group of test users were brought in to use and test the application. This provided a feedback channel of real users to improve non-logic areas of the application, such as UI/UX design and user flows. In addition to finding errors when users

perform unexpected actions. Getting hands-on testing from users is a great testing technique towards the end of the development cycle.

Manual testing plan

Test ID	Test Description	Expected Outcome
T001	Verify the user can launch the app and reach the home screen.	The app loads correctly, and the home screen is displayed.
T002	Test user login with valid and invalid credentials.	Valid login redirects to dashboard; invalid shows error.
T003	Place an order with multiple customisations (milk, size, extras).	Custom orders are accepted and confirmed with correct pricing.
T004	Navigate to previous orders and reorder an item.	Order history is displayed; reorder populates the current basket.
T005	Ensure payment flow works with valid card details.	Payment is processed securely, and confirmation is shown.
T006	Test offline behaviour and error handling when the network is unavailable.	The app shows an offline state and prompts retry without crashing.
T007	Check navigation across all app tabs (Menu, Orders, Profile, Cart).	All navigation tabs are functional and respond correctly.
T008	Verify that a user can edit and save profile details successfully.	Profile details are updated and reflected across the app.
T009	Ensure the app sends push notifications for order status updates.	The user receives timely and accurate push notifications.
T010	Test logout functionality and ensure session data is cleared.	The user is logged out, and sensitive data is cleared from memory.

Following the above, below is the summary of the test performed with any issues and suggestions for improvements, with the addition of other issues found during testing that are not outlined in the table x test plan.

Summary of Manual Testing

User ID	Role	Test Tasks Performed	Issues Found	Suggested Improvements
U001	Casual User	Placed a coffee order, customised milk & size, checked out	Button text unclear on “Review Order” screen	Rename to “Confirm Order” for clarity
U002	Staff Member	Navigated admin order dashboard	Confused by filter options and date selection UI	Add date picker and tooltips

U003	Power User	Used mobile pre-order flow	Error message not shown when skipping required drink option	Add field validation with instant feedback
U004	First-time user	Used kiosk to order and cancel	No back button from confirmation screen	Add a “Go Back” option on confirmation screen
U005	Accessibility Test	Used screen reader & keyboard navigation	Some buttons unlabelled, and tab order inconsistent	Add aria-labels and logical tab index
U006	Elderly user	Ordered via kiosk and paid	Text size too small and there is poor contrast on button labels	Increase text size and use higher contrast colours

Appendix 4, Stripe Qualified Domains

```
a.stripecdn.com,
api.stripe.com,
atlas.stripe.com,
auth.stripe.com,
b.stripecdn.com,
billing.stripe.com,
buy.stripe.com,
c.stripecdn.com,
checkout.stripe.com,
climate.stripe.com,
connect.stripe.com,
dashboard.stripe.com,
express.stripe.com,
f.stripecdn.com,
files.stripe.com,
hcaptcha.com,
hooks.stripe.com,
invoice.stripe.com,
invoicedata.stripe.com,
js.stripe.com
m.stripe.com,
m.stripe.network,
manage.stripe.com,
```

```
merchant-ui-api.stripe.com,
pay.stripe.com,
payments.stripe.com,
q.stripe.com, qr.stripe.com,
r.stripe.com, stripe.com,
terminal.stripe.com,
uploads.stripe.com,
verify.stripe.com
```

https://docs.stripe.com/ips?locale=en-GB

Appendix 5, Stripe Payment Service

Uni-Rails-1 [WSL: Ubuntu] - stripe_paymentintent_service.rb

```ruby
# frozen_string_literal: true

require 'stripe'

class StripePaymentintentService
  def initialize(amount, current_user, order)
    @amount = amount
    @current_user = current_user
    @order = order
  end

  def create
    Stripe::PaymentIntent.create(create_params)
  end

  def update(payment_intent_id)
    Stripe::PaymentIntent.update(
      payment_intent_id,
      update_params
    )
  end

  def status
    Stripe::PaymentIntent.retrieve(
      @order.payment_intent
    ).status
  end

  def capture
    Stripe::PaymentIntent.capture(
      @order.payment_intent
    )
  end

  private

  def retrieve; end

  def create_params
    {
      amount: @amount,
      currency: 'gbp',
      receipt_email: @order.user.email,
      customer: @order.user.stripe_id,
      automatic_payment_methods: { enabled: true },
      setup_future_usage: 'off_session',
      capture_method: 'manual'
    }.compact
  end

  def update_params
    {
      amount: @amount,
      currency: 'gbp',
      receipt_email: @order.user.email
    }.compact
  end
end

```

Also available at:
https://github.com/TidalCub/Alpaca-Cafe/blob/main/app/services/stripe_paymentintent_service.rb

Appendix 6, Google Tag Service


```ruby
# frozen_string_literal: true

require 'net/http'
require 'json'
require 'googleauth'

class GoogleRetailTagService
  GOOGLE_API_URL = 'https://retail.googleapis.com/v2/projects/aplaca-cafe/locations/global/catalogs/default_catalog/userEvents:write'
  SERVICE_ACCOUNT_JSON = Rails.root.join('config/google_service_account.json')

  def initialize(user)
    @user = user
    @access_token = fetch_access_token
  end

  def new_event(event_type, product, qty = nil)
    return unless @access_token

    send_request(request_body(event_type, product, qty))
  end

  def home_event
    return unless @access_token

    send_request(home_request_body)
  end

  private

  def send_request(request_body)
    uri = URI(GOOGLE_API_URL)
    request = Net::HTTP::Post.new(uri)
    request['Authorization'] = "Bearer #{@access_token}"
    request['Content-Type'] = 'application/json; charset=utf-8'
    request.body = request_body

    response = Net::HTTP.start(uri.hostname, uri.port, use_ssl: true) do |http|
      http.request(request)
    end

    log_response(response)
  end

  def home_request_body
    {
      eventType: 'home-page-view',
      visitorId: @user.id.to_s,
      sessionId: SecureRandom.uuid,
      eventTime: Time.now.utc.iso8601,
      userInfo: {
        userId: @user.id.to_s
      }
    }.compact.to_json
  end

  # rubocop:disable Metrics/MethodLength
  def request_body(event_type, product, qty)
    {
      eventType: event_type,
      visitorId: @user.id.to_s,
      sessionId: SecureRandom.uuid,
      eventTime: Time.now.utc.iso8601,
      productDetails: [{
        product: { id: product.id.to_s },
        quantity: qty
      }],
      userInfo: {
        userId: @user.id.to_s
      }
    }.compact.to_json
  end
  # rubocop:enable Metrics/MethodLength

  def fetch_access_token
    scope = 'https://www.googleapis.com/auth/cloud-platform'
    authorizer = Google::Auth::ServiceAccountCredentials.make_creds(
      json_key_io: File.open(SERVICE_ACCOUNT_JSON),
      scope: scope
    )
    authorizer.fetch_access_token!['access_token']
  rescue StandardError => e
    Rails.logger.error "Failed to fetch Google Retail API access token: #{e.message}"
    nil
  end

  def log_response(response)
    if response.code.to_i == 200
      Rails.logger.info "\n\n\n\n\n\n\nSuccessfully sent event to Google Retail API: #{response.body}"
    else
      Rails.logger.error "\n\n\n\n\n\n\nFailed to send event: #{response.code} - #{response.body}"
    end
  end
end
```

Also available at:
https://github.com/TidalCub/Alpaca-Cafe/blob/main/app/services/google_retail_tag_service.rb

Appendix 7, Product Catalogue

The product catalogue is too long to be shown in full here. The product catalogue is available here: https://github.com/TidalCub/Alpaca-Cafe/blob/main/products.json.

Appendix 8, Config Manager

AlpacaPoS - config_manager.py

```python
from cryptography.fernet import Fernet
import json

class Config_Manager:
  def __init__(self):
    self.key = None
    self.load_key()
    self.host = "localhost"
    self.port = 1883
    self.username = "None"
    self.password = "None"
    self.pos_type = "printer"
    self.config = None

  def load_key(self):
    try:
      with open("master.key", "rb") as key_file:
        key = key_file.read()
        self.key = Fernet(key)
    except (FileNotFoundError):
      print(f"There is no {filepath} found. Please add the key file")

  def create_config(self, config_json):
    self.config
    return json.dumps(config_json)

  def encrypt(self, config):
    try:
      config_json = self.create_config(config).encode()
      encrypted_config = self.key.encrypt(config_json)
      with open("config.json.enc", "wb") as enc_file:
        enc_file.write(encrypted_config)
      print("File encrypted successfully")
    except Exception as e:
      print(f"An error occurred while encrypting the file {e}")

  def decrypt(self):
    try:
      with open("config.json.enc", "rb") as enc_file:
        encrypted_config = enc_file.read()
      decrypted_config = self.key.decrypt(encrypted_config)
      return decrypted_config.decode()
    except Exception as e:
      print(f"An error occurred while decrypting the file {e}")

  def load_config(self):
    try:
      decrypted_config = self.decrypt()
      data = json.loads(decrypted_config)
      self.host = str(data['MQTT_BROKER']['host'])
      self.port = int(data['MQTT_BROKER']['port'])
      self.username = str(data['authentication']['username'])
      self.password = str(data['authentication']['password'])
      return True
    except Exception as e:
      print(f"An error occurred while loading the config {e}")

if __name__ == "__main__":
  Config_Manager().decrypt()
```

Also available at https://github.com/TidalCub/AlpacaPoS/blob/main/config_manager.py

Appendix 9, Setup Wizard

The Setup wizard allows non-technical staff to configure devices quickly by entering the MQTT broker's host, port, username, and password. The wizard then performs a test connection to ensure successful communication with the broker before encrypting the credentials into config.json.enc, making them securely available for the device's application.

AlpacaPoS [WSL: Ubuntu] - setup_wizard_CL.py

```python
import json
import socket
import paho.mqtt.client as mqtt
from config_manager import Config_Manager

CONFIG_FILE = "config.json"

def get_input(prompt, default=""):
    value = input(f"{prompt} [{default}]: ").strip()
    return value if value else default

def test_mqtt_connection(host, port, username="", password=""):
    client = mqtt.Client()
    client.username_pw_set(username, password)

    try:
        client.connect(host, int(port), 60)
        client.loop_start()
        print("Connection successful!")
        return True
    except Exception as e:
        print(f"Connection failed: {e}")
        return False

def setup_wizard():
    print("Welcome to the Setup Wizard!\n")

    # Step 1: Get MQTT Configuration
    host = get_input("Enter MQTT Host", "localhost")
    port = get_input("Enter MQTT Port", "1883")
    username = get_input("Enter MQTT Username", "")
    password = get_input("Enter MQTT Password", "")

    # Step 2: Test Connection
    if not test_mqtt_connection(host, port):
        print("Warning: Connection failed, but continuing...\n")

    # Step 3: Select Device Type
    device_types = ["Printer", "POS Terminal", "Order Screen", "Kitchen Display", "Kiosk"]

    print("\nSelect a device type:")
    for i, device in enumerate(device_types, 1):
        print(f"{i}. {device}")

    while True:
        choice = input("\nEnter choice (1-5): ").strip()
        if choice.isdigit() and 1 <= int(choice) <= len(device_types):
            device_type = device_types[int(choice) - 1]
            break
        print("Invalid choice, try again.")

    # Step 4: Save Configuration
    config = {
        "authentication": {"username": username, "password": password},
        "MQTT_BROKER": {"host": host, "port": port},
        "device_type": device_type,
    }

    cm = Config_Manager()
    cm.encrypt(config)

    print("\nConfiguration saved successfully in 'config.json.enc'. Setup complete!\n")

if __name__ == "__main__":
    setup_wizard()
```

Appendix 10, Role-Based Access Control

RBAC offers multiple benefits of application security and management. Using RBAC and policies, it allows for more granular permissions, giving access to the users who require different access to resources and is in line with organisational structures, where different job roles have different access in line with a simple approach to access control where there are tiers of users who get more and more access, which may not be relevant to their job function. The table shows an example of roles, the description and the typical user that would have the role.

Access Control Roles

Role Name	Description	Typical User
Customer	A normal customer who would be ordering on the app.	A normal customer
Barista	The employee who would make the orders	The Barista
Store Manager	Needed more access to information related to a store, such as orders, order statistics, revenue, and control over product and ingredient availability	The Store manager
Admin	Needs access to the technical and administrative side of the business, access to statistics, revenue information, and store information, but not able to control stock availability on a store level	Office staff, corporate staff, regional manager