COM3014

Advanced Challenges in Web Technologies

Group UG-3

Vendur

Cloud-Native Retail Platform

Team Members

Aran Jannson
Alexander Kyriacou
Thomas Rouse
William Bang
Noah Esguerra
Ismaeel Anjum

Contents

1 Requirements and Problem Statement **2**

1.1 Problem Statement 2

1.2 Requirements 4

2 Architecture **6**

2.1 Overview 6

2.2 Microservices 6

2.3 Database and Backend 7

2.4 Version Control with Git 9

2.5 Containerisation with Docker 10

2.6 Summary of Benefits 10

3 Teamwork **11**

3.1 Team Organisation and Communication 11

3.2 Work Distribution 12

3.3 Conflict Management 12

4 Implementation **14**

4.1 Use of Cloud-Native Software Development Processes 14

4.1.1 Microservices 14

4.1.2 Github 19

4.1.3 Docker 21

4.2 Testing 23

5 Learning from Class and from Good Examples **27**

5.1 Class Concepts 27

5.2 Amazon 27

5.3 Shopify 27

5.4 eBay 28

6 Roadmap and Business Model **29**

6.1 Start-up Description 29

6.2 Project Motivation 29

6.3 Vision and Future Plans 29

6.4 Target Market Group (TMG) 30

6.5 Competitive Analysis 30

6.6 Key Stakeholders 31

6.7 Ethical and Legal Considerations 31

7 Conclusion **32**

A Testing Evidence **35**

Submission URLs:

Public Git Repository:

- https://github.com/AranJannson/Vendur

Hosted Website:

- https://www.vendur.shop
- https://vendur.shop

Page Routes:

- `/auth/signin`
- `/auth/signup`
- `/basket`
- `/basket/checkout`
- `/basket/checkout/success`
- `/handler/[...stack]`
- `/orders`
- `/organisations`
- `/organisations/[shop]`
- `/organisations/create`
- `/organisations/management`
- `/organisations/management/[item_id]/edit`
- `/organisations/management/add`
- `/products`
- `/products/[item]`
- `/search`

Declaration on AI-Generated Content:

Some of the example items used in the development and testing of the database for this project were generated using artificial intelligence tools. These items were created for illustrative purposes only, to simulate realistic data for functional and non-functional testing of system features. No real user data or commercially sensitive information has been included.

The use of AI-generated examples was conducted ethically and in accordance with the guidelines of the University of Surrey and relevant academic standards, ensuring that the synthetic data did not impact the validity or originality of the project outcomes.

1 Requirements and Problem Statement

1.1 Problem Statement

In the increasingly competitive digital commerce landscape, small to medium-sized businesses (SMBs) face significant challenges in establishing and maintaining a meaningful online retail presence. Unlike large retail enterprises, these businesses often lack tools that enable them to manage and optimise an e-commerce platform that can meet ever-evolving customer expectations.

Meanwhile, modern consumers demand smooth and fast online shopping experiences, including intuitive product browsing and secure checkout and payment. Failure to meet these standards can often lead to lost sales opportunities and hurts the overall reputation and presence of brands in a highly saturated market.

This is further amplified by the operational complexity of providing a technically sound and responsive platform that meets these needs. In order to analyse sales trends and respond to customer feedback, many smaller organisations are forced to develop sub-par systems, or rely on platforms such as social media, which are not purpose-built retail solutions. This not only negatively impacts customer experience, but poses a significant security risk as well.

To approach these challenges, we proposed Vendur; a modular, scalable, and robust retail platform with cloud-native capabilities, and an emphasis on allowing organisations to manage their business intuitively and efficiently. The key services we aimed to provide to organisations were:

- List and categorise products with flexible promotion management.
- Securely process online payments through an integrated basket and checkout system.
- Analyse key business metrics through an accessible real-time analytics dashboard.
- Interact with customers efficiently through item reviews.
- Streamline administrative processes such as verification and profile management.

Our solution focuses not only on solving the immediate operational needs of small businesses, but also on future-proofing their retail operations. Through a modular and scalable microservice architecture, Vendur will allow organisations to adapt dynamically as their needs grow.

Ultimately, Vendur aims to provide smaller organisations with the tools necessary to compete in the digital marketplace, foster customer loyalty, and support sustainable business growth.

Below is a summary of our problems, solutions, and motivations behind them.

Problem	**Solution**	**Motivation**
Small businesses struggle to establish an online retail presence.	Develop an accessible online retail platform for small to medium-sized organisations.	Enable smaller organisations to compete in digital commerce and improve visibility and customer engagement.
Consumers expect effortless shopping and secure payments.	Build a platform that supports product browsing and secure payment processing.	Meet modern customer expectations to improve conversion rates and loyalty.
Organisations face operational inefficiencies using fragmented tools and manual processes.	Centralise product listing, order management, analytics, and customer interaction features into one platform.	Increase operational efficiency and provide a smooth experience for businesses and customers alike.
Existing e-commerce solutions can be costly, rigid, and hard to customise.	Design a modular, microservice-based architecture to allow flexibility, scalability, and gradual enhancements.	Allow businesses to adapt and grow the system affordably without needing to migrate platforms.
Managing customer feedback across multiple platforms is challenging.	Implement structured feedback and reply features directly within the platform.	Improve customer relations and allow businesses to manage reputations effectively from one dashboard.

Table 1: Summary of Problems, Solutions, and Motivations for Vendur

1.2 Requirements

The functional requirements for the Vendur platform define the core features and operations that our system had to support to meet user and business needs.

ID	Requirement	Justification
F1	Product Catalogue Management	Organisations must be able to add, update, and categorise products easily, including setting up discounts and promotions.
F2	Product Search and Filtering	Customers should be able to efficiently search and filter products based on attributes such as category, price, popularity, and available discounts.
F3	Secure Payment Processing	The platform must securely handle online payments, manage customer baskets effectively, and ensure reliable transaction processing.
F4	Analytics Dashboard	Organisations will have access to a dashboard that provides insights into product popularity, sales trends, seasonal variations, and overall performance metrics.
F5	Organisation Verification and Management	Administrators need tools for verifying organisations, managing reviews and product listings, and performing standard CRUD operations for organisational data.
F6	Customer Interaction and Feedback	Organisations should have the ability to view customer reviews to improve customer engagement and build trust.

Table 2: Functional Requirements

The non-functional requirements for the platform define the quality attributes that ensure a robust, scalable, and user-friendly service. These requirements focus on areas such as system performance, security, usability, and future scalability. Each requirement is justified based on the needs of both customers and organisations.

ID	**Requirement**	**Justification**
N1	The platform should support multiple concurrent users performing actions such as browsing products, managing profiles, and completing transactions.	To provide a seamless shopping experience even during peak usage, the system must handle many users at once without significant performance degradation.
N2	Users should experience minimal delay between logging in and interacting with the platform's features.	Quick access to browsing, checkout, and profile management is important to ensure user satisfaction and reduce bounce rates.
N3	The platform should be compatible with major browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.	Browser compatibility increases accessibility, ensuring that users across different devices and preferences can access the service without technical barriers.
N4	The platform's interface should be clean, simple, and easy to navigate.	A user-friendly design ensures inclusivity and encourages engagement from users of all technical abilities, supporting effective shopping and profile management.
N5	Sensitive user data such as passwords and payment information must be securely encrypted.	Encryption safeguards users' privacy and security, building trust in the platform and complying with data protection regulations.
N6	Storage and compute resources should be scalable to accommodate growing user bases and inventory expansion.	Initially, basic cloud resources will be sufficient, but future deployments must plan for scalable solutions to meet demand without downtime.
N7	The platform should achieve high availability and fault tolerance to avoid critical service interruptions.	Ensuring continuous availability strengthens user trust and supports uninterrupted e-commerce transactions, particularly during peak shopping periods.

Table 3: Non-Functional Requirements

2 Architecture

2.1 Overview

The Vendur application was designed as a modular, cloud-native web platform with a Next.js frontend, and a backend composed of five microservices, each with their own PostgreSQL database. Each microservice provides one key functionality which can be independently scaled, tested, and deployed. The system uses Docker to containerise and deploy microservices across different environments, and a Git repository for version control and to manage parallel development efficiently across the team.

2.2 Microservices

Five microservices comprise the Vendur platform:

- Catalogue - Manages product listings, categories, discounts and filtering options
- Payment service – Handles secure payment processing, basket management, and order confirmation.
- Analytics service – Provides sales, product, and user activity insights to organisations.
- Organisation Management service – Allows organisations to manage their product listings and request verification.
- Admin service – Enables administrators to verify organisations, oversee reviews, and perform CRUD operations on organisational data.

By separating these services, our team was able to isolate faults, independently scale bottleneck services, and introduce new features without impacting existing functionality. This promoted scalability, flexibility, and maintainability in our design.

The microservices are connected as follows:

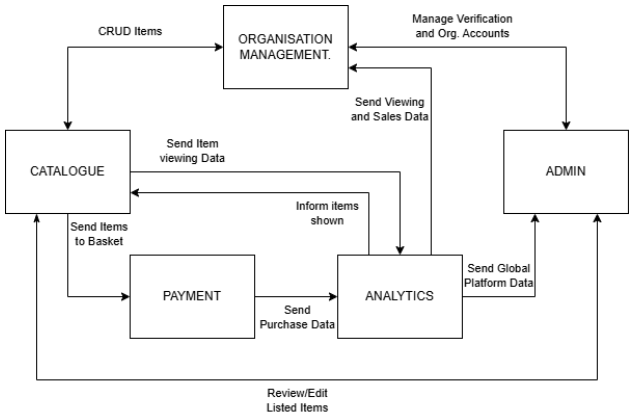

Figure 1: Microservices Overview

2.3 Database and Backend

Each microservice is paired with its own PostgreSQL database instance, each of which is managed via Supabase. This ensures that each microservice only has access to the data necessary for its own function, improving security through data isolation, as one data breach will not compromise the entire system. The backend is built using Express.js [10] with TypeScript [26], providing a robust and strictly typed environment to improve code consistency and mitigate runtime errors.

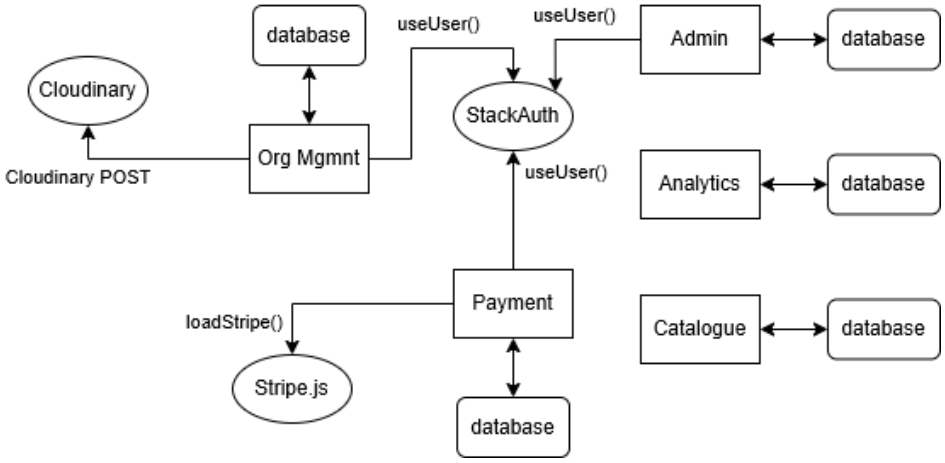

Figure 2: API Flows

The databases had the following architectures:

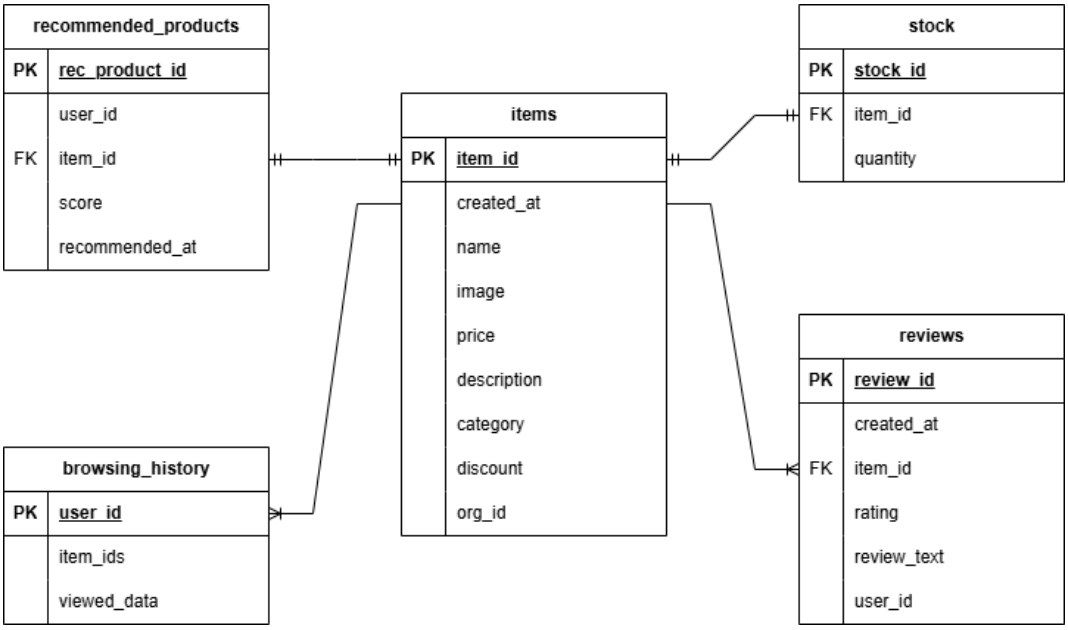

Figure 3: Catalogue Entity-Relationship Diagram

The catalogue database is centred around the `items` table. The `reviews`, `recommendations` and `stock` tables each contain a foreign key of `item_id` to link each object to the item they are related to. The `browsing_history` table contains `item_ids` attribute, which is an array containing one or more `item_id` objects, and one item can have many `browsing_history` objects, creating a many-to-many relationship between them.

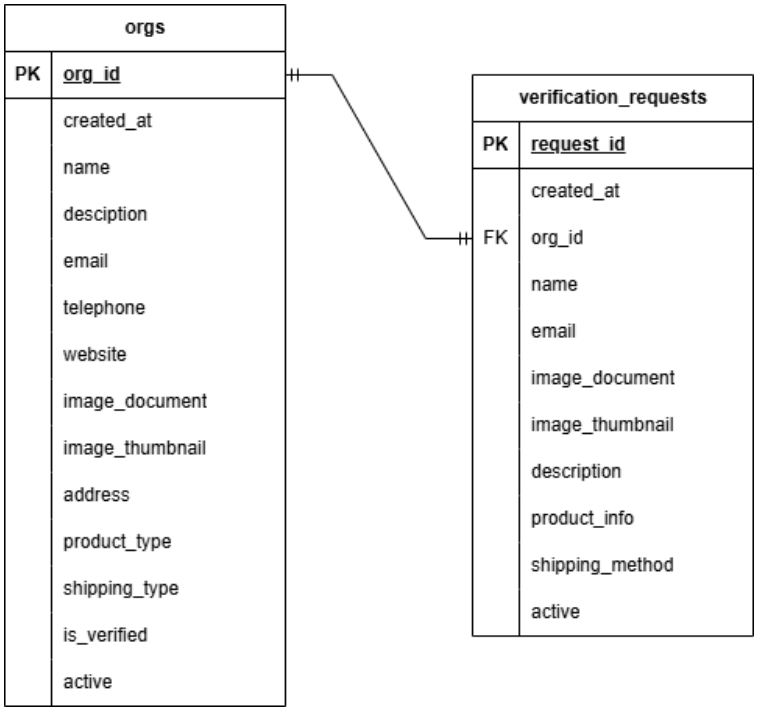

Figure 4: Organisation Management Entity-Relationship Diagram

The organisation management table consists of a table containing all organisations and one containing verification requests. The `verification_requests` table contains a foreign key of `org_id` to link the organisation that made each request. The request's current status of the request is indicated by the `active` attribute.

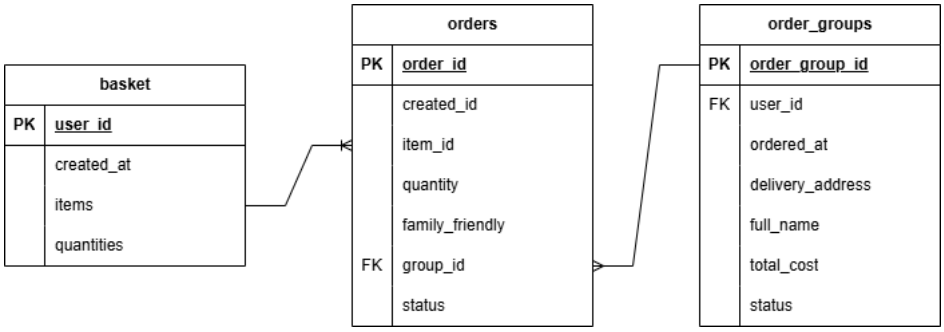

Figure 5: Payment Entity-Relationship Diagram

The payment table is centred around the `orders` table. The `order_groups` table keeps track of which items were ordered together, and the `orders` table contains a foreign key of `group_id` to link each ordered item to the group to which it belongs.

The `basket` table's `items` field stores an array of `item_id` objects from the `orders` table, creating an implicit relationship between the two.

API	**Endpoint/Link**
Cloudinary [8]	https://api.cloudinary.com/v1_1/${process.env.NEXT_PUBLIC_CLOUDINARY_CLOUD_NAME}/image/upload
Stripe.js [22]	loadStripe()
StackAuth [2]	useUser()

Table 4: API endpoints

2.4 Version Control with Git

Git was used as the version management system to track code changes across the project, as this has long been industry best practice. The frontend and each microservice had its own branch in the repository, allowing the team to effectively distribute work and work in parallel without affecting the main branch. We also opted to use Git's review system, meaning merge requests required the approval of two other team members before being integrated. This greatly reduced the likelihood of introducing errors to the live build. The use of Git also allowed us to revert to previous iterations of code if needed. An example of this structure is shown below:

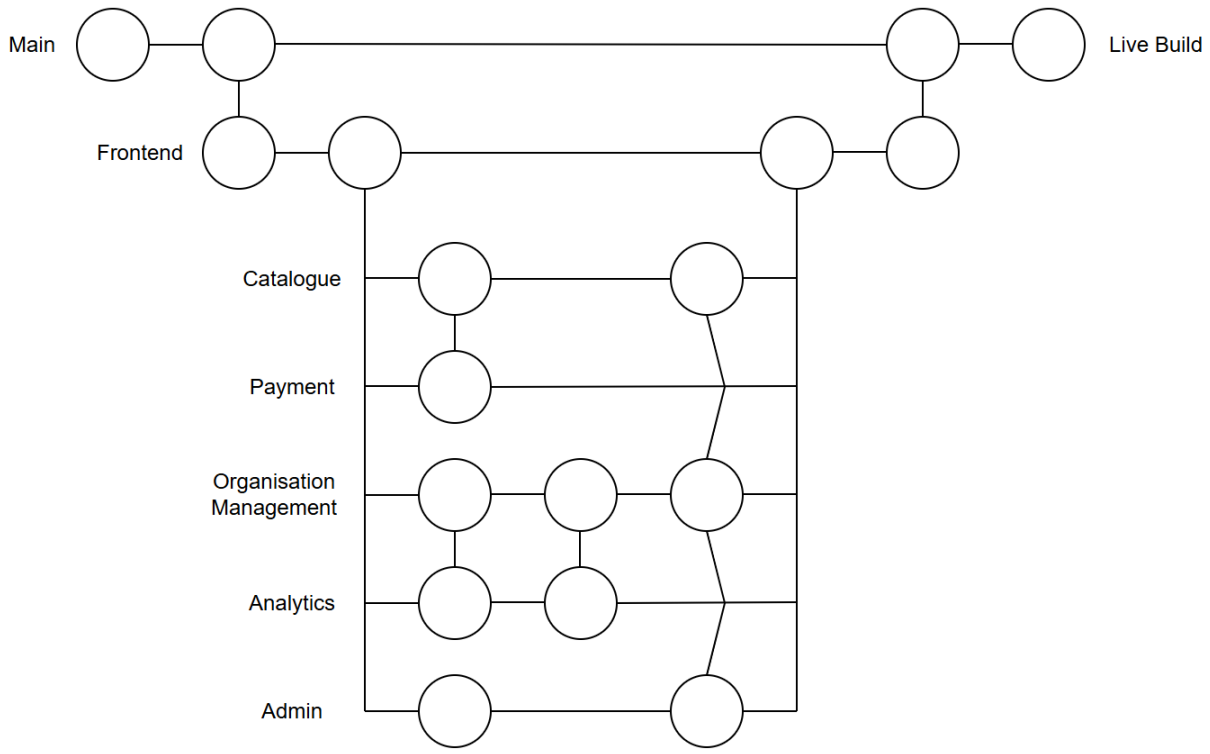

Figure 6: Example Git Workflow Diagram

2.5 Containerisation with Docker

Docker was used to containerise each microservice, packaging all dependencies within each container. This made local development, testing, and deployment consistent across environments and reduced the risk that the system ran on some machines and not others. We used Docker Compose to run the services locally, making this efficient during development. Docker supports our modular design and simplified updates, testing, and scaling of individual services.

2.6 Summary of Benefits

Overall, our architecture ensured that we met the key expectations of cloud-native software. These include:

- Scalability - Features can be scaled independently based on demand
- Resilience - Failures are isolated to specific services, reducing impact across the system
- Security - Each service has controlled API access and separate database
- Maintainability - Independent development and deployment allows for efficient feature updates and debugging

3 Teamwork

3.1 Team Organisation and Communication

From our initial project definition, our team emphasised transparency, consistency, and even contribution. The main means of ensuring this came in holding weekly in-person lab meetings to discuss and implement our respective features together, supporting each another where needed. Between these meetings, we communicated multiple times per week via WhatsApp and Discord chats to ensure an alignment in vision for the various features being worked on, in addition to reporting and resolving issues, discussing ongoing progress and assigning tasks to group members.

We used Trello to manage assigned tasks and monitor progress, which allowed us to visually organise our workflow into clearly defined states such as "To Do," "In Progress," "Under Review" and "Complete." We created a card for each individual task, feature, or fix, and these were assigned to team members with deadlines and labels.

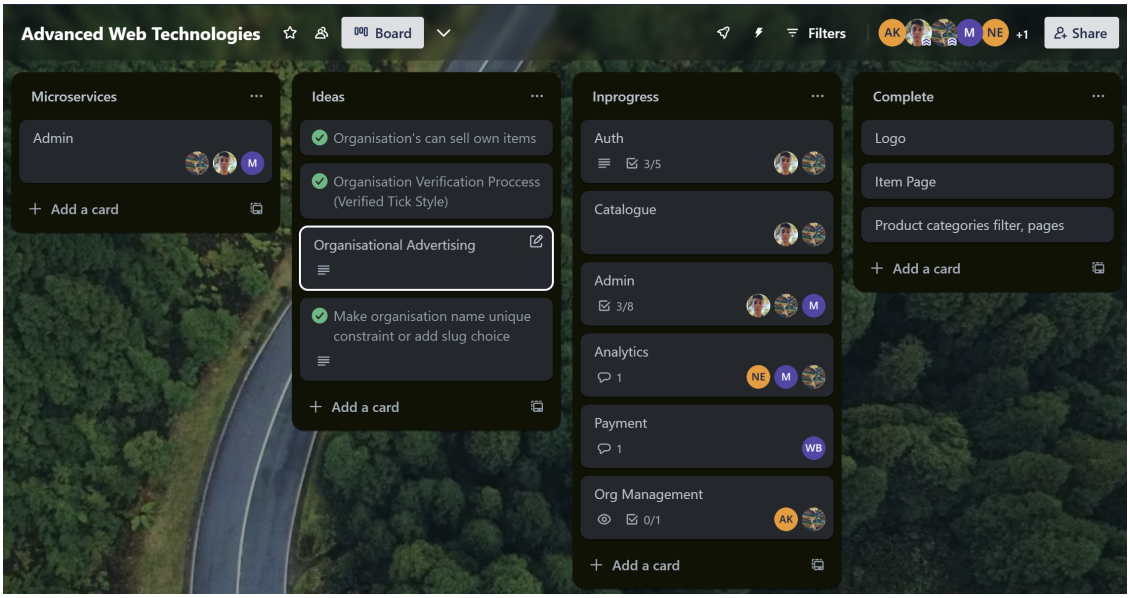

Figure 7: Vendur's Trello Board

This was ideal for our workflow, as it allowed us to explicitly view what tasks were being worked on by which team members and how they were progressing with each. This also helped us deploy support where appropriate, if a specific task were to require more manpower. It also meant that our workflow was centralised and readily visible to every member of the team, without the need for all members of the team to be online at one time. Team members could leave comments on tasks, attach resources, and move cards into the appropriate state, and Trello notifications would be sent to all team members when changes were made, furthering the asynchronous workflow between members, along with visibility and access for the entire team.

3.2 Work Distribution

At the start of the project, each team member was assigned ownership of one microservice based on their interests and skills, along with one secondary service based on expertise and support needed. These secondary tasks also included leading the frontend and shared backend infrastructure, and leading project documentation, though all group members contributed to these segments at some stage. These tasks were assigned as follows:

Team Member	**Responsibility**
Alexander Kyriacou	Organisation Management microservice, project documentation lead
Aran Jannson	Frontend application, Docker containerisation support, Hosting, general microservice backend development
William Bang	Payment microservice, data modelling and logic development
Thomas Rouse	Catalogue microservice, frontend integration support, general backend development
Noah Esguerra	Analytics microservice, statistical logic and graphing
Ismaeel Anjum	Admin microservice, backend integration and API linking

Table 5: Team Members and Responsibilities

3.3 Conflict Management

Although our team worked well throughout the project, we recognised early that in a group project like ours, with differing ideas, priorities, and working styles, conflict was inevitable at some stage. To mitigate the impact of this, we adopted several proactive methods:

- **Establishing Clear Expectations** Before any development took place, we created and signed two team contracts outlining our goals for the Vendur development, as well as the above-stated plans for communication, attendance, deadlines, and collaboration. Each team member signed these contracts, solidifying that we all agreed to these policies and would adhere to them.
- **Respectful and Open Commnunication** We made sure to foster an environment in which everyone felt comfortable expressing their thoughts and concerns. WhatsApp became our main platform for ongoing discussion, with technical collaboration taking place in the Discord server, and any member of the team could make suggestions, ask for support, or challenge decisions constructively. The chats had a professional, but informal and friendly tone to ensure that team members did felt comfortable coming forward with issues or concerns, without becoming frustrated with each other.

- **Active Listening and Issue-Driven Dialogue** During meetings, we encouraged active listening by allowing each person to present their point of view without interruption. We made a point of focusing on issues rather than individuals, preventing blame being placed on members, and keeping conversations productive and moving forward.
- **Consensus-Based Decision Making** In situations with multiple viable solutions, we discussed the positives and trade-offs of each approach as a group before making a decision based on a group poll. This process ensured that all group members felt a level of ownership over the direction of the project.
- **Resolving Issues Diplomatically** If team members temporarily missed an assigned task due to external circumstances, part of their workload would be reassigned for a period of time. We referred to the expectations established and agreed upon, and had respectful conversations to resolve these issues without escalation.

4 Implementation

4.1 Use of Cloud-Native Software Development Processes

4.1.1 Microservices

The Vendur platform adopted a microservice architecture to support the modular, scalable, and fault-tolerant design principles of cloud-native software. Each microservice was containerised with Docker and developed as an isolated, self-contained component that could then be implemented into the main system. This ensured efficient deployment and testing while reducing the risk of errors in the broader system when debugging or introducing features to individual services. Services communicate through RESTful HTTP API routes.

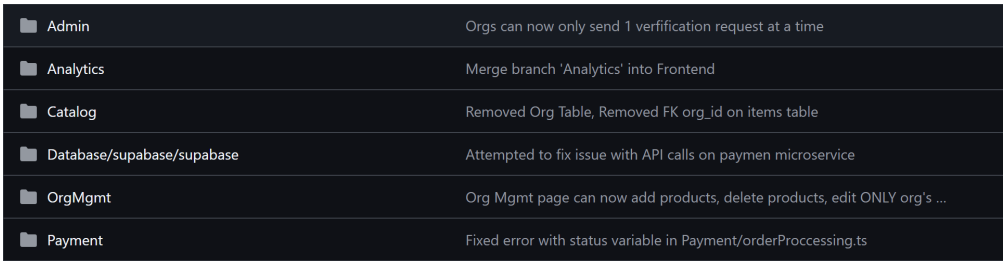

Figure 8: Microservices on Github

Our technology stack was selected to balance effective implementation of our design, familiarity with the team, and industry best practices. With these goals in mind, we opted to implement the backend using Express.js, built on Node.js, with TypeScript [26]. Each microservice had its own PostgreSQL database to ensure complete data isolation and resilience, and these were managed using Supabase CLI [23]. The data analytics features also used this technology to fetch the required data. The frontend was built with Next.js 15 [30], which is itself routed from React 19.

We used shadcn/ui to implement a responsive carousel component on the homepage [20]. This allowed us to showcase featured products with minimal configuration, using Tailwind-compatible styling out of the box.

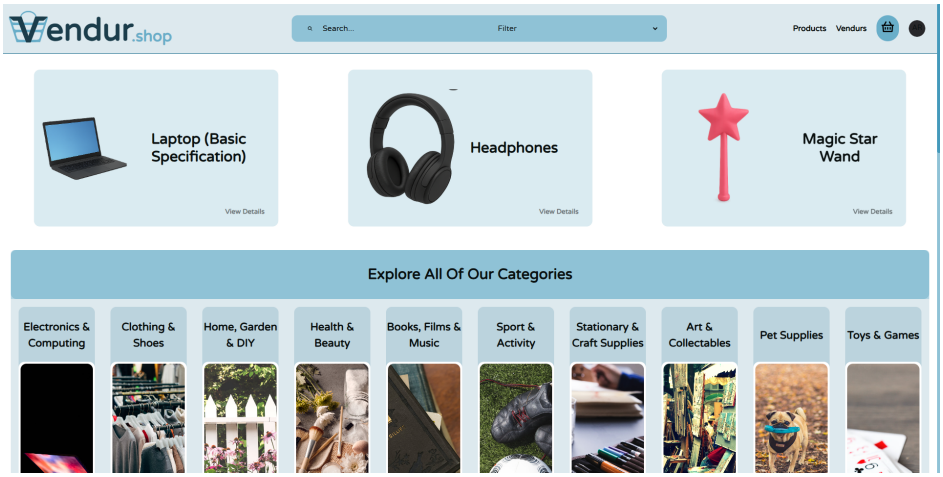

Figure 9: The Homepage for the Vendur Application

We also provided a 'spotlight' page with featured verified organisation pages. This gives SMBs that have been reviewed by the admin team an opportunity to boost customer engagement and sales, as well as garner a positive reputation.

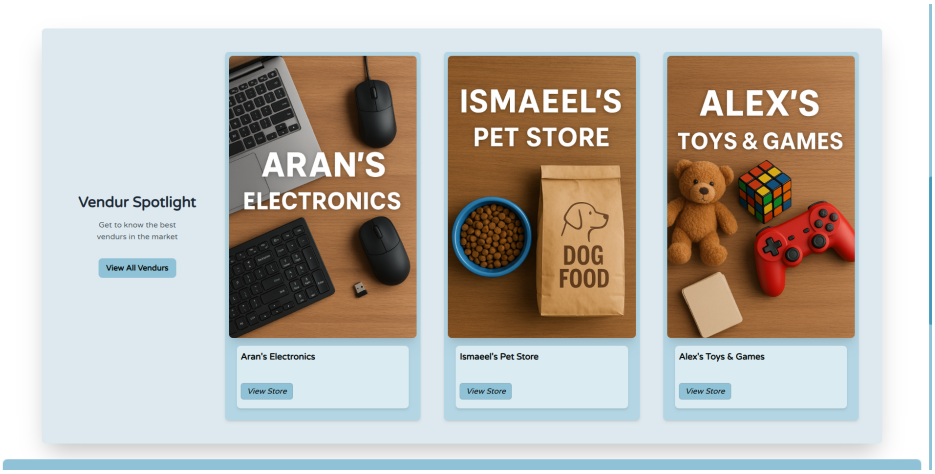

Figure 10: The Vendur Spotlight

For the frontend of the application, Stack Auth [2] was employed to handle authentication. Stack Auth provides an open-source solution with a robust and developer-friendly API, enabling integration with Next.js. Its flexibility and ease of implementation made it a good choice for managing user sessions and securing access to frontend components.

Stack Auth also contains a 'teams' system, which was used to implement organisation accounts. Organisation users belong to a Stack Auth team, and user accounts are given access to the organisation panel for the team to which they belong. The admin team was also implemented this way.

We used Nginx [11] as a reverse proxy to route incoming HTTP requests to our Next.js frontend services. We also made use of PM2 [17] to ensure that the server was consistently running, and handle any crashes. Nginx handles static file serving, SSL termination, and load balancing, while PM2 ensures availability of our backend by managing application processes, enabling automatic restarts on failure, and supporting zero-downtime reloads.

These microservices are broken down as follows:

1. Catalogue Microservice This service manages all product-related data. Sellers can add new items, set prices, apply percentage-based discounts, and manage stock levels. Customers can retrieve all listings, or perform targeted searches using query parameters such as category, price range, discount availability, or keyword. Filtering is performed server-side for performance. This service also exposes endpoints to fetch 'most popular' items, supporting integrations with the analytics service. Items are linked to verified organisations via a foreign key relationship. Products listed by verified organisations are more heavily weighted in the listing algorithm when filtering by popularity.

Items are stored in the database with fields such as title, description, price, discount percentage, and stock count. Each item is linked to a specific organisation through a foreign key, allowing sellers to manage their own listings independently. Sellers can apply discounts to their products, which are reflected in the price calculations on the frontend. Stock is tracked per item and is decremented upon successful checkout via the Payment microservice. These mechanisms ensure real-time inventory management and accurate pricing during the customer shopping experience.

Customers who are logged in can leave reviews for individual items. Each review consists of a star rating from 1 to 5 and an optional comment field to provide feedback. Reviews are tied to specific items and stored in a related table using the item's ID as a foreign key.

2. Payment Microservice The Payment microservice is intended to implement requirement F3, a secure payment processing system. The payment microservice has many features that ensure a secure user experience with robust techniques during implementation, including the use of secure APIs, and sensible authorisation and authentication. The following sections will describe how a functional payment system, yet a user-friendly environment, was implemented to satisfy requirement F3.

Unfortunately, we had a problem with CORS (cross-origin site request) settings that interfered with cookies on the hosted website between sites, which is a security measure to prevent cross-site scripting attacks. Therefore, the intended solution was not used. Instead, we opted to simulate the behaviour of cookies using a table.

After the user has added items to their basket, they can choose to check out. The user can choose from a selection of payment methods, Klarna [1], debit card, to name a few. More details would need to be provided by the user to ensure reliable delivery, specifically delivery and billing addresses. Until payment confirmation, items are held in their basket, and after confirmation, they will be

notified of their payment and shown their order number and intended delivery address.

Initially, cookies were implemented to store the user's current state, even in the event of website termination, and to fluidly manage data across multiple users concurrently with cookie-parser [15]. Although cookie-parser offers complete customisation of secure sending using options such as `httpOnly: true` and `secure: true`, it mitigates data interception.

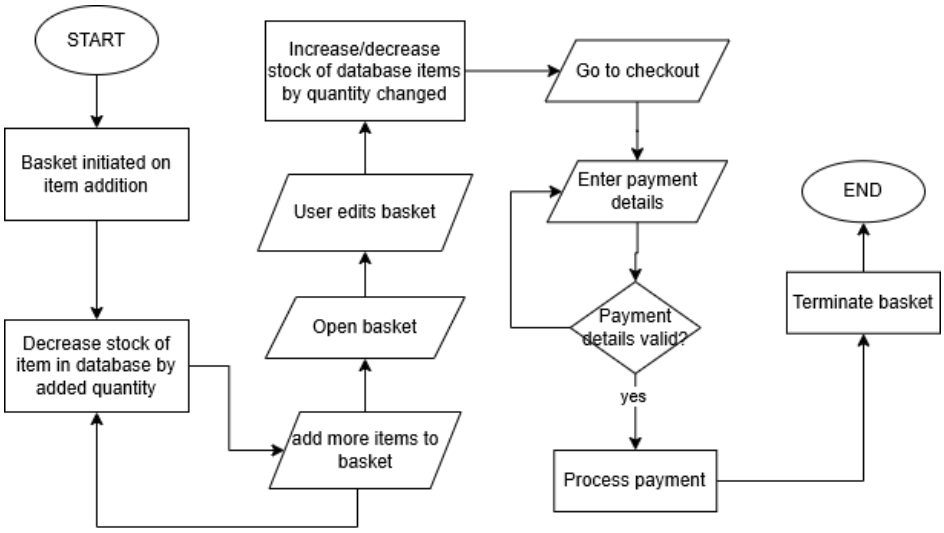

Figure 11: Payment flow

3. Organisation Management Microservice

This service manages all organisation-specific data and functionality. It allows sellers (organisations) to register, view, and edit their profiles, request verification, and manage their product listings. Each organisation is uniquely identified and linked to its corresponding products via a foreign key relationship. The service ensures that organisations can manage their storefronts independently through an intuitive dashboard.

Through the organisation dashboard, users can add new products to the Catalogue microservice. Products are linked to the organisation that created them, ensuring ownership is clearly maintained. Organisations can also update or delete their existing listings via secure endpoints, with changes reflected in the catalogue in real-time. The dashboard also provides a hub for relevant data analytics about organisations' products and sales.

To enhance credibility and visibility on the platform, organisations can request verification. This action triggers a submission to the Admin microservice, where the request is reviewed by an administrator. Verified organisations are prioritised in popularity-based product sorting, offering tangible benefits for sellers who meet trustworthiness criteria. Once verified, the organisation's status is stored persistently and is displayed on their profile and product listings.

4. Analytics Microservice This service creates clear and convenient graphs to provide essential

insight for an online retail platform. The service has been integrated within the organisation management and admin panels.

For each analytic on each panel, either the Payment service database or the Catalogue service database is queried, and the result is returned to the microservice. With this, appropriate aggregation is performed on the data, using the TypeScript record data structure when mapping a singular attribute to another singular attribute and a TypeScript array when a record is not sufficient. Once complete, the data is made available to an appropriate Chart.js [7] graph component on the frontend and illustrated on the necessary page. The Organisation Management analytics required the `org_id` as a parameter to ensure analytics related to only the given organisation were displayed.

In the organisation management panel, the following statistics were created:

- Average price of a product by category
- Number of listed items per category
- Item-Stock Value per item
- Average rating for each product
- Number of Reviews made per day
- Number of sales per item
- Item Revenue

These analytics provide a respective organisation, given that they are verified and not banned, with appropriate bar graphs to offer useful metrics for decision making. These can be utilised to provide an understanding of an organisation's retail presence.

In the admin panel, the following statistics were created:

- Top 5 most interacted pages
- The average price of an item per category
- The number of items listed per category
- Average value of an order per day
- The total order value per day
- Number of orders per day
- Total Revenue per organisation
- Total number of sales ever
- Total revenue ever

These analytics provide the Vendur admin with appropriate bar graphs and general data for an insightful perspective into the performance of the entire platform and of every organisation on the

website.

5. Admin Microservice

The Admin microservice provides the Vendur team with elevated access and tools to oversee organisation activity, manage verification, moderate items in the catalogue, and maintain system integrity. This is essential for ensuring trust between buyers and sellers, and supporting the broader moderation and oversight of the Vendur system. The system grants access to users if their user ID belongs to the IDs in the admin team.

A main responsibility of the Admin microservice is managing the organisation verification process. When a seller submits a verification request through the Organisation Management service, the Admin service retrieves and presents this request for review. Administrators can then inspect the requesting organisation's profile and listings, and choose to approve or reject the request via a secure endpoint. Once a decision is made, the verification status is updated in the database and reflected across the platform. This process ensures that only trustworthy organisations receive verified status, such as increased visibility and customer confidence.

Administrators are also granted the ability to view and moderate product listings submitted by organisations. The Admin microservice can fetch items across the platform and inspect their details, including product descriptions, pricing, category assignments, and associated organisation data. If a product violates the content guidelines, admins have the authority to modify or remove the listing directly. This moderation helps uphold the integrity of the marketplace, ensuring that all visible products meet platform standards and provide a safe, shopping experience.

Lastly, the admin panel contains a section for relevant system-wide analytics. This allows the Vendur team to keep up to date with system performance and identify areas of the system that are not being positively engaged with and need to be reviewed.

4.1.2 Github

Throughout the development of Vendur, we used GitHub to manage version control, and support collaboration across our team. GitHub was an essential part of our cloud-native development workflow, allowing us to synchronise code contributions, isolate feature development, and preserve a stable main branch for integration.

Each team member worked within their own dedicated branch, corresponding to either a specific microservice or the frontend. This branching strategy enabled parallel development and reduced the risk of merge conflicts. It also allowed for team members to help one another with developing their microservices in their own branches. Once development was complete and locally tested, each branch was merged into the frontend branch for integration testing, and into the main branch once validated.

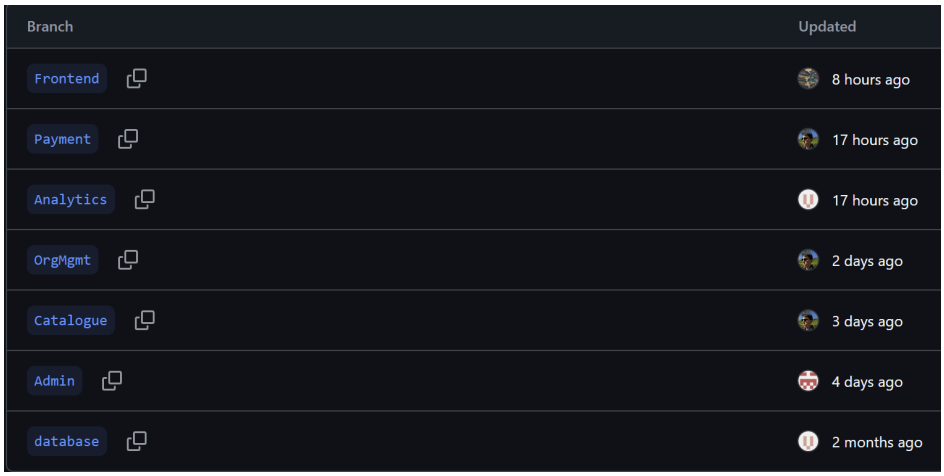

Figure 12: Git Branches

Commits were made regularly, using clear messages to document the purpose of each change. This improved traceability and made it easier to identify and reverse problematic changes when needed. GitHub's visual commit history and pull request interface also helped facilitate peer reviews, discussion, and team awareness of ongoing work.

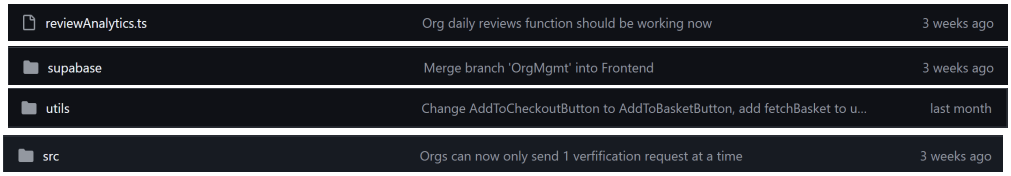

Figure 13: Git Commit Examples

Each feature was tested during development to verify route behaviour, request validation, and expected responses under different input conditions. This ensured that major issues or bugs could be ironed out as soon as possible before moving on, and minor bugs could be documented to be fixed at a later date. This continuous integration of tested features allowed us to ensure that the features we were implementing were working as intended, and informed any improvements that were needed at the time.

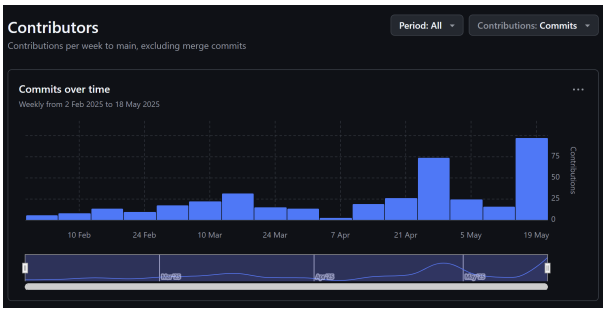

Figure 14: Git Commits Throughout the Project

Figure 15: Git Contributors Table

As stated in section 3, coding expectations were different for each team member according to expertise, and these were balanced with design and documentation responsibilities. In particular, Alexander Kyriacou possessed the least familiarity and understanding of the web technologies but the highest proficiency with academic writing and technical graph drawing. To complement this, they had few coding responsibilities, instead focusing on design of the web application and spearheading the documentation initiative throughout the project.

4.1.3 Docker

Docker was used to encapsulate each microservice within its own lightweight container. This approach enabled each microservice to be independently developed, tested, and deployed, without the risk of interfering with the behaviour or dependencies of other services.

Each of the five backend microservices was containerised using its own Dockerfile. These files define the base image for the service, Node.js in our case. They also install dependencies, copy the source code into the container, and specify an entrypoint command to start the service.

```dockerfile
FROM node:18

WORKDIR /app

COPY package.json ./

RUN npm install -g typescript
RUN npm install

COPY . .

RUN npm run build

EXPOSE 5078

CMD ["node", "dist/index.js"]
```

Figure 16: The Admin Microservice Dockerfile

Once each Docker image was created, Docker Compose was used to orchestrate the system. The `docker-compose.yml` file defines how each microservice is connected and configured to run in a local networked environment.

```yaml
version: "3.8"
services:
  microservice:
    build: .
    ports:
      - "5078:5078"
    env_file:
      - .env
    restart: unless-stopped
```

Figure 17: The Analytics Microservice docker-compose.yml

This setup allows each microservice to be spun up or shut down with a single command:

- `docker-compose up` starts the service and its dependencies
- `docker-compose down` halts all services and removes the container

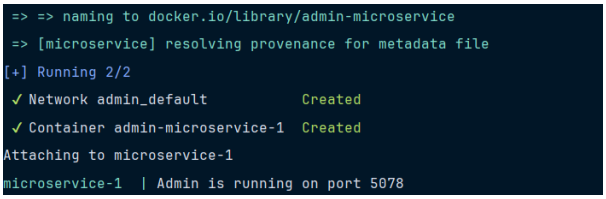

Figure 18: docker-compose up on the Admin Service

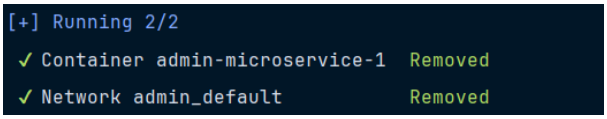

Figure 19: docker-compose down on the Admin Service

4.2 Testing

Platform testing was carefully planned and carried out based on our initial functional requirements detailed in Table 2. These tests are detailed in the following tables:

Test ID	Req.	Description	Inputs	Expected Output	Status
TC-F1-01	F1	Add a new product with all fields populated.	Organisation submits item with title, description, price, category, stock	Item is added to catalogue	Pass [A.26, A.27]
TC-F1-02	F1	Update stock and apply discount for existing product.	Organisation edits stock and applies discount	Catalogue reflects changes	Pass [A.28, A.29, A.30]
TC-F2-01	F2	Search products by keyword.	User searches 'shirt'	Matching products returned	Pass [A.31]
TC-F2-02	F2	Filter by category and price.	User filters for a category	Filtered list meets criteria	Pass [A.32]
TC-F6-01	F6	Submit a review with rating and comment.	User leaves 5-star review	Review shown on product page	Pass [A.33, A.34]

Table 6: Catalogue Microservice Test Plan

Test ID	Req.	Description	Inputs	Expected Output	Status
TC-F3-01	F3	View desired items in the basket.	User adds products to basket from the catalogue page	Products added are listed in the basket, including quantity	Pass [A.35]
TC-F3-02	F3	Complete checkout flow using Stripe test card.	Card: 4242 4242 4242 4242	Payment confirmed, order placed	Pass [A.36, A.36]

Table 7: Payment Microservice Test Plan

Test ID	Req.	Description	Inputs	Expected Output	Status
TC-F4-01	F4	Organisation specific analytics are retrieved	Organisation views dashboard	Analytic graphs for just the organisation are illustrated	Pass [A.44, A.45]
TC-F4-02	F4	Admin analytics are retrieved	Admin views dashboard	Analytic graphs for just admins are illustrated	Pass [A.46, A.47]

Table 8: Analytics Microservice Test Plan

Test ID	Req.	Description	Inputs	Expected Output	Status
TC-F1-03	F1	Remove a product from the catalogue.	User deletes an item from the catalogue via the admin panel	Item removed from catalogue	Pass [A.41, A.43]
TC-F5-01	F5	Approve an organisation verification request.	Admin reviews and accepts request	Organisation marked as verified	Fail (Pass on localhost)
TC-F5-02	F5	Admin edits organisation info.	Change description in dashboard	Info updated successfully	Pass [A.42]

Table 9: Admin Microservice Test Plan

Test ID	Req.	Description	Inputs	Expected Output	Status
TC-F1-04	F1	Manage listed items in the catalogue.	Organisation deletes a listed item	Item will be removed from catalogue and listed items.	Pass [A.38]
TC-F3-04	F1	Manage pending orders.	Organisation changes the status of an order	Order status will be update	Pass [A.39, A.40]

Table 10: Organisation Management Microservice Test Plan

To test API routes, we used Postman [18]. This allowed us to simulate HTTP requests such as GET, POST, PUT, and DELETE with custom headers, payloads, and authentication tokens. It was particularly useful for validating request-response behaviour across our microservices before frontend integration. Each endpoint was tested for expected functionality, response structure, and error handling, under both valid and invalid input conditions. Postman's history and collection features also enabled efficient retesting during the iterative development and regression testing phases.

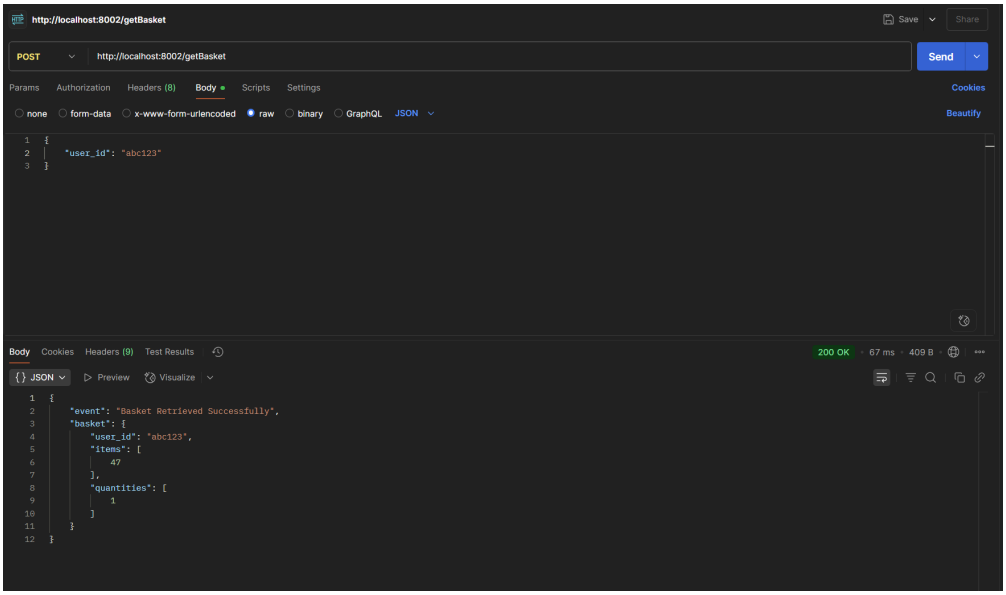

Figure 20: An Example Request To Get User Basket

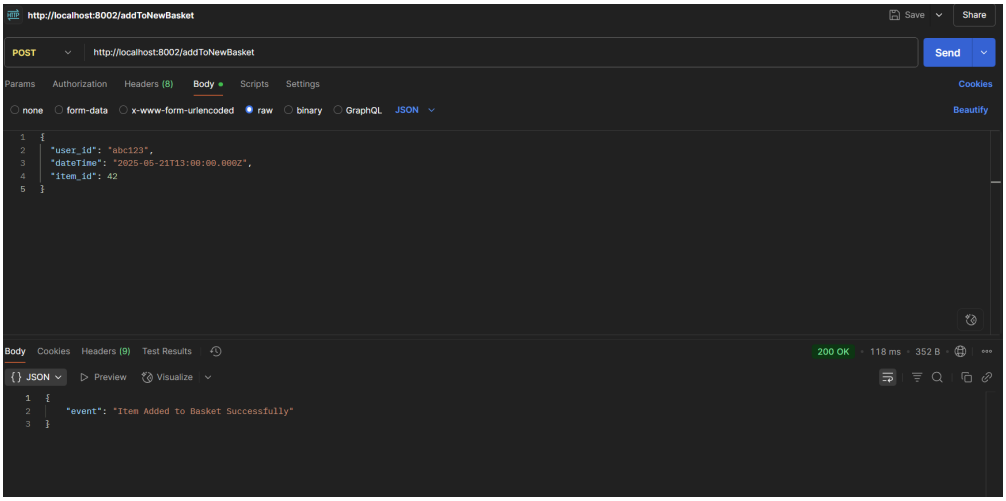

Figure 21: An Example Request to Add a Product to the User Basket

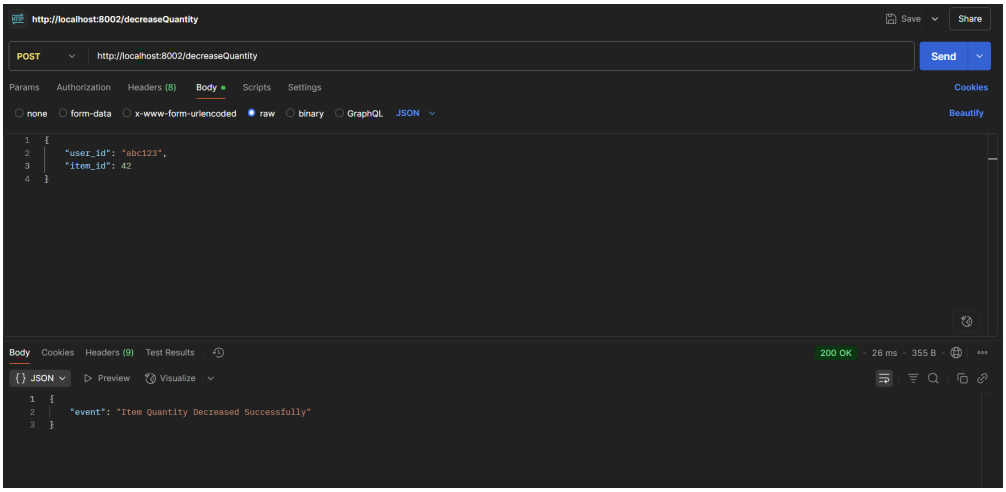

Figure 23: An Example Request to Decrease Stock when Added to Basket

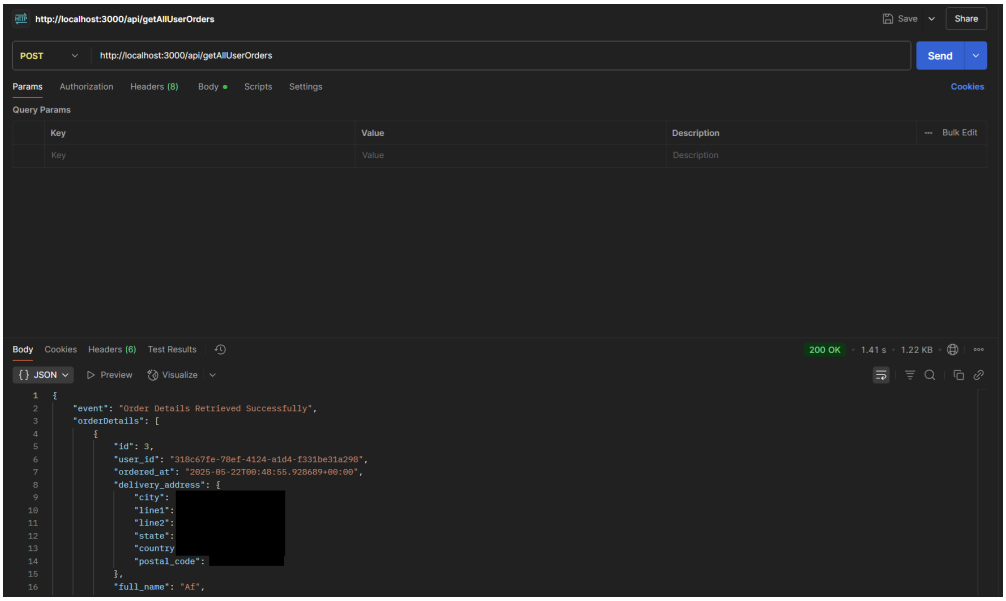

Figure 22: An Example Request to Get All Orders

5 Learning from Class and from Good Examples

5.1 Class Concepts

One of the most important takeaways from what was covered over the semester was the importance of the microservice architecture, as well as the value of containerising said microservices using Docker.

A common challenge presented by large-scale, collaborative coding projects is the inconsistency of environments between team members. This often leads to issues such as build errors or feature bugs on some machines, while others presented no issues. This can also cause application errors for end users. Using Docker enabled us to efficiently package each microservice with all its dependencies, ensuring that they will all reliably run across any environment.

Additionally, containerisation supports modularity and isolation, both key concepts in web development, allowing features to be independently developed, tested, deployed and maintained. This makes the system more resilient, as a failure in one service will not compromise the entire system.

5.2 Amazon

Amazon [3] is another retail platform that adopted a microservice architecture [25], albeit on a much larger scale, with hundreds of independent services available. This was a great inspiration for the highly scalable and isolated microservice architecture which would make up Vendur.

Vendur's backend mimics Amazon's in its structure, with each microservice being independently developed and deployed, internally connected through API routes. This is what leads to the highly scalable and resilient architecture for which Amazon is widely recognised.

Another key point of inspiration was Amazon's highly analytics-driven decision making [13]. The Amazon platform makes use of large datasets to optimise product listings, pricing, and recommendations. Although not on the same scale, this inspired our Analytics microservice to provide real-time business insights to companies using Vendur. This was particularly important to provide for SMBs which may lack in-house data analytics tools.

5.3 Shopify

Shopify [21] offers a ready-to-use e-commerce platform that is targeted at smaller businesses, overlapping with the target audience for Vendur. This inspired our decision to focus on rapid user onboarding with an emphasis on seller autonomy, as well as influencing the design of the organisation management microservice and dashboard.

Shopify presents merchants with an intuitive dashboard with access to analytics, product management, and customer interaction, so we designed Vendur's organisation management system with these concepts in mind. This inspired the modular structure of the microservice, which enables

businesses to list products and manage their profiles, all independently of other backend functionality. Our system also similarly supports organisation verification by the administration team, partly inspired by Shopify's tiers and merchant status.

5.4 eBay

eBay [9] has long had a significant presence in the e-commerce space, leading to the evolution of an efficient system of managing user-generated content in extremely high volumes. This provided a lot of inspiration for Vendur's own item search and filtering capabilities present in the catalogue microservice. While on a smaller scale, our database schema is designed to accommodate filtering by product categories, directly inspired by eBay's layered search capabilities.

The Vendur user experience was also greatly informed by eBay's use of progressive disclosure in the UI [29]. This concept helped us minimise interface clutter in our frontend design, as well as gradually presenting more advanced information (such as filters and reviews) to users when contextually relevant. This streamlined the user interface so as not to overwhelm new users while still providing important depth to experienced clients.

Our final takeaway from eBay was the use of real-time elements. Due to the nature of eBay's bidding system, consistent real-time updates are crucial, and these influenced how system responses were handled in Vendur. This included real-time updates such as discounts being applied to products, and reviews being made. While the Vendur platform does not support live bidding, the focus on consistent and responsive updates remained relevant to our design.

6 Roadmap and Business Model

6.1 Start-up Description

Vendur is a web-based retail platform designed to enable SMBs to effectively compete in the digital commerce space. The platform offers a range of fully integrated services, including product catalogue management, secure payment processing, customer and product analytics, and organisation account and profile management. Using a microservice-based architecture, Vendur allows organisations to scale operations efficiently, boost customer engagement, and optimise sales performance. Initially, Vendur will target SMBs within the UK e-commerce market, focusing on retailers with limited technical resources who need additional support in the online landscape.

6.2 Project Motivation

The motivation behind Vendur is to level the playing field for SMBs in an increasingly competitive online market. This can help small and often family-owned businesses get a foothold and survive with the decline in high-street shopping and the increase in convenient online shopping [14].

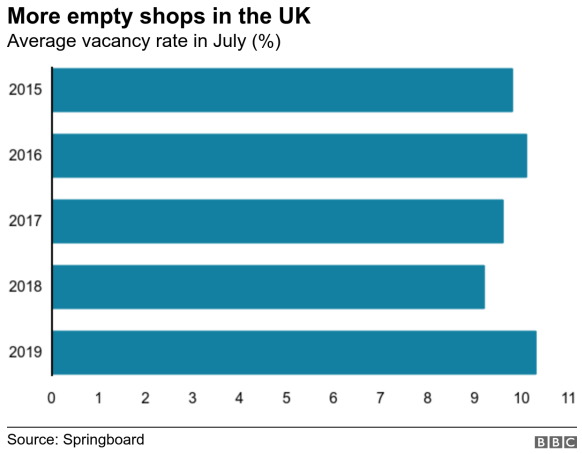

Figure 24: Rise in Empty High-Street Shops

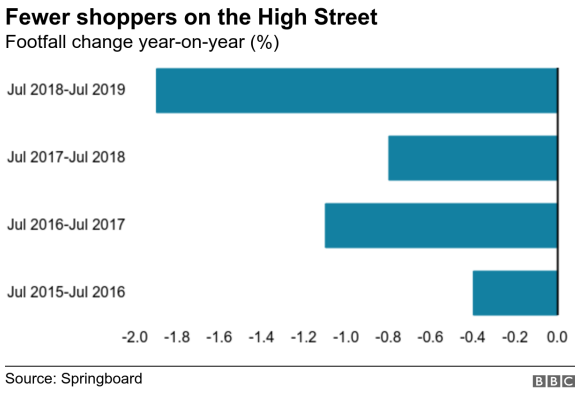

Figure 25: Decline in High-Street Shoppers

6.3 Vision and Future Plans

On a short-term scale, the main plan for Vendur is to fix the initially defined features that did not materialise in the final hosted version. From our testing, the only failure was in the organisation verification feature, as organisation accounts cannot currently submit a verification request (but admins are still capable of verifying an organisation manually). As this issue was not present in the locally hosted application, this would likely prove to be a small fix; however, the team simply ran out of time and resources before submission.

Our long-term vision for Vendur is to expand its reach to international markets, as well as scale and develop features to enhance the user experience, enabled by our modular design philosophy.

To support this scale, we plan to migrate the platform to cloud services such as AWS or Google Cloud, leveraging the robust infrastructure already in place for scalable computing and global deployment.

We also aim to draw inspiration from industry leaders such as Amazon [3], eBay [9], and Shopify [21] in refining the user experience. This includes introducing frictionless onboarding [28] to minimise the amount of information required to create an account, as well as offering one-click check out and integrated shipping options. This would eventually lead to the Vendur company having its own warehouses and storage in many locations, and hiring drivers to deliver some products to customers.

This scale will of course require monetisation, so we will evaluate income strategies which prioritise user growth. For example, we may offer a tiered membership model; we can allow users to have a limited free experience with basic e-commerce features, and provide advanced features such as more in-depth analytics or dedicated customer service behind an optional monthly paywall. This allows small businesses to access essential tools, while offering growing or mid-sized organisations enhanced capabilities. Another source of monetisation is advertising; however, we must ensure that advertisements are relatively nonintrusive so as not to negatively impact the user experience.

6.4 Target Market Group (TMG)

Vendur targets SMBs in the retail sector, particularly with limited technical capacity or online presence. Demographically, the platform appeals to business owners and operators of all ages but will likely be emphasised to relatively young business owners who are trying to find their way and build a presence in the online world. Businesses in almost any retail segment can make use of Vendur, from upmarket fashion boutiques to niche artisanal souvenir shops, so long as items can be packaged and delivered.

6.5 Competitive Analysis

Vendur competes with all-in-one e-commerce solutions such as Shopify, and marketplace platforms such as Amazon and Etsy. However, Vendur's tailored focus on SMBs is what sets the platform apart.

Unlike major competitors which offer rigid packages, Vendur provides an integrated solution that is ready-to-use and designed specifically for smaller retailers. The modular back-end architecture allows the Vendur team to deliver high-quality services as well as scaling and developing features which will most refine the experience for these companies.

In addition, Vendur stands out with its inbuilt analytics, organisation management tools, and flexible customer interaction support typically only available to larger enterprises. By focusing exclusively on SMBs, Vendur delivers an effective and affordable solution for getting started in the

e-commerce space.

6.6 Key Stakeholders

Vendur's primary users are small and medium-sized retail businesses. These organisations will use the platform to manage product listings, process payments, analyse customer data, and engage with their audiences. The platform is designed to meet the needs of SMBs that often lack large technical teams, providing them with an affordable and accessible solution to compete in the online market.

End customers are also critical stakeholders. Your experience browsing, purchasing, and interacting with Vendur businesses directly affects the reputation and commercial success of the platform. Ensuring secure transactions, fast performance, and user-friendly interfaces is essential to meet their needs.

The development team has played and will continue to play a key role in building, maintaining, and continuously improving the Vendur platform. This includes implementing new features, fixing bugs and enhancing system performance. Keeping the development team aligned is vital for long-term success.

Finally, investors and strategic partners provide the financial and operational backing to scale the business. They may offer capital, cloud services, or business expertise, helping Vendur expand into new markets and improve its product offerings. Engaging these stakeholders with a clear growth strategy will be crucial to secure the resources needed for expansion.

6.7 Ethical and Legal Considerations

During development, we assumed that customer and organisation data used would remain within a controlled test environment. Currently, the application uses placeholder data and credentials, meaning no personally identifiable information is stored or processed. However, if Vendur were to be implemented in a real-world setting with live users, strict adherence to the UK General Data Protection Regulation (GDPR) [12]and the Data Protection Act 2018 [27] would be legally required.

If deployed commercially, Vendur would collect and process user data such as names, organisation details, email addresses, product information, and customer payment details. In this case, it would be essential to implement transparent privacy policies, obtain informed user consent for data collection and storage, and offer opt-out mechanisms where appropriate. In particular, organisations would need to be notified about how their profiles and public responses are displayed on the platform, and customers would need to agree to terms and conditions that outline the visibility of their reviews.

All passwords and sensitive login information are currently hashed using industry-standard encryption methods by the authorisation system. This provides sufficient security to ensure that user credentials are not susceptible to being leaked in the event of a cyber attack. Data isolation with

separate databases for microservices further reinforces this.

Integration with third-party payment providers, such as Stripe, require compliance with PCI DSS (Payment Card Industry Data Security Standard) to ensure that financial data is handled securely [16]. Before deployment, Vendur would also need to verify that any partner services or libraries used comply with relevant security and privacy standards.

Ethically, the platform aims to empower small and medium-sized businesses by offering fair access to e-commerce infrastructure. Sellers are treated equally, with verification serving only as a means to prioritise trust, rather than exclude participation. Reviews are publicly visible, but tied only to product IDs and not customer names or accounts, supporting transparency while avoiding personal exposure.

Throughout the project, we adhered to the BCS Code of Conduct, particularly in regard to user privacy, fair treatment, and responsible data handling [4]. As part of our commitment to ethical software development, we ensured that all testing was conducted with placeholder data, with no real customers involved, and no actual commerce occurred. Should the project be deployed in the future, a full Data Protection Impact Assessment (DPIA) would need to be conducted before real data collection begins [5].

7 Conclusion

Overall, the Vendur project demonstrates a strong approach to building a scalable, cloud-native retail platform. Through the use of microservices, containerisation with Docker, secure payment processing, and an intuitive frontend, the system addresses our goals of supporting small and medium-sized businesses. Effective collaboration, testing practices, and adherence to cloud-native principles have ensured a maintainable and extensible codebase, providing the project with a solid foundation for future enhancements.

References

[1] Klarna - Compare prices and pay in 3 methods — klarna.com. https://www.klarna.com/uk/.

[2] Stack Auth — the open-source Auth0 & Clerk alternative — stack-auth.com. URL: https://stack-auth.com/.

[3] Amazon. Amazon. URL: https://www.amazon.co.uk/?tag=admpdesktopuk-21&ref=pd_sl_9555ef09d69d14d19c77d608afd2267c21d858116e164eb4f850b79e.

[4] British Computer Society (BCS). Bcs code of conduct. URL: https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/.

[5] British Computer Society (BCS). Data protection impact assessments (dpias). URL: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/accountability-and-governance/data-protection-impact-assessments-dpias/.

[6] Burst by Shopify. Burst - free stock photos for websites and commercial use. URL: https://www.shopify.com/stock-photos.

[7] Chart.js Dev Team. Chart.js. URL: https://www.chartjs.org/docs/latest/.

[8] Cloudinary. Cloudinary — Media API for Images, Video and Rich Media. URL: https://cloudinary.com/.

[9] eBay. ebay. URL: https://www.ebay.co.uk/e/coupon-offers/generic?mkcid=1&mkrid=710-53481-19255-0&siteid=3&campid=5337314663&customid=pcgbebayebaysd&toolid=10001&mkevt=1.

[10] Express.js. Express.js documentation. URL: https://expressjs.com.

[11] F5, Inc. Welcome to NGINX — F5. URL: https://www.f5.com/go/product/welcome-to-nginx.

[12] Information Commissioner's Office (ICO). Uk gdpr overview. URL: https://ico.org.uk/for-organisations/data-protection-and-the-eu/data-protection-and-the-eu-in-detail/the-uk-gdpr/.

[13] Michael Ampofo. How amazon uses data science and analytics to drive e-commerce success. URL: https://www.linkedin.com/pulse/how-amazon-uses-data-science-analytics-drive-success-michael-ampofo/.

[14] BBC News. High street: How many uk shops have closed? URL: https://www.bbc.co.uk/news/business-49349703.

[15] npm. cookie-parser documentation. URL: https://www.npmjs.com/package/cookie-parser.

[16] PCI Security Standards Council. Payment card industry data security standard. URL: https:

//docsprv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0_1.pdf.

[17] PM2. PM2 — Production Process Manager for Node.js Apps. URL: https://pm2.io/.

[18] Postman Dev Team. Postman. URL: https://www.postman.com.

[19] Russell Heimlich. Dynamic dummy image generator. URL: https://www.shopify.com/stock-photos.

[20] shadcn/ui. Carousel — Shadcn UI Docs. URL: https://ui.shadcn.com/docs/components/carousel.

[21] Shopify. Shopify. URL: https://www.shopify.com/uk/free-trial?term=shopify&adid=565751946441&campaignid=15439902872&branded_enterprise=1&BOID=brand&utm_medium=cpc&utm_source=google&gad_source=1&gad_campaignid=15439902872&gclid=Cj0KCQjw0LDBBhCnARIsAMpYlAqLXV-X3NiW2ar61v5v3nXwkXb6dM2c1ZnWccVRzlDpbv_NxSrzNXQaApk-EALw_wcB.

[22] Stripe.js. Stripe.js documentation. URL: https://docs.stripe.com/js.

[23] Supabase. Supabase CLI — Supabase Docs — supabase.com. URL: https://supabase.com/docs/guides/local-development/cli/getting-started?queryGroups=platform&platform=linux&queryGroups=access-method&access-method=kong.

[24] Tailwind Labs. Tailwind CSS — Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/.

[25] The New Stack. What led amazon to its own microservices architecture. URL: https://thenewstack.io/led-amazon-microservices-architecture/.

[26] TypeScript. Typescript. URL: https://www.typescriptlang.org.

[27] UK Government. Data protection act 2018. URL: https://www.legislation.gov.uk/ukpga/2018/12/contents.

[28] Userpilot. Frictionless onboarding. URL: https://userpilot.com/blog/frictionless-customer-onboarding/.

[29] UXPin. What is progressive disclosure? show hide the right information. URL: https://www.uxpin.com/studio/blog/what-is-progressive-disclosure//.

[30] Vercel. Next.js — The React Framework for the Web. URL: https://nextjs.org/.

A Testing Evidence

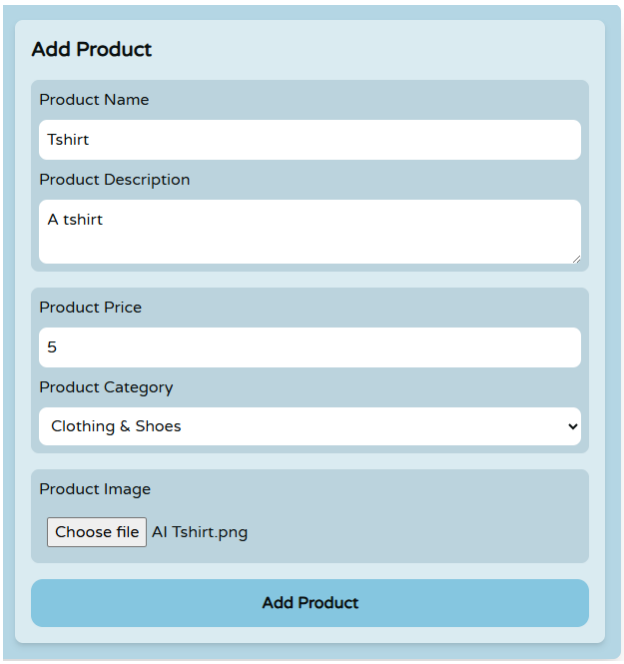

Figure A.26: Add Product Form (TC-F1-01)

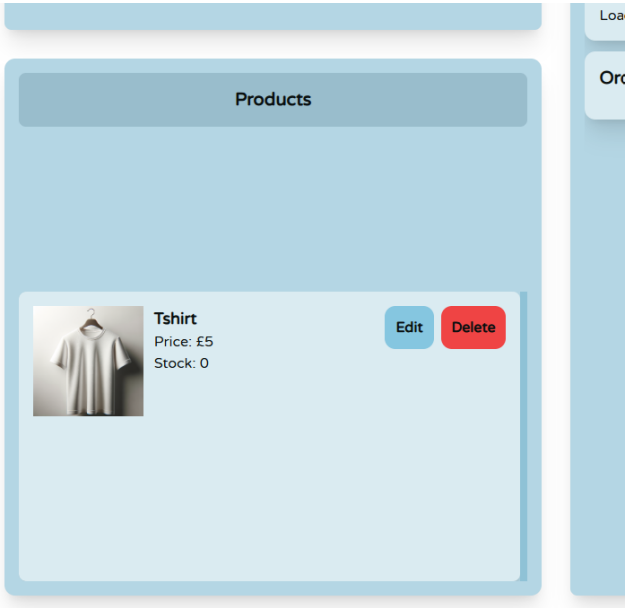

Figure A.27: Item Added to Catalogue (TC-F1-01)

Figure A.28: Editing a Products Stock (TC-F1-02)

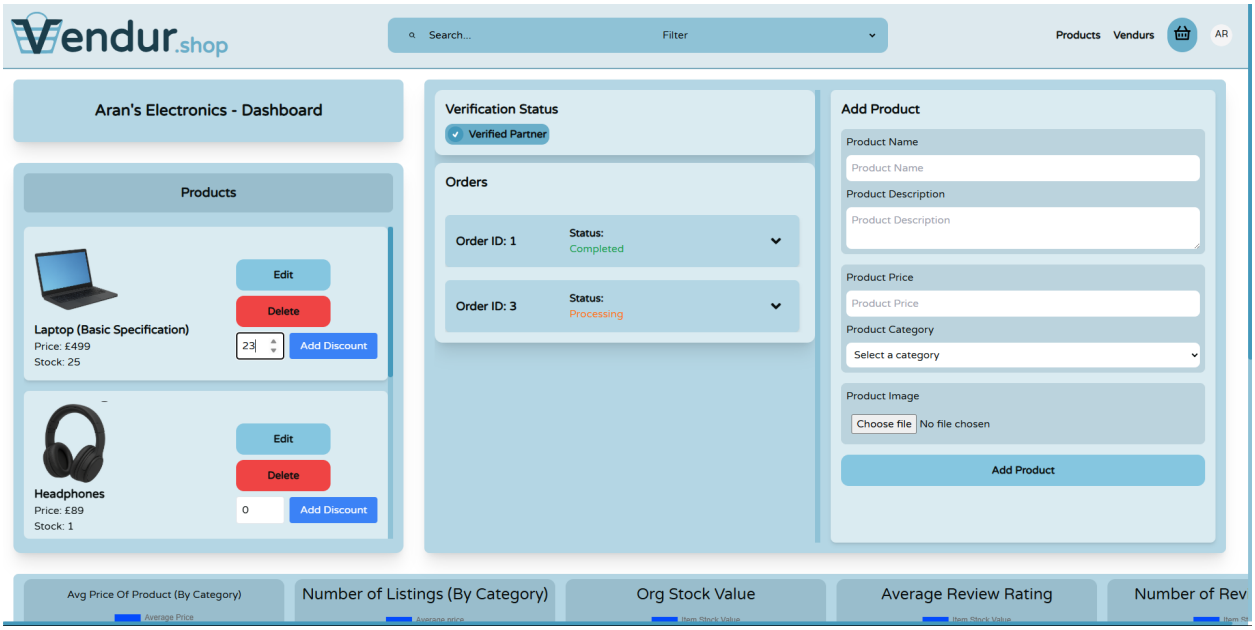

Figure A.29: Adding a Discount to a Product (TC-F1-02)

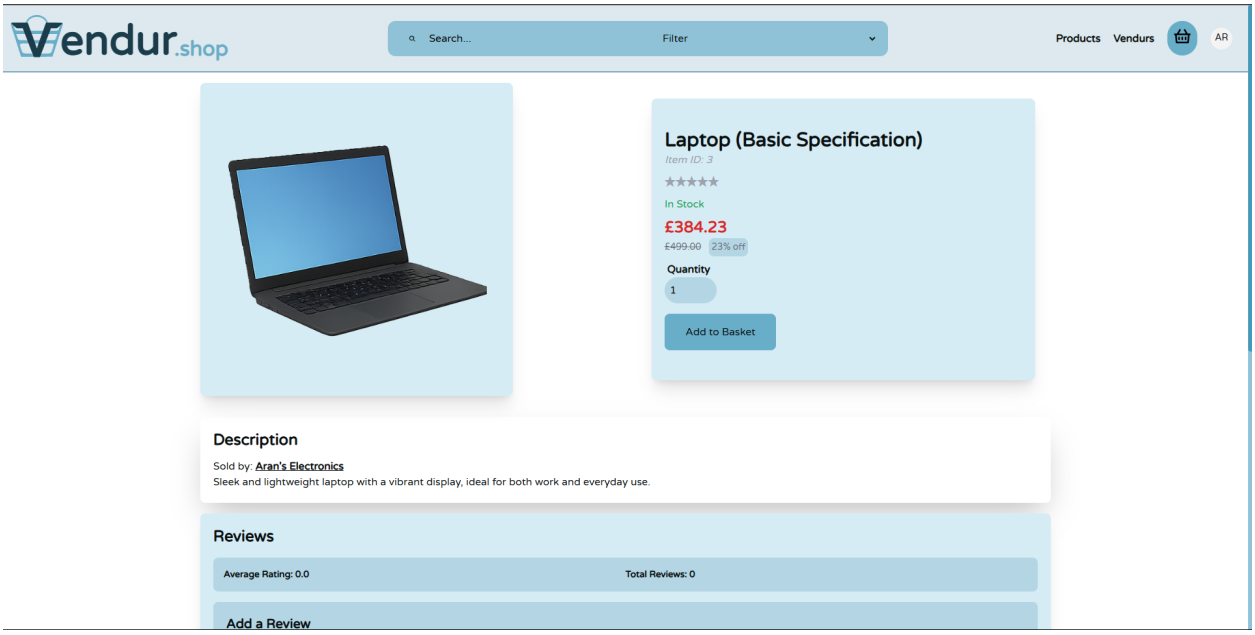

Figure A.30: Discount Added to the Product (TC-F1-02)

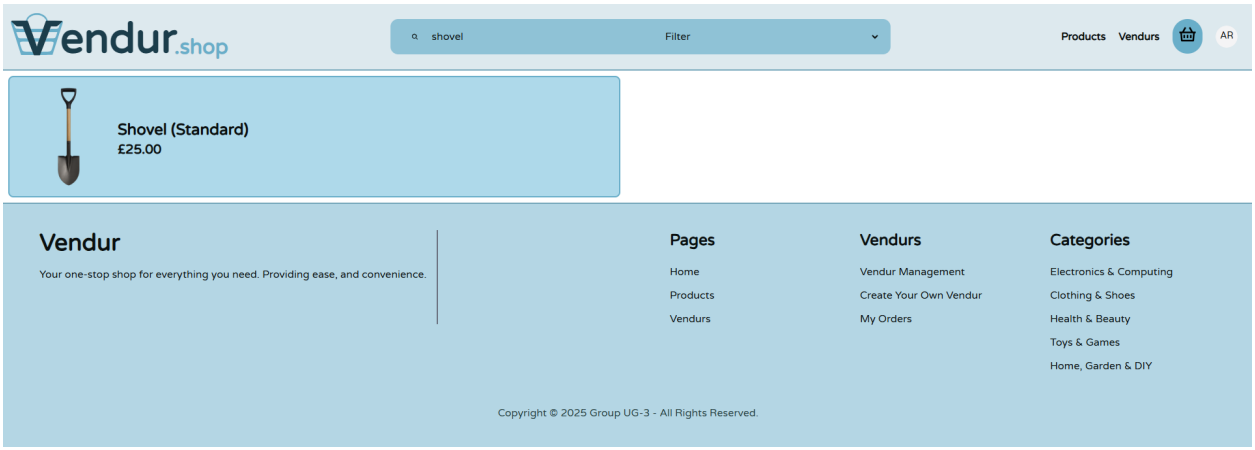

Figure A.31: Keyword Search (TC-F2-01)

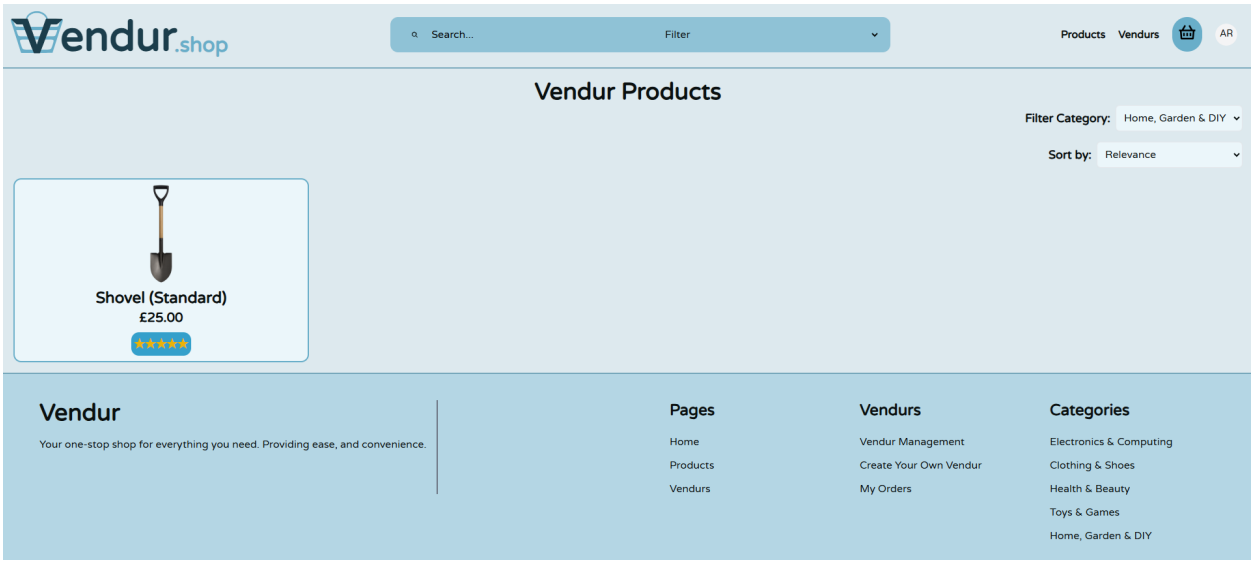

Figure A.32: Category Filtering (TC-F2-02)

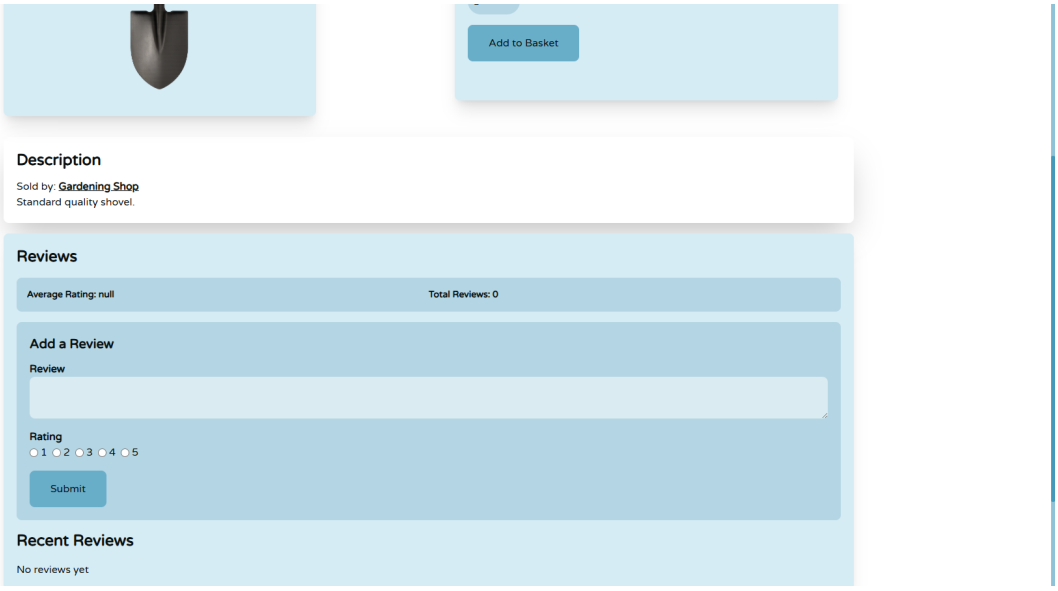

Figure A.33: Product Review Form (TC-F6-01)

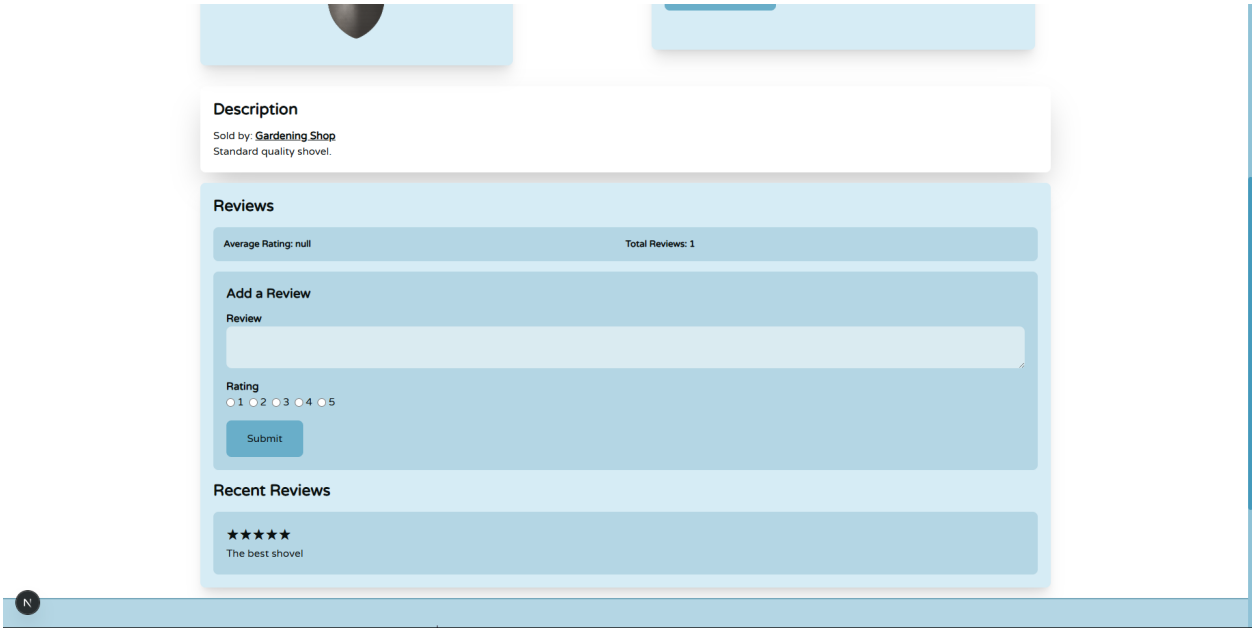

Figure A.34: Product Review Example (TC-F6-01)

Figure A.35: Items in the Basket (TC-F3-01)

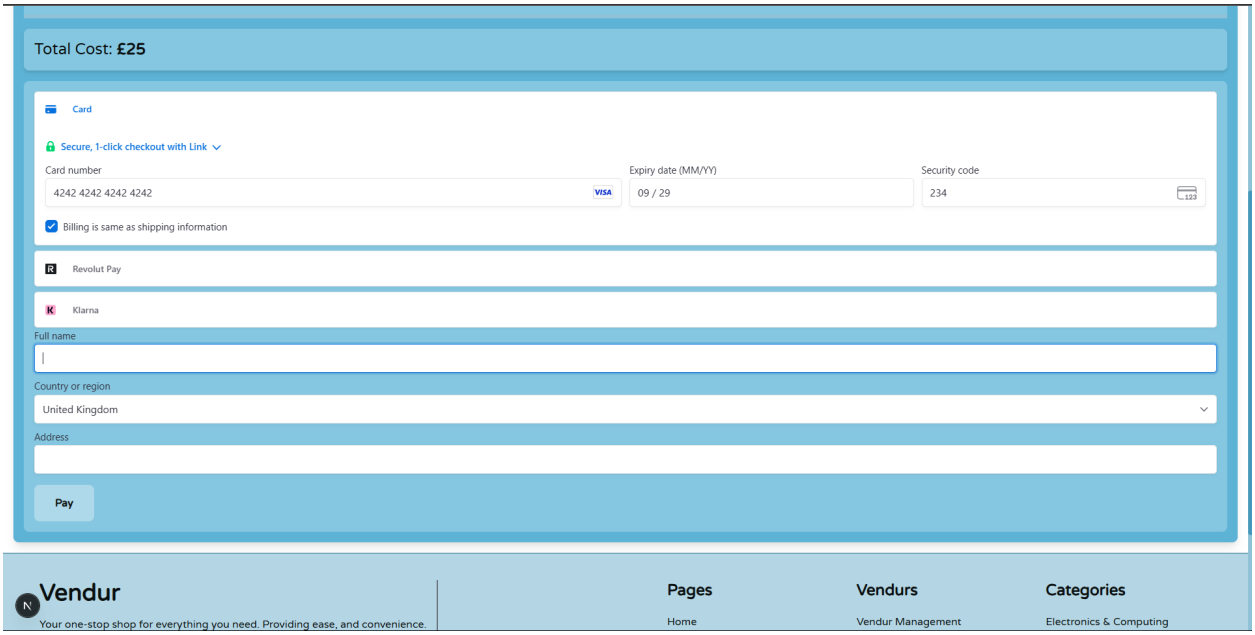

Figure A.36: Stripe Checkout (TC-F3-02)

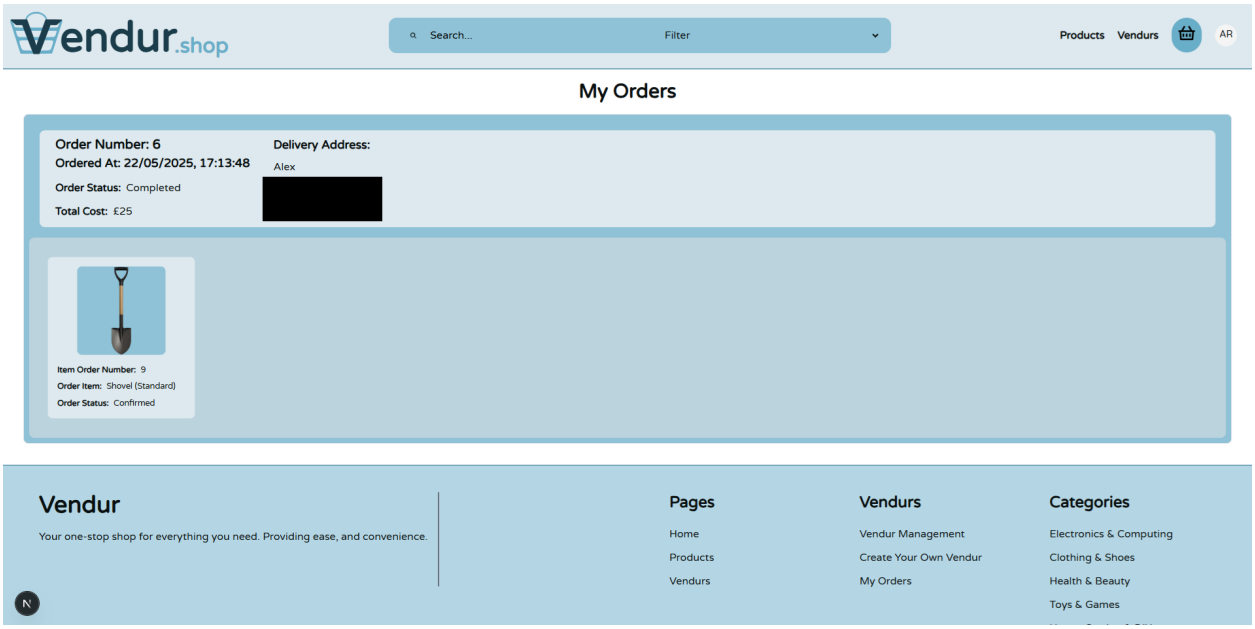

Figure A.37: Order Success (TC-F3-02)

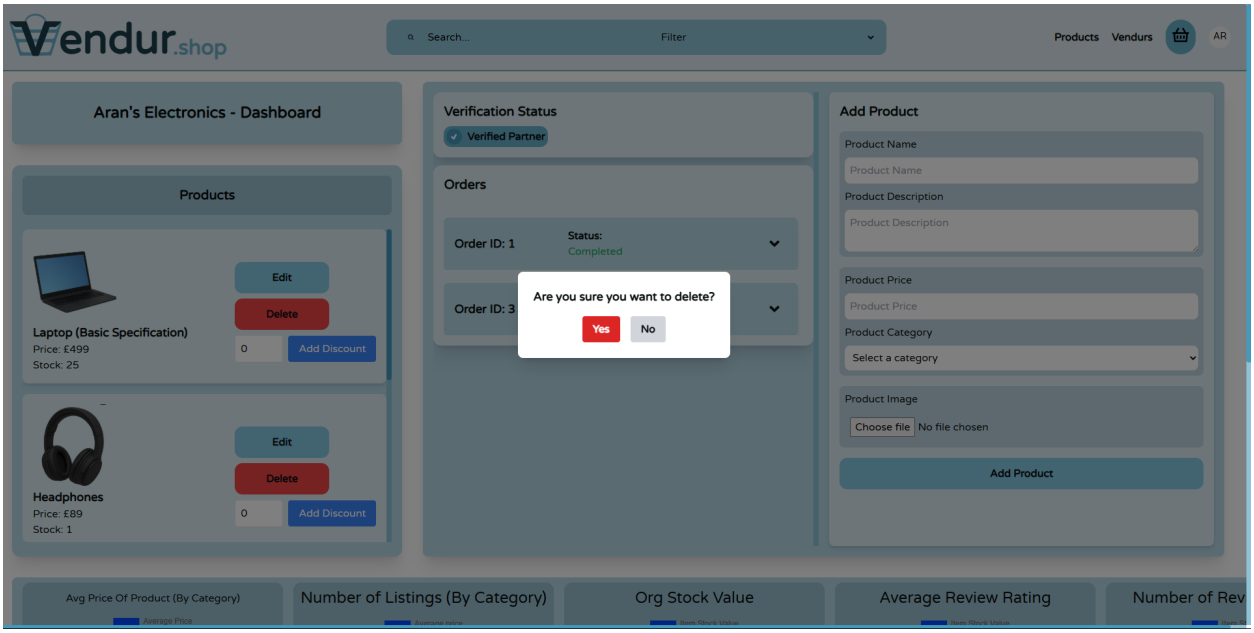

Figure A.38: Deleting an Item (TC-F1-04)

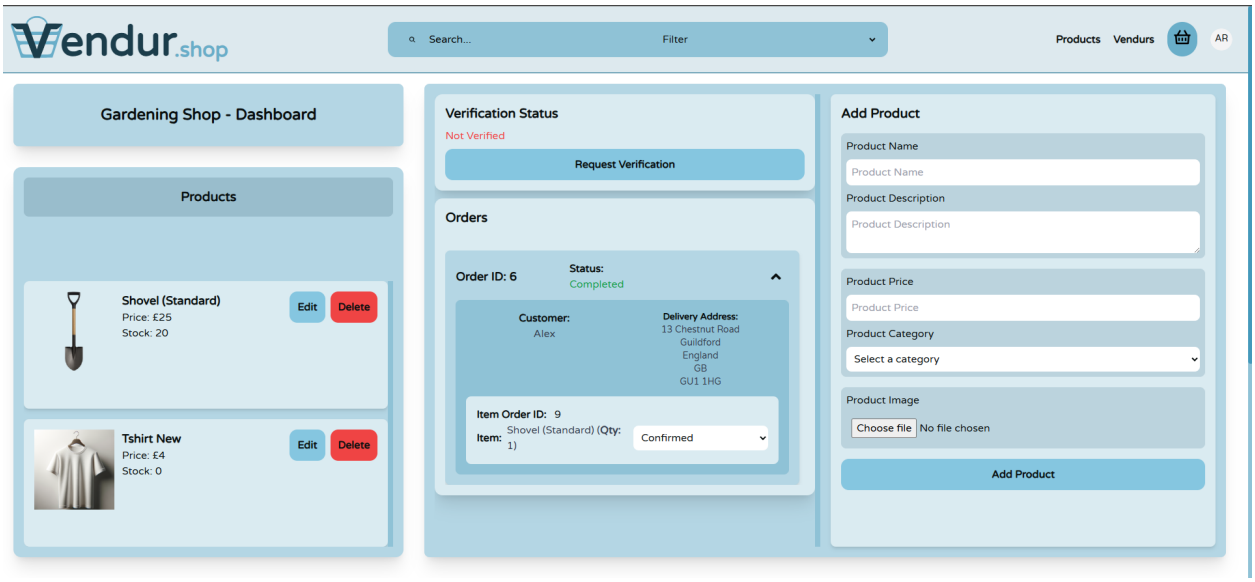

Figure A.39: Setting an Order to Complete (TC-F3-04)

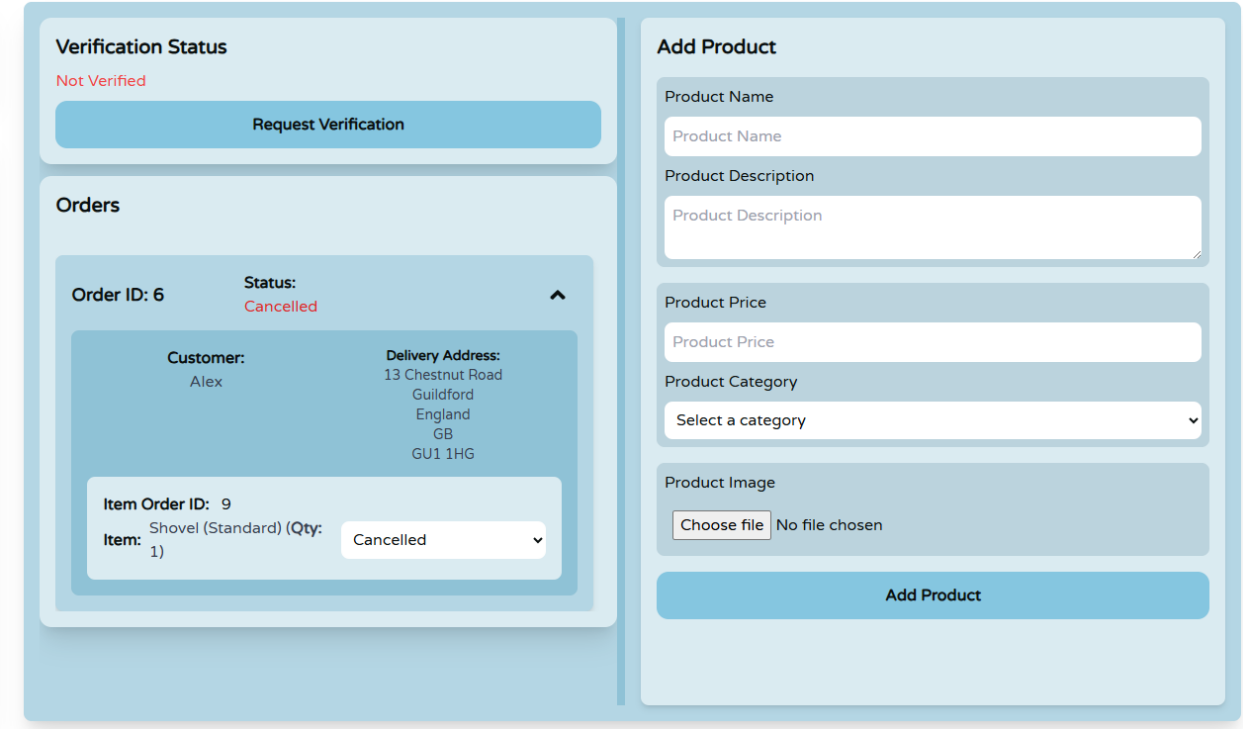

Figure A.40: A Cancelled Order

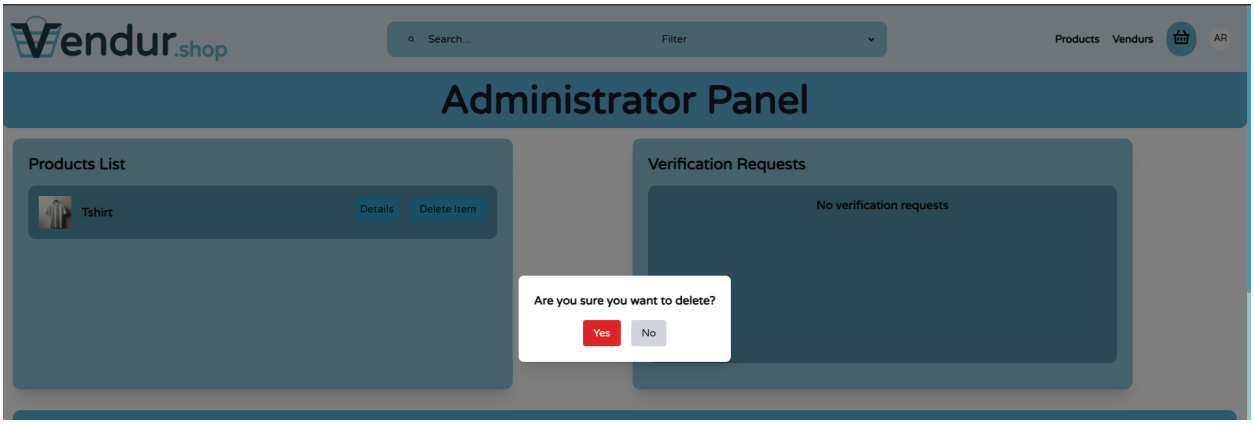

Figure A.41: Deleting a Product from the Admin Panel (TC-F1-03)

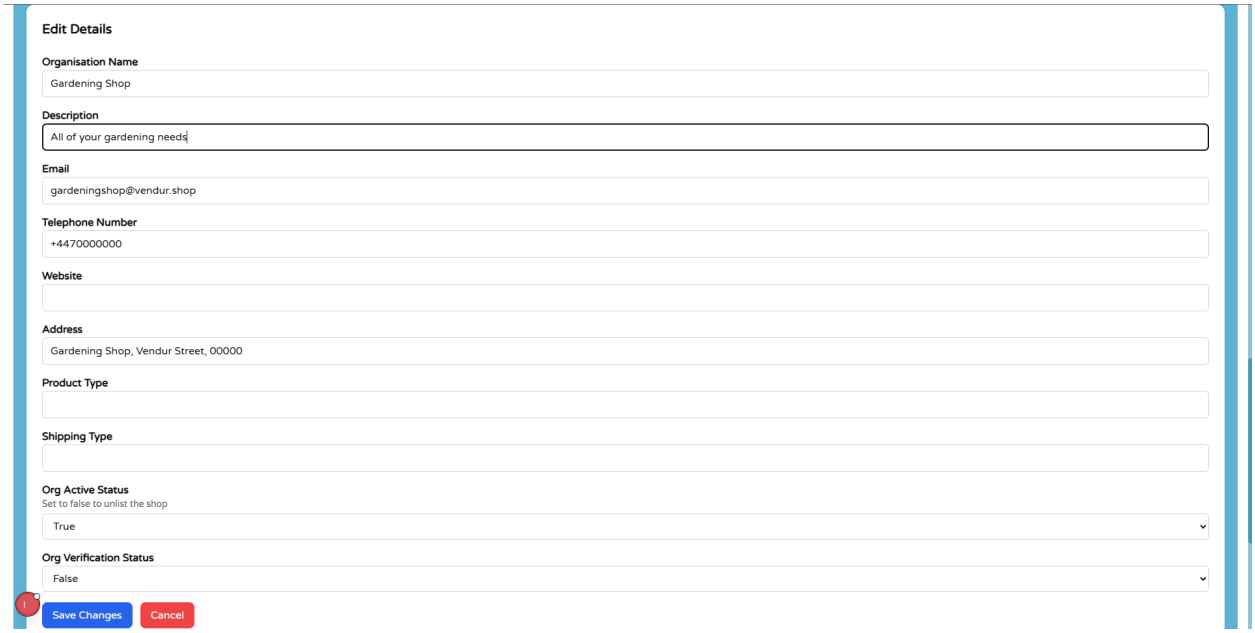

Figure A.42: The Edit Organisation Form (TC-F5-02)

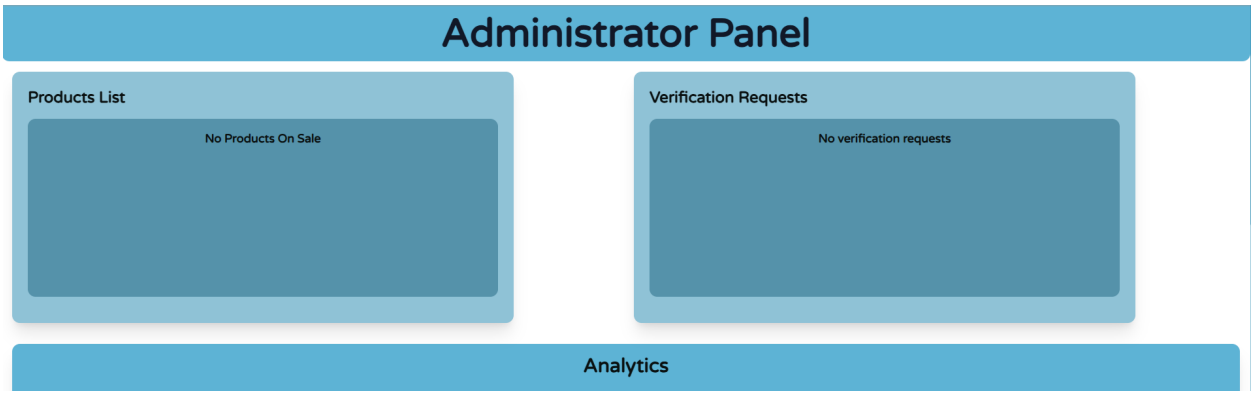

Figure A.43: Product Removed by Admin (TC-F1-03)

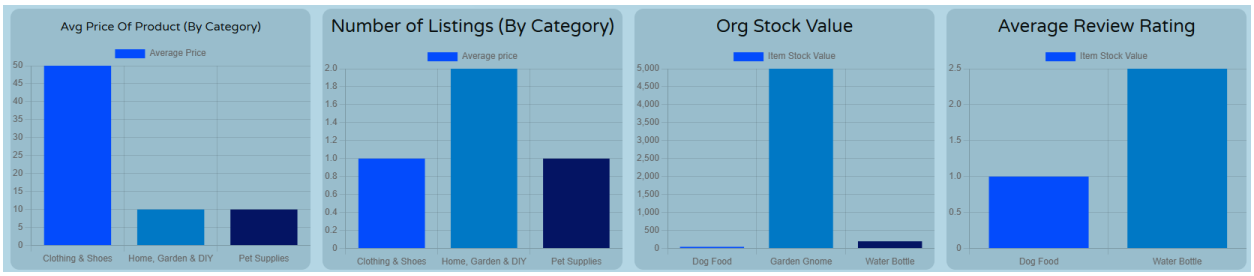

Figure A.44: Organisation Analytics (TC-F4-01)

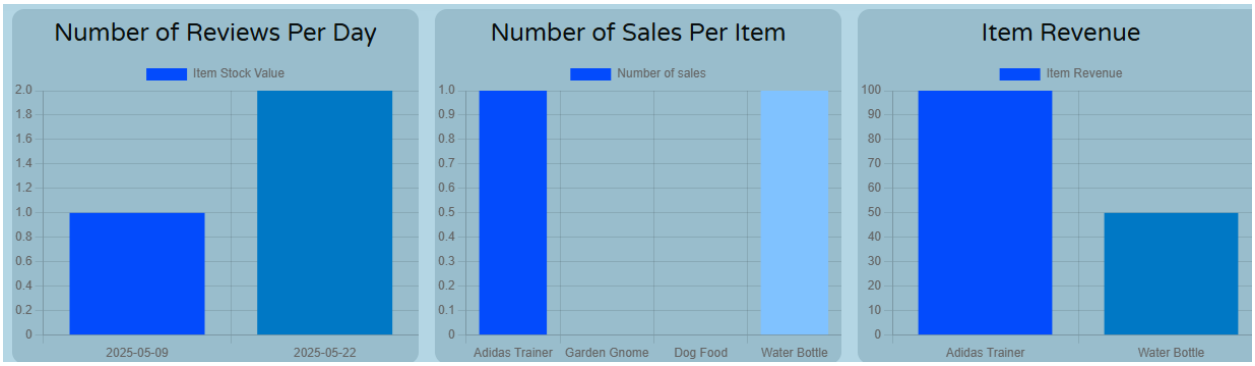

Figure A.45: Organisation Analytics (TC-F4-01)

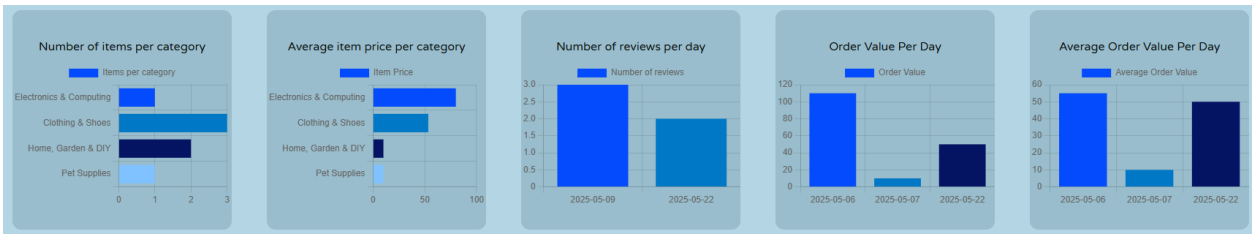

Figure A.46: Admin Analytics (TC-F4-02)

Figure A.47: Admin Analytics (TC-F4-02)