BAHRAIN POLYTECHNIC بوليتكنك البحرين

Assessment Cover Sheet

Assessment	Industry Project -Thesis		
Assessment	Controlled	Individual	Must-pass
Due Date	31st December 2024	**Course Code**	IT8X99
Course Title	Cooperative Learning Program		
Internal Moderator's	Mr. Shimaz Khan		
External Examiner's			

Instructions:

1. This cover sheet must be completed (section in blue below) and attached to your assessment before submission in hard copy/soft copy.
2. The time allowed for this assessment is XXX minutes/hours/days. (Check Moodle.)
3. This assessment carries 30 marks distributed to a total of 100 questions assessing CILO X and CILO X.
4. The materials allowed for use in this assessment are XXX, XXX, and XXX.
5. The **use of generative AI tools is strictly prohibited**.
6. References consulted (if any) must be properly acknowledged and cited.
7. The assessment has a total of XXX pages.

Learner ID	202101314	**Date Submitted**	
Learner Name	Mohamed Yanis Moussai		
Programme	IT8X99		
Programme	Industry Project		
Lecturer's			

By submitting this assessment for marking, I affirm that this assessment is my own work.

Learner	

Do not write beyond this line. For assessor use only

Assessor's Name			
Marking Date		**Maks Obtained**	

Comments:

by

Mohamed Yanis Moussai

A Thesis Submitted in

(Partial) Fulfillment of the

Requirements for the Degree of

Bachelor of Science

in Information & Communication Technology

at

Bahrain Polytechnic

Dec 2025

Title

BAYG IT Operations, Infrastructure, & Development

Copyright

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature: Yanis

Name: Mohamed Yanis Moussai

Date: 31st December 2025

Abstract

The 3rd Asian Youth Games (BAYG), a major international multi-sport event bringing together over 4,000 athletes from 45 countries, mandated the rapid establishment of a stable, scalable, and standardized Information and Communication Technology (ICT) environment under the Bahrain Olympic Committee (BOC). This project addresses the critical challenges faced by the small IT team, documenting the author's comprehensive contributions across Planning, Application Development, and Core Infrastructure Deployment.

The project's integrated approach focused on three critical deliverables:

1. **Standardization**: Developing and formalizing essential Standard Operating Procedures (SOPs) for core processes to ensure operational consistency and risk mitigation across IT support functions.
2. **Application Stabilization**: Contributing to the design, bug fixing, and optimization of the critical e-commerce website, which was built on the PERN stack (PostgreSQL, Express, React, Node.js).
3. **End-to-End Infrastructure Deployment**: Executing the entire physical and logical network architecture, including configuring servers, routers, printers, IP telephone (ground phones), and managing IP address allocation. This also involved creating and validating automation scripts for the mass, consistent provisioning of nearly 800 client laptops.

The project successfully delivered a complete, validated framework for operational readiness. Key results include the deployment of formalized SOPs, functional stabilization of the PERN-stack application components, and the robust configuration of the full network and endpoint infrastructure, significantly reducing manual effort and ensuring the stability required for the Games.

In conclusion, this work provided a vital, multi-layered ICT solution that spanned planning, application development, and comprehensive infrastructure management, ensuring the efficiency and successful execution of the BAYG's technology requirements.

Acknowledgments

First, I would like to thank the course coordinator of the CLP, Mr. Shabaz Ali, for his continuous support throughout the project.

Second I would also like to thank my supervisor Mr. Hasan Mansoor, for his continued diligence in guiding me throughout the project.

Third, I express a very deep and sincere gratitude to Mr. Rafiq Bhatti, BAYG, and BOC for accepting me as a trainee and allowing me to be a part of the IT team of such a unique and critical event.

Fourth, I am very thankful for all the teaching staff at Bahrain Polytechnic for teaching me all the knowledge that I applied into this project and for providing me with all necessary materials.

Lastly, I would like to thank my family and friends for always keeping me focused on completing the project and providing me with the best environment to work in.

Table of Contents

List of figures

List of Abbreviations

Abbreviation	Meaning
BOC	Bahrain Olympic Committee
OCA	Olympic Council of Asia
BAYG	Bahrain Asian Youth Games
ITCC	Information Technology Command Center
HQ	Headquarters (in Manama)
GMS	Games Management Systems
MIS	Mobile System (app on google play store)
OTT	Live streaming platform
RC	Rate card system (e-commerce website)
WRS	Web results System
SOP	Standard of Procedure
SKU	Stock Keeping Unit
NOC	National Olympic Committee

List of Tables

1. Introduction

This project focuses on delivering comprehensive IT operations, systems development, automation, and technical support throughout the preparation and execution phases of the Bahrain Asian Youth Games (BAYG). The scope of work involves supporting multiple internally developed platforms, coordinating with international vendors, implementing automation for large-scale device provisioning, and ensuring reliable IT performance across various event venues.

The introduction is structured into four parts: project rationale, project objectives, the proposed solution, and an overview of the report structure.

1.1 Project Rationale

The IT department plays a central role in ensuring that operational activities remain stable, efficient, and resilient; especially in large, multi-venue events where problems must be resolved quickly and systems must remain continuously available. The project required a combination of development, operations management, troubleshooting, documentation, and automation.

Several challenges highlighted the need for a structured IT approach:

- multiple systems from different vendors needing integration
- the requirement for smooth coordination between technical teams
- the necessity of reliable helpdesk operations during high-pressure periods
- a large volume of equipment that needed standardized configuration
- the importance of maintaining clear operational procedures and monitoring tools

Because IT directly affects event operations, broadcasting, logistics, staff workflows, and security, a consolidated and well-documented technical framework was essential.

1.2 Project Objectives

The project aimed to achieve the following objectives:

Main Objectives

1. Assist in the development, improvement, and deployment of internal systems including websites and GMS components.
2. Develop automation scripts using PowerShell to accelerate and standardize laptop provisioning.

Secondary Objectives

3. Coordinate with Bornan on vendor-supplied systems such as WRS, MIS, TSR, and OTT.
4. Establish Standard Operating Procedures (SOPs) for Command Center IT operations.
5. Test, track, and manage IT assets across venues.

6. Deliver helpdesk and technical support during the setup phase and the games period.

These objectives ensure operational readiness, efficient workflows, and minimized manual overhead.

1.3 Proposed Solution

To meet the objectives, the project adopted an operational-development hybrid solution consisting of:

- Enhancement of internal systems using React.js, PERN stack, AWS, and Vercel.
- Coordination with vendor platforms to ensure timely updates, stability, and data flow.
- Command Center monitoring, including servers, applications, network health, and assets.
- Automation tools designed to provision large numbers of laptops using PowerShell GUIs and scripts.
- Centralized helpdesk operations through Freshdesk for tracking and resolving technical issues across venues.

This approach increased efficiency, reduced system downtime, and ensured consistency across IT processes.

1.4 Description of the Report

The remainder of this document is structured as follows:

- Section 2 – Background Work:
  Covers relevant theory, technologies used, vendor systems, and comparisons with existing solutions.
- Section 3 – Methodology:
  Explains the requirements elicitation process, system design, implementation, and testing activities.
- Section 4 – Discussion and Conclusion:
  Summarizes achievements, unresolved challenges, ethical considerations, future improvements, and personal reflections.
- References
  Provides all cited scholarly and technical resources.
- Appendices
  Contain supporting materials such as diagrams, configurations, testing results, and user manuals.

2. Background Work

The successful delivery of IT operations for a large multi-sport event such as the Bahrain Asian Youth Games (BAYG) requires a multidisciplinary blend of technical knowledge, operational planning, and integration across multiple platforms and vendors. This section provides the theoretical and technological foundation necessary to understand the project, covering key concepts in IT service management, systems integration, automation, web architecture, and infrastructure deployment. It also examines the technologies selected for the project, critiques existing solutions, and justifies the need for customized, internally developed systems that aligned with the operational demands of the Games.

2.1 Related Theory

2.1.1 IT Service Management (ITSM)

Large-scale events function similarly to enterprise environments in terms of service expectations but operate under tighter timelines and higher volatility. IT Service Management (ITSM) frameworks; including incident response, change control, escalation paths, and service-level policies, provided structure to IT operations during BAYG.

Principles such as:

- incident ownership,
- log-based accountability,
- asset lifecycle tracking, and
- centralized command centre operation

shaped the IT Command Centre (ITCC) setup. These concepts justified the development of SOPs, ticketing workflows, escalation channels, and reporting structures. Without standardized ITSM practices, operational uncertainty and inconsistent resolutions would hinder venue readiness and compromise Games delivery.

2.1.2 Systems Integration

Major sporting events involve multiple systems developed by different vendors, each responsible for a specific domain: results, accreditation, mobile applications, timing, scoring, and online services. System integration theory explains how independent systems must communicate through defined boundaries, protocols, and coordination processes. For BAYG, internal applications such as the Rate Card System (RC-BAYG) had to operate in harmony with external vendor systems like WRS, MIS, OTT platforms, and games management solutions. Ensuring compatibility, reducing friction, and maintaining real-time operational readiness were all critical integration challenges.

2.1.3 Automation & Scripting

Automation plays a vital role in reducing manual effort, minimizing errors, and accelerating deployment, especially when configuring hundreds of devices in compressed timelines. PowerShell, widely used in enterprise environments, offers deep integration with Windows systems, allowing administrators to script repetitive actions such as installing updates, disabling services, applying configurations, and enforcing policies. Automation theory

emphasizes consistency, predictability, and scalability, objectives that directly influenced the development of laptop-setup scripts and provisioning tools used for nearly 800 devices during the Games.

2.1.4 Web Application Architecture

Modern web systems use modular architectures that separate frontend, backend, and database layers.

- Frontend manages user interface and user interactions (e.g., React.js).
- Backend handles business logic and API endpoints (Node.js / Express.js).
- Databases store persistent data (PostgreSQL).

Cloud hosting platforms such as AWS and Vercel utilize distributed infrastructure to ensure scalability, version control, and fault tolerance.

2.1.5 Asset Management Theory

Large-scale events handle thousands of laptops, radios, printers, Wi-Fi units, and network devices, each distributed across multiple venues. Asset management principles; identification, allocation, tracking, recovery, maintenance, and retirement, ensure controlled operations.

These principles informed:

1. checklists,
2. validation procedures,
3. return workflows, and
4. escalation pathways used by ITCC teams.

Without asset management theory application, devices risked loss, misallocation, downtime, or security risk.

2.2 Project Technology

The BAYG project depended on a diverse set of technologies across web development, infrastructure, automation, and collaboration. Each technology was selected based on suitability, scalability, and alignment with the operational environment of an international sporting event.

2.2.1 Web Platforms

The project utilized multiple web-based systems that served different operational needs:

React.js (Frontend):

Used for bayg.bh, offering fast UI updates and reusable components.
Strengths: high performance, component-based design.
Limitations: requires bundling and can be complex for beginners.

PERN Stack (Backend Systems):

Used for rc.bayg.bh and atm.bayg.bh.

PostgreSQL, **E**xpress.js, **R**eact.js, **N**ode.js.
Strengths: full JavaScript environment, scalable, flexible APIs.
Limitations: requires strong version control discipline and structured deployment pipelines.

Freshdesk (Helpdesk System):

A cloud platform providing ticketing workflows, SLAs, and escalation paths.
Used for: help.bayg.bh.

Table 1 Technology Stack

Technology	Purpose
Postgres	PostgreSQL is an open-source relational database used for storing structured data reliably and efficiently. In this project, it powered the backend of the PERN-based systems such as asset management and the rate-card platform. It was chosen for its stability and scalability during high-volume operations.
Express.js	Express.js is a lightweight Node.js web framework used to build fast and modular backend APIs. In this project, it handled routing, data processing, and communication between the frontend and PostgreSQL databases. It was chosen for its simplicity, flexibility, and strong ecosystem support.
React.js	React.js is a frontend JavaScript library used to build fast, interactive, and component-based user interfaces. In this project, it powered the BAYG public website, ensuring responsive design and smooth navigation. It was chosen for its reusability, performance, and strong community support.
Node.js	Node.js is a JavaScript runtime that allows backend applications to run on the server side. In this project, it powered the logic behind the PERN-based systems and handled communication with databases and APIs. It was chosen for its speed, scalability, and ability to support real-time operations.
PowerShell	PowerShell is a task automation and configuration scripting language built for Windows environments. In this project, it was

	used to automate bulk laptop setup and streamline device provisioning. It was chosen for its deep integration with Windows systems and ability to significantly reduce manual workload.
Draw.io	draw.io (now known as diagrams.net) is a web-based diagramming tool used to create system architectures, workflows, and technical documentation. In this project, it helped visualize network structures, ITCC processes, and system relationships. It was chosen for its simplicity, collaboration features, and ease of integration into documentation.
Microsoft Office	Microsoft Office is a productivity suite used for creating documents, spreadsheets, and presentations. In this project, it supported reporting, documentation, and communication across IT operations. It was chosen for its reliability, widespread compatibility, and ease of use in professional environments.
CrediMax	The CrediMax API is an online payment gateway interface used to process secure card transactions in web applications. In this project, it enabled seamless payment handling within the BAYG rate-card system. It was chosen for its security, local banking integration, and reliability for handling financial operations.

2.2.2 Vendor Systems

BAYG relied on multiple vendor technologies such as the Games Management System (GMS), Web Results System (WRS), Mobile Application (MIS), Ticketing, and OTT livestream platforms. These external systems required coordination, interface validation, and constant communication to ensure alignment with event timelines. The IT team frequently collaborated with vendor representatives to troubleshoot issues, synchronize event data, and maintain service availability.

Bornan provided core event systems:

- **WRS (Web Result System)** – publishes live scores and event outcomes.
- **MIS (Mobile Information System)** – portable event information platform.
- **TSR (Time Scoring & Results)** – captures and processes timing data.

- **OTT Broadcasting** – livestreaming and on-demand video distribution.

These systems required coordination, issue reporting, and operational monitoring.

2.2.3 Infrastructure Technologies

The physical IT ecosystem included switches, routers, servers, firewalls, IP telephony devices, printers, and access points deployed across venues. Configuration tasks covered:

- VLAN segmentation
- IP addressing schemes
- DHCP scopes
- Access control
- Printer routing
- Telephony configurations
- Monitoring and remote diagnostics

A reliable network backbone was mandatory to support media operations, results transmission, broadcasting, and venue logistics.

- **Amazon AWS:**
  Used for backend hosting, CI/CD, monitoring, and storage.
  Strengths: high reliability, scalability, strong documentation.
  Challenges: cost optimization and IAM policy management.
- **Vercel:**
  Used for hosting React-based frontends.
  Strengths: fast global CDN, automatic builds.
  Challenges: limited backend control.
- **Local Servers & Network Infrastructure:**
  Deployed for venue operations, broadcasting support, and internal testing.
  Challenges: physical maintenance, cross-venue coordination, and latency issues.

2.2.4 Automation Tools

PowerShell served as the primary automation tool for laptop provisioning. It enabled large-scale deployment by:

- Automating installations
- Enforcing security settings
- Disabling unnecessary services
- Accelerating setup workflows
- Ensuring uniform device configurations

Automation was essential to meet deadlines and reduce the workload on IT staff during the compressed pre-Games period.

2.2.5 Collaboration & Version Control

GitHub served as the central repository for code, issues, documentation, and version control.
Benefits: team collaboration, pull requests, issue tracking.
Challenges: requires strict discipline in branching strategy.

2.3 Related Work (Literature Review)

Several studies and industry practices highlight the importance of robust IT frameworks in large events.
Research on sports events shows:

- Results systems must maintain high accuracy and low latency, as scoring delays impact fairness and broadcast alignment.
- Automation significantly improves operational throughput in multi-venue events with limited time windows.
- Command Centers are essential in ensuring cross-team coordination and centralizing monitoring.
- Asset management systems reduce equipment loss and improve deployment efficiency.

Additionally, literature on ITSM emphasizes standardized processes (SOPs, escalation paths, ticketing systems) to ensure consistent service delivery.

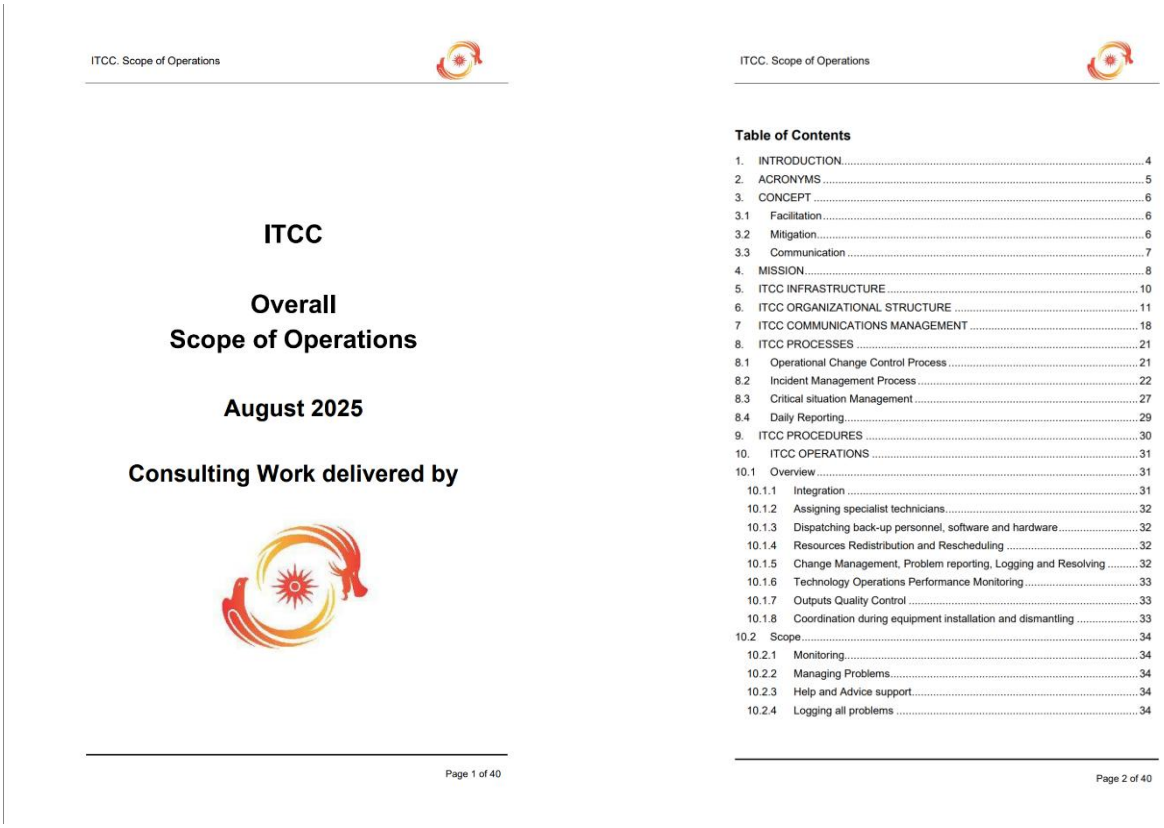
While vendor-provided systems exist for scoring and results, internal systems must complement them with asset management, helpdesk operations, and monitoring; areas often under documented in academic literature but essential in real-world events.

2.4 Market Research (Existing Solutions)

2.4.1 Existing Commercial Solutions

As part of the initial market research, the IT team received an official *ITCC Scope of Operations* document from the Olympic Council of Asia (OCA), which outlined an established framework for managing IT operations during multi-sport events. This document served as a reference model covering core operational domains such as incident management, change control, communication workflows, infrastructure requirements, organizational structure, and reporting cycles. While the ITCC framework provided a comprehensive high-level structure for Games-time IT management, it did not offer ready-to-deploy technical solutions or platforms tailored to the Bahrain Asian Youth Games' specific systems (such as the BAYG websites, GMS integrations, or asset management workflows). As a result, the document functioned primarily as a benchmark and guidelines reference rather than a complete commercial off-the-shelf (COTS) solution. This justified the decision to develop in-house systems and customized operational procedures that could align with the OCA standards while still meeting the unique requirements of BAYG's digital ecosystem.

Link to the document: https://drive.google.com/file/d/19FF6ImRIkSj8k3C-EunGkluoj1r1M1hP/view?usp=sharing



ITCC. Scope of Operations

ITCC

Overall
Scope of Operations

August 2025

Consulting Work delivered by

Page 1 of 40

ITCC. Scope of Operations

Table of Contents

Page 2 of 40

Figure 1 ITCC Scope of Operations Documentation by OCA

Several other IT management and event-support platforms exist, such as:

- **Atos Olympic Management Systems** – used in the Olympics for accreditation, results, and logistics.
- **Sportity** – event communication and information distribution platform.

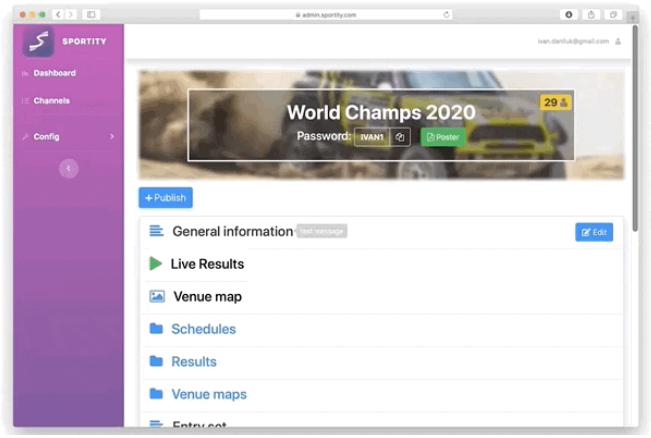


Figure 2 sporitiy system

- **ZenDesk / Freshdesk** – general helpdesk and ticketing systems.

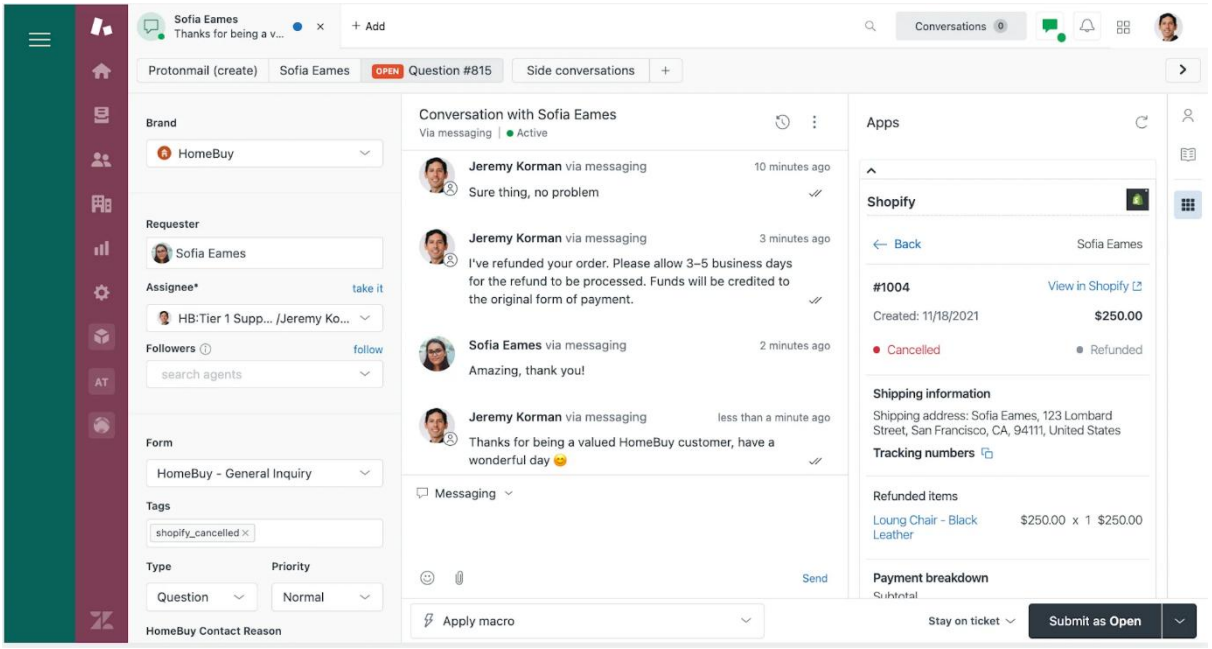


Figure 3 zendesk system

2.4.2 Comparison with Project Requirements

Table 2 project requirements comparison

Requirement	Existing Solutions	Limitations / Gaps
Results & Scoring	Atos, Bornan	no customization allowed
Helpdesk	Freshdesk, Zendesk	Works well but must be integrated with venue workflows
Laptop Provisioning	Intune, JAMF	Expensive, heavy setup, not feasible for short-term events

2.4.3 Justification for In-House Development

Customization: Event-specific requirements cannot be satisfied by generic commercial platforms.

Cost efficiency: Large enterprise solutions (e.g., Atos-level systems) are too costly for short-term events.

Flexibility: Internal team can rapidly deploy fixes, modify features, and respond to real-time operational needs.

Integration: In-house systems integrate easily with vendor APIs and internal workflows.

Control: Better oversight of hosting, performance, monitoring, and data.

Thus, a hybrid approach; vendor systems for scoring and results + in-house systems for internal operations, was the most effective and justified solution.

3. Gathering Requirements

The requirements for this project were gathered through a combination of direct communication, observation, and reference to official documentation. Meetings with IT supervisors and development leads helped define the technical expectations, system functionalities, and operational responsibilities that needed to be addressed. Coordination with Bornan and other vendors contributed additional requirements related to WRS, MIS, TSR, and broadcasting systems.

To ensure operational accuracy, requirements were also derived from observing daily workflows within the IT department and reviewing existing processes used during the Games preparation phase. The *ITCC Scope of Operations* document provided by the Olympic Council of Asia served as a foundational reference, outlining best practices for incident management, escalation, communication structures, and IT operations management. Discussions with end-users, including venue staff, administrators, and system operators, helped refine usability needs and identify practical challenges that the system and tools needed to solve.

This multi-source approach ensured that both technical and operational requirements were understood clearly, validated across stakeholders, and aligned with the overall BAYG event structure.

3.1 RC-BAYG Requirements

3.1.1 Functional Requirements

Functional requirements describe the core features and actions that the RC BAYG Website must support for it to operate correctly. They define what the system should do, the interactions between users and the application, and the essential processes required to complete tasks such as order submission, validation, payment, and administration. These requirements form the foundation of the system's behavior and ensure that all critical operations are implemented.

Table 3 RC BAYG Functional requirements

ID	Functional Requirement	Description
FR1	User Authentication	Users must be able to register and log in securely.
FR2	Order Creation	Users must submit new rate-card orders through the frontend. Users must have a calendar to choose from the beginning and end dates for renting the products.
FR3	Order Validation	Backend must validate order fields, quantities, and details.
FR4	Database Storage	All orders must be saved in PostgreSQL with unique IDs.
FR5	Admin Review	Admins must view, approve, or decline submitted orders.
FR6	Order Status Updates	System must update and display order status (pending/approved/declined).

FR7	Payment Processing	Web server must process payments via the CrediMax API.
FR8	Email Notifications	SMTP must send order confirmations and status email updates.
FR9	Activity Logging	System must log key actions for auditing and troubleshooting.
FR10	Admin Dashboard	Admins must access a dashboard with order summaries and controls, as well the ability to customize almost every aspect of the website.

3.1.2 Non-functional requirements

Non-functional requirements define the quality attributes and performance expectations of the RC BAYG Website. They describe how the system should operate rather than what it should do, covering aspects such as security, performance, scalability, usability, and overall reliability. These requirements ensure the platform remains stable, secure, and efficient during high-demand periods and throughout the event lifecycle.

Table 4 RC BAYG non-functional requirements

ID	Non-Functional Requirement	Description
NFR1	Security	Must use HTTPS, secure APIs, access control, and private DB network.
NFR2	Performance	Backend must respond quickly, even under high load.
NFR3	Scalability	System must support increased traffic during event peaks.
NFR4	Reliability	System should remain online with minimal downtime.
NFR5	Usability	UI must be simple, clear, and easy for all users.
NFR6	Maintainability	Codebase must support easy updates via GitHub.
NFR7	Compatibility	Must function on major browsers and mobile devices.
NFR8	Data Integrity	Data must remain accurate, consistent, and protected.

3.1.3 Constraints

Constraints identify the limitations and external conditions that affect the design, development, and operation of the RC BAYG Website. These include dependencies on third-party systems, security policies, hosting restrictions, strict timelines, and limited integration capabilities. Understanding these constraints is essential for evaluating design decisions and ensuring the system remains practical within real-world boundaries.

Table 5 RC BAYG Constraints

ID	Constraint	Description
C1	Vendor Dependency	Payment relies on CrediMax API availability.
C2	Hosting Limits	AWS resources and cost impact scaling ability.
C3	Time Constraints	Development had to be completed before event deadlines.
C4	Security Restrictions	Database required to remain private with restricted access.
C5	Integration Boundaries	System cannot deeply integrate with Bornan systems.
C6	Team Limitations	Limited development time and staff affected iteration speed.

3.2 Laptop Setup Requirements

The laptop setup process required a structured and standardized approach to ensure all devices were correctly configured before being issued to staff and volunteers. These requirements were derived directly from the official Laptop Setup SOP and cover all essential steps, including inspection, account creation, asset documentation, and re-issuance procedures. By defining these requirements, the setup process remained consistent, secure, and efficient across all deployed laptops throughout the event.

3.2.1 Functional Requirements

Table 6 laptop setup functional requirements

ID	Functional Requirement	Description
FR1	Initial Hardware Inspection	The system/process must include checks for laptop condition, ports, charger, mouse, keyboard, webcam, speakers, and microphone.
FR2	USB Setup Execution	The laptop must be configured using the provided USB drive and follow the instructions in the README and USB SOP.
FR3	Asset Verification	All asset tags (laptop, charger, mouse, bag) must match records before issuance.
FR4	Asset Transfer Documentation	The Asset Transfer Form must be completed, scanned, emailed, and stored in OneDrive.
FR5	Admin Account Creation	The system must support creation of a manual administrator account following defined credentials.
FR6	Standard User Account Creation	A standard user account must be created manually or via script for deployment.
FR7	Laptop renaming	Laptop must be renamed according to User account Name
FR8	Windows Update Configuration	Pending updates must be installed, and automatic updates must be disabled.
FR9	Time Zone Configuration	The time zone must be manually set to (UTC+03:00) and auto-adjust disabled.
FR10	Laptop Re-Issuance	Re-issued laptops must undergo account setup, activation, driver installation, and configuration steps.

3.2.2 Non-Functional Requirements

Table 7 laptop setup non-functional requirements

ID	Non-Functional Requirement	Description
NFR1	Consistency	All laptops must follow the same standardized setup process for uniform configuration.
NFR2	Usability	Steps must be simple enough for technicians to follow reliably, with optional automation via USB scripts.
NFR3	Accuracy	Asset information, forms, and documentation must be precise to avoid tracking errors.
NFR4	Reliability	The setup procedure must produce a fully functional laptop ready for deployment without repeated reconfiguration.
NFR5	Security	User data must be deleted during returns, and admin accounts must be protected with the correct credentials.
NFR6	Maintainability	The process must be easy to update when USB scripts or SOP versions change.

3.2.3 Constraints

Table 8 laptop setup constraints

ID	Constraint	Description
C1	Hardware Variations	Different laptop conditions or hardware issues may delay or prevent setup.
C2	Script Limitations	USB scripts may not cover all edge cases, requiring manual steps.
C3	Network Dependency	Activation, updates, and printer installation require stable internet access.
C4	Time Constraints	Large-volume laptop setup must be completed within tight event deadlines.
C5	Human Error	Manual documentation, scanning, and form handling may introduce mistakes.
C6	Resource Availability	Printer access, scanning equipment, or missing accessories may slow the process.

3.2.4 Laptop Setup Standard Operating Procedure (SOP) + Checklist

The Laptop Setup SOP establishes a standardized, repeatable process for preparing all BAYG laptops prior to deployment across event venues. Its purpose is to ensure consistency, minimize configuration errors, and reduce setup time, especially given the high volume of devices required for operations. The SOP outlines every step in the lifecycle of a laptop; from initial inspection and user account creation to driver installation, system activation, and final issuance or re-issuance.

By following this procedure, technicians can ensure that every laptop is fully functional, properly documented, and configured with the correct settings, applications, and security controls. The SOP also defines the workflow for handling returned laptops, guaranteeing that user data is wiped, assets are revalidated, and devices remain ready for redeployment. This standardized approach played a critical role in maintaining operational readiness and supporting reliable IT services during the Games.

Laptop Setup Standard Operating Procedure (SOP)

1. New Laptop Setup

Initial Inspection

For every new laptop, perform the following checks:

1. Verify the laptop is present and free from physical damage.
2. Confirm that all ports (e.g., HDMI, charging port) are functioning.
3. Ensure the charger is present and undamaged.
4. Check the laptop bag for availability and physical condition.
5. Verify the mouse is present and working properly.
6. Confirm the keyboard is functional and undamaged.
7. Test speakers and microphone through **Sound Settings**.
8. Test the webcam via the **Camera app**.

Document and report any malfunction in writing.

USB Setup

- Insert the provided USB drive and read the **README** file for setup instructions.
- Refer to the **USB SOP** for detailed guidance.

2. Pre-Issuance Procedure (First-Time Distribution)

Asset Verification

- Confirm that the asset tags on the laptop, charger, mouse, and bag are correct and match records.

Asset Transfer Form

- Ensure the **Asset Transfer Form** is completed accurately.
- Scan the signed form using the Xerox printer and email it to **hda@bayg.bh**.
- Save a copy of the scanned form in the **OneDrive folder: Asset Transfer Forms** (specific folder name to be used).

3. User Account Setup

Creating an Administrator Account (Manual)

1. Navigate to: **Settings > Accounts > Other Users > Add account**.
2. Select **"I don't have this person's sign-in information"**.
3. Choose **"Add a user without a Microsoft account"**.
4. Enter a **username** only (e.g., User1, User2, User3). Don't set a password.
5. Set all security question answers as **252525**.
6. Once created, select the account → **Change Account Type** → **Administrator**.

(Note: User accounts can also be created via the USB script.)

4. Install Wi-Fi driver (if required)

4. Windows Setup

Updates

- Install all pending Windows updates.
- Disable automatic updates either manually or through the USB script:
    - Go to **Settings > Updates > Pause updates for 5 weeks**.
    - Turn off **Get the latest updates**.

Time Zone Configuration

- Navigate to: **Settings > Time & Language > Date & Time**.
- Set the time zone to **(UTC+03:00)**.
- Disable automatic time zone adjustment.

5. Laptop Returns

When laptops are returned:

1. Inspect laptop, charger, mouse, and bag for functionality and damage.
2. Inform the user that all data will be permanently deleted.
3. Delete the user account.
4. Complete the Asset form with confirmation of asset return and signatures.

6. Laptop Re-Issuance

For re-deployment of laptops:

1. Create a new standard user account (via USB script or manually).
2. Complete the Windows setup process.
3. Reapply time zone and Wi-Fi settings.
4. Confirm the administrator account remains intact.
5. Verify user account login, Wi-Fi connectivity, and printer functionality.
6. Repackage the laptop with charger and mouse in the laptop bag.

Complete the **Asset Transfer Form**, obtain signatures, scan, and upload to the OneDrive folder.

Checklist

Technician: _____
Date: ____ / ____ / _____
Laptop asset tag: _____
Issued to (if applicable): _____

Laptop Hardware Process Check

Task	Is completed	Remarks
Laptop, Mouse, Bag, and Charger are checked	•	
Laptop screen, ports, and the rest are checked for damage	•	
Laptop keyboard is checked for functionality	•	
Laptop speaker is checked for functionality	•	
Touchpad is checked for functionality	•	
Microphone is checked for functionality	•	
Webcam is checked for functionality	•	
Charger is checked for damage and functionality	•	
Mouse is checked for damage and functionality	•	
Laptop and Mouse are checked to ensure proper asset labelling	•	

Laptop Software Set-Up Check

Task	Is Completed	Remarks
New user is created for requestee	•	
Automatic updates are disabled	•	
Time zone is set to UTC +3:00	•	
Wi-Fi is connected	•	
Office 365 is installed	•	
Windows and Office 365 are activated	•	
Windows is fully updated	•	
Test print is completed	•	
Test user login is verified	•	

Laptop Handover Process Check

Task	Is Completed	Remarks
Asset request is approved in the Helpdesk system	•	
Asset Transfer Form is filled out	•	
Laptop checkout is updated in the ATM system	•	
Laptop is handed over to the requestee	•	
Form is scanned at Xerox printer	•	
Form is sent to hda@bayg.bh	•	
Scanned copy is stored in the OneDrive folder	•	

Laptop Return Process Check

Task	Is Completed	Remarks
Laptop, Mouse, Bag, and Charger are checked for damage	•	
Asset Transfer Form is completed	•	
Form is scanned at Xerox printer	•	
Form is sent to hda@bayg.bh	•	
Scanned copy is stored in OneDrive	•	

User account is deleted	•	
Laptop check-in is updated in the ATM system	•	

3.2.5 USB SOP

This SOP defines the standardized process for using the BAYG PowerShell automation scripts to configure new or reissued laptops. The scripts ensure consistent device preparation, reduce manual setup time, and enforce IT configuration policies across all laptops deployed for the Games.

Running the Setup.ps1 Script

Purpose:
To perform the initial standardized configuration of a new or reset laptop.

Steps:

1. Right-click **Setup.ps1** and select **Run with PowerShell**.
2. The script checks for administrator privileges.
    - If not elevated, it relaunches itself with admin rights automatically.
3. The script applies core configurations including:
    - Creating a standardized administrator account
    - Renaming the device using the provided device code
    - Applying basic Windows configuration settings
4. When the script completes, it displays a summary of actions performed.
5. Select **Reboot** when prompted to finalize the setup.

Expected Outcome:
A standardized laptop with correct naming, admin access, and base configuration.

Running Install-WindowsUpdates.ps1

Purpose:
To install all available Windows updates automatically and consistently.

Steps:

1. Run the script as Administrator.
2. The script verifies if the PSWindowsUpdate module is installed.
    - If missing, it downloads and installs it.
3. The script scans for all available updates.
4. All updates are installed silently without user input.
5. A detailed update log is generated and saved on the desktop.
6. Restart the device if prompted.

Expected Outcome:
The laptop is fully updated with documented installation logs.

Running Disable-WindowsUpdates.ps1

Purpose:
To permanently disable automatic Windows updates in accordance with BAYG operational requirements.

Steps:

1. Run the script as Administrator.
2. The script stops and disables these services:
    - Windows Update (wuauserv)
    - Background Intelligent Transfer Service (BITS)
3. The script applies registry rules that:
    - Block access to Windows Update settings
    - Prevent automatic update checks
4. The script confirms that updates have been successfully disabled.

Expected Outcome:
Automatic updates remain permanently disabled, ensuring no unexpected restarts or background updates.

Verification Checklist

After completing all three scripts, technicians must confirm:

- The laptop name matches the assigned naming convention.
- The new admin account has been created successfully.
- No pending updates remain after running the update script.
- Windows Update settings cannot be accessed (after disabling script).
- The device restarts successfully and operates normally.

4. Methodology

This section outlines the key technical and operational methods used throughout the project, focusing on three major areas of contribution: development and enhancement of the RC BAYG website, creation of PowerShell automation scripts for device provisioning, and active involvement in the IT Main Operations Center during Games operations. Each subsection explains the approach taken, tools and technologies used, challenges encountered, and the final outcomes achieved.

4.1 RC BAYG Website (Rate Card System)

The RC BAYG website served as the digital platform for requesting, managing, and processing billable services and assets for external stakeholders. The system was built using the PERN stack and hosted on AWS, requiring continuous development, bug fixing, and deployment improvements throughout the event preparation period.

4.1.1 System Architecture Diagram

A system architecture diagram is a high-level structural design that shows how different components of a system are organized and interact with each other. It defines how data flows, how services communicate, and how security, performance, and scalability are maintained. This helps ensure that the system is built in a clear, efficient, and secure way.

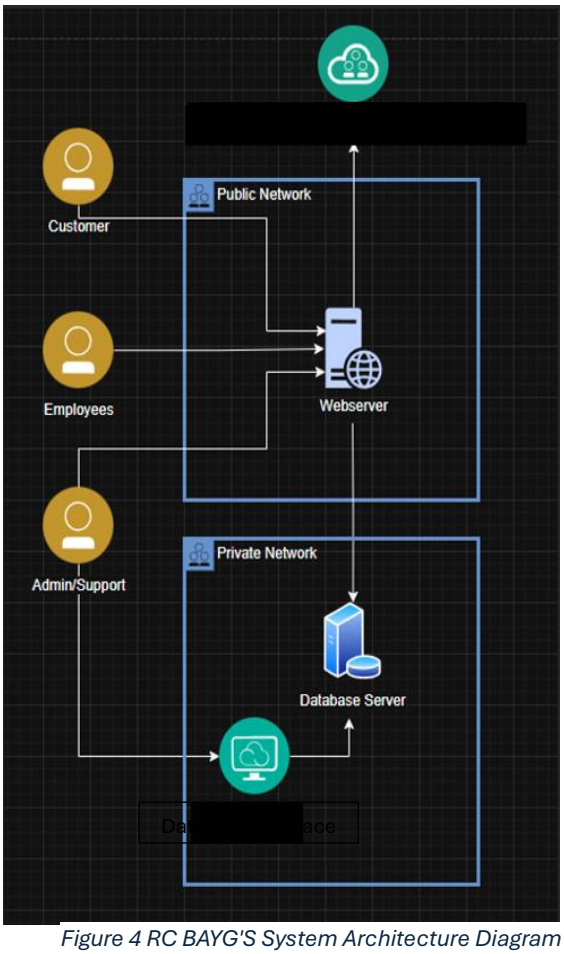


Figure 4 RC BAYG'S System Architecture Diagram

The system architecture consists of a public web server, a private database server, and an external payment integration using the CrediMax API. Customers and employees can access the web server through the public network, while the database remains isolated in a private network with access restricted only to the web server and authorized admin/support staff. When a payment is processed, the web server securely communicates with Credimax through its API, ensuring no direct exposure of sensitive data.

4.1.2 Sequence Diagram for creating orders in RC BAYG

A sequence diagram is a visual representation that shows the order of interactions between system components over time. It illustrates how processes, users, and services exchange messages step-by-step to complete a specific function or workflow. This helps clarify system behavior, timing, and dependencies during execution.

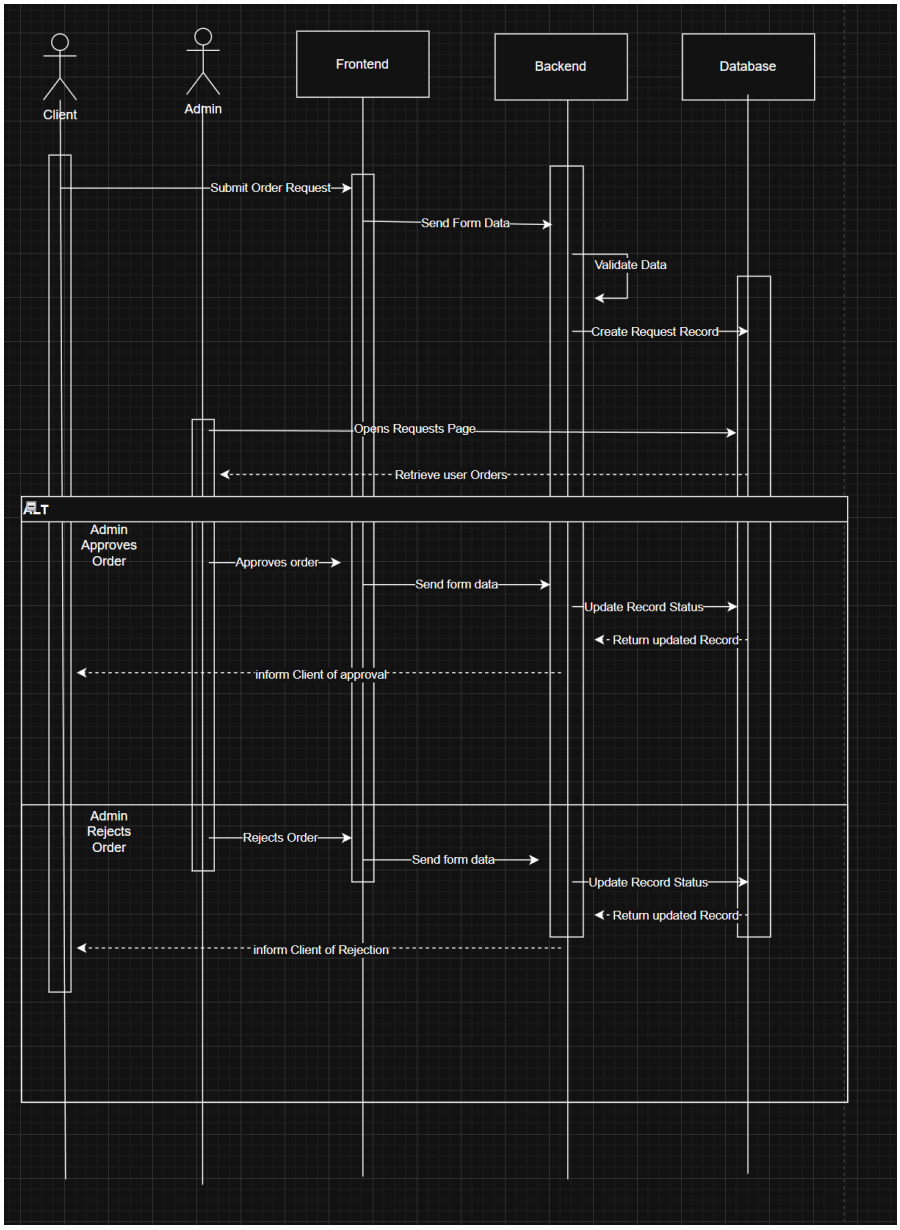


Figure 5 Sequence diagram for creating order in RC BAYG

The sequence diagram illustrates the order flow between the client, administrator, frontend, backend, and database components. The process begins when the client submits an order request through the frontend, which is then sent to the backend for validation. After validation, the order details are stored in the database, making it available for administrative review. The admin then accesses the system through the frontend to either approve or decline the order, and the final decision is updated in the database and reflected back to the client.

4.1.3 Website UI Design and user guide

This section provides an overview of the design, layout, and user interaction flow of the RC BAYG website. It explains the primary interface components, navigation structure, and key functions that guide users through browsing products, selecting rental periods, and submitting orders. The user guide ensures that customers and administrators can understand and operate the platform efficiently, supporting smooth and intuitive use throughout the event. The detailed manual is provided through the linked document and supplemented by screenshots and descriptions of the core pages, including the dashboard, categories, product listings, and order management interfaces.

Link to user guide:
https://docs.google.com/document/d/1BsfWb9JZbbbwlp7z9h43HAbAbOXXMbYI/edit?usp=sharing&ouid=112253711178265508181&rtpof=true&sd=true

Dashboard

After Signing in, You Should be greeted with the page below:

Figure 6 BAYG Store Main Page

This is the Dashboard. At the top of the page, you can see tabs to redirect you wherever you need.

Categories

After clicking on the Categories button, you will be redirected to the Categories section, where you can view the different categories to browse your desired products, such as in the image below:

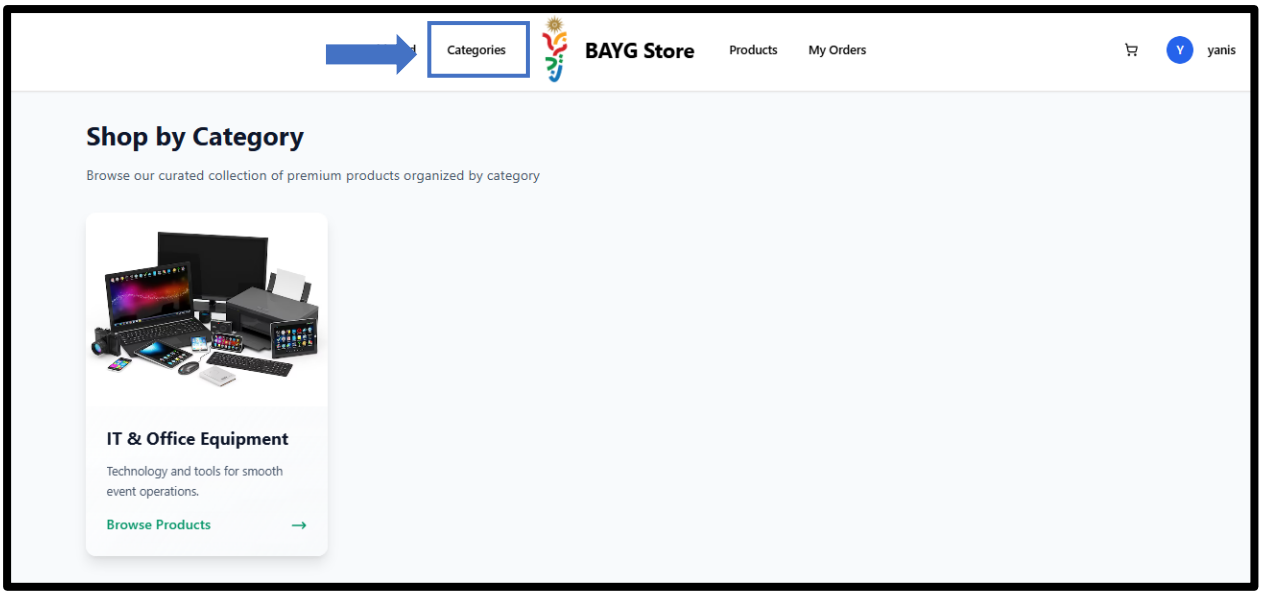


Figure 7 BAYG Store Tab Navigation

Products

Clicking on the Products tab will redirect you to the Products section, where you can browse all the products available, such as in the image below:

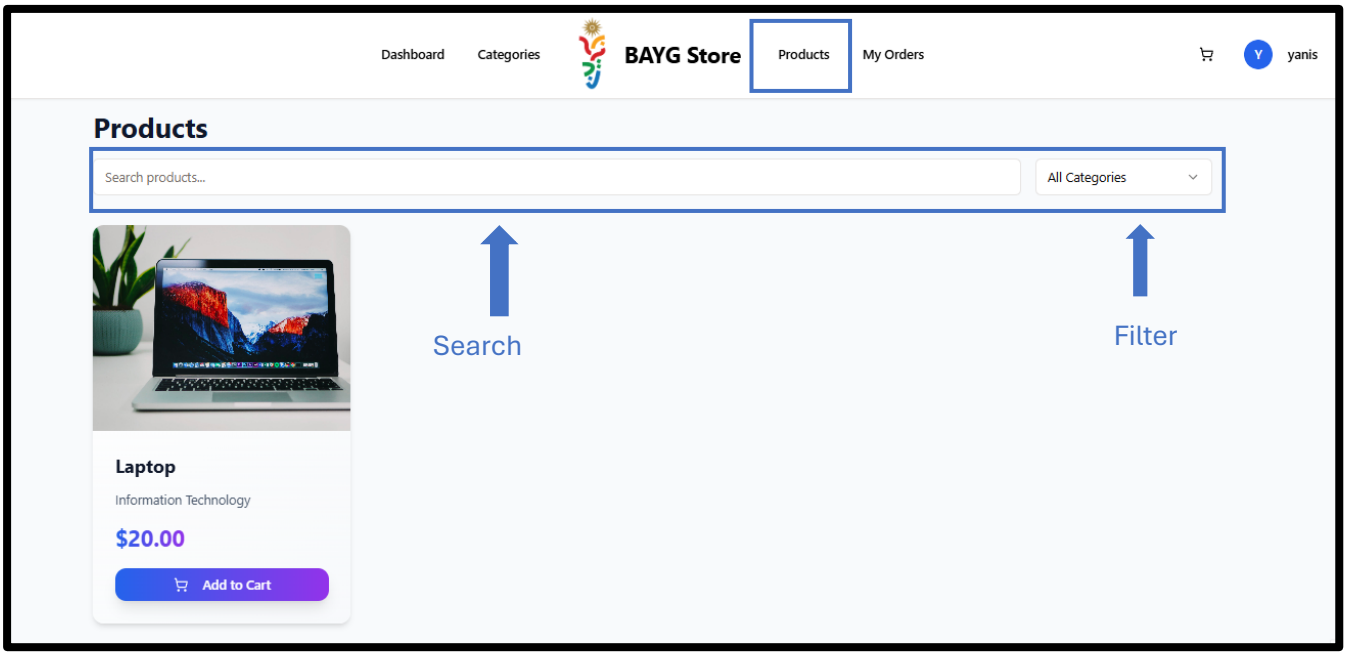


Figure 8 BAYG Store Product Search

You can use the search bar to look for any specific product you would like to purchase, as well as filter the products by category.

After clicking on your desired product, you will be redirected to the product's page where you can select your rental period and view the product's base price, description and your selection's total price.

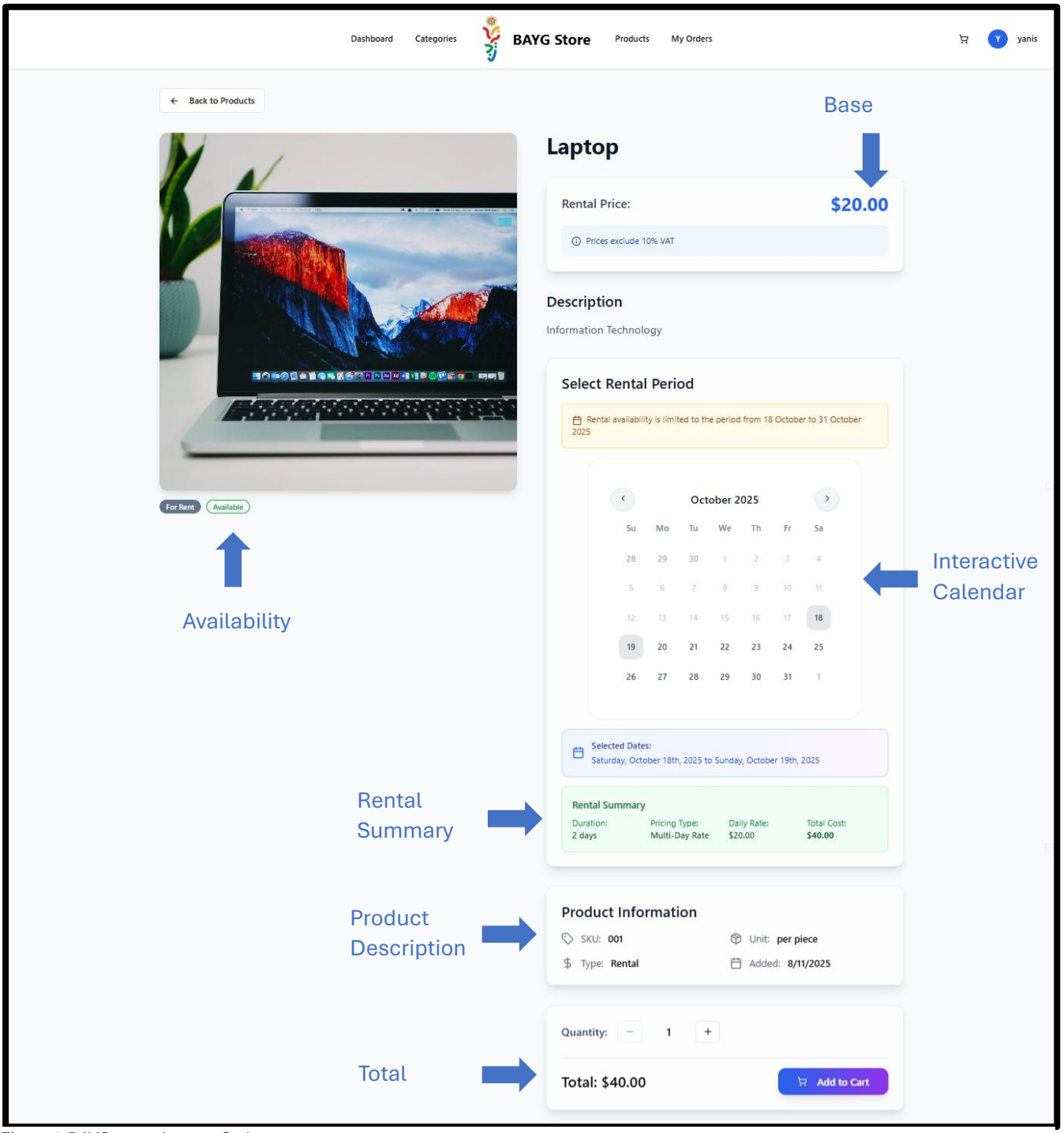


Figure 9 BAYG store how to Order

Orders

After adding your required products, you can access your cart for a final review of your order before submission, such as in the image below:

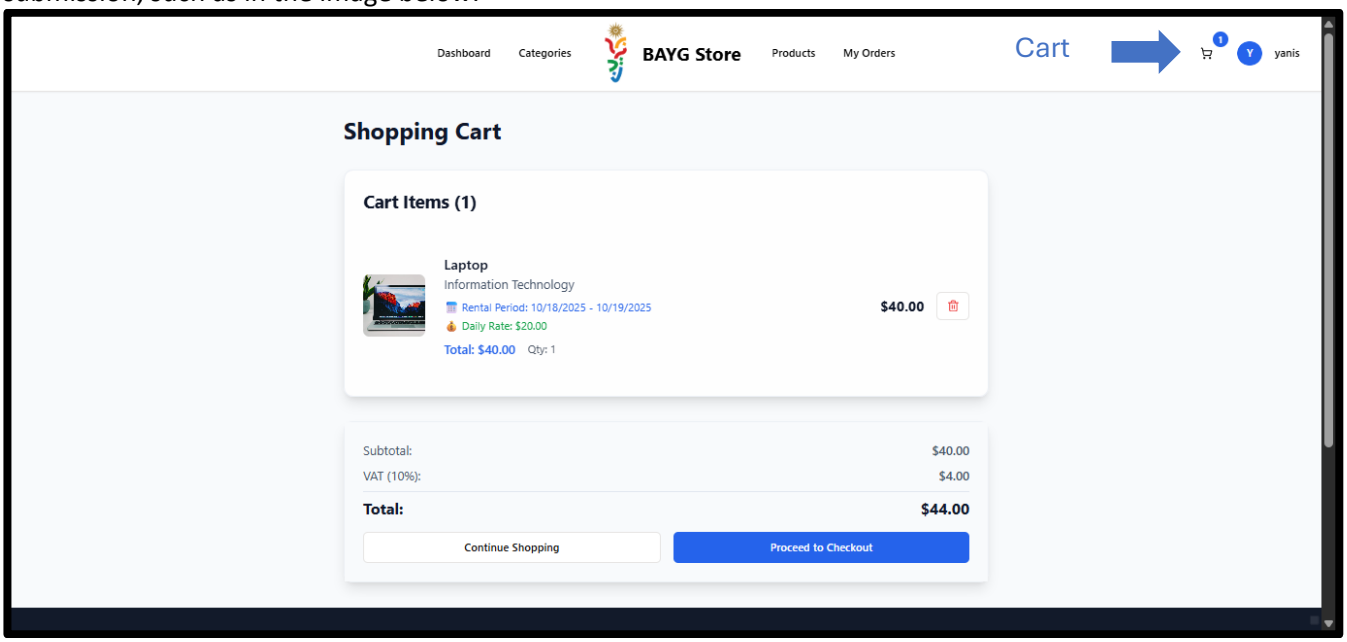


Figure 10 BAYG Store checkout page

You can delete items you don’t want, continue shopping, or proceed to checkout.

4.2 PowerShell Scripts

4.2.1 Flowchart Diagram

A flowchart diagram is used to visually represent the logical steps, decisions, and processing flow of a system or operation. It helps illustrate how tasks progress from start to end, including decision branches and possible outcomes. This provides a clear overview of the system's workflow, making it easier to understand, improve, and troubleshoot.

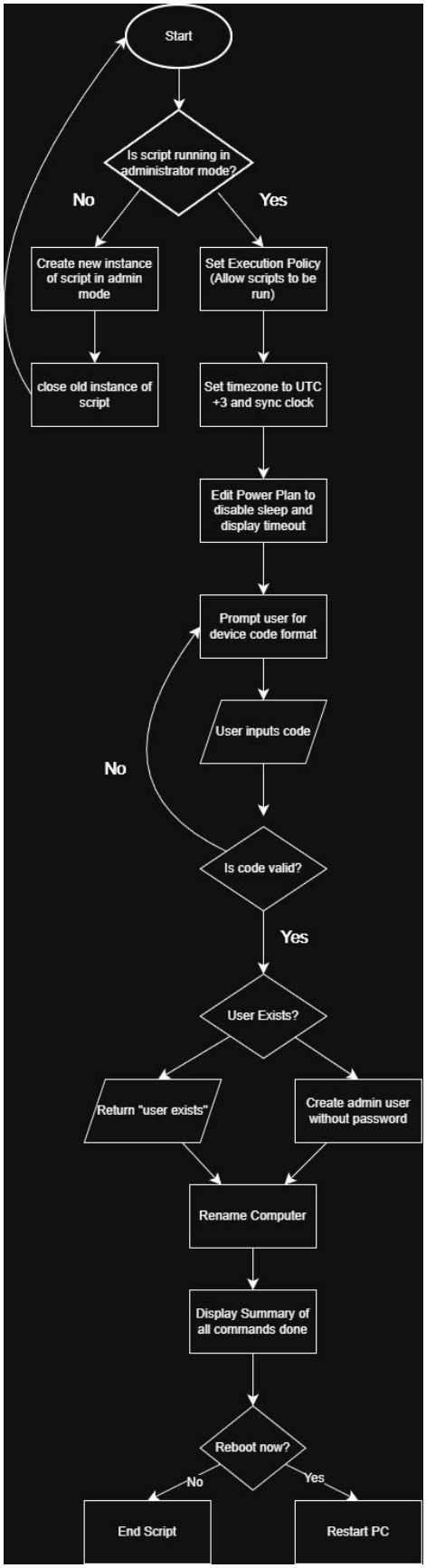


Figure 11 Setup script Flowchart

Setup.ps1

The **Setup.ps1** script is an automated Windows configuration tool designed to prepare new devices with standardized IT settings. It first ensures that it is running with administrator privileges, then configures essential system settings, creates a personalized admin account, and renames the computer based on a provided device code. Finally, it presents a summary of performed actions and offers a reboot option.

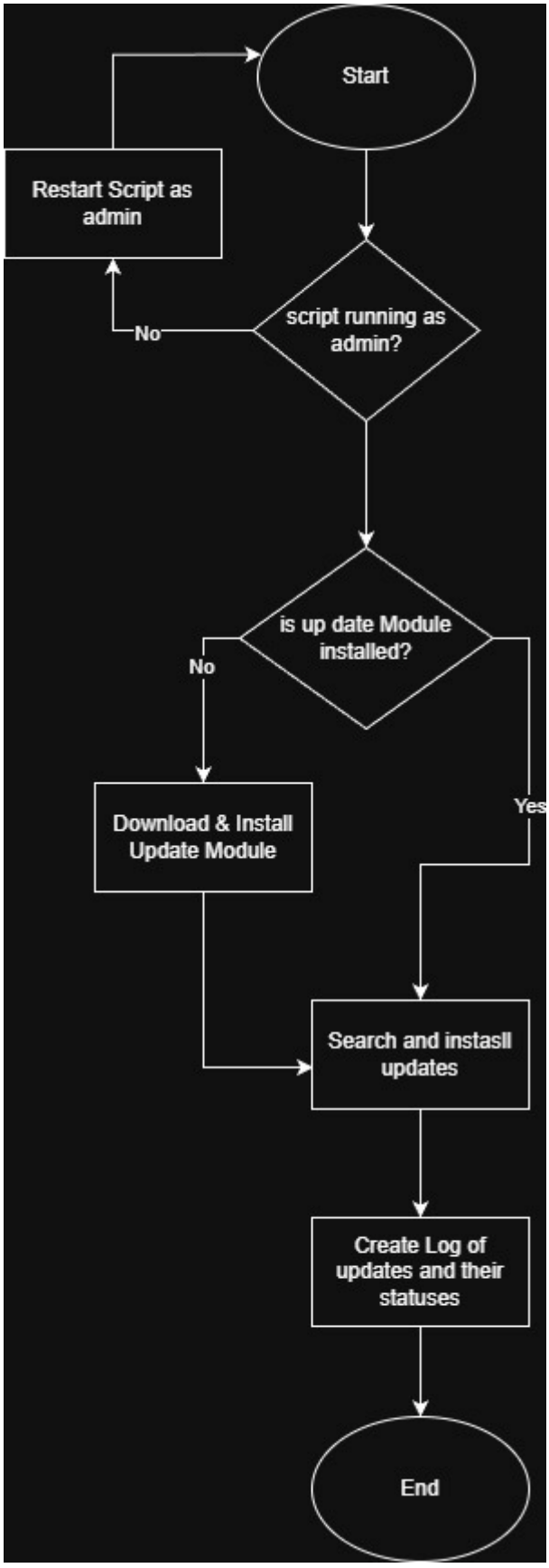


Figure 12 Update windows Script

Install-WindowsUpdates.ps1

This script automatically installs all available Windows updates by ensuring it runs with administrator rights, adding the required update module if it is not already installed, scanning for updates, installing them silently, and saving a detailed update log to the desktop before exiting.

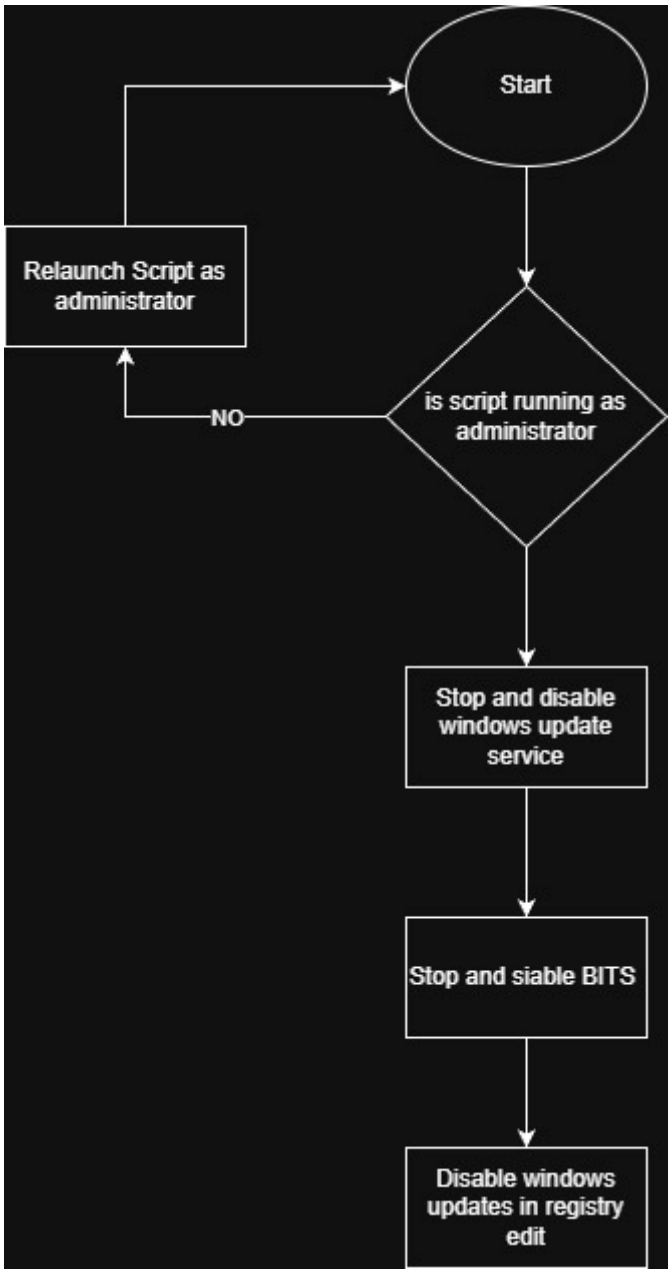


Figure 13 Disable windows Update Script

Disable-WindowsUpdates.ps1

This script permanently disables Windows Updates by ensuring it runs with administrator privileges, stopping and disabling the Windows Update and BITS services, and applying registry policies that block automatic updates and prevent access to update settings.

5. Implementation

The implementation phase focused on translating the system designs, workflows, and requirements into functional, operational components. This involved developing the RC BAYG website using the PERN stack, deploying backend and frontend services on AWS and Vercel, implementing PowerShell automation tools for laptop provisioning, and configuring IT infrastructure required for Games operations. This section outlines the most significant implementation steps and demonstrates how each component was built, integrated, and tested.

5.1 RC BAYG Website Implementation

The RC BAYG website uses a clear and organized folder structure to separate frontend, backend, and database components, making the system easier to maintain and update during development. This layout improves readability, simplifies debugging, and supports smooth collaboration between developers. Here is the file structure below:

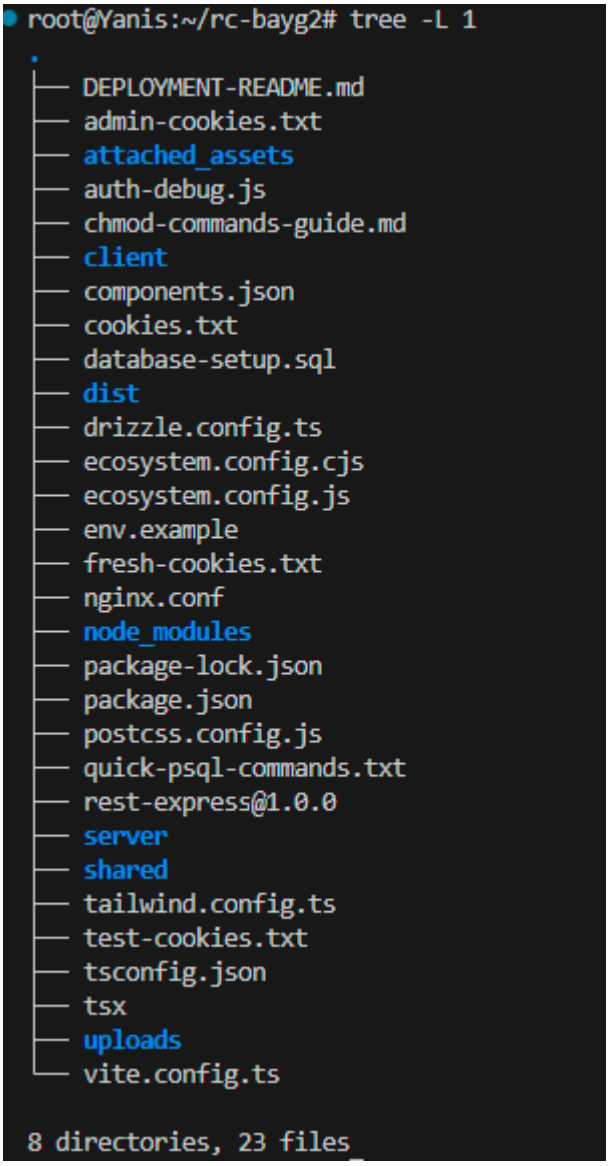


Figure 14 Level 1 depth Folder structure

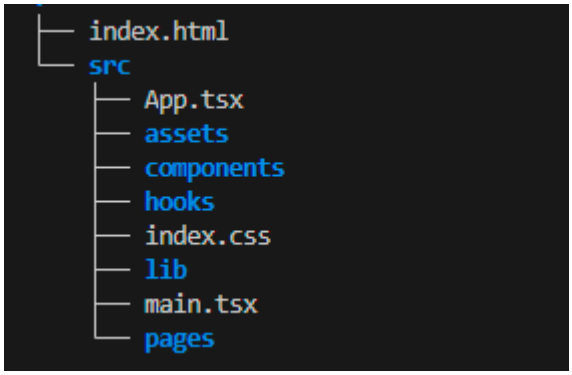


Figure 15 Client (Frontend) files

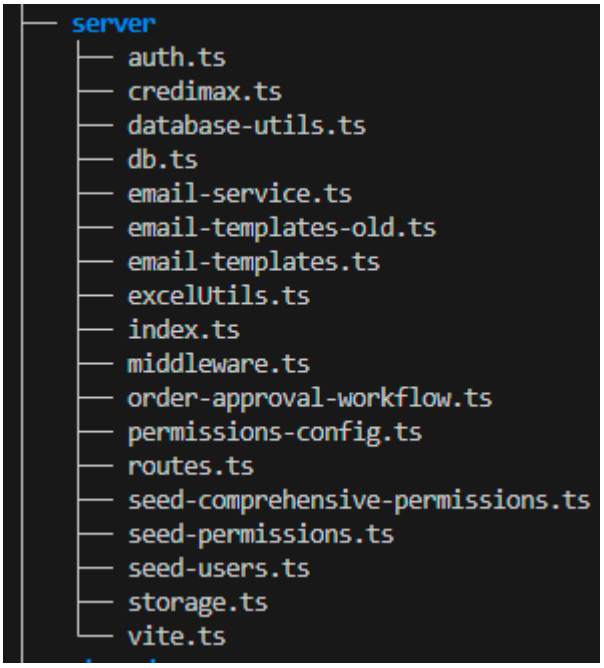


Figure 16 Server (backend) files

The RC-BAYG system is organized into a structured, modular project layout that separates the frontend, backend, shared resources, deployment assets, and infrastructure configurations. This ensures maintainability, clarity, and scalability across all system components.

Client (Frontend)

The client/ directory contains the React-based frontend, including:

- index.html – main HTML entry point
- src/ – React components, pages, hooks, and UI logic

This layer handles all user interactions, product browsing, order creation, and administrative dashboards.

Server (Backend)

The server/ directory contains the Node.js + Express backend, responsible for API routes, authentication, business logic, payments, email notifications, and database operations. Key files include:

- index.ts – backend entry point
- routes.ts – all API route definitions
- auth.ts – login and session logic
- credimax.ts – payment gateway integration
- order-approval-workflow.ts – automated admin workflow logic
- email-service.ts and email-templates.ts – email handling
- db.ts and database-utils.ts – PostgreSQL and Drizzle ORM logic
- seed-*.ts – scripts for creating initial users/roles/permissions

This is the operational core of the RC-BAYG platform.

Shared Schema

The shared/ folder contains:

- schema.ts – all database models and Zod validation schemas

These are shared across backend and potentially frontend to ensure consistency.

Deployment, Build, and Dev Tools

Important system-level files include:

- vite.config.ts – frontend build configuration
- ecosystem.config.js – PM2 deployment process file
- nginx.conf – reverse proxy configuration for production
- drizzle.config.ts – database migration and ORM config
- package.json – list of dependencies and scripts

The dist/ folder contains compiled backend output after running the build process.

Assets and Uploads

- uploads/ – images used on the website
- attached_assets/ – logs, keys, and development artifacts
  These support the frontend UI and development workflow.

Documentation and Miscellaneous

Contains supporting materials such as:

- DEPLOYMENT-README.md – deployment instructions
- chmod-commands-guide.md – server permission guide
- local debug cookies/tests

Overall Summary

The project follows a clean full-stack architecture consisting of:

- Frontend (React + Vite) = user interface
- Backend (Node.js + Express + TypeScript) = business logic + APIs
- Database (PostgreSQL) = data storage
- Shared Schemas = unified validation and models
- Deployment Layer (PM2 + NGINX + AWS) = production hosting

This structure separates responsibilities clearly, supports scalability, and provides a stable foundation for an e-commerce-style event management platform like RC-BAYG.

5.1.1 Frontend (React.js)

The website's user interface was implemented using React.js, following a component-based structure to ensure reusability and modularity. Pages such as the dashboard, product listing, categories, product details, and checkout were developed as independent components.
Key implementation steps included:

- Implementing routing using React Router
- Designing reusable components for buttons, cards, and forms
- Implementing state management using Context API
- Integrating frontend forms with backend API endpoints
- Adding client-side validation checks for order submission

More in depth Screenshots and frontend code snippets are included in Appendix III.

Example: App.tsx (Frontend Application Root Component)

This component serves as the central entry point for the RC-BAYG frontend system. It defines application-wide providers, routing logic, dynamic UI behavior, and global context layers such as authentication, theming, data fetching, and notifications. This file is essential for ensuring that every page of the system loads with the required configuration, security, and shared interface components.

```
1   import { Switch, Route } from "wouter";
2   import { queryClient } from "./lib/queryClient";
3   import { QueryClientProvider, useQuery } from "@tanstack/react-query";
4   import { Toaster } from "@/components/ui/toaster";
5   import { TooltipProvider } from "@/components/ui/tooltip";
6   import { AuthProvider } from "./hooks/use-auth";
```

Figure 17 app.tsx import

Explanation

- Wouter: Lightweight router used to handle all page navigation. Chosen for its simplicity and performance compared to larger frameworks (e.g., React Router).
- React Query (TanStack Query): Manages server state, caching, and background data synchronization across the website. This drastically improves performance for frequently accessed data (e.g., categories, products, orders).
- AuthProvider: Global authentication context that validates user sessions and protects routes.
- TooltipProvider / Toaster: UI helpers for tooltips and notifications across all pages.

Justification

Using these libraries reduces boilerplate code, ensures efficient state handling, and improves user experience with minimal overhead.

Page Imports

```
import NotFound from "@/pages/not-found";
import HomePage from "@/pages/home-page";
import AuthPage from "@/pages/auth-page";
import ProductsPage from "@/pages/products-page";
import CategoriesPage from "@/pages/categories-page";
import CategoryDetailPage from "@/pages/category-detail-page";
import ProductDetailPage from "@/pages/product-detail-page";
import CartPage from "@/pages/cart-page";
import CheckoutPage from "@/pages/checkout-page";

import OrdersPage from "@/pages/orders-page";
import UserDashboard from "@/pages/user-dashboard";
import AdminDashboard from "@/pages/admin-dashboard";
import ProfilePage from "@/pages/profile-page";
import { ProtectedRoute } from "./lib/protected-route";
import Footer from "@/components/footer";
import { NavigationHeader } from "@/components/navigation-header";
import { useTheme } from "./hooks/use-theme";
import { useLocation } from "wouter";
import { useEffect } from "react";
import type { SiteSettings } from "@shared/schema";
```

Figure 18 app.tsx page imports

Explanation

Each import represents a major page of the platform. Protected Route ensures that only authenticated users can access certain pages (e.g., dashboard, orders).

Justification

This demonstrates implementation of security, access control, and role-based routing as part of the system's operational requirements.

Dynamic Title Updater

```
function DynamicTitleUpdater() {
  const { data: settings } = useQuery<SiteSettings>({
    queryKey: ["/api/settings"],
  });

  useEffect(() => {
    if (settings?.browserTabTitle) {
      document.title = settings.browserTabTitle;
    } else if (settings?.siteName) {
      document.title = settings.siteName;
    }
  }, [settings?.browserTabTitle, settings?.siteName]);

  return null;
}
```

Figure 19 Dynamic title updater code snippet

Explanation

- Fetches site settings from /api/settings.
- Automatically updates the browser tab title based on admin-configured values.

- Ensures branding consistency across the platform.

Justification

This adds professionalism and improves usability. The use of React Query allows real-time updates without manually refreshing pages.

Router Configuration

```
function Router() {
  return (
    <Switch>
      <ProtectedRoute path="/" component={HomePage} />
      <ProtectedRoute path="/categories" component={CategoriesPage} />
      <ProtectedRoute path="/category/:categoryId" component={CategoryDetailPage} />
      <ProtectedRoute path="/products" component={ProductsPage} />
      <ProtectedRoute path="/products/:id" component={ProductDetailPage} />
      <ProtectedRoute path="/cart" component={CartPage} />

      <ProtectedRoute path="/orders" component={OrdersPage} />
      <ProtectedRoute path="/checkout/:orderId" component={CheckoutPage} />
      <ProtectedRoute path="/checkout" component={CheckoutPage} />
      <ProtectedRoute path="/dashboard" component={UserDashboard} />
      <ProtectedRoute path="/profile" component={ProfilePage} />
      <ProtectedRoute path="/admin" component={AdminDashboard} />
      <Route path="/auth" component={AuthPage} />
      <Route component={NotFound} />
    </Switch>
  );
}
```

Figure 20 router function code snippet

Explanation

1. Defines all available routes in the system.
2. Sensitive pages are protected using ProtectedRoute, which:
    a. checks session cookies
    b. verifies user role
    c. redirects unauthenticated users to /auth

Critical Evaluation

1. Strength: Lightweight and fast routing.
2. Weakness: Wouter does not support nested routes natively.
3. Alternative: React Router, although heavier, supports more advanced routing features.

5.1.2 Backend (Node.js & Express.js)

The backend was built using Express.js to handle API routing, business logic, and communication with PostgreSQL. Key implementation tasks included:

1. Creating secure REST API endpoints for orders, users, and categories
2. Implementing input validation middleware
3. Handling payment API integration with CrediMax (POST transactional requests, response validation)
4. Logging system actions for auditing
5. Implementing admin-exclusive endpoints for approving or rejecting orders

The backend runs on AWS EC2 with NGINX as a reverse proxy.

Example: Routing Architecture

The project uses a clear, modular routing structure where Express.js handles backend API routes and Wouter manages frontend page navigation. Backend routes are grouped by feature; such as products, orders, users, and cart, making them easy to maintain and extend. Middleware handles authentication and validation before each request reaches the main logic, ensuring secure and consistent behavior. On the frontend, Wouter maps URLs to pages like product details, checkout, and the dashboard, enabling fast, smooth navigation without reloading the page. This separation keeps the API organized and the user experience responsive.

Module	Description
/api/products	Create, update, delete, and retrieve products
/api/categories	Manage product categories
/api/orders	Order submission, approval workflow, and payment status updates
/api/cart	Shopping cart operations with rental-day price calculation
/api/admin/*	Super Admin and Manager operations (permissions, roles, reports)
/api/credimax/*	Integration with CrediMax payment gateway
/api/settings	SMTP, site branding, configuration
/api/slider-images	Homepage slider controls

For the sake of wordcount and time I will only explain the Order Creation Endpoint, check Appendix III for more in depth code analysis

Order Creation Endpoint (POST /api/orders)

This endpoint is responsible for generating a new order after a user completes the checkout process. It validates the cart contents, performs tax calculations, inserts the order into the database, creates each corresponding order item, and triggers email notifications. This ensures a complete, reliable, and auditable e-commerce workflow.

Authentication and request body

```
app.post("/api/orders", async (req, res) => {
  if (!req.isAuthenticated()) {
    return res.sendStatus(401);
  }

  try {
    const { customerInfo, paymentMethod, paymentIntentId } = req.body;
```

Figure 21 Order Code snippet

This first part defines the /api/orders POST endpoint. It immediately checks if the user is authenticated using req.isAuthenticated(). If the user is not logged in, the server returns HTTP 401 (unauthorized) and stops. If authentication passes, it safely enters a try block and extracts customerInfo, paymentMethod, and paymentIntentId from the request body, which are used later during order creation and payment handling.

Loading and validating the cart

```
// Get cart items
const cartItems = await storage.getCartItems(req.user!.id);

if (cartItems.length === 0) {
  return res.status(400).json({ message: "Cart is empty" });
}
```

Figure 22 get cart items

Here, the backend fetches all cart items for the currently logged-in user using their user.id. If the cart is empty, the endpoint immediately returns a 400 Bad Request response with a message explaining that there is nothing to create an order from. This prevents creating empty or invalid orders and enforces that an order must always be based on existing cart items.

Calculating subtotal, VAT, and total

```
// Calculate totals with 10% VAT - use totalPrice from cart items if available, otherwise
fallback to product price
const subtotal = cartItems.reduce(
  (sum, item) => {
    // Use calculated totalPrice from cart if available (for rentals), otherwise use
    contextual price
    const itemTotal = item.totalPrice ? parseFloat(item.totalPrice) : (() => {
      const displayPrice = item.product.productType === "rental" && item.product.rentalPrice
        ? item.product.rentalPrice
        : item.product.price;
      return parseFloat(displayPrice) * item.quantity;
    })();
    return sum + itemTotal;
  },
  0
);
const vatPercentage = 10.00;
const tax = subtotal * (vatPercentage / 100); // 10% VAT
const total = subtotal + tax;
```

Figure 23 calculate totals

This block computes all price totals for the order. It starts by calculating the subtotal using reduce over all cart items. If a cart item already has a totalPrice (typically pre-calculated for rentals), that value is used; otherwise, the code decides which price to use (rental price or normal price) based on the product type and multiplies it by the quantity. After getting the subtotal, it applies a fixed 10% VAT rate by multiplying the subtotal by 0.10, and then adds that tax to the subtotal to produce the final total for the order.

Creating the order record with approval workflow

```
// Create order with approval workflow - always starts as pending approval
const order = await storage.createOrder({
  userId: req.user!.id,
  subtotal: subtotal.toFixed(2),
  tax: tax.toFixed(2),

  total: total.toFixed(2),
  vatPercentage: vatPercentage.toFixed(2),
  paymentMethod,
  paymentIntentId,
  status: "pending", // Order status starts as pending
  adminApprovalStatus: "pending", // Always requires admin approval
});
```

Figure 24 start order approval process

In this part, the application creates a new order in the database using storage.createOrder. It stores the

user ID, the financial breakdown (subtotal, tax, total, VAT percentage) formatted to two decimal places, and the payment information collected from the request. The status field is set to "pending", and adminApprovalStatus is also "pending", which means that every order must be reviewed and approved by an admin before the user proceeds with payment or delivery. This implements the business rule that all orders go through an approval workflow.

Creating order items and handling rental days

```
// Create order items and update stock
for (const cartItem of cartItems) {
  // Calculate rental days if this is a rental item
  let rentalDays = null;
  if (cartItem.rentalStartDate && cartItem.rentalEndDate) {
    const startDate = new Date(cartItem.rentalStartDate);
    const endDate = new Date(cartItem.rentalEndDate);
    rentalDays = Math.ceil((endDate.getTime() - startDate.getTime()) / (1000 * 60 * 60 * 24)) + 1;
  }

  await storage.createOrderItem({
    orderId: order.id,
    productId: cartItem.productId,
    quantity: cartItem.quantity,
    price: cartItem.unitPrice || (() => {
      const displayPrice = cartItem.product.productType === "rental" && cartItem.product.rentalPrice
        ? cartItem.product.rentalPrice
        : cartItem.product.price;
      return displayPrice;
    })(),
    totalPrice: cartItem.totalPrice || (() => {
      const displayPrice = cartItem.product.productType === "rental" && cartItem.product.rentalPrice
        ? cartItem.product.rentalPrice
        : cartItem.product.price;
      return (parseFloat(displayPrice) * cartItem.quantity).toFixed(2);
    })(),
    rentalStartDate: cartItem.rentalStartDate,
    rentalEndDate: cartItem.rentalEndDate,
    rentalDays: rentalDays,
  });
}
```

Figure 25 create order item

This loop iterates through every cart item and converts it into a corresponding order_items entry linked to the newly created order. If the cart item is a rental and contains both a start and end date, the code calculates rentalDays by measuring the time difference between the two dates and converting it into days, making sure to include both the start and end dates. For each item, it stores the quantity, the unit price (either from unitPrice or by deriving it from the product), and the total price (either from totalPrice or recalculated based on quantity). Rental-related fields such as rentalStartDate, rentalEndDate, and rentalDays are also stored, which makes it possible later to show detailed rental information in invoices and reports.

Clearing the cart after order creation

```
// Clear cart
await storage.clearCart(req.user!.id);
```

Figure 26 clear cart

Once the order and its items have been successfully created, this line clears all cart items for the current user. This prevents users from accidentally placing the same order twice using the same cart contents and keeps the cart logically in sync: after an order is submitted, the cart is emptied.

Sending admin notification asynchronously

```
// Send email notifications asynchronously (don't block the response)
setImmediate(async () => {
  try {
    const { sendAdminOrderNotification } = await import("./order-approval-workflow");
    const user = req.user!;
    const customerName = `${user.firstName || ''} ${user.lastName || ''}`.trim() || user.username;

    await sendAdminOrderNotification({
      orderNumber: order.id.slice(-8).toUpperCase(),
      customerName: customerName,
      customerEmail: user.email,
      total: parseFloat(order.total),
      shippingAddress: customerInfo?.address || "", // Add missing required field
      items: cartItems.map(item => ({
        productName: item.product.name,
        quantity: item.quantity,
        price: (() => {
          const displayPrice = item.product.productType === "rental" && item.product.rentalPrice
            ? item.product.rentalPrice
            : item.product.price;
          return (parseFloat(displayPrice) * item.quantity).toFixed(2);
        })()
      }))
    });
  } catch (emailError) {
    console.error('Failed to send admin notification:', emailError);
  }
```

Figure 27 asynchronous admin order notification

Here, the backend schedules an asynchronous task using setImmediate so that the HTTP response is not delayed by email sending. Inside this async block, it dynamically imports the sendAdminOrderNotification helper from the order approval workflow module. It then builds a payload containing the formatted order number (using the last 8 characters of the ID), the customer's name and email, the total value of the order, the shipping address, and a simplified list of ordered items. An email is then sent to the admin team to notify them that a new order is waiting for approval. Any errors during email sending are logged but do not break the main request flow.

Final HTTP response to the client

```
    res.status(201).json({
      id: order.id,
      total: order.total,
      message: "Order submitted for admin approval. Admin has been notified and you will receive an email once approved."
    });
  } catch (error: any) {
    res.status(400).json({ message: error.message });
  }
});
```

Figure 28 return response to client

Finally, if everything goes well, the API returns a 201 Created status along with the order.id, the total amount, and a clear message explaining that the order has been submitted and is now awaiting admin approval. If any error occurs anywhere in the try block (for example, database issues or validation problems), the catch block handles it and responds with a 400 status and the error message. This gives the frontend a consistent way to inform the user if something went wrong.

5.1.3 Database Implementation (PostgreSQL)

The PostgreSQL schema was implemented with relational tables representing:

1. Users
2. Orders
3. Products
4. Categories
5. Order items
6. Audit logs

Key database work included:

1. Defining primary and foreign keys
2. Indexing frequently queried columns
3. Creating SQL functions for retrieving order summaries
4. Securing database access through private AWS subnets

The database diagram is included in **Appendix II**.

Database Setup Explanation (database-setup.sql)

```sql
-- Create database and user (PostgreSQL syntax)
CREATE DATABASE bayg_production;
CREATE USER bayg_user WITH ENCRYPTED PASSWORD 'BaygSecure2024!';

-- Grant privileges
GRANT ALL PRIVILEGES ON DATABASE bayg_production TO bayg_user;
ALTER DATABASE bayg_production OWNER TO bayg_user;

-- Connect to the database
\c bayg_production;
```

Figure 29 Database-setup.sql query snippet

The queries inside *database-setup.sql* perform the essential steps required to prepare PostgreSQL for the BAYG platform:

1. CREATE DATABASE bayg_production;
   Creates the main database used by the application.
2. CREATE USER bayg_user WITH ENCRYPTED PASSWORD 'BaygSecure2024!';
   Creates a dedicated PostgreSQL user for secure authentication.
3. GRANT ALL PRIVILEGES ON DATABASE bayg_production TO bayg_user;
   Gives the new user full access to the database.
4. ALTER DATABASE bayg_production OWNER TO bayg_user;
   Sets the user as the owner of the database.
5. \c bayg_production;
   Connects to the newly created database.
6. Schema and permission commands (GRANT ALL..., ALTER DEFAULT PRIVILEGES...)
   Ensure the user can manage all current and future tables and sequences.
7. CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
   Enables UUID generation for primary keys.

These queries fully prepare the database so the backend can read, write, and manage all required records.

How to Run the Database Setup

Run the setup script on the server using the following command:

psql -U postgres -f database-setup.sql

Once executed, the database is ready for migrations, seeding, and backend connection.

5.2 PowerShell Automation Scripts Implementation

Three automation scripts were implemented to standardize laptop provisioning:

5.2.1 Setup.ps1

1. Creates admin and user accounts
2. Configures Windows settings
3. Renames the device based on technician input
4. Confirms operations and prompts for reboot

Windows Device Setup Automation (Setup.ps1)

Auto-Elevation to Administrator

```
# --- Auto-elevate if not running as Administrator ---
if (-not ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.
Principal.WindowsBuiltInRole] "Administrator")) {
    Write-Host "[INFO] Restarting script as Administrator..."
    Start-Process powershell.exe -ArgumentList "-ExecutionPolicy Bypass -File `"$PSCommandPath`"" -Verb RunAs
    exit
}

Write-Host "[SUCCESS] Running as Administrator."
```

Figure 30 auto-elevation

This section ensures the script always runs with full administrative privileges. If the user launches the script without admin rights, it automatically restarts itself with elevated permissions using Start-Process -Verb RunAs. Once restarted, execution continues with full access needed for system-level changes.

Summary Tracking Array

```
# Summary
$summary = @()
```

Figure 31 create summary array

A simple array is initialized to store a summary of each completed step. At the end of the script, this summary is printed to give the technician a quick overview of what was successfully applied.

Setting Time-zone to UTC+3

```
#  Step 2: Set timezone to UTC+3
Write-Host "`n[STEP 2] Setting timezone to UTC+3 (Jordan Standard Time)..."
try {
    Set-TimeZone -Id "Jordan Standard Time" -ErrorAction Stop
    w32tm /resync | Out-Null
    Write-Host "[SUCCESS] Timezone set to UTC+3 and clock synced."
    $summary += "Timezone set to UTC+3 and synced."
} catch {
    Write-Host "[FAILED] Could not set timezone: $_"
    $summary += "Failed to set timezone."
}
```

Figure 32 set timezone

This block configures the device's system timezone to UTC+3 by selecting "Jordan Standard Time." The script then forces a clock resynchronization with Windows time servers. Any errors are caught and logged into the summary. This ensures all devices remain synchronized during BAYG operations.

Disabling Sleep & Display Timeout

```
#  Step 3: Disable sleep & display timeout on AC and battery power
Write-Host "`n[STEP 3] Disabling sleep and display timeout on AC and battery power..."
try {
    powercfg -change standby-timeout-ac 0
    powercfg -change monitor-timeout-ac 0
    powercfg -change standby-timeout-dc 0
    powercfg -change monitor-timeout-dc 0
    Write-Host "[SUCCESS] Sleep and display timeout set to Never on AC and battery."
    $summary += "Power plan set to Never sleep or turn off display (AC and battery)."
} catch {
    Write-Host "[FAILED] Failed to set power settings: $_"
    $summary += "Failed to set power plan."
}
```

Figure 33 change power settings

This section configures power settings for both AC and battery mode. The script removes all automatic sleep or display timeout triggers to ensure that the workstation stays awake continuously; critical during event operations where laptops must remain available without going to sleep.

Creating Local Admin Account & Renaming Device

```
#  Step 4: Create new local admin user and rename computer
Write-Host "`n[STEP 4] Creating new local admin account and renaming computer..."

$code = Read-Host "Enter the custom code for the new device/user (e.g., SALES01)"
$code = $code.Trim().ToUpper() -replace '\s+', '-'
```

The script prompts the technician to enter a custom device code (e.g., *MEDIA01*, *VENUE05*). It cleans the input by trimming spaces, converting to uppercase, and replacing spaces with hyphens. This code determines both the username and the computer name.

```powershell
if ($code -match "^.+$") {
    $username = "BAYG-$code"

    try {
        # Check if user already exists
        net user $username | Out-Null
        if ($LASTEXITCODE -eq 0) {
            Write-Host "[INFO] User '$username' already exists."
            $summary += "User '$username' already exists."
        } else {
            # Create account with no password and no password requirement
            net user $username "" /add /active:yes /passwordreq:no | Out-Null

            # Add to Administrators group
            net localgroup Administrators $username /add | Out-Null

            Write-Host "[SUCCESS] Admin account '$username' created with no password."
            $summary += "Admin account '$username' created (no password)."
        }

        # Rename computer
        Rename-Computer -NewName $username -Force -ErrorAction Stop
        Write-Host "[SUCCESS] Computer renamed to '$username'. A reboot is required for changes to take effect."
        $summary += "Computer renamed to '$username' (pending reboot)."
    } catch {
        Write-Host "[FAILED] Error creating user or renaming computer: $_"
        $summary += "Failed to create user or rename computer."
    }
} else {
    Write-Host "[FAILED] Invalid input. Please enter a valid code."
    $summary += "Invalid code entered."
```

This part creates a new **password-less local administrator account** using the format BAYG-CODE. Before creating it, the script checks whether the account already exists, ensuring the process is idempotent. It then renames the computer to match the same identifier, helping asset tracking and deployment. Errors are handled gracefully and recorded in the summary.

Final Summary Output

```powershell
#  Final Summary
Write-Host "`n==================== Setup Summary ===================="
foreach ($item in $summary) {
    Write-Host "- $item"
}
Write-Host "========================================================="
```

At the end, the script prints a clean summary of all completed steps, allowing technicians to quickly verify whether the configuration was successful.

Reboot Prompt

```powershell
#  Offer reboot option
Write-Host "`nDo you want to reboot now? (Y/N)"
$response = Read-Host
if ($response -match '^[Yy]$') {
    Restart-Computer -Force
} else {
    Write-Host "Please reboot later to apply changes."
}
```

The script gives the technician a choice to reboot immediately. Since computer renaming requires a restart, the script advises users to reboot to finalize changes.

5.2.2 Install-WindowsUpdates.ps1

1. Installs the PSWindowsUpdate module
2. Scans and installs all pending Windows updates
3. Generates a log file for verification

5.2.3 Disable-WindowsUpdates.ps1

1. Stops Windows Update and BITS services
2. Applies registry rules blocking updates
3. Verifies update services are disabled

In depth code analysis are included in **Appendix III**.

6. Testing

The testing phase ensures that all systems, scripts, and procedures implemented throughout the project function correctly, reliably, and securely. Because the project spans multiple domains; including web development, automation scripts, and IT operations, different testing approaches were applied depending on the component being evaluated. This section describes the test plans, participants involved, test cases, acceptance criteria, and results.

6.1 Test Plan

Testing was conducted across three main areas of the project:

6.1.1 RC BAYG Website

- Functional testing
- Input validation testing
- Payment workflow testing (CrediMax sandbox)
- Admin dashboard testing
- Database integrity testing

6.1.2 PowerShell Automation Scripts

- Execution and privilege testing
- Environmental validation
- Script logic validation
- Correct system configuration
- Error handling

6.1.3 Laptop Setup SOP Implementation

- End-to-end provisioning tests
- Driver installation testing
- Activation testing
- Account creation verification
- Re-issuance workflow testing

All tests followed a controlled, iterative cycle:
Prepare → Execute → Validate → Log → Fix (if needed)

6.2 Participants

The testing process involved three key individuals who represented the stakeholders interacting with the system: development, helpdesk operations, and IT support. Their combined expertise ensured that the functionalities were evaluated from multiple operational perspectives, including technical accuracy, usability, administrative workflow, and end-to-end performance. Each participant contributed based on their real-world responsibilities during the BAYG preparation and operations period, allowing the testing phase to accurately reflect the conditions and scenarios encountered during the event.

Name	Role	Background
Muhammad Owais	Lead Developer & IT Support	7-8 years experience in development, Owais has a major background in developing IT systems as well as an understanding in creating a user friendly interface to use.
Younis Hasan	Helpdesk Administrator Manager	Currently working at Zain in IT, Younis Hasan Has an extensive background in creating and managing IT systems.
Mohamed Yanis Moussai	IT Support & System Developer (Author)	Currently at university, in my last semester of completing my Programming Major. With an extensive background in creating coding projects, he has built a strong base in the LAMP stack.

6.2.1 Functionality Test Cases and Results for RC BAYG (With Participant Validation)

Test Case	Description	Expected Result	Actual Result	Owais	Younis	Yanis
TC1 – Order Submission	User submits order with all required fields	Order stored in DB with unique ID	Passed	Pass	Pass	Pass
TC2 – Date Selection	System calculates rental pricing based on selected dates	Correct total price displayed	Passed	Pass	Pass	Pass
TC3 – Admin Approval	Admin approves/declines an order	Status updated + email notification sent	Passed	Pass	Pass	Pass
TC4 – Payment Processing	User completes transaction via CrediMax API	Order marked as Paid	Passed	Pass	Pass	Pass
TC5 – Input Validation	Invalid inputs trigger validation errors	Error messages displayed	Passed	Pass	Pass	Pass

6.2.2 PowerShell Script Test Cases

Test Case	Description	Expected Result	Actual Result	Owais	Younis	Yanis
TC6 – Admin Privilege Detection	Script auto-relaunches with elevated permissions	Elevated execution	Passed	Pass	Pass	Pass
TC7 – Device Renaming	Technician inputs device code	Computer name updated	Passed	Pass	Pass	Pass
TC8 – Update Installation	Install-WindowsUpdates.ps1 installs all updates	System fully updated + log generated	Passed	Pass	Pass	Pass
TC9 – Update Disabling	Disable-WindowsUpdates.ps1 disables update services	Updates permanently disabled	Passed	Pass	Pass	Pass
TC10 – Account Creation	Setup.ps1 creates admin account	Account appears in Users list	Passed	Pass	Pass	Pass

7. Discussion and Conclusion

7.1 System Functionality

The final system successfully delivers a fully functional e-commerce and rental management platform designed specifically for the Bahrain Asian Youth Games (BAYG). Users can authenticate, browse categories, view products, select rental periods through a date-validated calendar, submit orders, and proceed with integrated CrediMax payments. The admin dashboard provides complete control over approvals, user management, order workflows, and content configuration, ensuring smooth operational oversight.

The solution closely reflects the design created earlier in the architecture and methodology sections. The separation of concerns between the frontend, backend, and database allowed the system to maintain strong security, high performance, and operational reliability. The routing architecture, protected endpoints, and permission-based access all reinforce the system's alignment with best practices in secure web application design.

In addition to the website, the custom PowerShell automation scripts enabled the IT team to deploy, configure, and prepare over 800 laptops efficiently. This ensured consistent standards across devices used during the Games and supported the broader digital operations strategy.

7.2 Summary of Achieved Objectives

The project met all major objectives established at the beginning:

Develop a scalable and secure e-commerce/rental portal

The RC-BAYG website includes all required features such as product browsing, cart functionality, checkout, admin approval workflow, and CrediMax payment integration.

Build an admin panel to manage BAYG operations

All system data; products, categories, users, orders, and site settings can be controlled through the admin interface, with permissions applied based on roles.

Implement automated deployment tools for device setup

The PowerShell script automated timezone configuration, sleep settings, admin user creation, hostname assignment, and overall system preparation.

Support operational monitoring during the event

Working as part of the Operations Centre allowed me to connect the technical system with real-world workflows, assisting in approvals, user support, and content updates.

Ensure security and reliability using a structured architecture

A public–private network separation, protected routing, input validation middleware, and secure database structure were all achieved.

Objectives not met

There were no major objectives left unfinished. Some optional features (such as multi-language support and advanced analytics) were excluded due to time constraints but do not affect the core functionality of the system.

7.3 Project Issues and Proposed Solutions

During development, several challenges emerged:

Issue 1: Calendar date validation

Early versions of the calendar allowed users to select invalid ranges.
Cause: Missing date constraints and timezone normalization.
Solution: Custom validation logic was implemented with strict rental date boundaries and UI feedback.

Issue 2: Payment API inconsistencies

CrediMax sometimes returned unexpected responses during testing.
Cause: Sandbox environment instability.
Solution: Additional error handling and response verification were added to the backend.

Issue 3: Deployment difficulties on EC2

Nginx reverse proxy misconfiguration caused routing loops.
Cause: Incorrect proxy path rules for React single-page routing.
Solution: Standard / rewrite rules and fallback routing were applied.

Issue 4: Laptop setup delays

Many batches of laptops were completed beyond their designated dates.
Cause: Reliance on third party vendor to deliver devices.
Proposed Solution: To communicate with more vendors to split the request workload.

7.4 Legal, Ethical, Social & Professional Issues (LESPI)

Developing the RC-BAYG system and setup scripts introduced a wide range of legal, ethical, social, and professional considerations due to its role in handling operational data during an international sports event. These concerns were not theoretical; they actively shaped design decisions, deployment processes, and operational workflows.

Legal

The system collected identifiable information such as usernames, order metadata, email addresses, and payment-related details. Although Bahrain does not enforce GDPR directly, the Olympic Council of Asia and Bahrain Olympic Committee operate under confidentiality expectations similar to GDPR principles, requiring data minimization, secure access, and non-disclosure of personal information.

For example, integrating the CrediMax payment gateway required strict compliance with PCI-aligned practices. To mitigate legal exposure, the backend was deliberately architected so that no stored or logged credit card information ever touched the application layer, ensuring legal compliance through architectural exclusion rather than post-processing.

The PowerShell automation work extended this responsibility. When provisioning nearly 800 laptops, the scripts created accounts, system settings, and logs. Here, legal concerns included:

preventing unauthorized administrative access,

avoiding persistence of personal or credential data, and

ensuring consistency in configuration to prevent accidental policy breaches.

Design choices reflected this:

The Setup.ps1 script generated administrative accounts without personal information, reducing traceability risk.

The Disable-WindowsUpdates.ps1 script enforced update policy via system registry, ensuring compliance with BAYG's IT governance framework.

Future enhancement could include automated secure wiping routines to guarantee that returned laptops undergo full legal-compliance data sanitation.

Ethical

Ethical challenges emerged in three domains:

Access to privileged functionality

Administrators could approve requests, view logs, and modify operational configurations. Ethical risk existed where power imbalance could result in bias or misuse.

This was mitigated through:

audit logging,

controlled role assignment,

and SOP-governed admin privilege distribution.

Transparency and accountability

The logging subsystem, including server logs and PowerShell output, ensured that decisions were attributable, strengthening ethical accountability.

Automation integrity

PowerShell configuration tools had the ability to alter system behaviour silently. Ethical responsibility

required ensuring scripts **did not introduce harmful actions** such as hidden backdoors, credential exposure, or forced vulnerabilities.

For example:

The automation scripts intentionally removed Windows automatic updates not to weaken security, but to prevent unexpected reboots during competition days, which could jeopardize downstream workflows.

Ethically, this required balancing system hardening with operational stability; the scripts installed all updates first, then froze future updates. This reflects ethical decision-making: prioritizing user impact and system reliability over questionable shortcuts.

Social

The RC-BAYG system had direct social consequences, affecting hundreds of people: volunteers, delegations, venue staff, and technical operators. System downtime, provisioning delays, or unclear processes could affect logistics, timing, accreditation, and service output.

Automation work amplified this responsibility:

- The **Install-WindowsUpdates.ps1 script** accelerated device preparation by standardizing patch deployment, allowing larger workforce coverage without delay.
- The **Disable-Updates script** ensured machines did not randomly reboot during live games, protecting social confidence and functional trust in ICT infrastructure.
- The **Setup.ps1 script** reduced technician dependency and prevented inconsistencies that would otherwise create disparities in how laptops were issued to individuals.

These tools collectively contributed to fairness, availability, and reliability, which are inherently **social** responsibilities in event-scale operations.

One risk identified was digital exclusion; some technicians and volunteers lacked scripting familiarity. This was mitigated by introducing interface pauses, printed summary messages, and clear SOPs, improving accessibility and reducing social barriers.

Professional

Professionally, the project required adherence to ICT best practices, organizational protocols, and cross-departmental discipline. PowerShell scripting increased this responsibility because unlike code running on isolated servers, these scripts modified hundreds of physical machines, meaning errors could scale harmfully.

Professional behaviour therefore manifested in:

- Code clarity, documenting script logic for handover
- Safety features such as privilege checks and automatic relaunch elevation
- Human-oriented features (pause-on-exit, summaries) to support varied technician skill levels
- Testing procedures before mass rollout (further detailed in Appendix IV)

A key professional learning outcome was understanding that automation magnifies mistakes, leading to heightened quality control, review, and structured deployment.

Additionally, because scripts enforced security settings and operational constraints, they became part of the governance ecosystem, reinforcing that professional engineers must build not only tools, but responsible tools.

7.5 Future Work

Although the system is fully functional and successfully supported the 2025 Bahrain Asian Youth Games, several enhancements could significantly improve long-term scalability, operational reach, and efficiency.

A major future improvement is **multi-language support**, particularly Arabic and Mandarin, which would broaden accessibility for regional volunteers and international delegations. This aligns with inclusivity standards applied in global sporting events and would reduce dependency on English-only interfaces.

A second opportunity is a **dedicated mobile application** for administrators, warehouse staff, and delivery teams. While the web system functions on mobile devices, a native app could offer offline mode, push notifications, barcode scanning, and faster interaction in field environments such as stadiums, logistics centers, and media areas.

Real-time inventory syncing across multiple warehouses would become increasingly valuable if the system is reused across future OCA or IOC-affiliated events. This would support distributed stock ownership, predictive restocking, and automatic availability resolution when multiple venues share the same resource pool.

Communication efficiency could be improved through **automated SMS or WhatsApp notifications** for approvals and delivery statuses. This would reduce delays in time-sensitive events and increase transparency across stakeholders.

More advanced upgrades include **AI-based demand forecasting**, helping organizers understand which equipment types are in highest demand, when they peak, and how procurement can be optimized. This could reduce waste, improve preparation lead times, and inform decision-making across committees.

Finally, a **conversational AI chatbot** could assist users and administrators in navigation, troubleshooting, and live support during events. Such a system would reduce human support overhead and enable faster onboarding for temporary staff and volunteers.

Collectively, these enhancements would improve usability, extend system lifespan, reduce operational bottlenecks, and support future editions of the Games or similar regional events.

7.6 Synopsis of Experience

Working on this project provided a valuable full-stack development experience, combining backend engineering, frontend UI design, database management, and DevOps-aligned deployment work. The opportunity to operate within the Operations Centre also exposed me to real-world IT processes, cross-department communication flows, escalation paths, and decision-making dynamics that cannot be fully understood through classroom learning alone.

From a technical perspective, I gained strong practical skills in API design, authentication, and payment integration, especially while implementing the CrediMax gateway. Designing and validating financial transaction flows demanded attention to reliability, edge case handling, and security considerations that I previously underestimated. I also developed competency in large-scale automation by scripting configuration and provisioning tasks for over 800 laptops. This work demonstrated how automation is not a convenience but a necessity when timelines compress and manual effort becomes infeasible.

A major learning experience was adopting an entirely new technology stack, the PERN stack, within a live project environment. This forced me to rapidly absorb new patterns, debug emerging issues, and apply concepts directly to production-grade use cases. The process strengthened my adaptability and reinforced the importance of continuous learning in software engineering.

Equally important were the **non-technical skills** gained throughout the project. Coordinating with managers, helpdesk staff, and branding departments revealed organizational and political complexities that technical skill alone cannot solve. Communicating requirements clearly, responding to feedback,

balancing conflicting priorities, and supporting decision-making under time pressure became vital competencies. In high-stakes moments, particularly during peak workload periods; troubleshooting required calmness, prioritization, and an understanding that technical failures often carry operational consequences for multiple stakeholders.

Working under this pressure highlighted areas for future improvement. At times, my instinct was to focus on implementation speed rather than structured planning or documentation. I learned the value of testing boundaries earlier, validating assumptions sooner, and involving stakeholders more frequently to avoid misalignment. Future projects would benefit from more formal risk assessments and communication checkpoints before deployment phases.

Overall, this experience significantly strengthened my professionalism, resilience, and technical confidence. It demonstrated that ICT roles extend far beyond writing code; they encompass reliability, governance, communication, ethical responsibility, and the ability to design systems that support people under operational stress. These lessons will influence how I plan, build, and deliver systems in future enterprise settings and have shaped my career interest toward event-driven IT, DevOps, and large-scale digital operations.

7.7 Conclusion

This project successfully delivered a production-ready resource management system that supported the Bahrain Asian Youth Games' operational workflows. By integrating a custom-built web platform, secure authentication and approval logic, and mass-deployment automation using PowerShell, the work demonstrated how software engineering can directly enable event management efficiency at national scale.

The system met all operational objectives set at project outset, including enabling controlled rentals, administrative approval flows, inventory visibility, and real-time communication between departments. The supporting infrastructure scripts further ensured consistency across nearly 800 physical laptops, which strengthened reliability and aligned with the Games' fast-paced logistics requirements.

Beyond functional delivery, the project generated meaningful insight into event-based ICT deployment. It illustrated how technically simple actions, such as automated patching or disabling unplanned operating system updates can materially affect social and organizational outcomes when executed across large venues. It also highlighted the importance of clear governance, secure design, and accountability mechanisms when handling personal data, financial transactions, and administrative power.

The lessons learned extend beyond BAYG's use case. The experience demonstrated that modern event operations require holistic approaches where applications, automation, network environments, and human workflows coexist as one ecosystem. The project reinforced that secure coding practices, maintainability, documentation, and LESPI considerations are not optional; they are determinants of trustworthiness and operational success.

While there is room for improvement, such as multilingual interfaces, predictive analytics, and mobile extensions, the current system provides a strong baseline for future Games, OCA events, or wider institutional deployments. The outcomes prove that relatively small ICT teams can achieve large impact when leveraging automation, iterative learning, and cross-department collaboration.

Ultimately, the project strengthened both technical capability and professional judgment. It offered proof that ICT systems can meaningfully support real-world operations when designed with foresight, ethical consideration, and resilient execution practices. These insights will contribute significantly to future development work and professional engagements beyond the academic environment.

Final Word count (Excluding Appendix): 12695 words.

Appendix

The appendix section provides supplementary material that supports the development, implementation, and evaluation of the RC-BAYG system. While the main chapters focus on analysis, design, methodology, and system behavior, the appendix contains the detailed technical artifacts that are essential for full transparency and reproducibility. These include the system's folder structure, database setup scripts, table schemas, API endpoint samples, testing documentation, configuration files, and other reference materials.
By placing these items in the appendix, the main body of the report remains clear and concise, while still allowing readers, developers, and evaluators to access the complete technical evidence behind the system whenever needed.

Appendix I: System Documentations

1. Deployment Documentation

This section provides a complete and reproducible procedure for deploying the BAYG e-commerce platform. The deployment process applies to any Ubuntu-based server (e.g., Ubuntu running on AWS EC2), and covers installation, configuration, environment variables, database setup, build procedures, and service operation. The platform consists of a React frontend, an Express/Node.js backend, and a PostgreSQL database.

1.1. Server Requirements

Operating System

- Ubuntu 20.04 / 22.04 / 24.04 LTS (recommended)

Software Dependencies

- Node.js (v18+)
- npm
- PostgreSQL
- Git

1.2. System Preparation

1. Update the package manager

sudo apt update && sudo apt upgrade -y

2. Install Git

sudo apt install git -y

3. Install Node.js (LTS recommended)

curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -

sudo apt install -y nodejs

4. Install Postgres

sudo apt install -y postgresql postgresql-contrib

sudo systemctl start postgresql

sudo systemctl enable postgresql

5.Run the Database Setup File

sudo -u postgres psql -f database-setup.sql

IMPORTANT

Sometimes setup fails to seed the users into the database, requiring the admin to manually insert the superadmin user into the database:

First enter this command into the shell

psql 'postgresql://bayg_user:BaygSecure2024!@localhost:5432/bayg_production'

then enter the sql query:

INSERT INTO users (id, username, email, password, first_name, last_name, is_admin, is_super_admin, role_id) VALUES (gen_random_uuid(), 'superadmin', 'superadmin@bayg.com', 'admin123', 'Super', 'Admin', true, true, '52896663-b6e6-4a05-be41-3c188c5c7141');

```
                  id                  |  username  |        email        |  password  | first_name | last_name | is_admin | is_super_admin |
   role_id              |         created_at
---------------------------------------+------------+---------------------+------------+------------+-----------+----------+----------------+------------
------------------------+----------------------------
 415f2b45-a05d-4288-a2a0-cec5c1b064eb | superadmin | superadmin@bayg.com | admin123   | Super      | Admin     | t        | t              | 52896663-b6
e6-4a05-be41-3c188c5c7141 | 2025-11-25 12:19:12.889253
```

Figure 34 Sql query result

1.3. Cloning and setting up project

1. Clone the project:

sudo git clone https://github.com/owaisqarnikhan/rc-bayg2.git

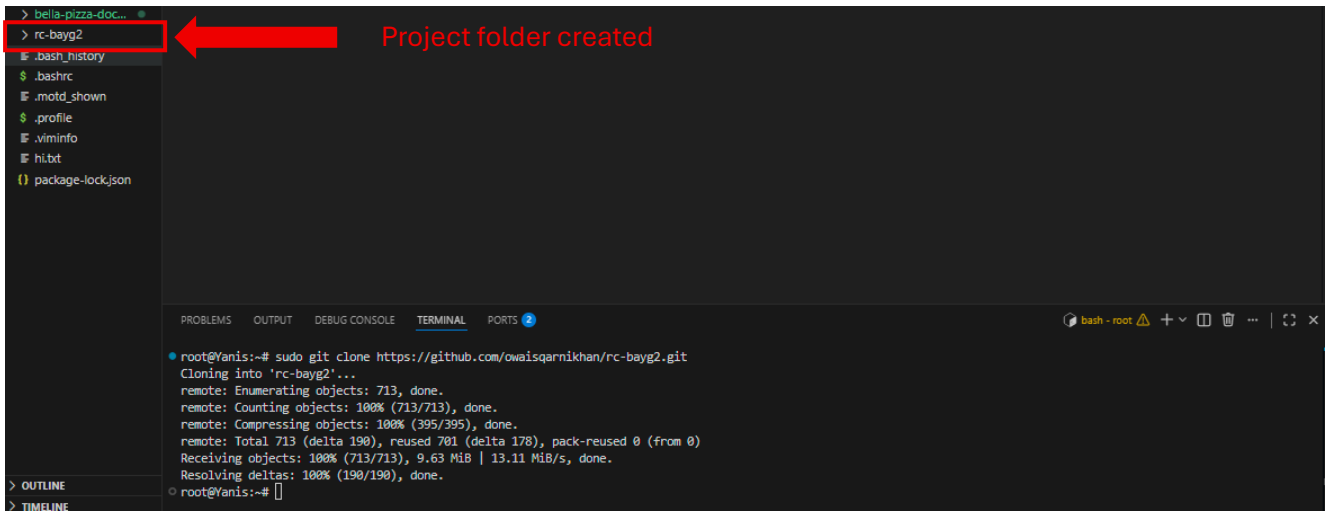


Figure 35 cloning project github repository

2. Move to project root directory

cd rc-bayg2

3.Edit the .env File

Copy code below into your .env file (replace the original):

DATABASE_URL=postgresql://bayg_user:BaygSecure2024!@localhost:5432/bayg_production

SESSION_SECRET=dev-secret

NODE_ENV=development

PORT=5000

Save with:

CTRL+O -> Enter -> CTRL+X

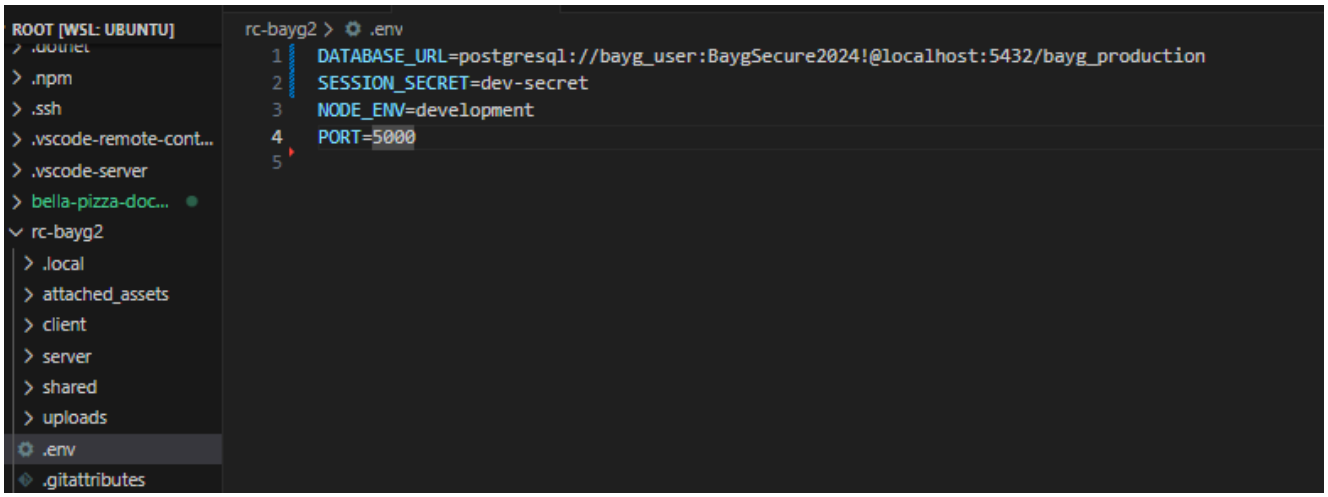


Figure 36 .env file configured

4.Install project dependencies

Move to project root directory and enter this command:

npm install

npm install dotenv

```
root@Yanis:~/rc-bayg2# npm install
npm install dotenv
13 vulnerabilities (3 low, 7 moderate, 3 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
root@Yanis:~/rc-bayg2#
```

Figure 37 npm and dotenv dependencies installed

IMPORTANT

Make sure to add "import 'dotenv/config';" to your index.ts file in the server folder.

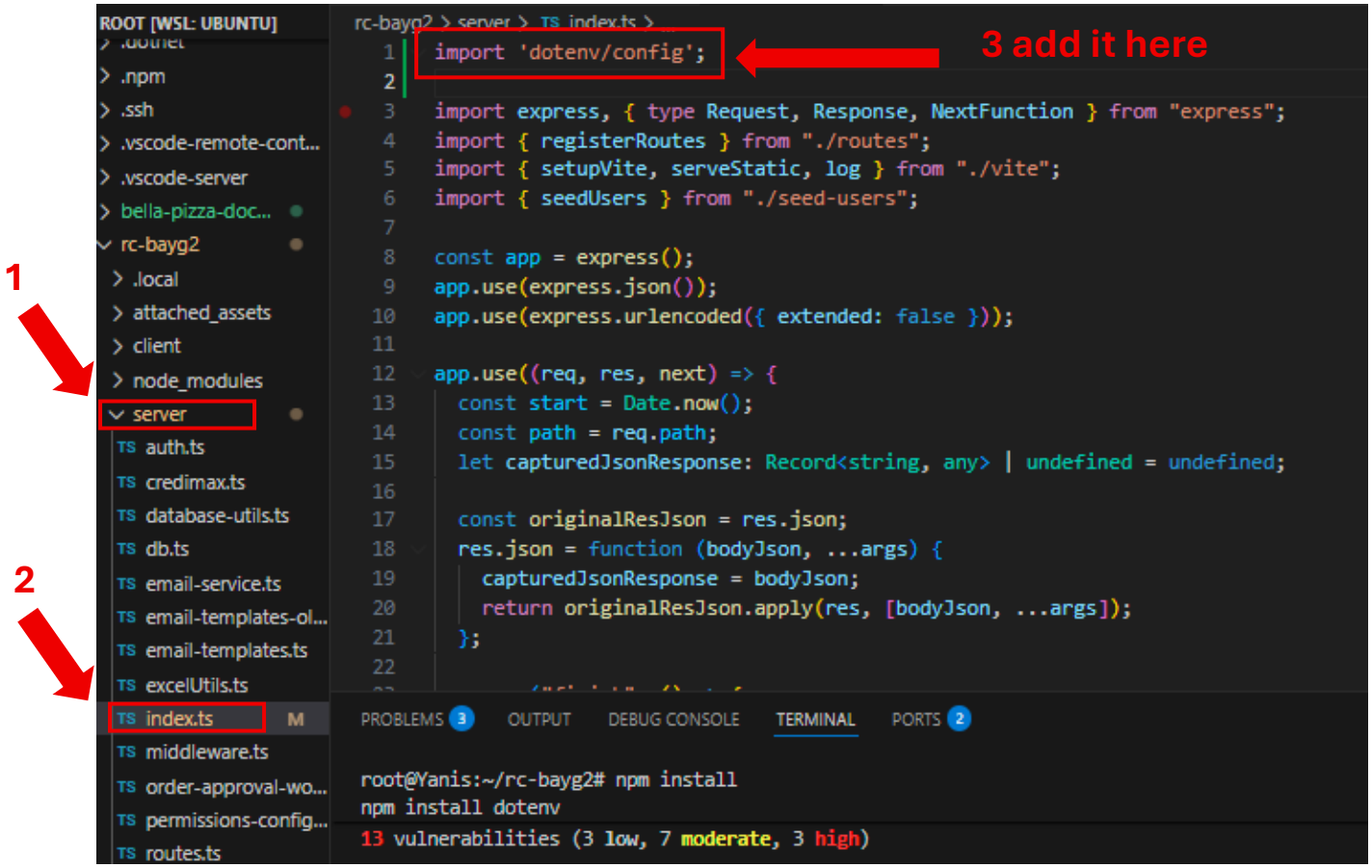


Figure 38 adding dotenv import

Also for the thesis we will be removing password hashing which will require us to edit the **hashPassword** and **comparePasswords** functions in **auth.ts in the server folder.** make sure they look like the picture below:

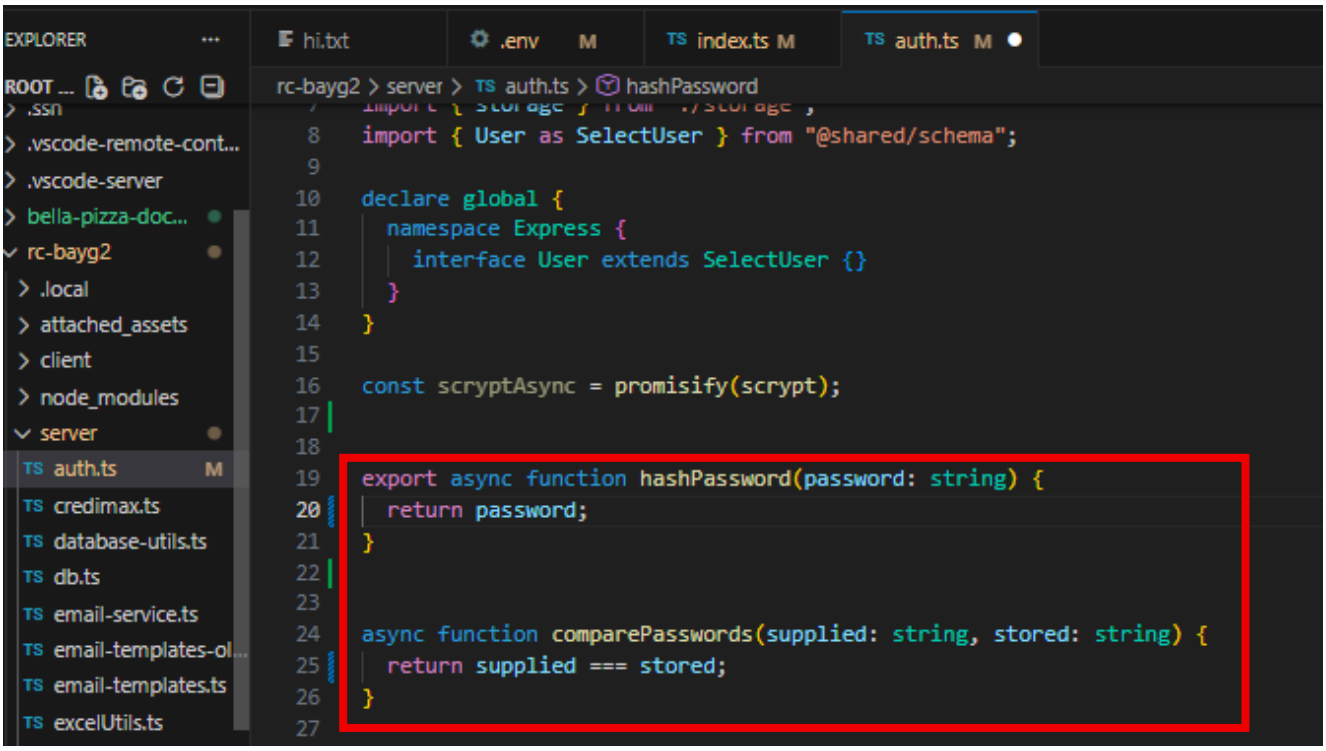


Figure 39 removing Hashing from authentication

5.Push database to code

npm run db:push

6.Run the build

npm run dev

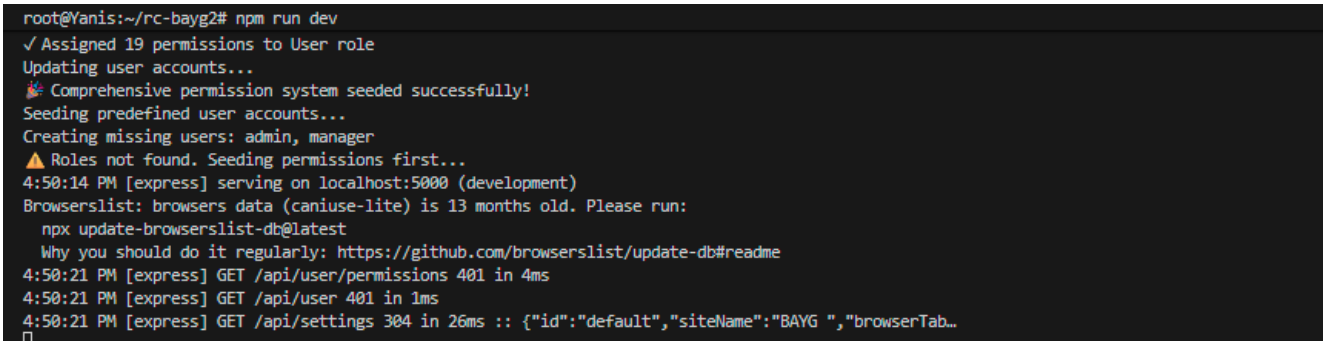


Figure 40 running dev build

7. Open your browser at the link

Open chrome and type in the URL section: localhost:5000 or 127.0.0.1:5000

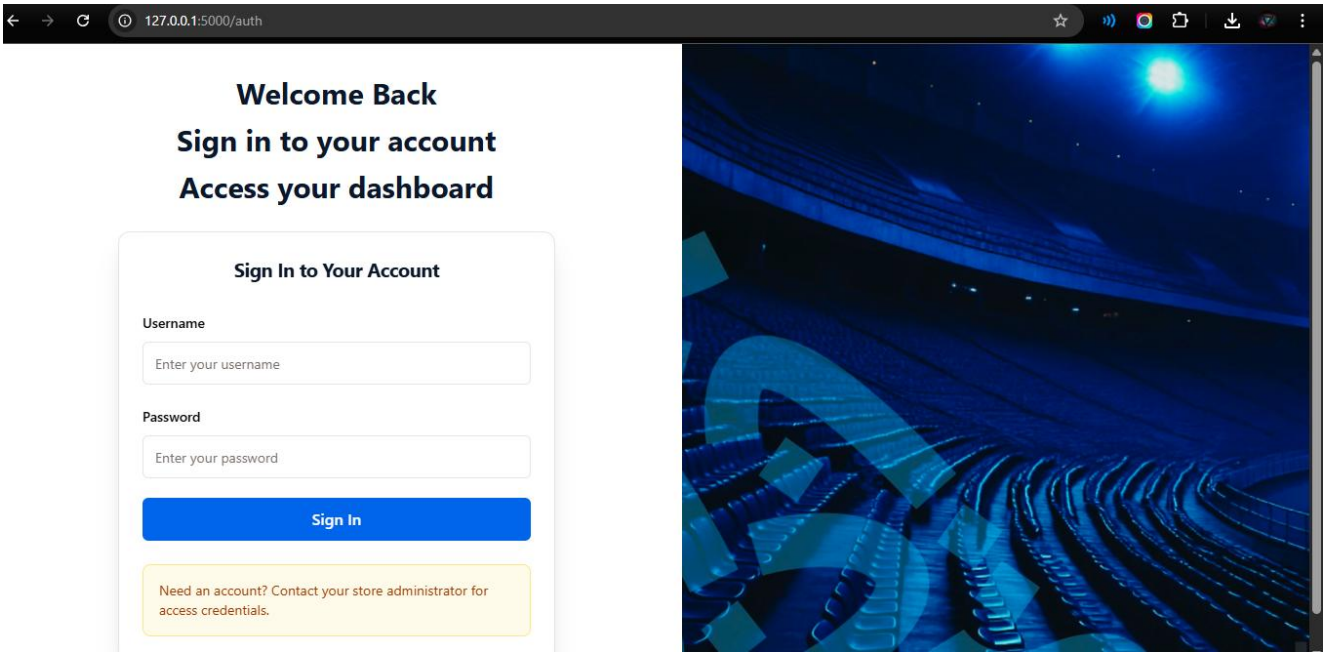


Figure 41 website is running

2. Superadmin Documentation

Login credentials

Username: **superadmin**

Password: **admin123**

2.1 Homepage

This should be the first thing you see once you log in:

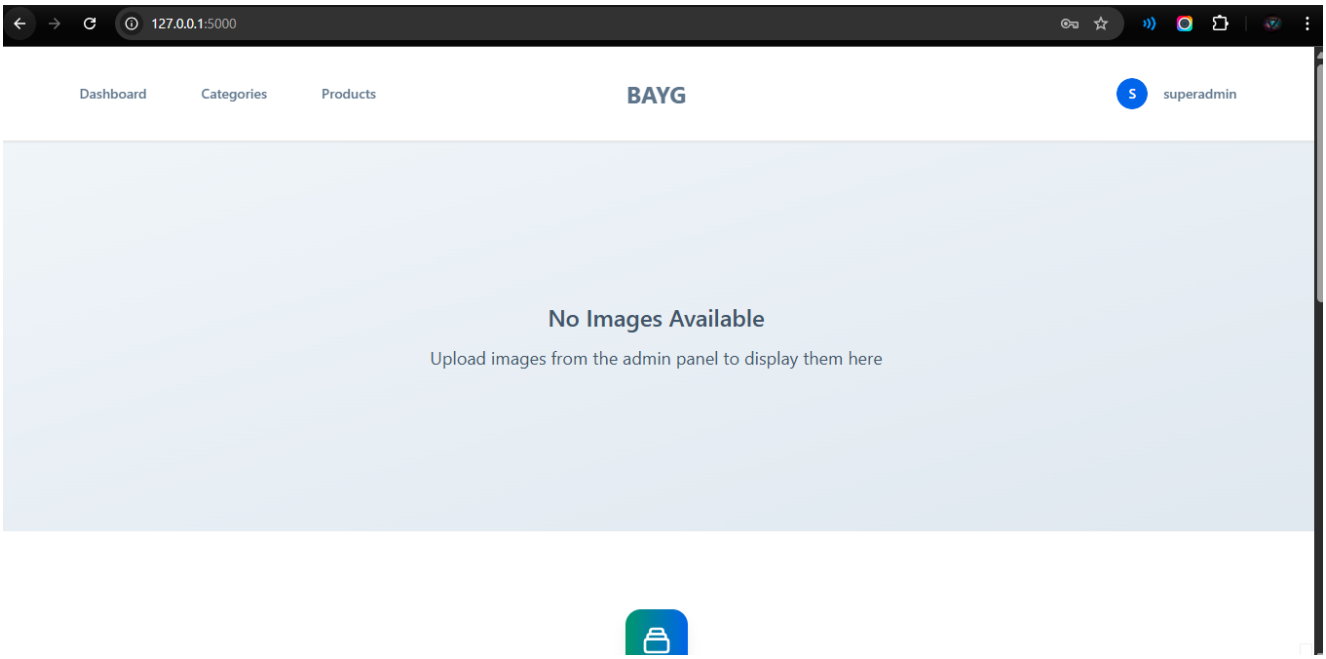


Figure 42 superadmin homepage

2.2 Dashboard

To navigate to the superadmin dashboard, click on the username at top-right most of the screen, and click on 'super admin Panel' as shown below:

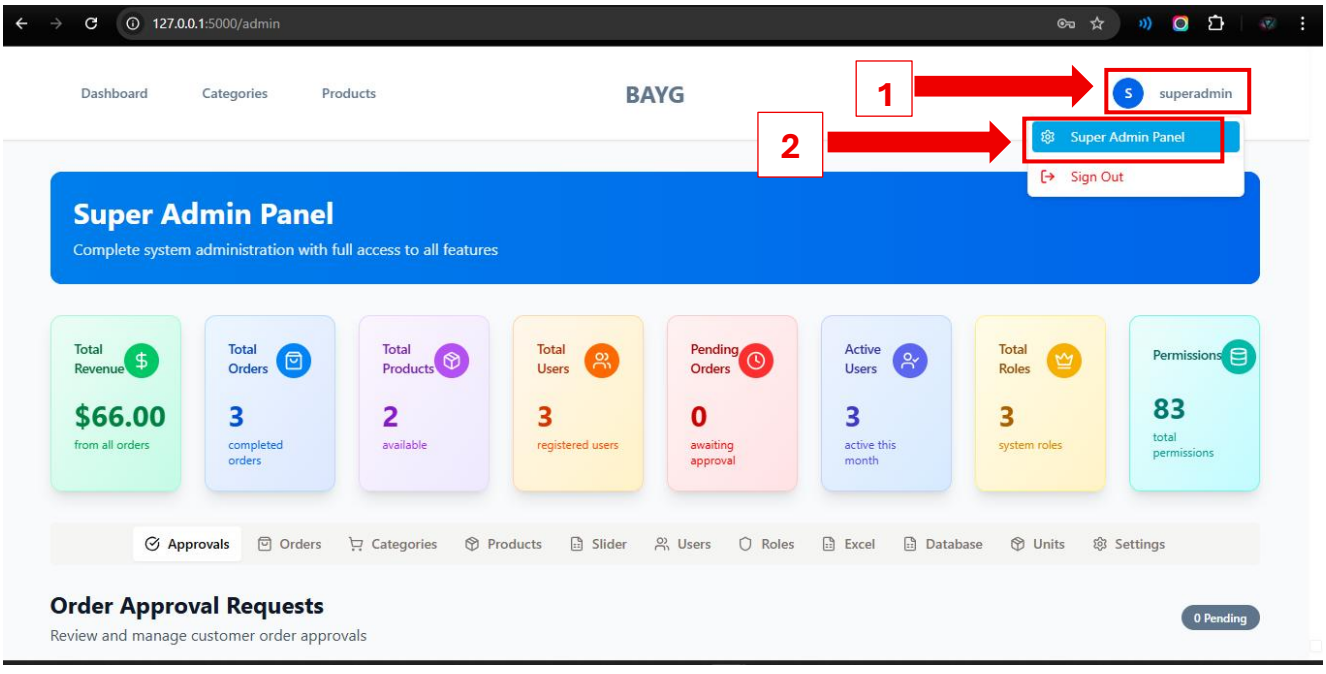


Figure 43 superadmin dashboard

The Admin Dashboard is the central environment where all administrative tasks are carried out. It is designed to be modular, permission-controlled, and data-driven. Every component on the screen exists to give administrators quick access to essential functions while protecting sensitive actions behind permission checks. Below is a detailed explanation of what the user sees and why each part of the dashboard is important.

At the top of the dashboard, the interface presents a header that clearly identifies the user's role. A Super Administrator sees the label *"Super Admin Panel"*, while a Manager sees *"Manager Panel."* This is not just a cosmetic feature. Making the role visible from the first moment helps set expectations and reduces confusion, especially in environments where multiple staff members interact with the system. For example, a Manager immediately understands that certain sections may not be available to them, which prevents unnecessary attempts to access restricted features. At the same time, Super Administrators see a clear signal that they are in a high-control environment where their actions can affect the entire system.

Statistical Overview

Directly below the header, the system presents a collection of real-time statistics that summarize the current state of the platform. These include total revenue, the number of completed orders, total products available, and total registered users. For Super Administrators, the overview expands to include pending orders, the number of active users for the current month, the total number of roles, and the overall count of permissions defined in the system.

These metrics do more than provide general information; they act as an ongoing assessment of the system's health. Because they update dynamically, administrators can quickly notice irregular activity or emerging issues. A sudden drop in revenue or completed orders may suggest checkout problems, while a growing number of pending orders could indicate workflow delays or insufficient staff availability. For Super Administrators, the statistics regarding roles and permissions offer insight into the system's internal security setup. If roles or permissions begin to grow unexpectedly, it may indicate misconfiguration or permission creep, which can weaken overall access control. Furthermore, the use of loading skeletons when data is being fetched ensures that administrators receive constant feedback from the interface rather than interpreting a blank area as a failure, which contributes to a smoother and more trustworthy user experience.

Permission-Controlled Tabs

The main body of the dashboard is organized into a tab-based navigation system. Each tab corresponds to a management module; such as products, categories, orders, users, or settings, and the system shows or hides these tabs based on the permissions assigned to the logged-in user. This structure creates an adaptive interface that remains clean and relevant to each administrator.

From a usability and security perspective, this is one of the most important aspects of the dashboard. By removing access to modules that the user does not need, the interface becomes less cluttered and easier to navigate, reducing cognitive load and preventing mistakes. At the same time, it enforces the "least privilege" principle by ensuring that no user is exposed to tools that exceed their authority. This approach also makes the system more scalable: new permissions or modules can be added without redesigning the entire dashboard layout.

Purpose and Importance of Key Modules

While each module provides different functionality, they all contribute to the overall administration of the system. They include:

1. **Approvals:** Used for reviewing orders that require administrator approval before being processed. This protects the system from fraudulent, incomplete, or unusually large orders.
2. **Orders:** A full view of all customer orders, allowing administrators to track order status, investigate customer issues, and monitor trends in purchasing behaviour.

3. **Categories:** Ensures that the product catalogue remains organized and easy for customers to browse, especially in stores with many product lines.
4. **Products:** Arguably the most vital module for daily operations, as it controls the entire product inventory, pricing, descriptions, stock levels, availability, and more.
5. **Slider:** Manages promotional banners on the homepage. Although simpler than the other modules, it directly influences the user experience and marketing effectiveness.
6. **Users:** Provides tools for managing customer and staff accounts, which helps resolve account issues, update roles, or deactivate problematic accounts.
7. **Roles & Permissions (Super Admin Only):** Controls the system's access structure. This is a critical security component, as it shapes what each user type can or cannot do. Poor configuration here can expose the system to misuse.
8. **Excel Export:** Allows administrators to download structured data for reporting, auditing, or external analysis.
9. **Database Tools:** Provides backup and export functionality, which is essential for data protection, disaster recovery, and migration.
10. **Units of Measure:** Maintains consistency in how product quantities are presented, preventing confusion and ensuring accurate product listings.
11. **Site Settings:** Controls the branding and front-end configuration of the website, allowing non-technical administrators to update logos, titles, and visual themes without editing code.

Each section is designed to support a specific part of the system's operations, and together they provide a complete administrative environment for managing the entire e-commerce platform.

2.3 Site Settings

The Site Settings page is the administrative interface responsible for configuring the global appearance, branding, and operational settings of the e-commerce system. Unlike modules that manage products or orders, this page affects the entire public-facing website as well as internal administrative tools. Because of its reach, the component is designed to be highly structured, validation-driven, and fully permission-restricted. Administrators use this page to customize everything from logos, colors, and page headings to email settings and notification behavior. The component relies heavily on React Query, React Hook Form, Zod validation, and modular UI components to deliver a real-time, error-resistant environment for system-wide configuration.

When the administrator opens the Site Settings page, they see a multi-tab interface that allows them to control the appearance, content, and communication settings of the entire website. The page is organized into clear categories so the administrator can easily update different parts of the system without needing technical knowledge. Everything they change here affects what customers and other users see on the platform.

General Tab

In the General tab, the user can edit the most visible elements of the website's identity:

- The site name
- The text shown on the browser tab
- The main logo
- The favicon (the small icon shown in the browser tab)

The interface also shows previews of the uploaded images, allowing the administrator to immediately check how the logo or favicon will appear. They can also adjust the size of the header logo so the branding fits neatly in the navigation bar.

What the user can do:

Upload new logos, replace outdated icons, adjust sizes, and update the platform name that appears throughout the site.

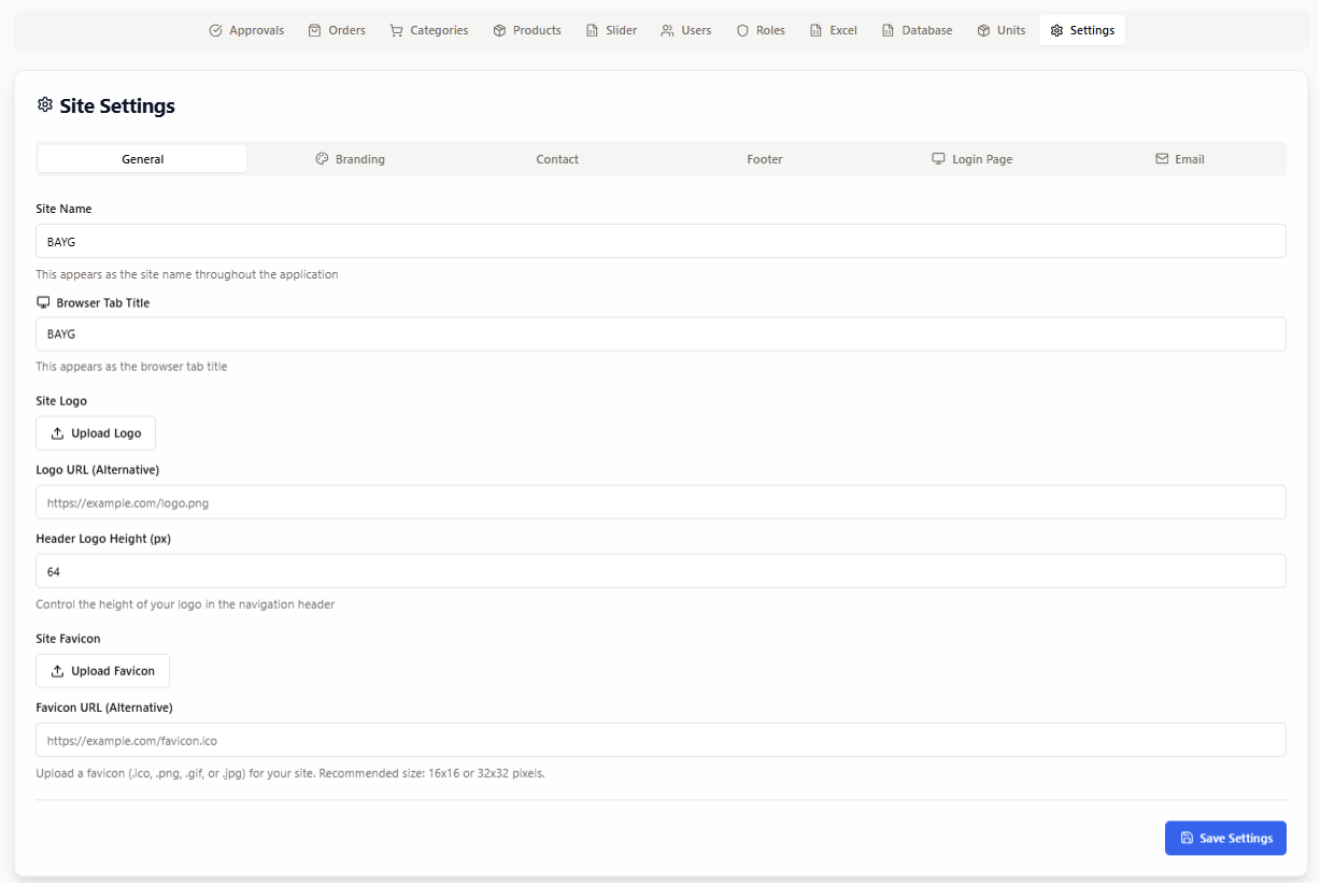


Figure 44 site settings general tab

Branding Tab – Themes and Colors

The Branding tab allows the administrator to control the overall style of the website. They can select from preset themes or choose their own text and navigation colors. the user can change the website's look and feel instantly: such as switching the color scheme for an event, rebranding, or improving readability. This tab is as crucial as it is simple as the website's look has to abide by BAYG standards and must be approved by the Marketing and Branding Departments.

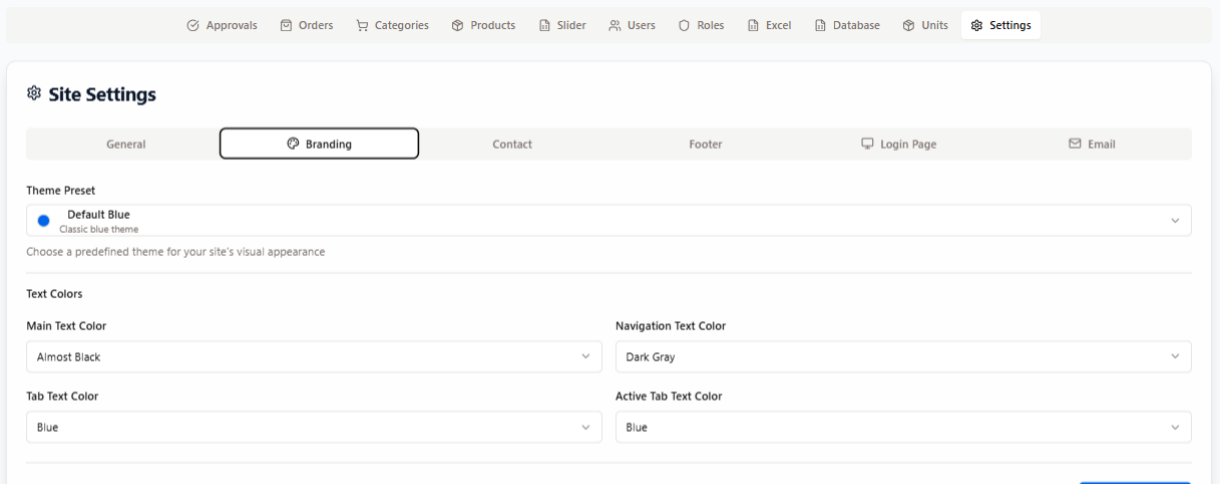


Figure 45 Branding Tab

Contact Tab – Public Contact Information

The Contact tab lets the user update the details that appear on the website's contact section. This includes:

- Contact email
- Support email
- Admin email
- Phone number
- Physical address
- Business hours

What the user can do:
Keep public-facing information up to date so customers always see current contact details.

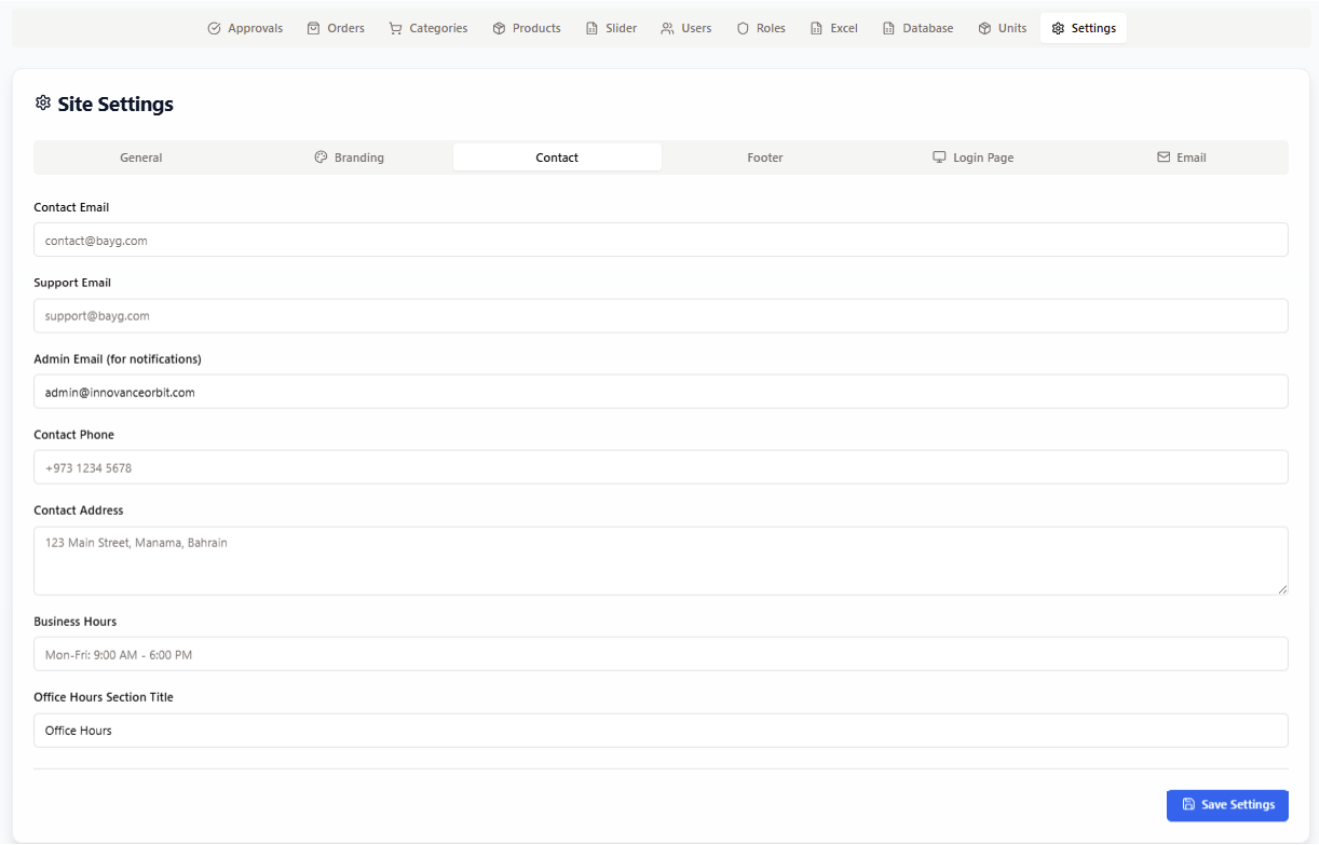


Figure 46 public contact tab

Footer Tab – Website Footer Details

In the Footer tab, administrators manage the content displayed at the bottom of the website. This includes:

- A footer description
- Copyright notice
- Extra footer text
- Quick link labels (Home, Products, About, etc.)
- Social media links
- Footer images (e.g., sponsor logos or payment method icons)

Uploaded images appear immediately as previews, and the admin can adjust their width to match the site layout.

What the user can do:
Customize the footer to reflect branding, add sponsors, update social links, and maintain important navigation shortcuts.

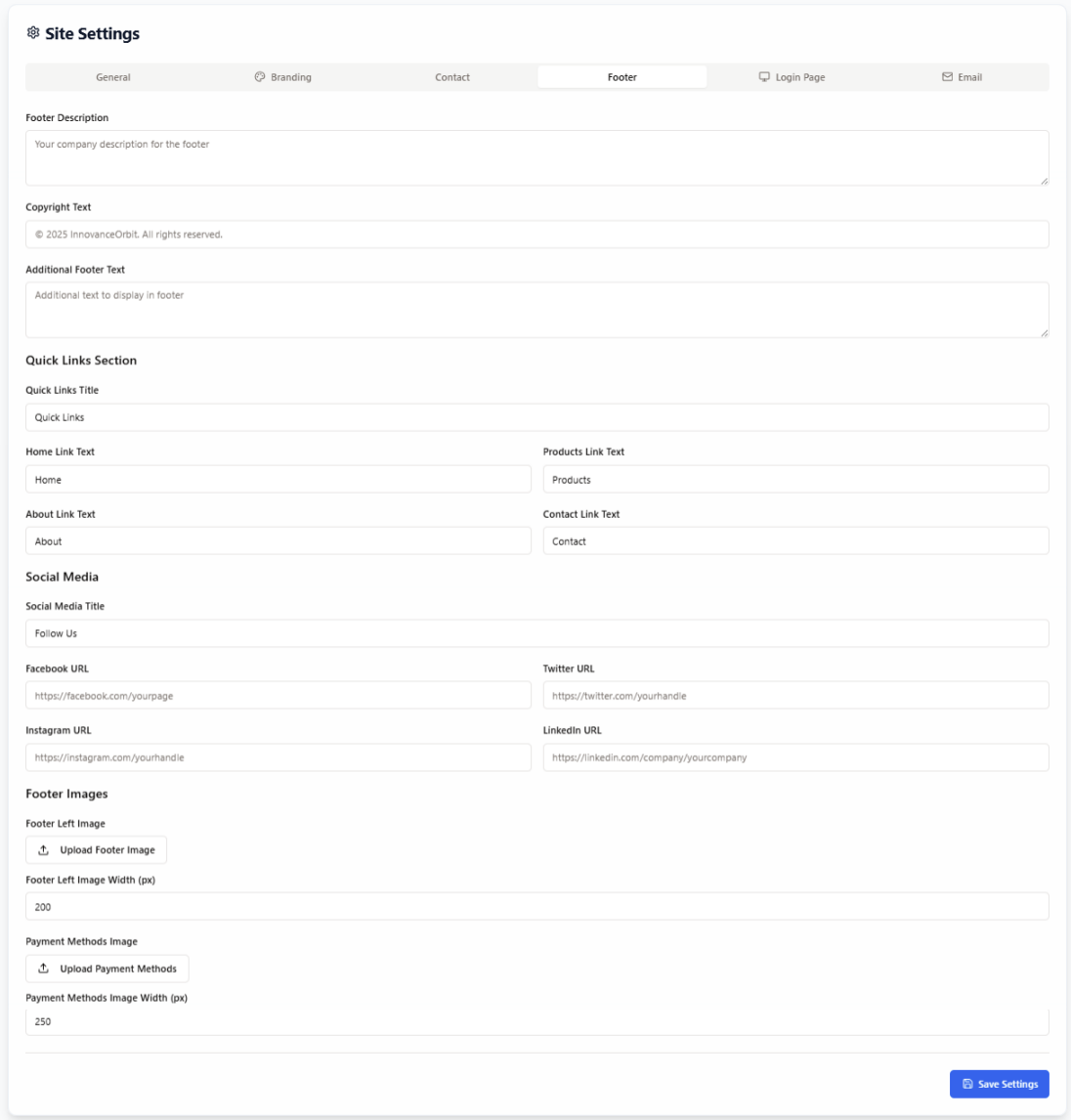


Figure 47 footer tab

Login Tab – Customizing the Login Page

The Login tab lets the administrator personalize the login page for staff and managers. They can change:

- The main heading text
- Subtitle text
- The login page title
- The access message shown below the form
- The logo displayed on the login page
- The logo width

A preview of the current logo is immediately shown on the page.

What the user can do:
Adjust the login screen to match organizational branding or update messaging for users who need access.

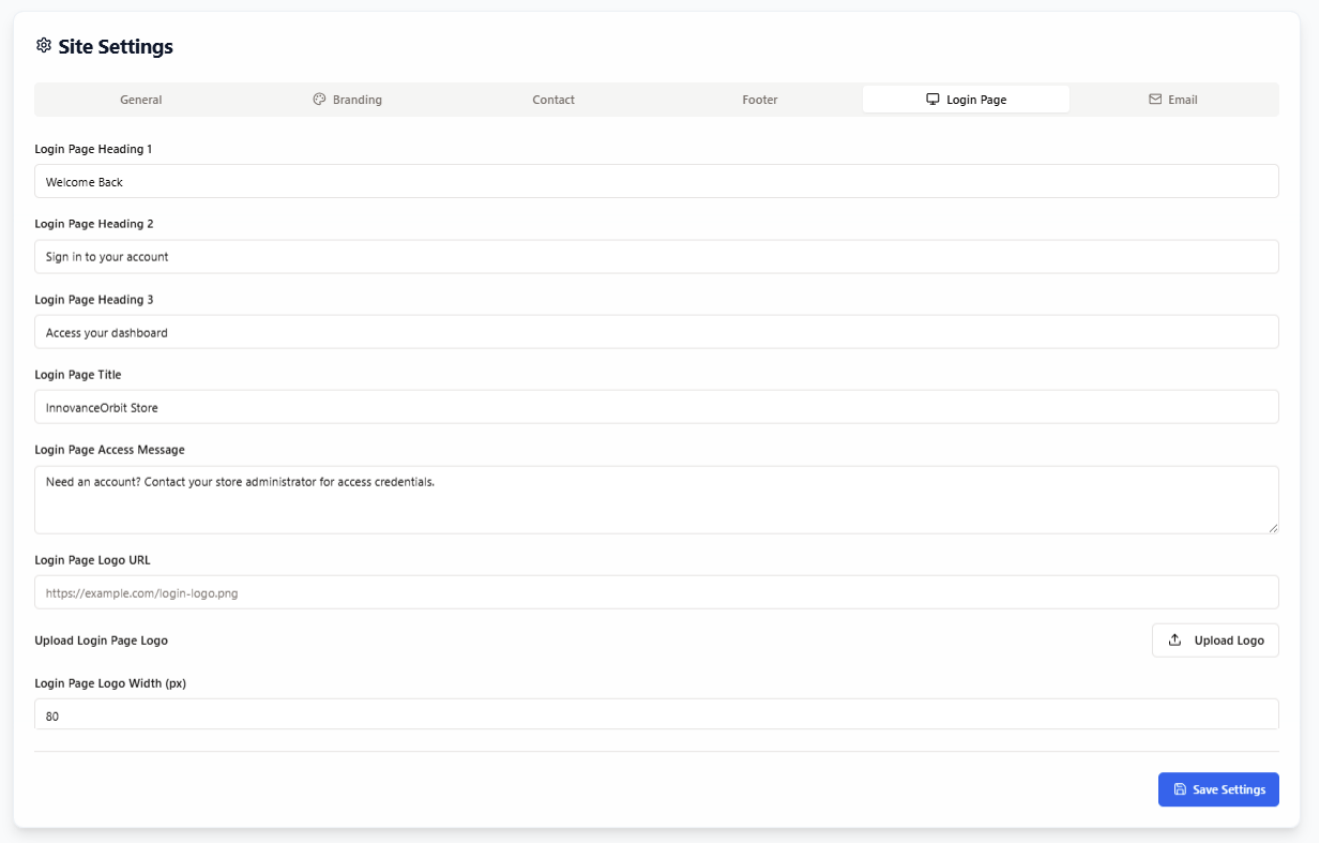


Figure 48 login tab

Email Tab – Email Notifications and SMTP Setup

The Email tab deals with automated email functionality. Here, the user can:

- Turn email notifications on or off
- Enter SMTP settings (provider, port, username, password)
- Specify the “From Name” and “From Email” used in outgoing messages
- Check if the email system is configured correctly
- Send a test email to verify everything works

The page also includes helpful instructions for common email providers such as Gmail, Outlook, and Yahoo.

What the user can do:
Enable and configure email notifications, manage how order confirmation emails are sent, and test the setup without leaving the page.

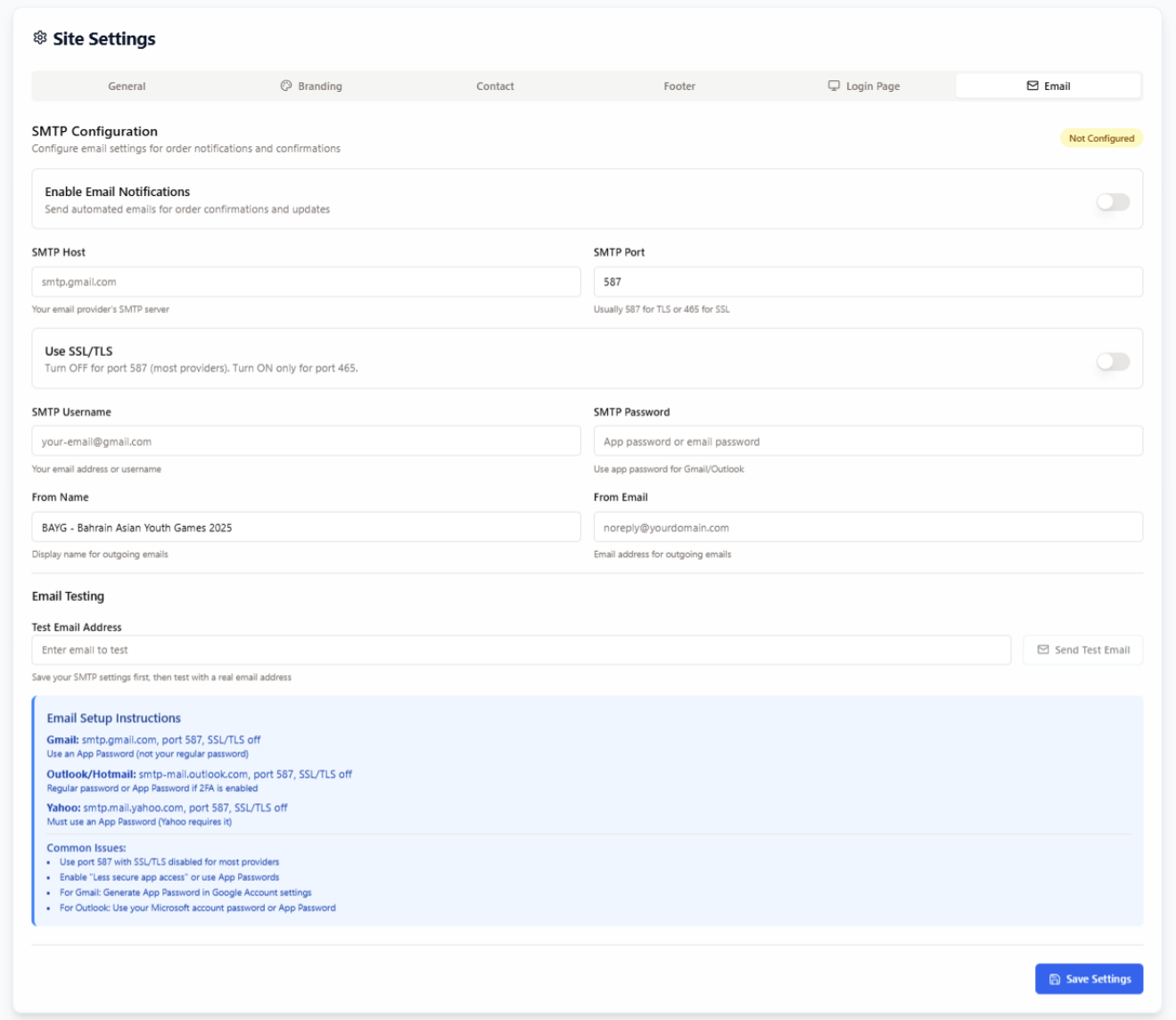


Figure 49 email tab

2.4 Order Approvals

At the top of the page, the user sees:

1) The title "Order Approval Management"
2) A small summary showing:
    a) How many orders are pending
    b) How many have been approved

This gives the administrator an immediate sense of workload and recent activity without having to open individual orders.

Pending Orders Section

If there are orders waiting for approval, they appear at the top inside a highlighted card. Each pending order is displayed as its own small, easy-to-read block. The administrator sees:

- The order number
- Customer name and email
- How long ago the order was submitted
- The order total
- A colored badge showing the current approval status (Pending)

The layout makes it easy to quickly scan and understand the most important information immediately.

What the user can do with each pending order

Each pending order includes three action buttons:

Approve

Clicking Approve opens a confirmation dialog.
Inside the dialog, the user can:

- Add optional remarks for the customer
- Confirm the approval

After approval:

- The order is marked as approved
- The customer automatically receives an email notification
- The order moves from “Pending” to “Processed Orders”

Reject

Clicking Reject opens a rejection dialog.
Here, the administrator must:

- Provide a reason for rejecting the order
- Confirm the action

After rejection:

- The order is marked as rejected
- The customer receives an email explaining the rejection

- The order moves to the “Processed Orders” section

View Details

The View Details button is provided for cases where the administrator wants to inspect the full details of the order before deciding. (In this component, it acts as a placeholder, but from the user’s perspective, it represents the ability to review the order more deeply.)

Below is an example of how the page looks like:

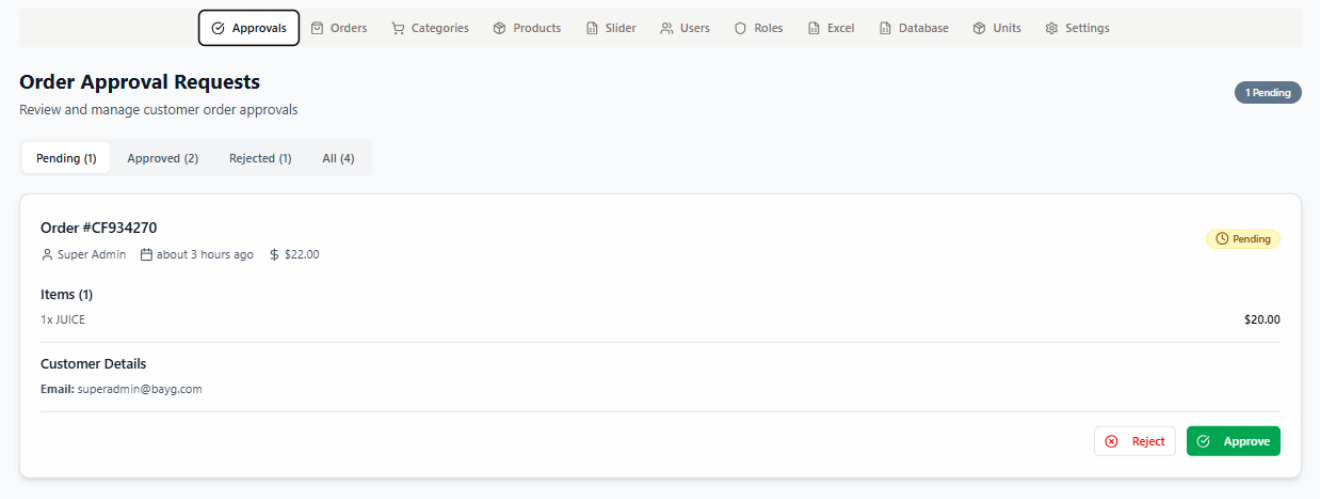


Figure 50 Order Approval requests tab

Processed Orders Section

In the Approved/Rejected/All tabs, administrators see a list of the most recent approved or rejected orders.
For each processed order, the user can see:

- The order number
- The customer’s name
- The approval status (Approved / Rejected)
- The total amount
- The admin’s remarks, if any
- How long ago the order was processed

This section gives administrators a clear historical view of their decisions and provides context for order trends or recurring issues.

No Orders State

If there are no pending or processed orders, the page shows a simple empty-state message with an icon, telling the administrator that there are currently no orders requiring approval. This keeps the interface clean and avoids confusion.

Overall Purpose

From the user’s viewpoint, the Order Approval Management page provides:

- A clean list of all orders waiting for administrative approval
- Clear information about who placed each order and when
- Tools to approve or reject orders with remarks

- Automatic communication to customers via email
- A record of recently processed orders for transparency and review

This page is designed to make the approval workflow fast, organized, and reliable for administrators, ensuring that every order is reviewed properly and customers are promptly informed of the decision.

2.5 Orders Management tab

The Order Management page provides administrators with a clear and organized overview of every order placed on the platform. It displays each order in a card layout, making it easy to scan essential information and quickly take action. The design emphasizes clarity, status visibility, and smooth administrative workflow.

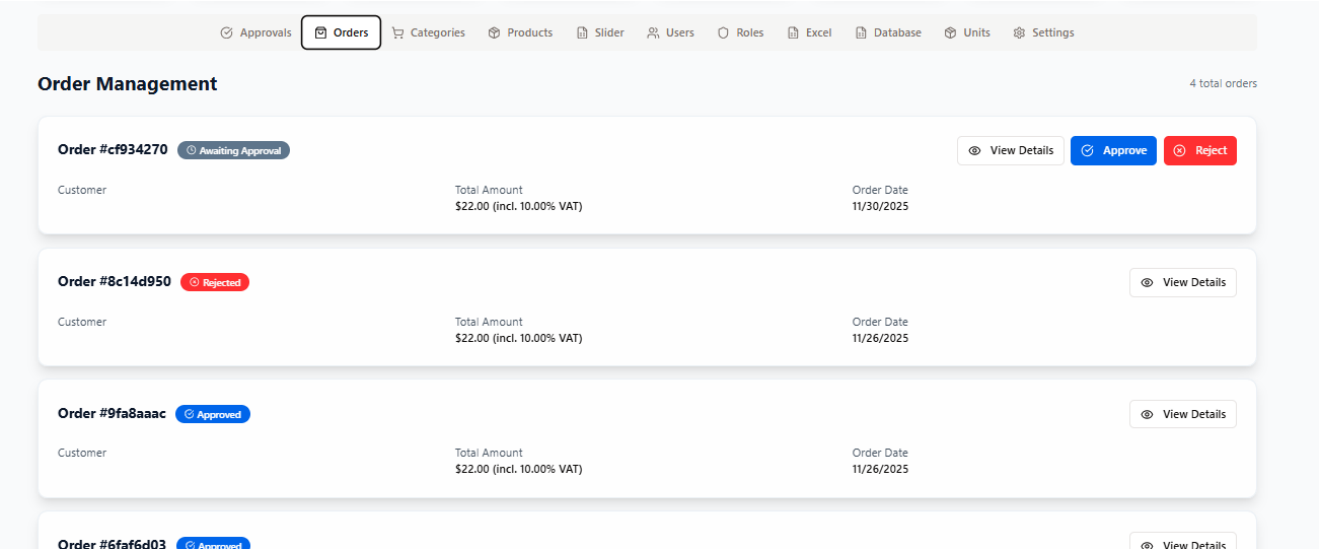


Figure 51 orders management tab

Order List

All orders are displayed in a scrollable list. Each order appears inside a card that summarizes the most important details:

1) **Order number**
2) **Customer name**
3) **Total amount and VAT**
4) **Order date**
5) **Current status**, shown using a coloured badge such as:
    a) *Awaiting approval*
    b) *Approved*
    c) *Rejected*
    d) *Processing*
    e) *Shipped*
    f) *Delivered*
    g) *Cancelled*
    h) *Payment pending*

The badge system helps administrators quickly identify which orders need attention.

Actions Available for Each Order

Each order card includes buttons that allow administrators to take appropriate actions depending on the order's state.

View Full Order Details

Every order has a "View Details" button.

When clicked, a detailed pop-up appears showing:

- Customer information (name and email)
- Order date and estimated delivery timeframe
- Payment method used
- Every item in the order with quantities and prices
- A full cost breakdown (subtotal, tax, VAT rate, and total)
- Any remarks from administrators about that specific order
- Shipping information, when applicable

This pop-up allows the administrator to understand the order fully before taking action.

Approve an Order

If an order is still waiting for approval, the administrator will see an **"**Approve**"** button.

Clicking it opens a confirmation dialogue where the admin can:

- Add optional notes or instructions for the customer
- Approve the order with one click

After approval:

- The system notifies the customer by email
- The order status updates to *Approved*
- The order moves forward in the processing workflow

Reject an Order

For orders that require approval, there is also a **"**Reject**"** button.

When clicked, a rejection dialogue appears. Here the admin must:

- Provide a reason for the rejection
- Confirm their decision

Once rejected:

- The customer receives an email explaining the reason
- The order status updates to *Rejected*
- The order is removed from the approval queue

This ensures that customers are kept informed and that internal order records remain consistent.

Mark an Order as Delivered

For orders that have passed payment and are ready to be closed, a **"Mark Delivered"** button may appear (depending on the order status).

When clicked:

- The order is marked as delivered
- The customer receives a delivery confirmation message
- The order moves into the completed section of the system

This button simplifies the final step of the order lifecycle.

Empty State

If the system contains no orders at all, the page shows a simple empty-state message with an icon and a short explanation informing administrators that orders will appear once customers start placing them. This prevents confusion and keeps the interface clean.

Overall User Purpose

From a user's standpoint, the Order Management page is designed to:

- Give a clear overview of all orders in one place
- Highlight orders that need immediate attention
- Offer full visibility into each order's details
- Allow quick approval, rejection, or completion of orders
- Ensure customers are automatically informed of every decision

The entire page supports efficient and transparent workflow, enabling administrators to manage customer orders confidently and with minimal friction.

2.6 Product management

The Product Management page is the central area where administrators create, edit, organize, and maintain all products available in the system. The page is designed to be straightforward and functional, presenting products in a clean list and offering an intuitive dialogue-based interface for managing product details.

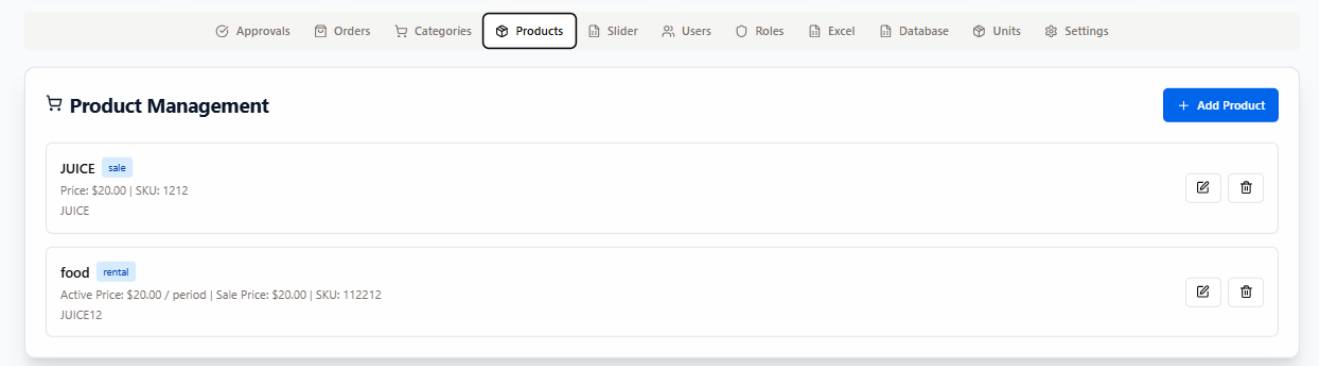


Figure 52 product management tab

Product List

All existing products are displayed in a vertically scrolling list of cards.
Each product card shows key information such as:

- Product name
- Product type (**sale** or **rental**)
- Price
- SKU
- Whether the product is **featured**
- Whether the product is **active or inactive**
- A short preview of the description (if available)

This gives administrators a quick snapshot of their inventory without opening each product individually.

Actions Available on Each Product

Each product entry includes two buttons:

1. **Edit** – opens the product in an editable form
2. **Delete** – removes the product from the system

The "Edit" option allows the administrator to update the product's name, image, price, category, status, and all other attributes.
The "Delete" option gives them a straightforward way to remove products that are no longer offered.

If the system has no products, the page displays a message informing the user that no products exist yet.

Adding or Editing a Product

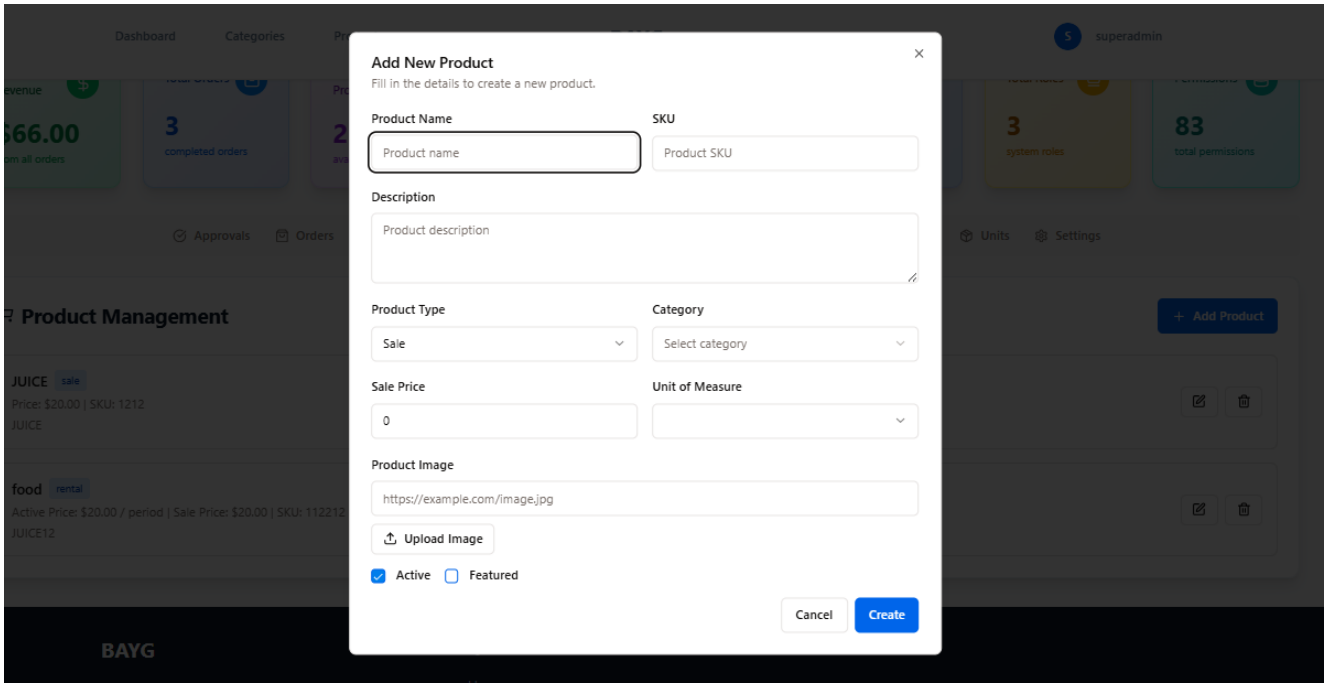


Figure 53 adding product dialogue box

Whether the user is adding a new product or editing an existing one, the interface uses a pop-up dialog that covers the center of the screen.
This form is organized into clear sections and guides the administrator through each required detail:

Basic Product Information

The user can enter:

- Product name
- SKU
- Description

These fields provide identifying information that customers and staff will later see.

Product Type

The user selects whether the product is:

- Sale: a regular purchasable item
- Rental: an item that can be rented for a certain period

This ensures the system handles pricing and display correctly based on the type.

Category and Unit of Measure

Using dropdown selection fields, the administrator can assign:

- A category (e.g., Electronics, Apparel, Equipment)
- A unit of measure such as *piece*, *box*, *set*, or any custom unit defined in the system

This helps keep inventory organized and searchable.

Pricing

The form adjusts based on the product type:

- Sale products show a **sale price**
- Rental products show a **rental price**

The user simply enters the price as a number.

Product Image

Administrators can either:

- Paste an image URL
- Or upload an image directly from their computer

A small preview of the uploaded image appears underneath, allowing the user to confirm what will be displayed on the site.

Status and Visibility Options

Two checkboxes allow the user to control how the product appears on the website:

- Active: whether the product is available for use
- Featured: whether it is highlighted in promotional areas

These quick toggles help manage seasonal items, promotions, and discontinued products efficiently.

Saving or Cancelling Changes

At the bottom of the form, the user sees:

- A Cancel button, which closes the dialog without saving
- A Create or Update button, depending on whether it is a new or existing product

Once saved, the product list refreshes to reflect the changes immediately.

Overall Purpose

From the administrator's perspective, the Product Management page serves as a complete toolkit for controlling the product catalog.
It enables them to:

- Add new products quickly
- Edit or update existing products
- Upload and manage product images
- Assign categories, units, and pricing
- Mark products as active, inactive, or featured
- Remove outdated products

The interface keeps everything organized and accessible so administrators can manage the catalog effectively without any technical knowledge.

2.7 Category Management

The Category Management page allows administrators to manage the different product categories used throughout the system. Categories help structure the catalogue by grouping related items together, improving both internal organization and customer browsing. The page is presented in a simple list-based layout, making it easy to create, edit, and remove categories.

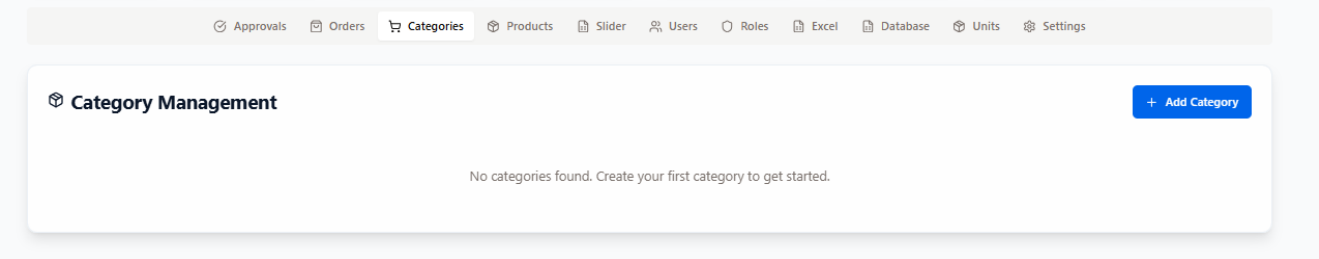


Figure 54 product management tab

Category List

All categories are shown in individual cards arranged in a vertical list.
Each category card displays:

- The category name
- A short description (if one was provided)
- The image URL or a preview image (if one exists)

This gives the administrator an at-a-glance overview of all available categories and the details associated with each.

Actions Available for Each Category

To the right of every category, the user can choose to:

- **Edit** – open the category details to update the name, description, or image
- **Delete** – permanently remove the category

These options allow administrators to easily keep the system's category structure up to date.

If no categories exist yet, a centered message explains that the list is empty and invites the user to create their first category.

Adding or Editing a Category

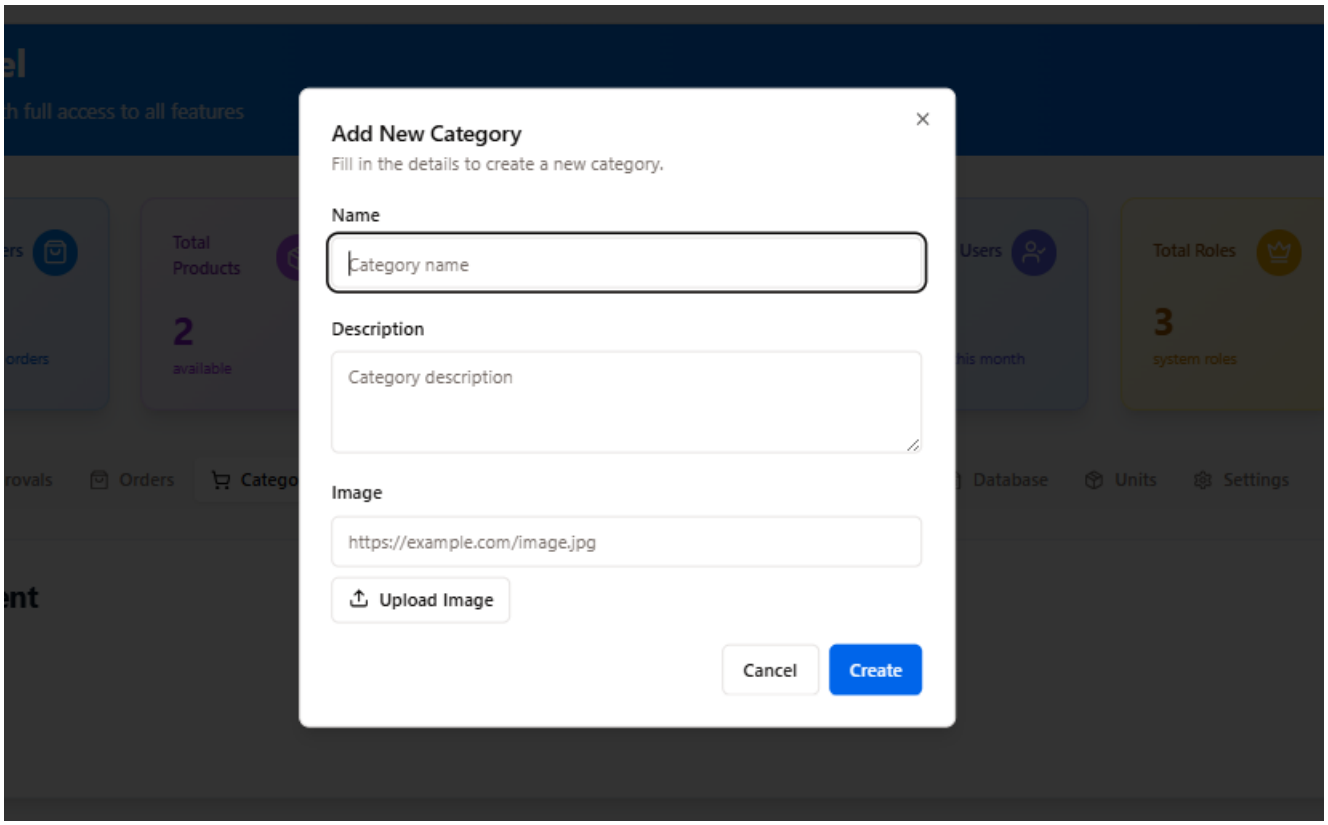


Figure 55 add category

Whether the administrator is adding a new category or editing an existing one, the system uses a pop-up dialog to capture all the required information in a clean and guided format.

Inside this dialog, the user can provide:

Category Name

A short, descriptive title for the category (e.g., “Electronics”, “Sports Equipment”, etc.).

Category Description

An optional text field where the administrator can include additional context or details.
This helps clarify the purpose of the category for other administrators.

Category Image

The administrator can:

- Paste the URL of an image, **or**
- Upload an image directly from their device

If an image is provided, a preview is displayed immediately within the form. This allows the administrator to confirm the correct image before saving.

Saving or Cancelling Changes

At the bottom of the form, administrators are offered:

- A **Cancel** button to close the dialog without saving
- A **Create** or **Update** button depending on the action

Once the user submits the form, the category list refreshes automatically to show the newly added or updated category.

Overall Purpose

From the administrator's perspective, the Category Management page provides a simple and efficient way to:

- Create new product categories
- Organize the catalogue structure
- Edit category names, images, and descriptions
- Remove categories that are no longer needed

The interface gathers all essential actions into a single page, making category organization straightforward and helping maintain a clean and navigable product catalogue.

2.8 Slider Management

The Slider Management page gives administrators full control over the large rotating images displayed on the website's homepage. These images are often used for promotions, announcements, or branding, so the page focuses on making it easy to create, update, reorder, and activate or deactivate each slide.

The interface is structured to be straightforward and intuitive, allowing administrators to manage visual content without any technical knowledge.

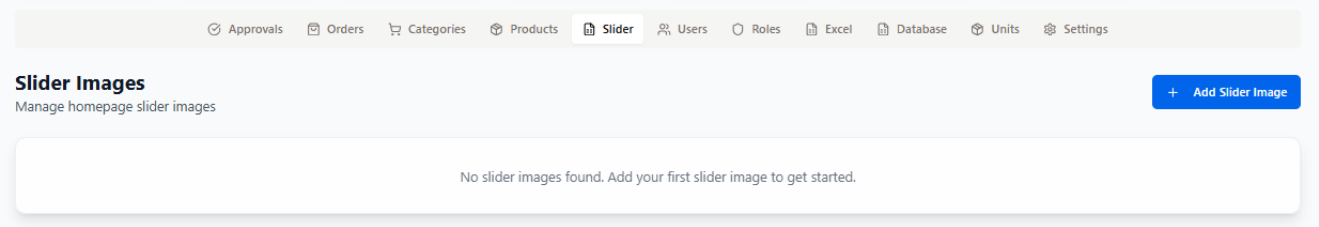


Figure 56 Slider Images tab

Adding a New Slider Image

When the user clicks "Add Slider Image", a pop-up dialog appears in the center of the screen.
This is where the administrator creates a new slider item for the homepage.

Inside this form, the user can enter:

Title (Optional)

A short caption or heading for the slide.

Description (Optional)

A brief subtitle or supporting text.

Image (Required)

The user can either:

- Paste an image URL
  or
- Upload an image from their computer via a file picker

Once an image is selected, a live preview shows how it will appear on the homepage slider.

Sort Order

A number that determines the position of the slide in the rotation.
Lower numbers show first.

Active Toggle

A simple switch that determines whether the slide is currently visible on the homepage or hidden.

At the bottom of the form, the user can:

- Save the new slide
- Cancel and close the dialog

Editing an Existing Slider Image

Next to every slider item in the list, the user can click an Edit button.

This opens the same dialog used for adding a slider, but filled with the existing slide's information.
The administrator can update:

- Text
- Image
- Display status
- Sort order

Changes apply immediately once saved.

Toggling Visibility (Active / Inactive)

Each slide has an eye-shaped icon next to it:

- Eye icon ("show") means the slide is active and currently shown on the homepage.
- Eye-off icon ("hide") means the slide is inactive and not displayed.

The user can toggle visibility with a single click, without opening the edit dialog.
This is useful when temporarily hiding a slide (e.g., after a promotion ends).

Deleting a Slider Image

A trash icon allows the user to permanently remove a slide.

After deletion:

- The slide disappears from the list
- It is removed from the homepage

A confirmation toast appears to notify the user that the slide was deleted successfully.

Slider List

All existing slider images appear in a clean, scrollable list of cards.
Each card displays:

- A thumbnail image
- The slide title and description
- The current sort order
- Whether the slide is active or inactive

- A drag handle icon indicating that slides can be ordered visually (if future features support dragging)

To the right of each card, action buttons allow:

- Toggle active/inactive
- Edit
- Delete

This gives the administrator full control directly from the list without needing to open multiple menus.

Loading and Empty States

If the system is still loading images, skeleton placeholders appear to indicate that content is being fetched.

If no slider images exist yet, the user sees a simple message encouraging them to add their first slide.

Overall Purpose

From the user's perspective, the Slider Management page enables them to:

- Add new promotional slides
- Change their text, image, or order
- Hide or show slides with one click
- Delete old or outdated content
- Maintain a professional and up-to-date homepage banner

This page ensures that administrators can keep the website's visual presentation fresh and relevant without relying on developers or editing files manually.

2.9 User Management

The User Management page provides administrators with a complete overview of all registered users in the system, along with the tools needed to create accounts, update user information, manage administrative privileges, and assign roles. The interface is designed to simplify the process of managing user access and maintaining control over the system's security and structure.

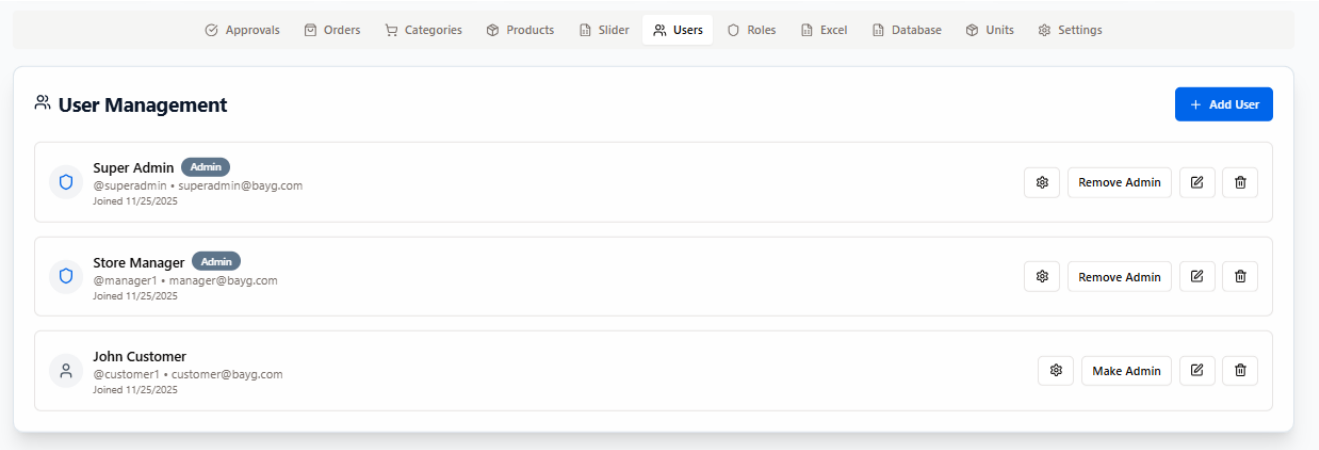


Figure 57 User Management Tab

User List

All registered users are displayed in a clean list, with each user shown inside an individual card.
Each user entry presents:

- The user's full name (or username if no names are provided)
- Email address
- Username
- The date the user joined
- An icon that indicates whether they are a **normal user** or an **admin**
- A small "Admin" badge if the user has admin privileges

This gives administrators a snapshot of the system's user base and their respective roles.

Actions Available for Each User

Each user card includes action buttons that allow administrators to manage that specific account.

Assign Role & Permissions (Super Admin Only)

If the administrator viewing the page is a **Super Admin**, they will see a Settings button next to each user.

Clicking this opens a role assignment dialog where they can:

- Select a role from a dropdown menu
- Review role descriptions
- Assign the selected role to the chosen user

This feature provides structured permission control based on predefined roles.

Promote or Remove Admin Access

Every user can be granted or stripped of admin privileges using a "Make Admin" or "Remove Admin" button.

This allows administrators to:

- Elevate a user to admin status when additional system oversight is needed
- Remove admin access when it is no longer appropriate or secure

A confirmation toast notifies the administrator of the successful change.

Edit User Information

Clicking the Edit icon opens a dialog where administrators can update:

- First name
- Last name
- Username
- Email
- Password (left blank to keep the existing one)
- Admin status

The form automatically fills in existing user information and allows edits without affecting other parts of the user's account.

Delete a User

A Delete button allows administrators to permanently remove a user from the system.
This action:

- Removes the user's account
- Refreshes the list to reflect the change
- Displays a success toast confirming the deletion

This option is typically used to manage inactive or unauthorized accounts.

Adding a New User

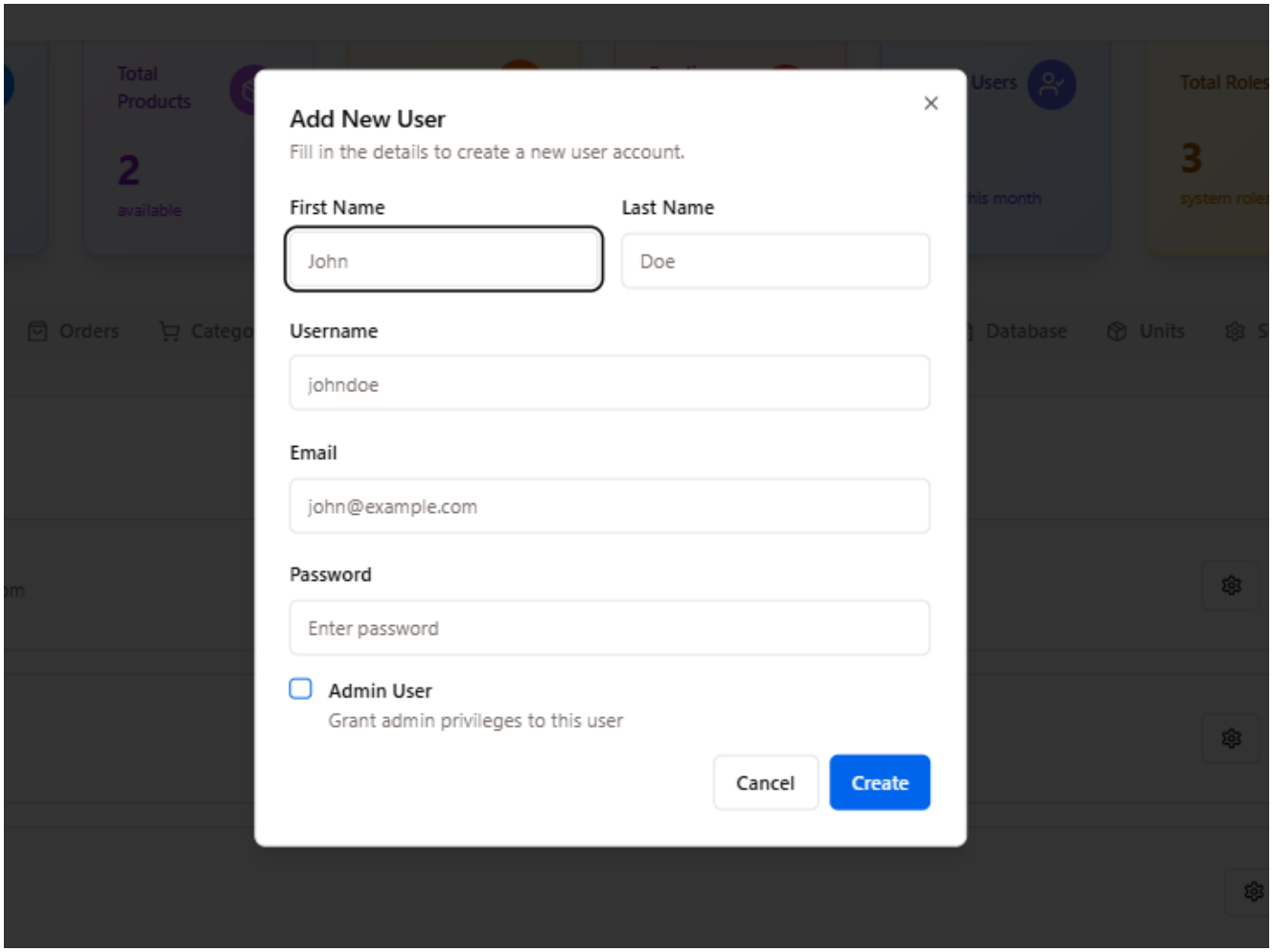


Figure 58 adding user dialogue box

When the admin clicks "Add User", a pop-up form appears.
This form requests:

- First name
- Last name
- Username
- Email address
- Password
- Admin checkbox (optional)

The administrator fills in the required details and submits the form.
Upon success, the new user appears immediately in the list.

Role Assignment Dialog (Super Admin Only)

When a Super Admin chooses to manage a user's role, a dedicated dialog is displayed.
Inside this dialog, they can:

- Choose a role from a dropdown list
- Read the role's description
- Apply the new role to the selected user

Once assigned, the role takes effect immediately.

Loading and Empty States

If user data is still loading, the page displays a simple "Loading users..." message.
If there are no users in the system, the page shows a helpful message inviting the admin to create their first user.

Overall Purpose

The User Management page gives administrators full control over the system's user accounts by enabling them to:

- Create new accounts
- Update user profiles
- Assign roles and permissions
- Control admin privileges
- Remove users when needed

This ensures a secure, organized, and scalable user ecosystem, where permissions and responsibilities can be managed as the system grows.

2.10 Permission and Role Management

The Role and Permission Management section is accessible only to the Super Admin, and it provides a structured interface for controlling how access is distributed across the system. It is divided into three main tabs; Manage Permissions, Assign Roles to Users, and System Overview, each focusing on a different part of the system's access control.

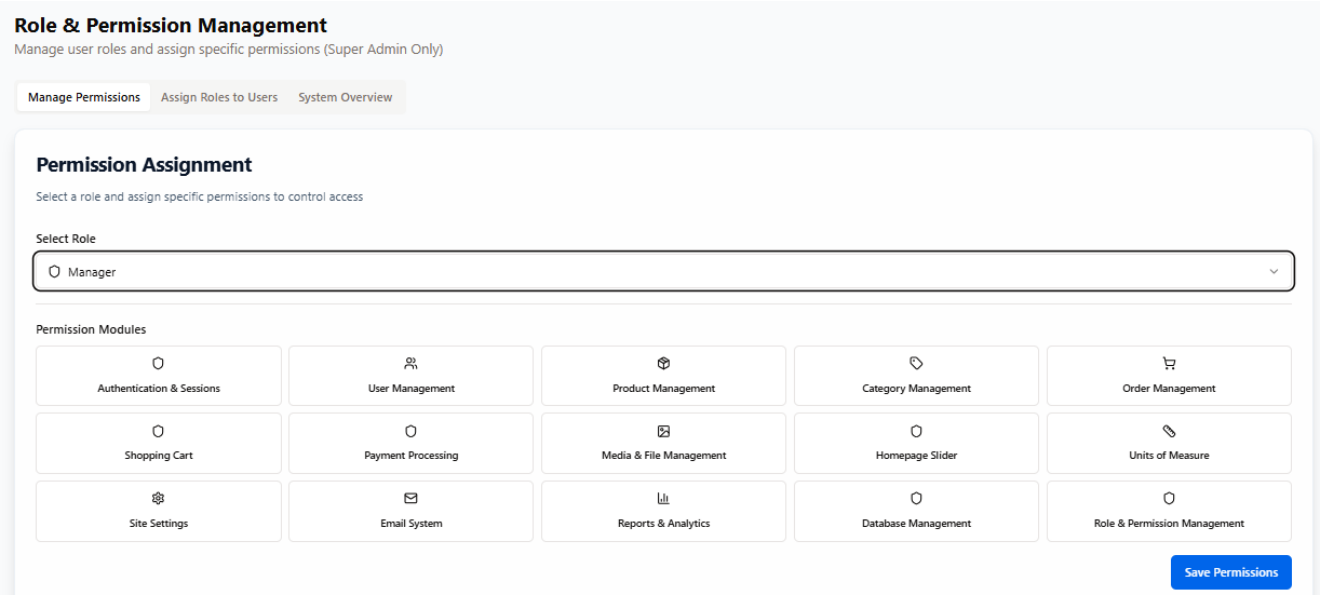


Figure 59 Role and permissions tab

Manage Permissions Tab

Selecting a Role

At the top of the page, the Super Admin sees a dropdown menu that lists all available roles in the system (except system-protected roles like Super Admin).
Choosing a role determines which permissions are displayed and edited.

Choosing a Permission Module

Below the role selector, the interface shows a grid of "permission modules."
Each module represents a major feature area of the system such as:

- Users
- Products
- Slider Images
- Categories
- Orders
- Units of Measure
- Roles & Permissions

Each module is represented through an icon and a label, making it easy to identify.

Viewing and Editing Permissions

Once a module is selected:

1) All permissions associated with that module appear in a checklist.
2) Each permission has:
    a) A title (e.g., *"View Orders"*)

b) A short description of what it allows

c) A small badge indicating the action type (e.g., *view*, *edit*, *delete*, *create*)

The Super Admin can check or uncheck these permissions to grant or remove access for the selected role.

Saving Permissions

A "Save Permissions" button finalizes changes.
A confirmation message appears when the update is successful.

This section allows the Super Admin to define precisely what every role can or cannot do.

Assign Roles to Users Tab

This tab focuses on assigning roles directly to specific users.

Selecting a User

A dropdown allows the Super Admin to choose from all registered users (except other Super Admins, who cannot be reassigned for safety).

Each user entry displays:

- Username
- Email
- Admin status badge if applicable

Selecting a Role

Next to the user selector is another dropdown listing all non-system roles.

Assigning the Role

After choosing both:

- Assign Role updates the user instantly.
- A confirmation appears when the change is complete.

Current Role Assignments

Below the role assignment section, the page displays a live summary of all users, showing:

1) Their username and email
2) Their current role, if any
3) Badges indicating whether they are:
    a) Super Admin
    b) Admin
    c) Standard User

This provides a clear overview of who has what type of access in the system.

System Overview Tab

The System Overview tab provides a quick snapshot of the system's access-control structure.

It displays three high-level statistics:

- Total number of roles
- Total permission modules

- Total permissions active in the system

Below these statistics is a full list of all permission modules.
Each module card shows:

- The module's icon
- Its display name
- A short explanation of what part of the system it controls

This tab is particularly helpful for understanding the scale and organisation of the system's permissions.

Overall Purpose of This Section

The Role & Permission Management interface gives the Super Admin complete control over:

- What a role can access
- Which permissions are granted to which users
- How system-wide access is structured and monitored

By separating permissions, roles, and user assignments into clear sections, the system ensures:

- Security: Users only see and do what they are permitted to.
- Scalability: New roles and permissions can be defined as the system grows.
- Accountability: Administrators can track who has been assigned which role.

This section is central to maintaining a secure and organized multi-user administration environment.

2.11 Excel Management

The Excel Data Management page gives administrators full control over exporting and importing the system's data using Excel files. This is designed to make bulk data editing, analysis, backup, and large-scale updates easier and more efficient.
The page is organized into two main sections: **Individual Sheets** and **Bulk Operations**.

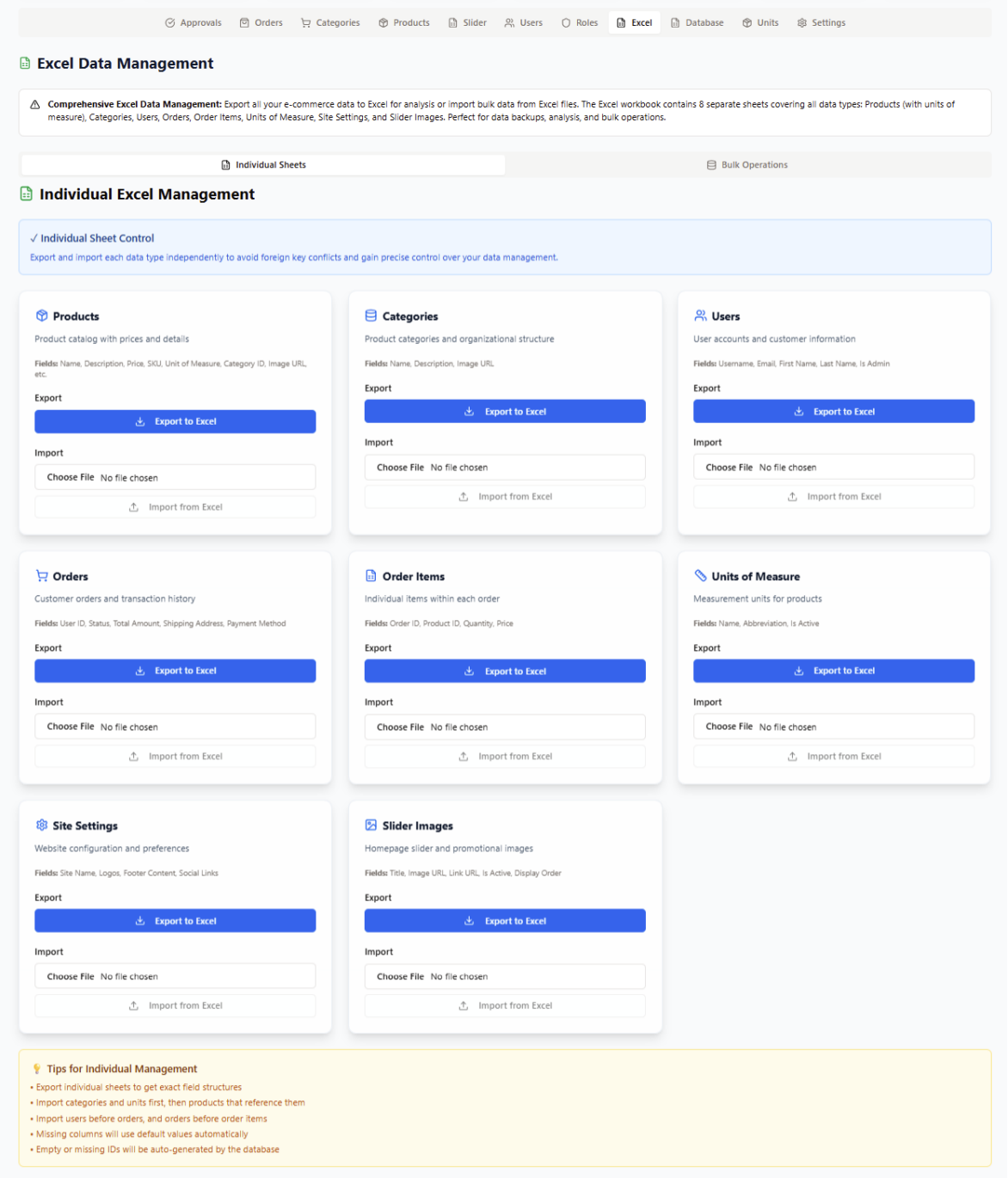


Figure 60 excel management tab

Individual Sheets Tab

When switching to the Individual Sheets tab, the user is given tools (via the IndividualExcelManager component) to export or import specific sheets separately.
For example:

- Exporting only Products
- Exporting only Categories

- Importing a single sheet without affecting other data

This provides finer control for administrators who do not want to perform full-system imports or exports.

(The system automatically loads the component, so the user sees additional tools designed for individual sheet management.)

Bulk Operations Tab

The Bulk Operations tab is the main interface for handling complete Excel imports and exports. It is divided into two large cards: Export to Excel and Import from Excel.

Export to Excel

The Export card allows administrators to download the entire dataset in a single Excel workbook.

What the User Sees

1) A description explaining that the exported file contains all store data
2) A detailed list showing the eight sheets included in the export:
    a) Products
    b) Categories
    c) Users (without passwords)
    d) Units of Measure
    e) Orders
    f) Order Items
    g) Site Settings
    h) Slider Images
3) An Export to Excel button

What the User Can Do

- By clicking the button, the system generates the Excel file and automatically downloads it.
- A confirmation message appears once the export succeeds.
- If export fails, the user receives an error message.

Use Cases

This feature is ideal for:

- Backing up system data
- Analysing data offline
- Sharing structured information with other departments
- Migrating or synchronizing data with other systems

Import from Excel

The Import card allows administrators to upload a complete Excel workbook and replace existing system data.

What the User Sees

1) A file selector for uploading an Excel file

2) File validation messages (e.g., incorrect file type)
3) A list describing the expected structure of all eight sheets
4) Notes explaining how the system handles missing columns, IDs, or passwords
5) A preview of the selected file showing:
    a) File name
    b) File size
6) An Import Excel Data button

What the User Can Do

1) Upload an Excel file containing updated system data
2) Confirm and start the import process
3) Receive feedback on:
    a) Successful imports
    b) Errors (e.g., wrong formatting, missing data)
4) Monitor import progress if the file is large

Important Notes Displayed to the User

- Importing a sheet overwrites existing data
- User passwords cannot be imported for security reasons
- Newly imported users receive random passwords unless changed later
- Missing fields use default database values

Use Cases

This feature is used when:

- Performing bulk updates (e.g., updating thousands of products)
- Restoring backed-up data
- Rebuilding the store's database quickly
- Setting up the system initially with structured data

Overall Purpose of the Excel Management Page

This page gives administrators powerful tools to manage large quantities of data reliably and efficiently. It supports:

- Full-system export for backup and analysis
- Full-system import for rapid data setup or restoration
- Individual sheet operations for precise updates
- Automatic validation, error handling, and feedback

By centralizing all Excel-related operations in one place, the system enables secure, fast, and user-friendly data management for the e-commerce platform.

2.12 Database Management

The Database Management page provides administrators with full control over backing up and restoring the system's database. This section is designed for high-risk, high-impact operations, so the interface is structured to be clear, cautious, and informative. Throughout the page, the design encourages safe handling of data, ensuring administrators understand the consequences of each action.

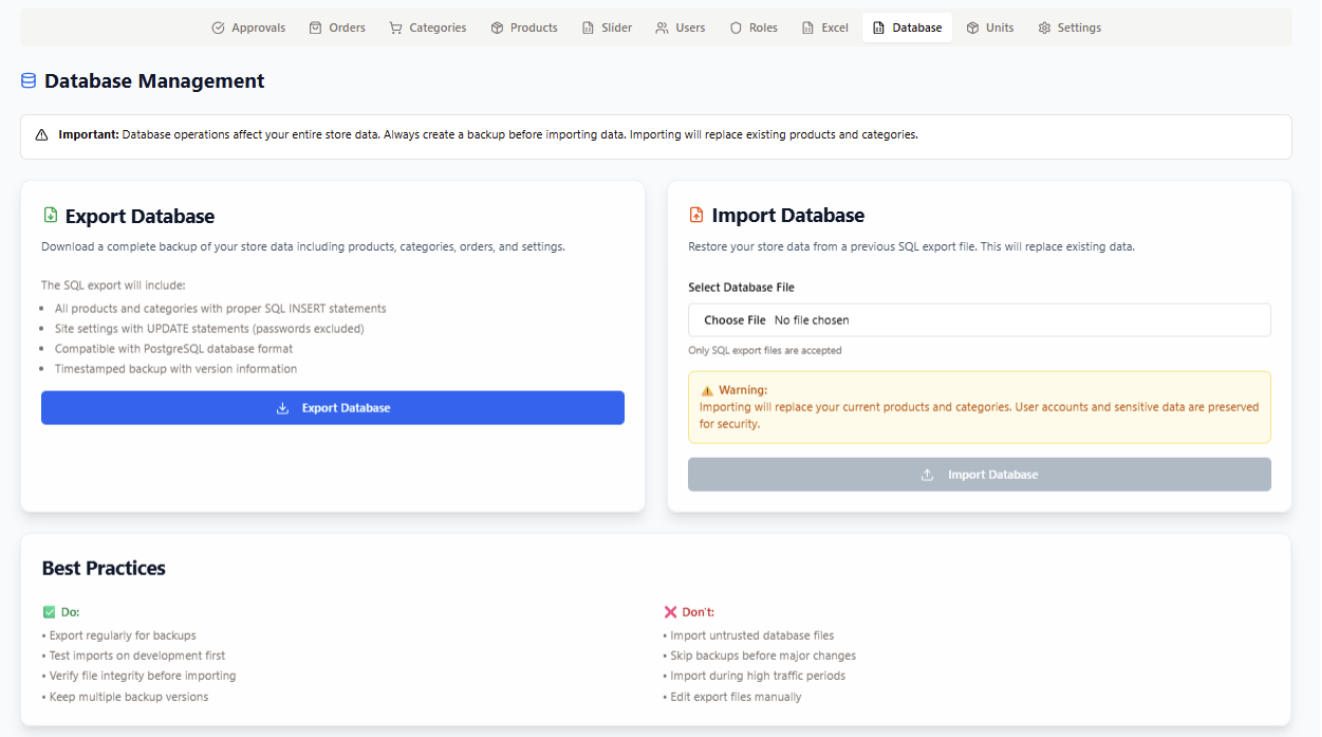


Figure 61 Database Management

Export Database Section

The left-hand card allows administrators to export the full database.

What the User Sees

1) A title labeled "Export Database"
2) A description explaining that this operation downloads a complete backup of the system, including:
    a) Products and categories
    b) Orders
    c) Site settings
3) A list summarizing what the SQL export file contains and how it is structured
4) A button labeled "Export Database"

What the User Can Do

- Start a full database export with a single click
- Automatically download a timestamped .sql file that can be used later for restoration
- Receive success or error notifications depending on the result

Purpose for the User

This tool is intended for:

- Creating regular backups
- Archiving system snapshots

- Preparing for major changes such as imports or migrations
- Maintaining disaster recovery procedures

The export section gives users confidence that they can safely save a full version of their system.

Import Database Section

The right-hand card handles **importing** a previously exported SQL file.

What the User Sees

1) A title labeled “Import Database”
2) A description stating the import will replace existing data
3) A file input where the administrator selects an .sql file
4) A highlighted warning that:
    a) Product and category data will be replaced
    b) Sensitive user accounts are preserved
5) A button labeled “Import Database”, which activates only when a valid file has been chosen

What the User Can Do

- Select a database export file from their computer
- Confirm the import by clicking the button
- Begin restoring the system to the state captured in that backup
- See progress notifications
- After completion, watch the page reload automatically to display the updated data

Why This Matters for the User

This functionality allows administrators to:

- Restore the store from a previous backup
- Revert major mistakes in data entry
- Quickly rebuild the database after corruption or errors
- Recreate production data in development environments

It gives the user a powerful form of control over the entire system, one that requires careful action but offers significant safety and recovery benefits.

Best Practices Section

At the bottom, the user finds a Best Practices card that is divided into two lists:
Do’s and Don’ts.

What the User Sees

1) Clear recommendations on how to handle database backup and import procedures
2) Advice encouraging responsible and safe use of this tool
3) Examples such as:
    a) Regularly exporting backups
    b) Avoiding imports during high-traffic periods

c) Not editing SQL files manually

What This Provides

This section educates administrators on safe operational behavior.
It reinforces the idea that database operations are not routine tasks; they are critical maintenance actions that require attention, planning, and care.

Overall Purpose of This Page

The Database Management interface enables administrators to:

- Securely download a complete backup of the system
- Restore the entire database from a reliable SQL file
- Build proper safety habits around data management

From the user's perspective, the page functions as the central tool for high-level system maintenance and disaster recovery.
Its layout, warnings, and structured workflow ensure administrators understand exactly what will happen before any major change takes place, helping them manage the platform confidently and responsibly.

3. Client Documentation

3.1 Homepage

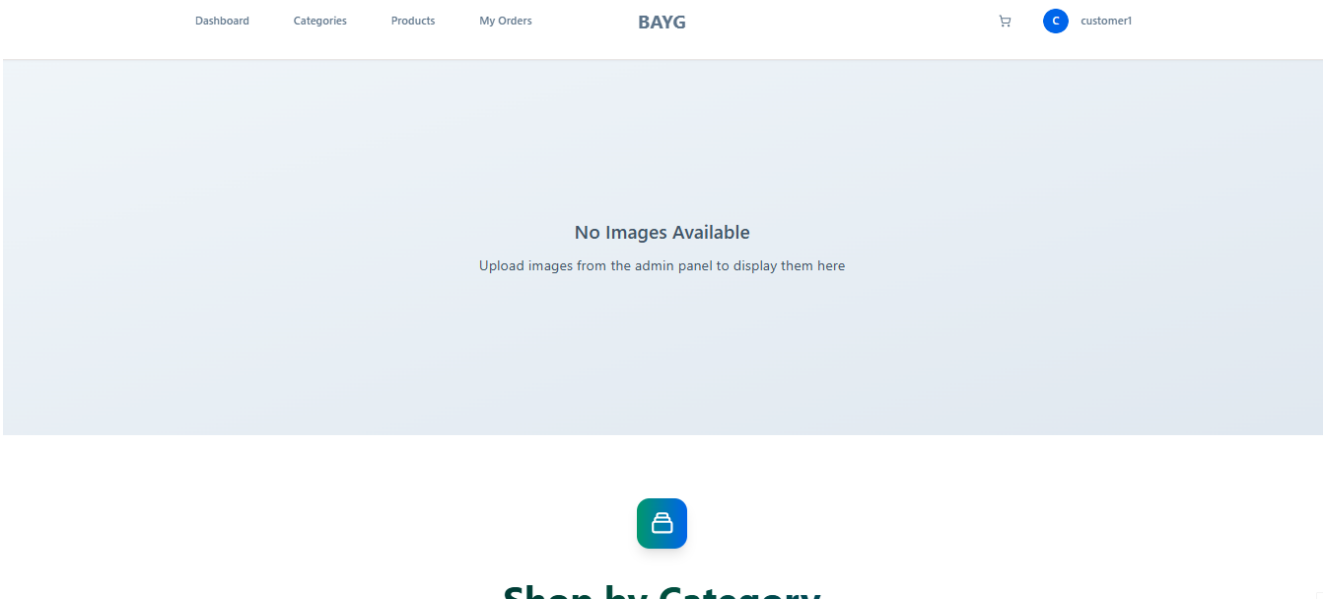


Figure 62 Client Homepage

3.2 User Dashboard and Panel

The User Dashboard serves as the central hub for regular users, giving them a personalized overview of their activity within the platform. Once logged in, users are greeted with an interface that summarises their account status, recent orders, personal details, and shopping cart activity.
The design aims to be welcoming, practical, and easy to navigate, giving users quick access to the actions they perform most frequently on an e-commerce platform.

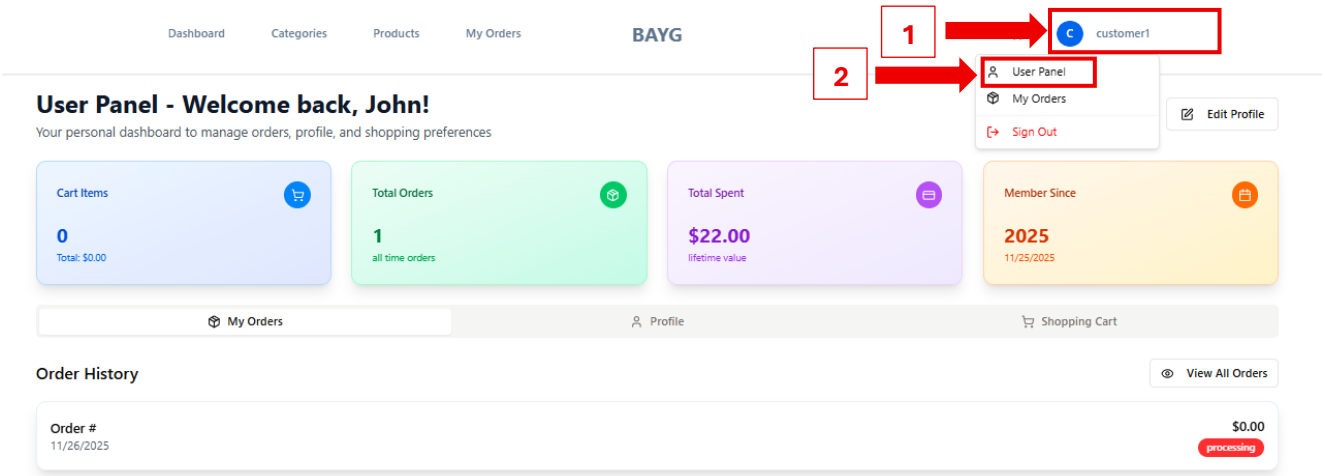


Figure 63 Client User Panel

Welcome Header and Profile Access

At the top of the dashboard, the user sees:

- A personalized greeting
- A short description explaining that this is their personal control panel
- A button labeled “Edit Profile”, which immediately takes them to their profile editing page

This header establishes a sense of familiarity while directing the user to essential account settings.

Summary Statistics Cards

Beneath the greeting, the user finds four colourful summary cards. These cards highlight key aspects of their account at a glance:

Cart Items

Shows:

- Number of items currently in the shopping cart
- The combined value of those items

Total Orders

Shows:

- Total number of orders placed by the user
- A small label indicating these represent all-time orders

Total Spent

Shows:

- The cumulative amount the user has spent on the platform
- Helps the user understand their lifetime purchasing value

Member Since

Shows:

- The year the user created their account
- The full join date underneath

These cards offer an immediate snapshot of the user's engagement and transaction history, presented in a visually engaging format.

Main Tabs for Navigation

Below the statistics is a tabbed section that organizes the user's information and actions into three areas:

a. My Orders

b. Profile

c. Shopping Cart

This structure helps the user quickly switch between different parts of their account without being overwhelmed.

My Orders Tab - Order History Overview

This tab displays the user's recent orders in a clean, card-based format.

What the User Sees

1) A small list of their **three most recent orders**
2) Each order card shows:
    a) Order number
    b) Date of purchase
    c) Total amount spent

d) The order's current status (e.g., delivered, pending, approved), shown using coloured badges

What the User Can Do

- Click "View All Orders" to see their full order history
- If they have no orders, a friendly empty-state message appears with a **"Start Shopping"** button

This page gives the user confidence that their purchases are recorded accurately and easy to revisit.

Profile Tab - Personal Information Overview

The Profile tab displays the user's main account information in a clear, readable format, including:

- Username
- Email address
- First and last name

Icons are used to visually highlight email and identification fields, making the information more approachable.

What the User Can Do

- Review their personal details
- Click the "Edit Profile" button to update their information

This tab ensures that users always have quick access to the information that identifies them on the platform.

Shopping Cart Tab - Recent Cart Items and Total

This tab shows the user a preview of their current shopping cart, limited to the first few items for quick browsing.

What the User Sees

1) A small list of up to three recent cart items, each showing:
   a) Product name
   b) Quantity selected
   c) Total cost for that item
2) A summary card showing the full cart total

If the cart is empty, the user sees an illustration with a message encouraging them to browse products.

What the User Can Do

- Click "View Full Cart" to open the full cart page
- Click "Start Shopping" if the cart is empty
- Review item totals before proceeding to checkout

This tab acts as a quick checkpoint for users to verify what they are planning to purchase.

Role-Aware Redirection

Although not visible to the user, the dashboard automatically redirects administrators and managers to the appropriate admin dashboard instead of this user panel. Regular users remain on this page.

This ensures that each user only sees pages relevant to their role and permissions.

Overall Purpose of the User Dashboard

The User Dashboard functions as a personalised space where users can:

- Monitor their order history
- Review their account details
- Track their shopping cart activity
- Access their profile and shopping tools instantly

Its structure supports efficiency, clarity, and user comfort, making the platform feel more organised and user-friendly.
Instead of searching through menus, users can find their most important information in one place, presented in a way that is visually appealing and easy to understand.

3.3 Categories Page

The Categories Page serves as the entry point for users who want to browse products by category. Its design centers on clarity, visual appeal, and ease of exploration, ensuring that users can quickly identify and navigate to the product collections they are interested in.

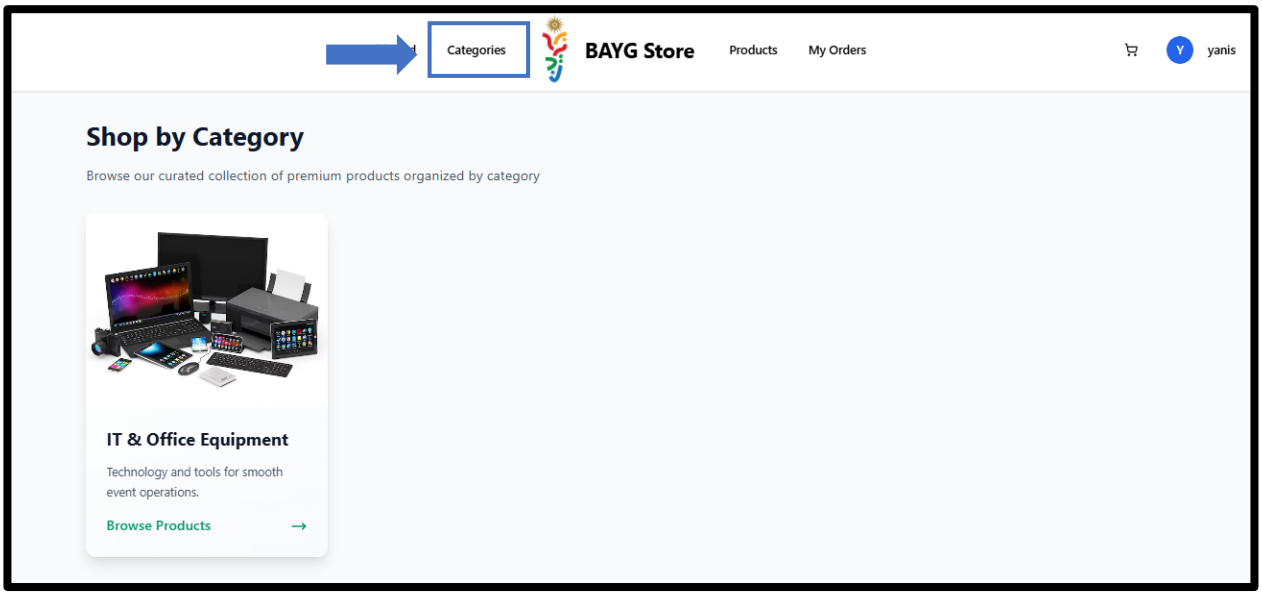


Figure 64 Categories section

When users arrive at the Categories Page, they are greeted with a clean and modern interface. At the top of the page, a heading titled “Shop by Category” introduces the purpose of the page, followed by a short description encouraging users to browse the available product collections.

The rest of the page is dedicated to displaying the list of categories in a responsive grid that adjusts seamlessly across mobile, tablet, and desktop screens.

Displaying Categories

Once the categories load, the user sees a grid of category cards. Each card is designed to be visually engaging and informative.

Each category card typically shows:

- An image representing the category
- The category name displayed below the image
- A short description summarizing the type of products in that category
- A small “Browse Products” indicator with an arrow

If a category has no image, the system automatically generates a colourful placeholder featuring the first letter of the category's name, ensuring the interface remains visually consistent.

The entire card responds to user interaction through subtle animations such as hover scaling and colour changes, making the page feel lively and polished.

What the User Can Do

The Categories Page is interactive and straightforward to use:

Click any category card to enter that category

When a user clicks a category card, they are taken directly to the product listing page dedicated to that category.
This allows them to begin browsing specific items immediately.

Explore the category descriptions

Even without clicking, users can read the short descriptions to understand what type of products are included in each category.

Navigate seamlessly across devices

The layout adapts automatically to different screen sizes, ensuring that users on mobile, tablet, or desktop can comfortably browse the available categories.

Empty State

If no categories have been added to the system yet, the user is shown a friendly empty-state message.

This includes:

- An icon illustration
- A title: "No Categories Available"
- A short explanation indicating that categories will appear once created by an administrator

This ensures that even in the absence of data, the interface remains informative and user-friendly.

Overall User Experience

The Categories Page provides a visually rich and intuitive browsing experience. Instead of forcing the user to search through menus, it presents all available categories as attractive, clickable cards. The transitions between states: loading, browsing, and empty view, are designed to feel smooth and professional.

This creates a welcoming and efficient shopping experience, guiding users naturally toward the products they want to explore.

3.4 Products Page

The Products Page is the core browsing interface where users explore the store's available items. It is designed to be flexible, intuitive, and responsive, allowing users to quickly find products of interest through search, category filtering, and direct browsing.

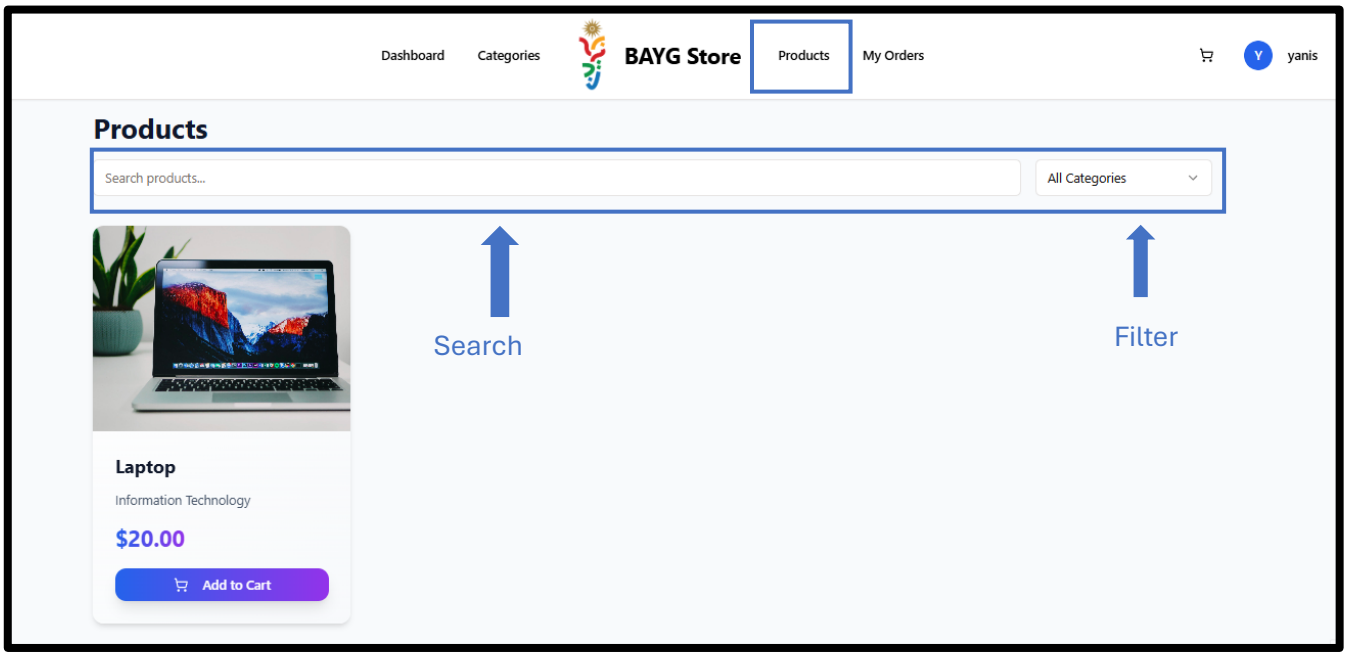


Figure 65 Products page

At the top of the page, users see a clear title **"Products"** or **"Electronics Products"** if a category filter is applied. This heading immediately communicates the scope of the results currently displayed. A helpful note appears alongside it when a filter is active, showing the user exactly which category is being used to refine the results.

This gives users a sense of control and clarity over what they are viewing.

Searching and Filtering

Just below the page header, the user is provided with a set of tools to narrow down the product list:

Search Bar

- Users can type keywords to search for products by name or description.
- When a category filter is active, the placeholder text changes (e.g., "Search in Electronics..."), reinforcing context.
- The search works instantly and updates the product list dynamically.

Category Filter Dropdown

- A dropdown menu allows the user to switch between "All Categories" and any specific category available.
- The dropdown is always visible, ensuring quick access to category filtering.

Together, these tools support efficient exploration, especially in stores with a large number of products.

Product Display Grid

Once the data loads, products appear in an organized grid layout. Each item is displayed through an interactive Product Card, which typically contains:

- A product image

- Product name
- Price
- A short description
- Icons or visual cues that maintain a consistent shopping experience

All cards are clickable, allowing users to open the detailed product page by selecting any item.

The design is responsive, meaning the number of columns adjusts depending on screen size, making the interface comfortable on mobile, tablet, and desktop devices.

What the User Can Do

On the Products Page, users can:

Browse all products visually

Scroll through the grid and visually explore items.

Search for items by name or description

Using the search bar, users can quickly narrow down results based on keywords.

Filter products by category

The dropdown menu allows switching between all categories or specific ones.

Combine search + category filtering

Users can refine their results very precisely (e.g., searching "chair" only inside the "Furniture" category).

Click a product card to view more details

Selecting any product sends the user to the product's dedicated page, where full details, pricing, and purchasing options are displayed.

Empty State (No Results Found)

If the filters or search query produce no matching results, the interface responds clearly:

- A centered message informs the user that nothing matches the current criteria.
- If no filters are applied at all, the message indicates that no products are available in the store yet.

This prevents confusion and helps guide the user back to modifying their search or filter settings.

Overall User Experience

The Products Page balances simplicity and flexibility. It supports casual browsing while also offering powerful search and filtering tools for users who know exactly what they are looking for. Through dynamic updates, clear feedback messages, and a visually consistent layout, the page provides a smooth, intuitive shopping experience that supports both discovery and decision-making.

3.5 Product Detail page

The Product Detail Page is the most comprehensive product screen in the system. It presents all information related to a specific item and allows users to interact with it; whether purchasing, renting, inspecting details, or adjusting order options. The page is structured to guide the user logically from general product information to specific configurations and finally to the purchasing action.

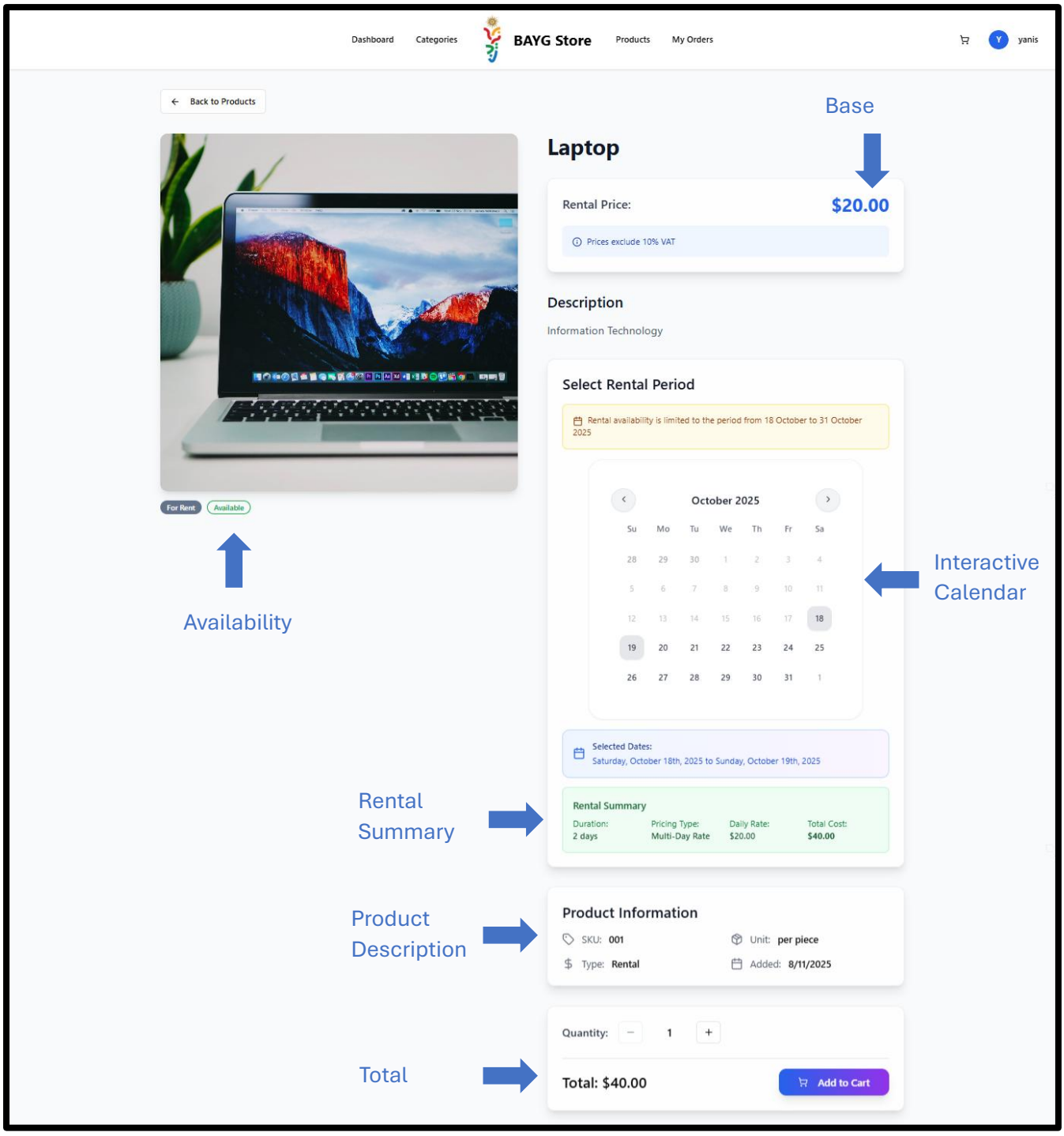


Figure 66 product detail page

At the top of the page, the user is offered a clear navigation option through a **"Back to Products"** button. This ensures that users can easily return to browsing without relying on browser controls. If the page loads and the product does not exist, a clear "Product Not Found" message appears, preventing confusion and maintaining a professional user experience.

Product Images and Visual Presentation

The left side of the page displays a large product image through a clean, centered container. If an image is unavailable, a placeholder graphic is shown instead. This ensures the user always has a visual context for the product they are viewing.

Just below the image, the page presents relevant badges, which may include:

- **Featured** - identifying premium or highlighted items
- **For Sale / For Rent** - indicating the product's type
- **Availability Badge** - reassuring users that the item is currently in stock

These visual markers help users quickly understand the product's category and purpose.

Core Product Information

The right-hand section of the top layout displays descriptive information about the item, including:

- **Product Name**, shown prominently as the main heading
- **Customer Rating**, represented visually through star icons with review counts (if available)
- **Price**, clearly labeled either as a purchase price or a rental daily rate

Sale-based products display a single price, while rental products specify the rate per rental period (e.g., *per day*). A short informational note underneath clearly states that prices exclude VAT, maintaining transparency.

A dedicated "Description" section provides a written overview of the item. If no description is available, the system communicates this explicitly

Rental Date Selection (For Rental Products Only)

For products that can be rented, the page introduces a detailed date-selection interface that guides the user through selecting a suitable rental period.

Key Features the User Sees

1) A highlighted notice explaining that rentals are limited to a specific date window (18–31 October 2025).
2) A toggle that allows switching between single-day rentals and multiple-day rentals.
3) Two date pickers:
    a) Start Date
    b) End Date (disabled automatically for single-day rentals)

The interface provides immediate validation feedback. If dates fall outside the allowed range or are selected in the wrong order, a red error box appears explaining the issue.

Automatic Rental Cost Display

Once valid dates are chosen, the system calculates:

- The number of rental days
- The daily rental price
- The total rental cost

This summary is shown in a green-highlighted box, reinforcing clarity and supporting informed user decisions.

Additional Product Information

Further down the page, a structured "Product Information" card presents the product's metadata:

- SKU
- Unit of Measurement
- Product Type (Sale/Rental)
- Date Added to the System

These details help users confirm that the product meets their technical or operational needs.

Quantity Selection and Pricing Summary

At the final stage of the page, a dedicated section allows users to adjust the quantity they wish to order. The interface provides:

1) Increment and decrement buttons, each visually distinct
2) A clearly displayed quantity counter
3) A dynamically updating **total price**, which recalculates based on:
    a) Quantity
    b) Selected rental period (if applicable)

This real-time updating ensures users always understand the financial aspect of their selection before committing.

Add to Cart Action

At the bottom of the page, a prominently styled "Add to Cart" button completes the interaction. When pressed, the system checks:

- Whether the user has selected valid rental dates (for rental items)
- Whether the quantity is acceptable

After successful validation, the item is added to the cart, and the user is redirected to their shopping cart to review their order.

A toast notification momentarily appears to confirm success, ensuring the user receives immediate feedback.

Overall User Experience

The Product Detail Page blends visual clarity, structured information, and guided interaction. For sale items, the ordering process is straightforward and focused on pricing and quantity. For rental items, the interface becomes more interactive, supporting date selection and cost breakdowns in a way that remains intuitive.

The user always understands:

- What the product is
- How much it costs
- Whether it is for sale or rent
- What dates are allowed for rentals
- How long the rental will last
- What the final cost will be

- What quantity they are ordering

In all cases, the interface ensures errors are clearly explained, selections are validated, and actions feel deliberate and informed.

3.6 Checkout Page

The Checkout Page represents the transition point between a user's shopping cart and the formal creation of their order. The interface is designed to guide the user through reviewing their selected items, providing their customer details, and submitting their order for administrative approval. If the order has already been created and the user is returning to complete payment, the page automatically switches to the relevant payment screen.

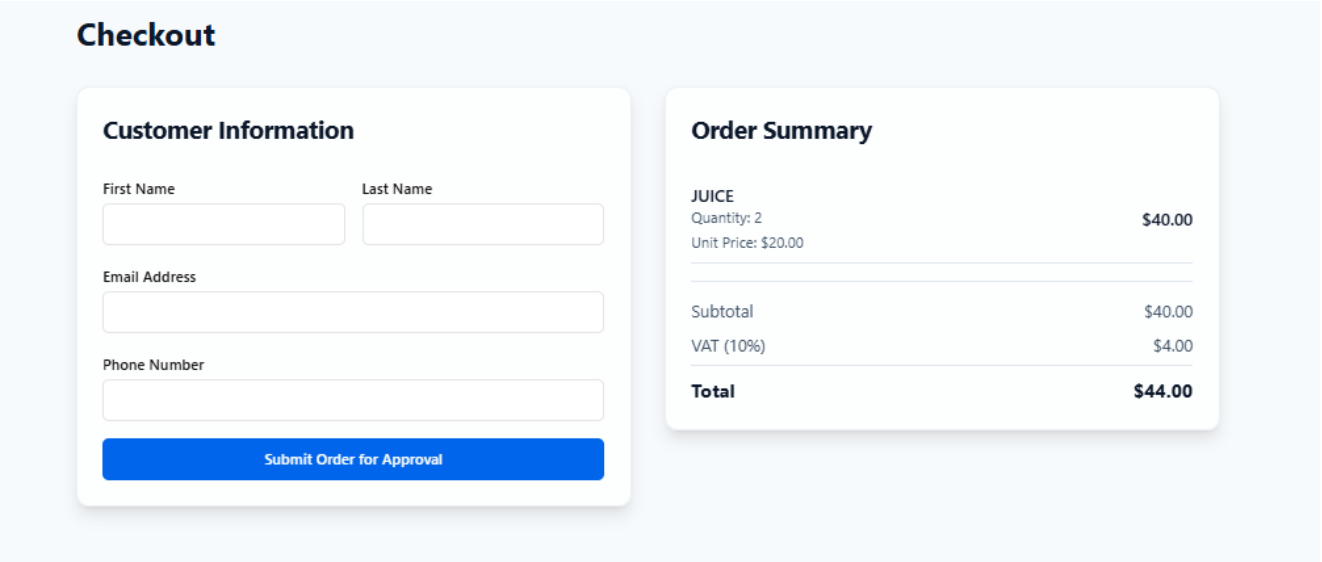


Figure 67 Payment Checkout

Empty Cart State

If a user navigates to checkout with no items in their cart, the system prevents progression and displays:

- A clear message stating that the cart is empty
- A short explanation
- A "Continue Shopping" button that redirects them to the product catalogue

This prevents accidental order creation and ensures users are aware of missing items before proceeding.

Checkout Form (Creating a New Order)

Users who have items in their cart are presented with a two-column layout:

Customer Information Form

This section collects the details required to create an order. The user fills in:

- First Name
- Last Name
- Email Address
- Phone Number

Each field provides instant validation feedback. If the user submits incomplete or incorrect information, the system displays error messages directly beneath the relevant fields.

Once all information is provided, the user selects "Submit Order for Approval."

What happens next

After submission:

- The order is created with a status indicating it is waiting for administrative approval
- A confirmation modal appears, informing the user that their order has been received successfully
- The user is guided to wait for approval before they can proceed to payment

The modal gives the user a clear and reassuring confirmation that the order has entered the approval pipeline.

Order Summary

The right column provides a detailed breakdown of the user's cart. For each item, the page displays:

- Product name
- Quantity
- Unit price
- Total price for that line item
- For rental products: start date, end date, and duration

This allows the user to verify everything before placing their order.

Beneath the itemized list, the system shows a summary of:

- Subtotal
- VAT (10%)
- Final Total Price

This gives complete transparency into how the total cost is calculated.

Order Approval Confirmation Modal

After a user submits their details:

- A modal window appears indicating the order was created successfully
- It informs the user that the order is now waiting for administrative approval
- Once the user closes the modal, the system automatically clears the cart and redirects them back to the products page

This workflow ensures the user cannot accidentally place duplicate orders.

Overall User Experience

The Checkout Page is structured to provide:

Clarity

Users see exactly what they are paying for and what information is required.

Guided Interaction

The page adapts automatically depending on whether the user is creating a new order or returning to pay for an existing one.

User Confidence

Immediate feedback through validation messages, modals, and summaries helps reassure users that their actions are correctly processed.

Error Prevention

Safeguards such as empty-cart blocking, required form fields, and post-submission confirmation prevent accidental or incomplete order submissions.

3.7 Payment Checkout page

The Order Payment Checkout page is the final stage in the purchasing workflow. It provides users with a complete summary of their order and allows them to securely finalize their payment. The page is carefully structured to give clarity, verify eligibility, and guide users through the last step of the transaction.

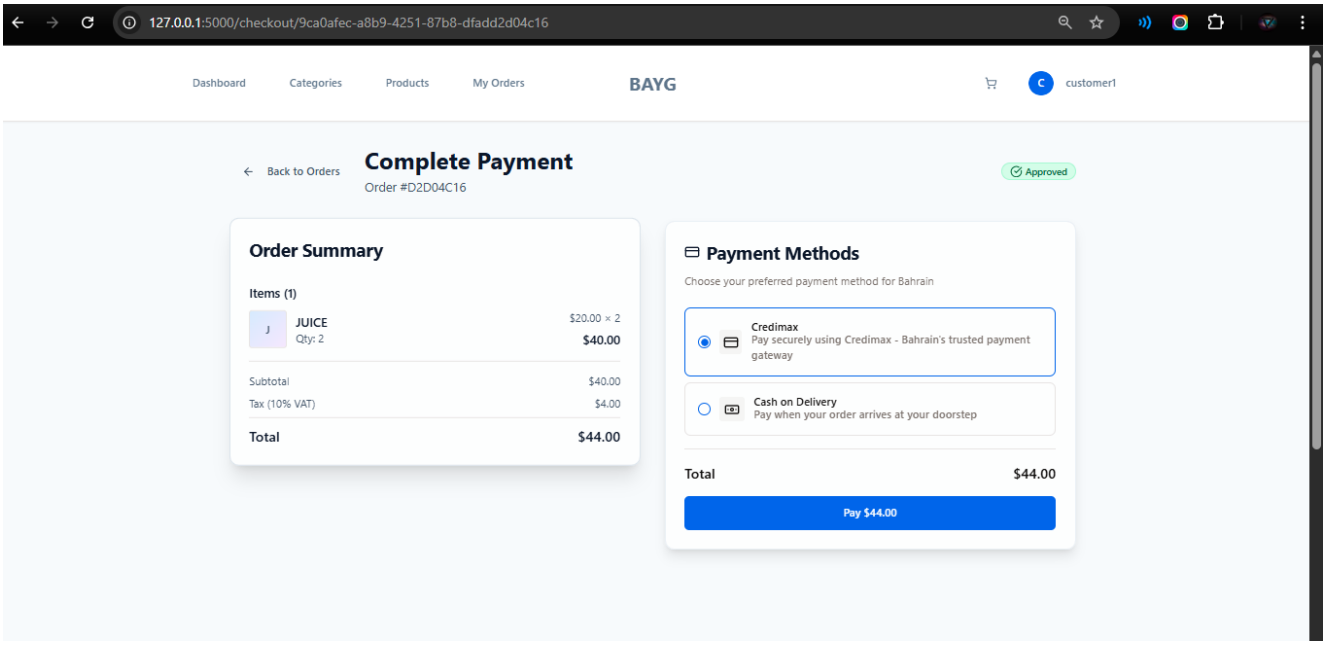


Figure 68 Payment Checkout Page

Payment-Ready State

If the order is approved and eligible for payment, the main checkout interface is displayed. It is divided into two main sections:

Order Summary Panel

This section presents a clearly organized breakdown of the user's order. The purpose is to verify accuracy before completing payment.

Header Information

- Page Title: "Complete Payment"
- Order Number: Displayed under the title for quick reference
- Approval Badge: A green "Approved" badge reassures the user that the order has passed internal checks and is ready for payment

Itemized Order List

Each product included in the order is shown with:

- A product thumbnail (or a fallback placeholder if no image exists)
- The product name
- Quantity ordered
- For rentals, the selected rental period and total rental days
- The unit price and total price for that line item

This ensures that both regular and rental products are presented with full clarity.

Pricing Breakdown

Below the item list, the system provides a structured summary:

- Subtotal
- 10% VAT calculation
- Final Total displayed prominently

These costs are itemized to ensure users understand exactly how the total was derived.

Payment Section (Right Side)

The right column is dedicated entirely to conducting the payment.

Supported Payment Methods

The system integrates Bahrain-specific payment options, which include:

- Credit and debit card payments
- Cash-on-delivery
- Any other digital payment method supported by Credimax

The payment module is interactive and guides the user through entering payment details securely.

IMPORTANT

The CrediMax payment option WILL NOT WORK at the time of the review of the thesis, due to not being provided with the necessary information to interact with the CrediMax API (e.g. Merchant ID, Merchant credentials, API KEY, etc..)

Payment Feedback

After attempting payment:

- Successful payments trigger a confirmation toast and automatically redirect the user back to their Orders page.
- Failed payments present an immediate error message instructing the user to retry or contact support.

This real-time feedback helps prevent uncertainty and ensures users always know the outcome of their transaction.

Navigation and User Control

Throughout the checkout page, a “Back to Orders” button is always available, allowing users to exit the process if they need to review or modify their orders.

Overall User Experience

The Order Payment Checkout page is designed around three principles:

Clarity

Users are given complete transparency:

- What items they are paying for
- How much the order costs
- Whether the order is approved
- What payment method they are using

Security

Users are only allowed to pay when the system confirms the order is valid and approved. All interaction feedback is immediate and visible.

Convenience

A clean two-column layout separates review from action, allowing users to focus on payment without losing sight of their order details.

Appendix II: Detailed Design

This appendix presents the complete detailed design of the ICT solution developed for the Bahrain Asian Youth Games (BAYG). While the main thesis outlines the high-level architecture and design approach, this appendix provides the full technical specifications, diagrams, and structural representations required to fully understand how the system was engineered.

The detailed design focuses on the internal components of the BAYG systems, including the architecture of the RC-BAYG platform, the PERN-based web systems, the device-deployment automation workflows, and the infrastructure required to support multi-venue operations. It explains how each subsystem interacts, how data flows between components, and how the design aligns with the operational requirements of the Games.

This section includes:

- System architecture diagrams
- Data flow representations
- Database structure and entity relationships
- API design and endpoint structure
- Frontend component design and state-management logic
- Backend logic, routing structure, and middleware flow
- Automation workflow diagrams and script logic
- Network and device configuration designs, where applicable

The purpose of this appendix is to provide full design transparency, ensuring the implementation can be understood, evaluated, maintained, and extended by future teams. It also serves as documentation to demonstrate how the design choices meet BAYG's requirements for scalability, reliability, integration with vendor systems, and compliance with operational standards set by the Bahrain Olympic Committee and BAYG management.

High-Level System Architecture and Data Flow of the RC BAYG Web Platform

This diagram illustrates the overall architectural structure of the BAYG web application, showing how the client-side interface, server-side logic, and database layer interact to deliver a seamless and reliable user experience. The architecture follows a modular, layered design built on modern web development technologies, ensuring scalability, maintainability, and efficient data flow across all components.

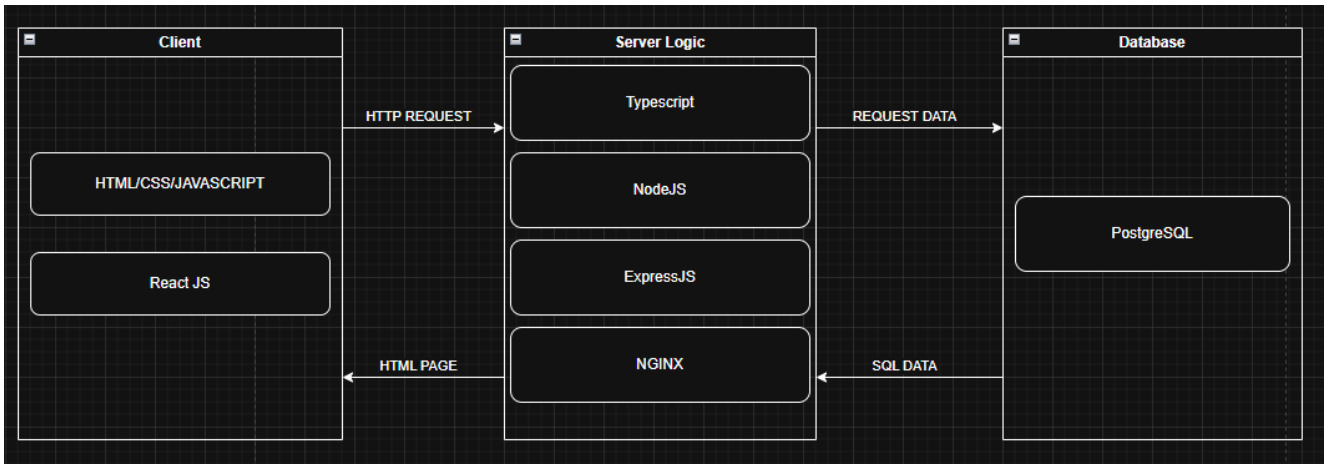


Figure 69 System Architecture and Data Flow

Analysis

Client Layer

The client layer consists of the HTML/CSS/JavaScript foundation supported by the React.js frontend framework.
This layer is responsible for:

- Rendering the user interface
- Managing client-side state
- Handling user interactions
- Sending HTTP requests to the server (e.g., placing an order, retrieving items, updating forms)

React's component-based structure allows the platform to dynamically update content without full page reloads, improving responsiveness and user experience. The client communicates with the server exclusively through RESTful API calls.

Server Logic Layer

The server logic is implemented using a multi-tier Node.js ecosystem composed of:

- TypeScript adds static typing, improving reliability, debugging, and large-scale maintainability
- Node.js provides the runtime environment for executing server-side JavaScript
- Express.js handles routing, API endpoints, middleware functions, validation, and business logic
- NGINX acts as a reverse proxy, load balancer, and static file server, optimizing performance and security

This layer processes all incoming HTTP requests, validates user input, applies business rules, interacts with the database, and returns structured responses to the client.

It effectively acts as the “core brain” of the system, transforming raw data into meaningful application behavior.

Database Layer

The backend uses PostgreSQL as the primary database engine.
This layer is responsible for:

- Storing structured application data (orders, users, items, categories, payments)
- Maintaining data integrity through relational constraints
- Executing SQL queries generated by the server
- Ensuring secure and optimized data retrieval

The system follows a transactional approach, ensuring that operations such as submitting orders or processing payments are reliable and reversible if needed.

Data Flow Summary

The interaction between layers follows a clear and predictable pattern:

1. The client sends an HTTP request to the backend via React.js.
2. The server logic (TypeScript + Node.js + Express) interprets the request, processes it, and prepares a database query.
3. The PostgreSQL database returns the relevant SQL data back to the server.
4. NGINX manages the delivery of responses and static assets back to the client.
5. The client receives an HTML/JSON response and updates the UI accordingly.

This creates a streamlined, secure, and high-performance architecture capable of supporting the operational requirements of the Bahrain Asian Youth Games.

Entity - Relationship Diagram of RC BAYG

The Entity–Relationship Diagram (ERD) represents the complete logical data model used by the BAYG e-commerce and rate-card platform. It shows how core business entities such as users, products, orders, and permissions are structured and how they interact through primary and foreign key relationships. The schema has been deliberately normalized into clear domains: user and access control, catalog and inventory, cart and order processing, and site configuration/content, to support maintainability, data integrity, and the operational requirements of the Bahrain Asian Youth Games.

TO VIEW THE ERD VISIT THIS LINK AND DOWNLOAD THE IMAGE:
https://drive.google.com/file/d/1stleFP4LbmU5SFma1pnRAI9D3-JkbYJ_/view?usp=sharing

User, Role, and Permission Domain

At the heart of the access-control model is the users table. It stores authentication and identity information (email, username, password, first_name, last_name) as well as flags such as is_admin and is_super_admin, which distinguish privileged accounts (e.g., IT administrators) from standard operational users. The role_id column links each user to a specific entry in the roles table, forming the first level of role-based access control (RBAC). This allows BAYG to assign different capabilities to groups such as logistics staff, finance officers, and system administrators without hard-coding permissions into the application.

The roles table defines these high-level roles, including metadata such as name, display_name, description, and operational flags like is_system_role and is_active. System roles can be protected from accidental modification, while the active flag allows roles to be temporarily disabled without deleting data, which is useful when restructuring responsibilities during different phases of the Games.

To support fine-grained authorization, the model introduces permission_modules, permissions, and role_permissions. permission_modules groups permissions into logical functional areas (for example, “Product Management”, “Order Approvals”, “Site Settings”), storing fields like name, display_name, icon, sort_order, and is_active. The permissions table then defines specific actions within those modules (such as “create product”, “approve order”, “edit slider images”) with attributes like name, display_name, action, and module_id. The foreign key module_id ties each permission back to a module, ensuring permissions are organized in a way that is intuitive for administrators configuring access.

The role_permissions junction table implements a many-to-many relationship between roles and permissions: each record maps one role_id to one permission_id, with timestamps such as created_at for auditing. This design allows the system to answer questions like “What can this role do?” and “Which roles can perform this action?” in a single join, making permission checks efficient and maintainable. Overall, this domain gives the platform a flexible security model that can adapt to BAYG’s organisational hierarchy and separation of duties (e.g., only certain roles can approve large-value orders or change site-wide settings).

Product, Category, and Measurement Domain

The categories and products tables model BAYG’s rate-card catalog; essentially all assets that can be rented or ordered by functional areas during the Games (e.g., laptops, printers, furniture, connectivity options).

The categories table provides top-level grouping: fields such as name, description, image_url, and created_at allow the IT and marketing teams to structure the catalog into intuitive sections, link them to imagery consistent with BAYG branding, and track when categories were introduced.

The products table stores detailed information about individual items. Core attributes include name, description, sku, image_url, and category_id (FK -> categories.id). Financial and rental behaviour is captured through fields such as price, rental_price, rental_period, product_type, unit_of_measure, rating,

review_count, is_featured, is_active, and created_at. This design supports both one-off purchases and time-based rentals, which is critical in a Games context where many items are rented per day or per event. Linking each product to a category via the category_id foreign key enforces consistency and allows the system to build category-based navigation and reporting.

Supporting this is the units_of_measure table, which standardizes the measurement units used across the platform (e.g., “per day”, “per event”, “per device”). Each unit has an id, name, abbreviation, is_active, and created_at. While products store the chosen unit in unit_of_measure, the separate table makes it easy to manage a controlled vocabulary and prevents inconsistencies such as different spellings for the same unit.

Together, these tables form a flexible catalog domain that can be updated rapidly throughout the Games lifecycle, while still preserving historical data for auditing and post-event analysis.

Cart and Order Processing Domain

The cart_items, orders, and order_items tables implement the full e-commerce flow: from items being added to a cart, through to confirmed orders and line-item level details.

cart_items acts as a temporary staging area for users building their requests. Each row links to both a user (user_id -> users.id) and a product (product_id -> products.id), and records commercial details such as unit_price, total_price, quantity, and temporal attributes like rental_start_date, rental_end_date, and created_at. These columns allow the system to calculate rental cost based on duration and quantity, and to pre-validate that orders align with event timelines (e.g., equipment needed only during competition days).

When a user submits their cart, its contents are transformed into one or more orders. The orders table represents the header of each transaction and includes financial fields (subtotal, tax, vat_percentage, total), payment integration fields (payment_intent_id, payment_method), and workflow/approval information (status, admin_approval_status, admin_remarks, admin_approved_at, admin_approved_by). The user_id foreign key associates each order with the requesting user, while admin_approved_by (FK -> users.id) identifies the staff member responsible for final approval. This is particularly important in BAYG’s context, where many orders represent internal cost allocations rather than public consumer purchases and must go through supervisory vetting.

The order_items table stores the line items belonging to each order. Each record links to an order (order_id → orders.id) and a product (product_id -> products.id) and contains item-specific data including price, quantity, total_price, rental_start_date, rental_end_date, rental_days, and created_at. This separation between order header and items follows standard e-commerce design and allows the system to:

1. handle orders containing multiple different products,
2. compute totals and rental periods per line, and
3. generate detailed invoices or reports for specific categories or venues.

Overall, this domain captures a complete transactional history, enabling both operational tracking during the Games and post-event financial reconciliation.

Site Configuration and Content Domain

The platform includes tables dedicated to presentation and configuration so that non-technical staff can adjust content and behaviour without modifying the code.

The site_settings table is effectively a key–value configuration hub but modeled in a structured way. It stores a wide range of attributes, including branding (site_name, browser_tab_title, header_logo, footer_logo, logo_url, favicon_url, colour fields such as theme, text_color, header_text_color,

tab_text_color, tab_active_text_color), layout options (footer_background_url, payment_methods_image, widths and heights for logos and images), communication details (contact_email, contact_phone, contact_address, support_email, admin_email, business_hours, office_hours_title), and even email templates (order_confirmation_template) and SMTP configuration (smtp_enabled, smtp_host, smtp_port, smtp_secure, smtp_user, smtp_password, smtp_from_name, smtp_from_email). Social media links (Facebook, Twitter, LinkedIn, Instagram) and textual content such as quick links and services descriptions are also stored here, along with timestamps created_at and updated_at.

This design means that changes to branding (for example, aligning with BAYG and Olympic Council of Asia guidelines), communication channels, or email behaviour can be made directly via the admin UI and reflected across the public site, without redeploying the application.

The slider_images table controls visual content for the homepage hero/slider section. It contains fields such as title, description, image_url, sort_order, is_active, created_at, and updated_at. This allows the marketing or communications team to schedule and reorder banners (for example, promoting specific venues or announcements) while maintaining a record of past campaigns.

These configuration tables are intentionally independent of transactional data to avoid coupling long-lived configuration with short-lived order and cart records, improving maintainability and simplifying backup strategies.

Referential Integrity, Normalization, and Design Rationale

The ERD enforces relationships using foreign keys between tables such as orders.user_id -> users.id, products.category_id -> categories.id, cart_items.product_id -> products.id, and role_permissions.role_id -> roles.id, among others. This prevents orphan records, e.g. an order cannot exist for a non-existent user, and an assigned permission must always reference a valid permission record. These constraints are essential in a multi-user operational environment like BAYG, where many concurrent users from different departments interact with the system.

The schema is normalized to reduce redundancy: product attributes are stored once in products and reused through foreign keys in cart_items and order_items; role and permission metadata is centralized in roles and permissions, while role_permissions only stores links between them. At the same time, some denormalization is applied in a controlled way when it improves performance or clarity (e.g., storing total_price on both cart_items and order_items to avoid recalculating it repeatedly during reporting).

From a wider system perspective, this ERD supports key BAYG use cases:

1. internal stakeholders browsing a categorized rate card of products,
2. building carts for specific time windows,
3. submitting orders that go through multi-level approval,
4. enforcing who can approve or modify what via the role/permission system, and
5. updating branding, notifications, and visual content in line with BAYG and OCA standards.

In summary, the ERD represents a well-structured, production-ready data model that balances flexibility, security, and operational control, making it suitable for managing the ICT and equipment-rental aspects of a large multi-sport event like the Bahrain Asian Youth Games.

Wireframe

The wireframe diagram illustrates the complete user flow and interface structure of the BAYG E-commerce & Rental platform, covering both client-side and admin-side interactions. The flow begins with the authentication module, where all users first encounter the login form. Depending on the user's role: client, admin, or super admin, the system redirects them to different homepages. Super admins access a more advanced dashboard with administrative control tabs, while regular clients are directed to the client homepage.

LINK TO WIREFRAME:

https://drive.google.com/file/d/1eYDMT6BqVV2aHDAZXdyJE632sOnukTkC/view?usp=sharing

Client Flow

From the client homepage, users can navigate through the main sections: Categories, Products, Orders, and Cart, all visible from the shared navigation bar. The categories page displays all available product groups, and selecting one transitions the user into the products listing page, which then leads to the product detail interface. The product detail page includes images, product information, and an "Add to Cart" option, completing the standard browsing and shopping cycle.

The cart page is shown as a combination of two primary components: a client information form and an order summary. Once the user proceeds, the system leads to the checkout page, where they choose a payment method. If Credimax is selected, the flow moves to an external Credimax payment page and returns to the platform upon completion. A successful payment redirects the user back to the order section, where they can monitor order status.

Superadmin Flow

On the administrative side, the diagram shows a parallel navigation structure, but with additional options such as Admin Panel, Categories Management, Product Management, and Dashboard. Super admins are shown accessing the Admin Homepage, which contains admin action tabs and UI elements for managing system operations. This includes actions like approving orders, updating products, or configuring site settings. After completing administrative tasks, the user can return to the super admin homepage.

Overall, the wireframe provides a high-level visualization of the system's UX layout, navigation flow, and separation between client and administrative user journeys. It demonstrates how each page is interconnected and how different user roles move through the system.

Appendix III: Detailed Implementation

This appendix presents a comprehensive breakdown of the RC-BAYG platform's implementation, detailing the structure of the source code, runtime directories, backend logic, frontend components, and deployment assets. The goal is to demonstrate how the project's architecture reflects principles of modularity, scalability, and maintainability. Each section describes the technical role, purpose, and functional responsibilities of the corresponding files and folders, showing how the system supports the operational requirements of the Bahrain Asian Youth Games.

1.Detailed Explanation of the Project Structure

The RC-BAYG project is structured according to modern full-stack engineering principles, ensuring a clear separation between application layers, shared logic, infrastructure components, and runtime assets. This structure is neither arbitrary nor stylistic; every directory and file serves an architectural purpose that supports maintainability, scalability, and operational reproducibility. The project's top-level layout establishes the foundation for how the system behaves under development and production conditions, while also providing transparency and predictability to future maintainers, auditors, or technical teams that may need to work with the system.

Root Project Structure

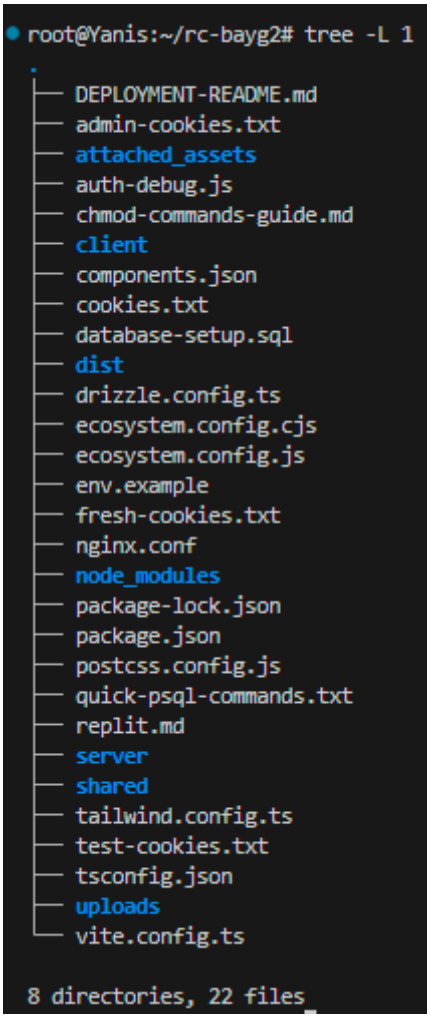


Figure 70 root project structure

The project is organized according to a three-tier architecture, separating the frontend, backend, and shared logic. Infrastructure and deployment-related files are stored at the root of the repository, ensuring reproducibility across environments.

Deployment Documentation and Operational Files

The DEPLOYMENT-README.md file documents the end-to-end deployment pipeline, including server configuration, build instructions, NGINX setup, and PM2 process management. It ensures that any developer or DevOps engineer can reproduce the production environment identically.

Files such as admin-cookies.txt, cookies.txt, fresh-cookies.txt, and test-cookies.txt contain saved HTTP session cookies used by the developer to test authenticated endpoints. Their purpose is diagnostic; they simulate logged-in sessions during testing, allowing protected routes to be checked without repeating the login process.

The attached_assets directory stores supporting artifacts that do not belong to the versioned source code, such as debug exports, logs, or auxiliary data used during development. These assets assist in troubleshooting issues, recording behavior, or supporting temporary development tasks.

Database Initialization and Reproducibility Files

The database-setup.sql script defines the initial PostgreSQL database creation process. It creates the bayg_production database, the associated PostgreSQL user (bayg_user), and assigns the necessary privileges. Its purpose is to ensure consistent database initialization on any new environment, enforcing reproducibility and reducing deployment errors.

Production Build Output and Runtime Data

The dist directory contains the compiled, production-ready JavaScript output of the backend server after TypeScript compilation. This folder represents the executable form of the backend, specifically the code that PM2 launches in production, separating source from deployment output.

Build System and ORM Configuration Files

Files such as drizzle.config.ts, tailwind.config.ts, vite.config.ts, postcss.config.js, and tsconfig.json define the build behavior, TypeScript compilation rules, CSS processing pipeline, and ORM migration settings. These configuration files govern how the system is built, ensuring deterministic behavior.

Runtime folders such as uploads contain dynamically uploaded content (slider images, product images) generated by administrators during system usage. They are intentionally separated from source code to avoid polluting version control and to enable backup strategies.

Backend Implementation Structure

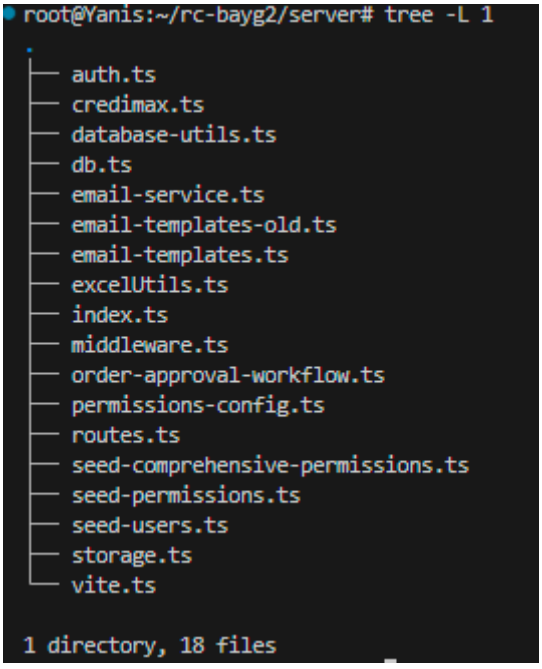


Figure 71 Server folder structure

The backend implements all business logic, authentication, authorization, payment processing, email workflows, database interactions, and API endpoints. It follows a layered architectural approach, ensuring modularity and testability.

Backend Entry Point and Application Bootstrap

The index.ts file acts as the backend's entrypoint. It constructs the Express application, registers middleware (JSON parsing, cookies, sessions, and Passport.js), initializes the PostgreSQL database connection, and mounts all API routes. It is responsible for launching the server and represents the composition root of backend logic.

Authentication and Session Management

Authentication is implemented in auth.ts, which defines the Passport.js local strategy for verifying user credentials. Its purpose is to authenticate users securely and maintain session state across requests. It also serializes and deserializes user sessions into cookies, enabling persistent login functionality. Authentication guards defined here ensure that only legitimate users can access protected features.

Authorization and Permission Enforcement

middleware.ts complements authentication by enforcing authorization logic. It provides reusable middleware functions such as ensureAuthenticated and role-based access checks. Its purpose is to validate user permissions on each request, preventing unauthorized access to administrative or sensitive functionalities.

The system's authorization policy is defined in permissions-config.ts, which maps system roles; such as super_admin, manager, and user to specific permissions. This file does not contain executable logic; rather, it encodes security rules, acting as the policy definition layer.

Database Connectivity and Data Access Layer

Database connectivity is established in db.ts, where Drizzle ORM creates a typed connection to PostgreSQL. The purpose of this module is to expose a consistent and type-safe interface for database interaction. Higher-level helpers are implemented in database-utils.ts, which provides reusable transactional patterns and utility functions frequently required by queries.

The primary data access layer resides in storage.ts, which implements the repository pattern. This module abstracts direct database operations behind human-readable functions such as createOrder, getCartItems, getUserByEmail, or createOrderItem. Its purpose is to isolate SQL and ORM-specific details from business logic modules, reducing duplication and improving maintainability.

API Routing and Request Handling

All HTTP API endpoints are registered in routes.ts. Its purpose is to define how incoming HTTP requests map to controllers and logic functions. This file structures routes related to products, categories, orders, carts, authentication, settings, and administrative operations.

Business Workflow Logic

Business workflows- specifically the multi-step order approval process, are implemented in order-approval-workflow.ts. This module encodes the rules governing how orders transition between states, such as pending, approved, and rejected. Its purpose is to formalize domain logic into a predictable, testable sequence of operations.

External Integrations: Payment and Email

Integration with the CrediMax payment gateway is implemented inside credimax.ts. The module constructs payment sessions, handles API signatures or tokens, validates callbacks, and forwards results to the client. Its purpose is to abstract external payment communication and ensure secure transaction processing.

Email functionality is implemented using email-service.ts (transport logic) and email-templates.ts (rendering HTML templates). Their purpose is to generate and send automated emails for events such as order submission, approval, or administrative notifications.

Database Seeding and Initialization Scripts

Finally, database seed scripts; seed-users.ts, seed-permissions.ts, and seed-comprehensive-permissions.ts populate the system with baseline accounts, roles, and permissions. Their purpose is to allow any fresh deployment of the system to be initialized consistently and reliably.

Frontend Structure

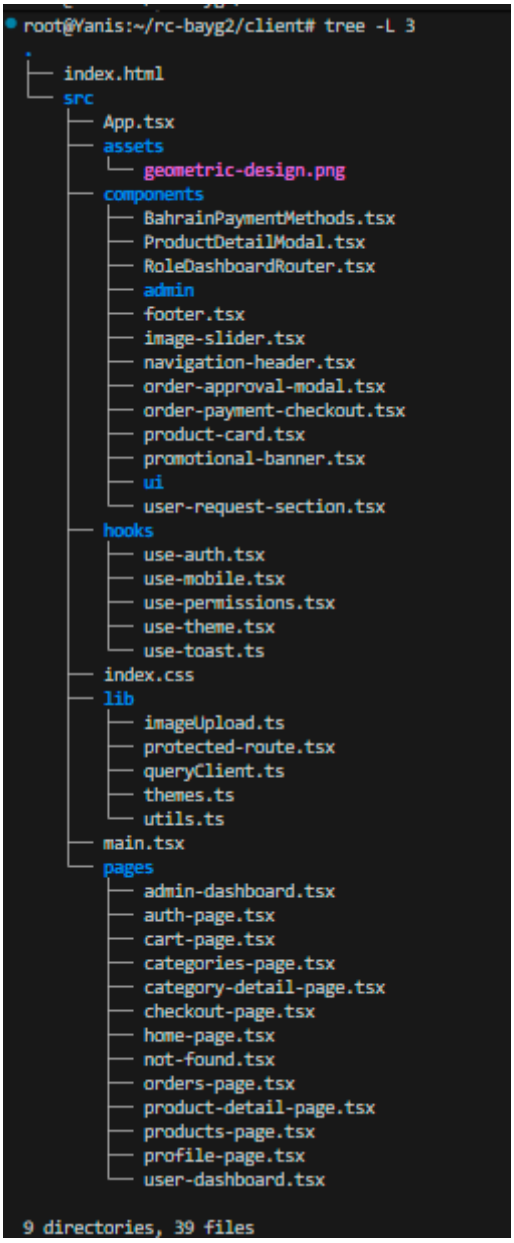


Figure 72 Frontend File Structure

The frontend is a modern React + Vite single-page application (SPA), designed to provide responsive and interactive user experiences for both customers and administrators.

Initialization and Global Providers

The application begins with index.html, which serves as the host container for the SPA. It contains the root into which the entire React application is rendered. Its purpose is to provide a minimal document structure while delegating all dynamic behaviour to JavaScript.

The main.tsx file initializes the React runtime, mounting the root App component and configuring global context providers such as React Query, authentication state, theming, and toast notifications. Its purpose is to prepare the application for rendering by injecting global dependencies.

Application Root and Routing Logic

The App.tsx component is the logical root of the UI. It defines the application's routing logic, loads global settings from the backend, applies themes, and orchestrates high-level navigation. Its purpose is to coordinate the entire frontend environment.

Page-Level Components

The pages/ directory contains the top-level views of the application. Each file corresponds to a distinct screen, such as the home page, product catalogue, product detail view, category pages, checkout workflow, login page, user profile, order history, and the administrative dashboard. These pages integrate reusable components and communicate with the backend through React Query.

Reusable and Domain-Specific Components

The components/ directory contains reusable UI elements and domain-specific features. Standard UI components like the navigation header, footer, product card, and promotional banner provide consistency across pages. Domain-specific components include the order approval modal, payment checkout handler, image slider, and role routing handler. Their purpose is to modularize the UI so that changes to one area do not impact unrelated parts of the application.

Stateful Logic Through Custom Hooks

The hooks/ directory contains custom React hooks that encapsulate stateful logic. These include authentication state management (use-auth), permission checks (use-permissions), toast notifications (use-toast), theming (use-theme), and mobile responsiveness detection (use-mobile). These hooks separate behaviour logic from presentation components, improving maintainability and reusability.

Frontend Utility Modules

The lib/ directory expands the functionality with supporting utilities, such as image upload helpers, route protection logic, formatting utilities, and theme definitions. Its purpose is to centralize cross-cutting logic that would otherwise lead to duplication across components or pages.

The assets/ directory holds static files (e.g. images) used by the interface, and index.css defines global styling and imports Tailwind CSS directives.

This demonstrates that the RC-BAYG system is built upon a disciplined and modular architecture. Every file and directory serves a deliberate purpose within the system, from database accessibility to component reusability and production deployment stability. The structured organization makes the platform highly maintainable and scalable, meeting the stringent requirements of an event-driven environment such as the Bahrain Asian Youth Games.

3. RC-Bayg Source Code Explanation and Analysis

This Section provides a detailed technical explanation of the core source code that forms the RC-BAYG E-Commerce and Rate-Card Management Platform. While the complete system consists of many functions, components, hooks, and modules across both the frontend and backend, this section focuses on analysing the most important and functionally significant parts of the codebase. Explaining every line of code would be unnecessarily extensive for the scope of this thesis; therefore, emphasis is placed on the modules that govern authentication, order workflows, payment integration, API routing, database access, and key user interface components.

The objective of this appendix is not only to describe what the code does, but to explain why it is structured in a particular way, how the different modules interact, and how the implementation reflects software engineering principles such as modularity, separation of concerns, maintainability, and security. Each selected code section will be discussed using clear technical reasoning, demonstrating how it fulfills its intended role within the system's architecture and contributes to the overall performance and reliability of the application.

In the following sections, the backend code will be examined first, covering authentication logic, permission validation, payment gateway integration, and business workflow handling. This will be followed by an analysis of the frontend code, highlighting the components, hooks, and state management strategies that enable the platform's user experience. By focusing on the most essential and impactful portions of the system, this appendix provides a concise yet comprehensive understanding of how the RC-BAYG platform operates internally.

2.1 Server Entrypoint Module (index.ts)

The index.ts file serves as the central entrypoint of the RC-BAYG backend system. It is responsible for configuring middleware, initializing the database, seeding essential system data, registering API routes, preparing development and production server environments, and ultimately launching the HTTP server instance that powers the entire platform. Because all backend features, from authentication to order management, depend on the server's correct initialization, this file plays a critical architectural role in ensuring the stability, reliability, and operational consistency of the system.

Environment Configuration and Express Initialization

The module begins by importing environment variables using dotenv/config, ensuring that all subsequent components have access to sensitive configuration values such as database credentials and session secrets. The Express application is then created, along with middleware for parsing JSON and URL-encoded bodies.

```
import 'dotenv/config';

import express, { type Request, Response, NextFunction } from "express";
import { registerRoutes } from "./routes";
import { setupVite, serveStatic, log } from "./vite";
import { seedUsers } from "./seed-users";

const app = express();
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
```

By loading .env variables at the earliest possible stage, the system guarantees that every downstream module: session handlers, database clients, authentication components, and Vite configuration, operates with the correct environment-dependent configuration.

The Express instance is initialized as the foundation of the backend. JSON and URL-encoded middleware allow the server to interpret API requests consistently, ensuring that all incoming data (such as login credentials or order submission payloads) is validated and parsed into accessible request fields. This standardization creates a reliable communication contract between the frontend and backend layers.

High-Resolution API Logging Middleware

Before registering routes, the entrypoint defines a custom logging middleware that measures the duration of every API call, captures JSON responses, and outputs summarized logs. This middleware plays a crucial role in debugging, monitoring performance, and auditing the behavior of deployed systems.

```
app.use((req, res, next) => {
  const start = Date.now();
  const path = req.path;
  let capturedJsonResponse: Record<string, any> | undefined = undefined;

  const originalResJson = res.json;
  res.json = function (bodyJson, ...args) {
    capturedJsonResponse = bodyJson;
    return originalResJson.apply(res, [bodyJson, ...args]);
  };

  res.on("finish", () => {
    const duration = Date.now() - start;
    if (path.startsWith("/api")) {
      let logLine = `${req.method} ${path} ${res.statusCode} in ${duration}ms`;
      if (capturedJsonResponse) {
        logLine += ` :: ${JSON.stringify(capturedJsonResponse)}`;
      }

      if (logLine.length > 80) {
        logLine = logLine.slice(0, 79) + "…";
      }

      log(logLine);
    }
  });

  next();
});
```

Figure 73 middleware definition

This middleware performs several sophisticated tasks:

Capturing response bodies
By overriding res.json, the middleware intercepts the payload before it is sent to the client. This allows logging without disrupting the normal response flow.

Timing execution
The difference between the start and finish timestamps provides precise latency metrics for each API call. This is essential for identifying slow queries or inefficient endpoints.

Structured request auditing
Logging includes the HTTP method, route path, status code, execution time, and truncated JSON response.
This is invaluable for:

a) debugging

b) performance monitoring

c) diagnosing production issues

d) tracing user interactions during BAYG operations

Selective logging
Only API routes (/api/...) are logged, preventing noise from static file requests in production.

This design demonstrates strong attention to observability, a critical requirement in real-world deployments.

Database Initialization and Health Check

The system performs a synchronous startup procedure inside an immediately invoked async function. The first major step is verifying that the database connection is operational.

```
// Test database connection first
const { db } = await import("./db");
await db.execute('SELECT 1 as test');
log("Database connection successful");
```

Figure 74 Connect to database and test connection

Rather than assuming the database is reachable, the entrypoint actively executes a test query (SELECT 1).
This has several important benefits:

- Ensures that the server does not proceed if the database is offline
- Prevents silent failure halfway through route initialization
- Guarantees that all features relying on Drizzle ORM can operate correctly
- Provides clear diagnostics during deployment, containerization, or VM-based hosting

A failing database check terminates the startup process early, helping maintain system integrity.

Seeding the Permission System and User Accounts

Before enabling routes, the backend seeds the role and permission system, followed by predefined administrative users.

```
// Seed comprehensive permission system first (creates roles)
const { seedComprehensivePermissions } = await import("./seed-comprehensive-permissions");
await seedComprehensivePermissions();

// Seed predefined user accounts after roles are created
await seedUsers();
```

Figure 75 seed users and User accounts

The ordering of seeding operations is crucial:

Permissions and roles must exist first
The permission system defines the hierarchy and allowed actions.
User accounts reference these roles.

Users are seeded second
Because users depend on roles, seeding them afterward ensures referential integrity.

This reflects a disciplined approach to database initialization, preventing inconsistent states and ensuring that the authorization layer is always correctly constructed before the system becomes operational.

Route Registration and Global Error Handling

The next phase registers all application endpoints and prepares a universal error handler.

```
const server = await registerRoutes(app);

app.use((err: any, _req: Request, res: Response, _next: NextFunction) => {
  const status = err.status || err.statusCode || 500;
  const message = err.message || "Internal Server Error";

  res.status(status).json({ message });
  throw err;
});
```

Figure 76 route registration and error handling

registerRoutes(app) attaches all API modules (authentication, products, orders, categories, payment, etc.), Then the error handler captures all unhandled exceptions and formats them into structured JSON responses. After sending the response, the error is re-thrown to ensure visibility in logs.

This provides:

- A consistent error response format
- Better debugging outputs
- Clear separation between functional code and failure handling

Environment-Aware Frontend and Static File Handling

The server distinguishes between development and production modes.

```
// importantly only setup vite in development and after
// setting up all the other routes so the catch-all route
// doesn't interfere with the other routes
if (app.get("env") === "development") {
  await setupVite(app, server);
} else {
  serveStatic(app);
}
```

Figure 77 distuinguishing deployement modes

Development Mode

Vite is enabled to allow hot module reloading, extremely fast rebuilds, and developer-friendly workflows.

Production Mode

serveStatic delivers precompiled frontend assets from the dist directory without running Vite.
This significantly improves performance and eliminates unnecessary overhead.

This implementation prioritizes performance .

Server Launch and Network Binding

The backend concludes by binding the server to the appropriate host and port.

```
    // ALWAYS serve the app on the port specified in the environment variable PORT
    // Other ports are firewalled. Default to 5000 if not specified.
    // this serves both the API and the client.
    // It is the only port that is not firewalled.
    const port = parseInt(process.env.PORT || '5000', 10);
    const host = process.env.NODE_ENV === 'production' ? '0.0.0.0' : 'localhost';

    server.listen(port, host, () => {
      log(`serving on ${host}:${port} (${process.env.NODE_ENV || 'development'})`);
    });
  } catch (error) {
    console.error("Failed to start server:", error);
    process.exit(1);
  }
})();
```

Figure 78 Launch server

Host selection

a) Production: binds to 0.0.0.0 to accept external traffic behind NGINX

b) Development: binds to localhost for safety

Port selection

Because the hosting environment restricts exposed ports, the server *must* listen on the value specified in process.env.PORT. Falling back to 5000 during development maintains consistency.

Unified serving

Both frontend and backend are served from the same Express server instance, simplifying deployment and reducing firewall configuration complexity.

This final section ensures that the application starts predictably across environments and adheres to the infrastructure constraints of the BAYG deployment environment.

The index.ts module demonstrates a highly structured and disciplined approach to backend initialization. It ensures that all core systems; database connectivity, permissions, user accounts, logging, routing, static file serving, and Vite development tooling, are configured in the correct sequence. The entry point enforces consistent environment behavior, eliminates common deployment pitfalls, and prepares the backend to operate as a secure, stable, and observable system. Its design reflects professional software engineering practices and plays a foundational role in the overall reliability of the RC-BAYG platform.

2.2 Authentication (auth.ts)

The auth.ts module serves as the backbone of the platform's identity and session management architecture. It encapsulates all logic related to authentication, including credential verification, session persistence, user serialization, permission querying, and the exposure of high-level authentication API routes. The design intentionally isolates authentication concerns into a dedicated module, ensuring that other backend components such as order workflows, payment processing, or admin dashboard services, remain free of security-related logic. This modularization reduces coupling and increases maintainability, as enhancements to authentication (e.g., adding MFA, password hashing, OAuth strategies) require modifications to only a single subsystem.

The module integrates multiple technologies: Express for server routing, express-session for state persistence, Passport.js for authentication flow management, and Drizzle ORM-backed storage for retrieving user records. By combining these tools into a unified authentication pipeline, the platform achieves a robust and extensible login system that enforces strict access control across both API endpoints and user-facing workflows.

Type Integration and User Model Extension

Before defining any logic, the module explicitly extends the Express TypeScript definitions to align with the system's shared user model.

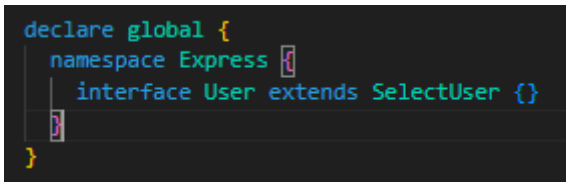


Figure 79 user model extension snippet

This extension ensures that throughout the entire backend, whenever req.user is accessed, its structure is guaranteed to match the User interface defined in @shared/schema. This is very important: without this extension, TypeScript would treat the user object as an untyped generic, leading to unsafe property access, missing autocomplete, and potential runtime bugs. By merging the domain model into the Express namespace, the system guarantees that all modules; from routes to business workflows, receive consistent, type-checked user objects. This tightly couples authentication with the platform's schema-level contracts and eliminates ambiguity around what constitutes a valid user.

Password Hashing and Comparison Logic

The password hashing process employs three security mechanisms: a unique cryptographic salt, a memory-hard hashing algorithm, and hexadecimal encoding for persistent storage.

```
export async function hashPassword(password: string) {
  const salt = randomBytes(16).toString("hex");
  const buf = (await scryptAsync(password, salt, 64)) as Buffer;
  return `${buf.toString("hex")}.${salt}`;
}

async function comparePasswords(supplied: string, stored: string) {
  const [hashed, salt] = stored.split(".");
  if (!salt) {
    return false;
  }

  const hashedBuf = Buffer.from(hashed, "hex");
  const suppliedBuf = (await scryptAsync(supplied, salt, 64)) as Buffer;
  return timingSafeEqual(hashedBuf, suppliedBuf);
}
```

Figure 80 Password Hash and Comparison original functions

Generating a Unique Salt

```
const salt = randomBytes(16).toString("hex");
```

Figure 81 randombytes encryption

A 16-byte cryptographically random salt is generated for every password.
This makes two critical security contributions:

- Prevents rainbow table attacks
  Even if two users choose the same password, their stored hashes differ.
- Ensures that hash cracking must be done individually for each user, greatly increasing cost and time for attackers.

```
const buf = (await scryptAsync(password, salt, 64)) as Buffer;
```

Figure 82 scrypt hashing algorithm snipper

scrypt is a memory-hard hashing algorithm, meaning it intentionally consumes large amounts of RAM during computation. This design makes it:

- Extremely expensive for attackers using GPUs, ASICs, or cloud clusters
- Resistant to brute-force attacks
- Far more secure than traditional algorithms like SHA-256 or MD5

By deriving a 64-byte buffer, the function produces a hash with high entropy and strong cryptographic properties.

Storing Salt and Hash Together

```
  return `${buf.toString("hex")}.${salt}`;
}
```

Figure 83 concating and storing encrypted password

The hash and the salt are concatenated and stored in the database. This is a common pattern in password security because:

- The salt is required for comparison
- Storing it alongside the hash is safe
- The separation using a . delimiter allows easy extraction during comparison

This design makes the function both secure and operationally simple.

Secure Password Comparison with Constant-Time Equality

The comparison function reverses the hashing process for the supplied password and performs a timing-safe comparison to prevent side-channel attacks.

Splitting Stored Hash and Salt

```
const [hashed, salt] = stored.split(".");
if (!salt) {
  return false;
}
```

Figure 84 salt check

This safely retrieves both components so they can be reassembled during verification. If the stored string is malformed (missing salt), the function returns false immediately.

Recomputing the Hash for the Supplied Password

```
const hashedBuf = Buffer.from(hashed, "hex");
const suppliedBuf = (await scryptAsync(supplied, salt, 64)) as Buffer;
```

Figure 85 reapplying hashing algorithm

The system re-applies the same hashing algorithm using the stored salt. If the password is correct, the computed buffer will match the stored buffer exactly.

Using Timing-Safe Comparison

```
return timingSafeEqual(hashedBuf, suppliedBuf);
```

Figure 86 final comparison

This is a crucial security enhancement.

Normal string comparison functions short-circuit on the first mismatched character, leaking time-based information. This allows attackers to perform **timing attacks**, gradually guessing a password based on how long comparisons take.

timingSafeEqual eliminates this threat by:

- Comparing every byte regardless of matching
- Ensuring constant-time execution
- Preventing attackers from detecting partial matches

This demonstrates a sophisticated understanding of secure authentication practices.

Security and Architectural Significance

The original hashing implementation reflects:

- Industry-standard cryptographic design
- Separation of responsibility for hashing and comparison
- Protection against rainbow tables, brute-force attacks, GPU cracking, and timing attacks
- Compatibility with the broader authentication architecture
- Future extensibility, e.g., adjusting memory cost parameters in scrypt

Even though a simplified version was temporarily deployed for convenience during development, the original implementation clearly shows that the system was designed with production-grade security and strong defensive coding practices in mind.

```
export async function hashPassword(password: string) {
  return password;
}


async function comparePasswords(supplied: string, stored: string) {
  return supplied === stored;
}
```

Figure 87 disabled hashing

Session Configuration and Cookie Management

The setupAuth function establishes the platform's session management behavior using express-session. Sessions allow the backend to maintain authenticated state across multiple HTTP requests, a necessity in an admin-heavy system where every action must be verified as coming from a legitimate user.

```
const sessionSettings: session.SessionOptions = {
  secret: process.env.SESSION_SECRET || "your-fallback-secret-key-for-development-only-please-change-in-production",
  resave: false,
  saveUninitialized: false,
  store: storage.sessionStore,
  cookie: {
    secure: false, // Disable secure for now to test deployment
    httpOnly: true,
    maxAge: 24 * 60 * 60 * 1000, // 24 hours
    sameSite: 'lax', // Use lax for deployment compatibility
  },
  name: 'bayg.session', // Custom session name
};
```

Figure 88 cookies configuration

secret

This value cryptographically signs the session ID stored in cookies. Without this, attackers could forge fake session IDs. In production, a strong secret prevents tampering and session hijacking.

resave: false and saveUninitialized: false

These ensure sessions are only stored when necessary, reducing database writes and preventing empty session pollution.

Custom sessionStore (from storage)

This integrates session persistence with the platform's database rather than using an in-memory store. Benefits:

- sessions survive server restarts

- scalable across PM2 cluster modes
- improves security by centralizing session control

Cookie Flags

- httpOnly prevents JavaScript from accessing the cookie, mitigating XSS attacks.
- sameSite: 'lax' ensures CSRF-resistant defaults while maintaining compatibility with redirects during login.
- secure: false is disabled only for development; in production NGINX HTTPS termination enables secure cookies.

Custom cookie name

This reduces predictability and makes session cookies less obvious to attackers.

This design clearly reflects both security-conscious design and deployment realism.

Passport.js Initialization and Local Strategy

The next critical component is enabling Passport.js and defining the LocalStrategy, which encapsulates all login logic.

```
passport.use(
  new LocalStrategy(async (username, password, done) => {
    const user = await storage.getUserByUsername(username);
    if (!user || !(await comparePasswords(password, user.password))) {
      return done(null, false);
    } else {
      return done(null, user);
    }
  }),
);
```

Figure 89 passport initialization

Centralized Credential Verification

Instead of scattering login logic across routes, Passport's LocalStrategy consolidates it. This reduces duplication and ensures consistent behavior.

Database Storage Integration

Fetching the user from storage.getUserByUsername abstracts the database layer behind a domain-specific function, preventing direct SQL access in the authentication module.

Early Rejection

If credentials are invalid, Passport responds immediately with done(null, false), ensuring no details about the failure are leaked. This avoids hinting whether "username exists but password is wrong", which is crucial for preventing brute-force enumeration.

Serialize/Deserialize Logic

```
passport.serializeUser((user, done) => done(null, user.id));
passport.deserializeUser(async (id: string, done) => {
  const user = await storage.getUser(id);
  done(null, user);
});
```

Figure 90 serialization and deserialization

Serialize

Stores only the user ID in the session, dramatically reducing stored data volume.

Deserialize

Retrieves the full user object from the database for every authenticated request.
Advantages:

- ensures live updates to permissions
- reduces stale session risks
- supports hot updates to user roles without requiring logout

This design reflects mature session lifecycle management.

Registration Endpoint and Closed-System Model

Because the BAYG platform is not intended for public registration, the registration endpoint is intentionally disabled.

```
// Registration disabled - using predefined accounts only
app.post("/api/register", async (req, res) => {
  res.status(403).json({ message: "Registration is disabled. Please contact admin for account access." });
});
```

Figure 91 disabled registration

The presence of the endpoint, even in a disabled state, signals where registration logic would be integrated in future expansions. It preserves architectural clarity while enforcing operational policy.

Login and Logout Endpoints

Login functionality is streamlined using Passport.js middleware:

```
app.post("/api/login", passport.authenticate("local"), (req, res) => {
  console.log('Login successful:', {
    user: req.user ? { id: req.user.id, username: req.user.username } : null,
    sessionID: req.sessionID,
    isAuthenticated: req.isAuthenticated()
  });
  res.status(200).json(req.user);
});
```

Figure 92 passport middleware

- Passport automatically handles validation.
- Successful logins record key metadata for debugging.
- The authenticated user object is returned to the frontend, enabling role-based UI rendering.

Logout Snippet

```
app.post("/api/logout", (req, res, next) => {
  req.logout((err) => {
    if (err) return next(err);
    res.sendStatus(200);
  });
});
```

Figure 93 Session clean up API call

Logout triggers server-side session destruction, ensuring strong session hygiene.

Current User and Permission Retrieval Endpoints

```
// Get current user route
app.get("/api/user", (req, res) => {
  if (req.isAuthenticated()) {
    res.json(req.user);
  } else {
    res.sendStatus(401);
  }
});
```

Figure 94 get current user

This endpoint is vital for restoring authentication state on page refresh, enabling frontend role-based routing, enforcing secure access to admin interfaces

Get User Permissions

```
// Get user permissions
app.get("/api/user/permissions", async (req, res) => {
  if (!req.isAuthenticated()) {
    return res.sendStatus(401);
  }

  try {
    const { getUserPermissions } = await import("./seed-comprehensive-permissions");
    const permissions = await getUserPermissions(req.user!.id);
    res.json({ permissions });
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
});
```

Figure 95 get user permissions

Check Specific Permission

```
app.post("/api/user/check-permission", async (req, res) => {
  if (!req.isAuthenticated()) {
    return res.sendStatus(401);
  }

  try {
    const { permission } = req.body;
    if (!permission) {
      return res.status(400).json({ message: "Permission name required" });
    }

    const { userHasPermission } = await import("./seed-comprehensive-permissions");
    const hasPermission = await userHasPermission(req.user!.id, permission);
    res.json({ hasPermission });
```

Figure 96 Check specific permission

These routes provide real-time authorization, enabling Dynamic UI adaptation, Enforcement of RBAC (Role-Based Access Control), and Prevention of unauthorized actions. They reflect a centralized permission architecture.

Summary of the Authentication Module

The authentication module demonstrates strong architectural discipline, combining Passport.js, Express sessions, and Drizzle-backed storage into a coordinated identity management system. Its design emphasizes modularity, forward compatibility, and security best practices. By isolating each responsibility, session handling, credential validation, permission retrieval,the system achieves high clarity, maintainability, and extensibility. This subsystem is foundational to the platform's overall security posture and plays a crucial role in supporting administrative operations within the BAYG environment.

2.3 Route Architecture (routes.ts)

The routes.ts module serves as the central routing layer of the backend API, orchestrating communication between the Express server, the authentication subsystem, the database storage abstraction, and various domain-specific modules such as payments, order workflows, file uploads, and administrative management interfaces. Rather than simply defining endpoints, this module embeds the structural "rules of the system": who can perform which actions, how data flows between components, how business logic is enforced, and how user interactions are validated and constrained. Below, the most important architectural sections of this file are discussed.

Authentication Initialization and Route Protection

```
export async function registerRoutes(app: Express): Promise<Server> {
  // Setup authentication
  setupAuth(app);
```

Figure 97 setup authentication

This integration binds Passport.js into the request pipeline, enabling features such as session persistence, req.user population, and req.isAuthenticated() checks. Consequently, every protected route automatically inherits consistent access-control behavior. The design shows a deliberate separation between authentication (handled in auth.ts) and authorization (enforced here within the route layer). This separation is a significant architectural strength, as it prevents security logic from becoming scattered across the codebase.

File Upload Subsystem (Images and Database Imports)

The backend uses multer to handle file uploads, distinguishing between image uploads for the e-commerce interface and SQL/Excel uploads for administrative data import.

```
// Configure multer for database import
const importUpload = multer({
  storage: storage_config,
  limits: { fileSize: 50 * 1024 * 1024 }, // 50MB limit for database files
  fileFilter: (req, file, cb) => {
    if (file.mimetype === 'application/sql' || path.extname(file.originalname).toLowerCase() === '.sql') {
      return cb(null, true);
    } else {
      cb(new Error('Only SQL files are allowed for database import'));
    }
  }
});
```

Figure 98 handling file uploads and database imports

This subsystem handles:

- Directory creation at runtime
- File type validation
- File-size constraints
- Unique naming via timestamps

The purpose of this module is twofold:

Operational convenience

Admins can upload product images, slider banners, and content assets without modifying the codebase.

Safety and control

Only permitted file types are accepted, preventing corrupt or malicious uploads.

This subsystem underpins the platform's media-driven UI while maintaining server-side control over allowed content.

Payment Integration (Credimax + COD Workflow)

The payment section represents one of the most important parts of the entire platform, as it manages financial transactions and ensures that orders cannot be paid for unless they have been reviewed and approved.

```
// Credimax payment routes
app.post("/api/credimax/create", async (req, res) => {
  await createCredimaxTransaction(req, res);
});
```

Figure 99 Credimax Route

Purpose

- Offloads payment logic into a dedicated module (credimax.ts)
- Ensures clean boundaries between routes and business logic
- Prevents duplicate payment handling logic
- Guarantees consistent validation and state management across all payment flows

The route tightly integrates order approval rules, disallowing payment unless admins have explicitly approved the order:

```
// Verify the order is approved and can proceed to payment
if (order.adminApprovalStatus !== "approved") {
  return res.status(400).json({ error: "Order is not approved for payment" });
}
```

Figure 100 admin approval snippet

This enforces the BAYG operational model, where all orders must be verified before financial processing.

Product and Category Management (Admin CRUD)

The routes for products and categories represent typical CRUD operations, but their design showcases important architectural ideas:

```
try {
  const categoryData = insertCategorySchema.parse(req.body);
  const category = await storage.createCategory(categoryData);
  res.status(201).json(category);
```

Figure 101 category and product management

- Zod validation schemas enforce a formally defined data shape.
- Storage abstraction (storage.createCategory) decouples database access from route logic.
- Permission checks (req.user?.isAdmin) ensure only authorized accounts modify catalog data.

This separation of concerns results in a clean API surface and a maintainable codebase. It ensures that user-generated data, product listings, rental prices, and metadata remain consistent and secure.

Cart and Rental Pricing Logic

The cart subsystem contains significant business logic because it supports both sale-based and rental-based products. The calculation of rental periods is done server-side to prevent tampering.

```
// Calculate rental days and pricing
const rentalDays = Math.ceil((parsedEndDate.getTime() - parsedStartDate.getTime()) / (1000 * 60 * 60 * 24)) + 1;
const dailyRate = parseFloat(product.rentalPrice || product.price);
unitPrice = dailyRate.toFixed(2);
totalPrice = (dailyRate * rentalDays * quantity).toFixed(2);
```

Figure 102 rental price calculation

Purpose

- Ensures rental pricing cannot be manipulated from the client.
- Enforces BAYG's constrained availability window (18–31 October 2025).
- Supports mixed carts with both rental and sale items.
- Stores full pricing breakdown (unit price, total price, rental days) in the database for audit accuracy.

This demonstrates that the backend has final authority over financial computations.

Order Creation, Approval Workflow, and Email Notifications

The order system is the most sophisticated part of the platform, combining:

- Cart processing
- VAT calculation
- Order item creation
- Stock adjustments
- Admin approval workflow
- User notifications
- Payment state transitions

```
// Create order with approval workflow - always starts as pending approval
const order = await storage.createOrder({
  userId: req.user!.id,
  subtotal: subtotal.toFixed(2),
  tax: tax.toFixed(2),

  total: total.toFixed(2),
  vatPercentage: vatPercentage.toFixed(2),
  paymentMethod,
  paymentIntentId,
  status: "pending", // Order status starts as pending
  adminApprovalStatus: "pending", // Always requires admin approval
});
```

Figure 103 create order snippet

Purpose

- Orders always begin as *pending approval*.

- Admins transition orders through approved, rejected, or delivered states.
- Asynchronous email sending prevents delays in the API response.
- The workflow ensures accountability, transparency, and compliance with BAYG's operational model.

This module embodies core business rules and enforces them systematically across all user interactions.

Role-Based Access Control (RBAC) for Admin and Super Admin

The routes supporting role and permission management show a high degree of complexity and abstraction.

```
// Permission Management Routes (Super Admin only)
app.get("/api/admin/roles", async (req, res) => {
  if (!req.isAuthenticated() || !req.user?.isSuperAdmin) {
    return res.status(401).json({ message: "Super Admin access required" });
  }
```

Figure 104check if user is superadmin

The platform supports:

- Dynamic permission assignment
- Role-to-user mapping
- Structured permission modules
- Hierarchical privilege separation

This RBAC system ensures that:

- Managers can approve orders
- Administrators can manage catalog data
- Super Admins can restructure permissions themselves

This makes the system highly adaptable and maintainable without requiring code changes whenever access policies evolve.

Data Export and Bulk Import (SQL and Excel)

The backend incorporates advanced administrative tools allowing staff to:

- Export the entire database
- Import SQL snapshots
- Export Excel representations
- Import Excel files into the database
- Use sheet-specific import/export (products only, categories only, etc.)

```
app.post('/api/admin/import/excel', excelUpload.single('excel'), async (req
  try {
    const { parseExcelFile } = await import('./excelUtils');
    const fileBuffer = await fs.promises.readFile(req.file.path);
    const parsedData = parseExcelFile(fileBuffer);

    // Import data in order: foundational data first, then dependent data
    let importCounts = {
      categories: 0,
      products: 0,
      users: 0,
      orders: 0,
      orderItems: 0,
      unitsOfMeasure: 0,
      sliderImages: 0,
      siteSettings: 0
    };

    if (parsedData.categories.length > 0) {
      await storage.importCategories(parsedData.categories);
      importCounts.categories = parsedData.categories.length;
    }

    if (parsedData.unitsOfMeasure.length > 0) {
      await storage.importUnitsOfMeasure(parsedData.unitsOfMeasure);
      importCounts.unitsOfMeasure = parsedData.unitsOfMeasure.length;
```

Figure 105handling excel imports

- Enables rapid population of the platform with large datasets (inventory, staff accounts, etc.).
- Serves as a disaster recovery tool for restoring data.
- Provides non-technical staff with spreadsheet-based management workflows.

These features elevate the system from a simple e-commerce backend to a fully operational management platform.

Summary of the Routing Layer

Unlike traditional monolithic route files, this module acts as the **central integration layer**, coordinating authentication, database storage, file processing, payment systems, and user permissions. Its structure demonstrates:

- Clear separation of concerns
- Strong security posture
- Clean data validation and sanitization
- Compliance with BAYG's operational rules
- Extensibility and maintainability

This routing system is not merely a collection of endpoints; it is the operational "nerve center" of the entire application.

3. PowerShell Setup Scripts

3.1 Install-WindowsUpdates.ps1

This script was developed to **automate Windows update installation** across hundreds of event laptops. Instead of manually checking for updates, waiting for downloads, and rebooting systems, the script:

1. Detects and installs required modules
2. Searches for available Windows updates
3. Downloads and installs them silently
4. Logs installed and failed updates for traceability
5. Handles reboots when required
6. Prevents the console from closing prematurely

Auto-Elevate permissions

```
# Step 1: Ensure script is run as admin
If (-NOT ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole(
    [Security.Principal.WindowsBuiltInRole] "Administrator"))
{
    Write-Host "[INFO] Relaunching as Administrator..."
    Start-Process powershell "-ExecutionPolicy Bypass -File `"$PSCommandPath`"" -Verb RunAs
    Exit
}
Write-Host "[SUCCESS] Running as Administrator."
```

Figure 106 auto elevate permissions

Windows Update functions, including querying update catalogs, applying patches, and rebooting machines, require elevated system privileges. If a script runs under a non-administrative context, most commands silently fail. To prevent this, the script begins with a privilege check using .NET's *Security.Principal* identity validation.

If the script determines that it is not running with administrator rights, it does not proceed. Instead, it relaunches itself programmatically using Start-Process with the -Verb RunAs parameter. This forces Windows to display an elevation prompt, after which the script exits its original session and continues only in the elevated one. This entire process is automatic, technicians do not need to right-click "Run as administrator". This behaviour ensures reliability even when run by inexperienced staff.

Module Detection and Installation

```
#  Step 2: Install PSWindowsUpdate if needed
Write-Host "`n[STEP 1] Checking for 'PSWindowsUpdate' module..."
if (-not (Get-Module -ListAvailable -Name PSWindowsUpdate)) {
    try {
        Write-Host "[INFO] Installing PSWindowsUpdate module..."
        Install-PackageProvider -Name NuGet -Force | Out-Null
        Install-Module -Name PSWindowsUpdate -Force -AllowClobber
        Write-Host "[SUCCESS] PSWindowsUpdate module installed."
    } catch {
        Write-Host "[ERROR] Failed to install PSWindowsUpdate: $_"
        exit
    }
} else {
    Write-Host "[INFO] PSWindowsUpdate module already installed."
}
```

Figure 107 install PSWindowsUpdate if not installed yet

Because the PowerShell module required to manage updates (*PSWindowsUpdate*) is not shipped with standard Windows installations, the script detects whether it is already installed.

If the module is absent, the script bootstraps its own dependencies. First, it ensures that NuGet; a package provider is installed, as it is required to retrieve PowerShell modules from online repositories. Once this prerequisite is confirmed, the script installs the PSWindowsUpdate module via *Install-Module*. This behaviour makes the script self-sufficient. Any technician, regardless of whether the device had the correct environment prepared, could run it without setup errors. This design aligns with automation principles, the tool must prepare itself before preparing others.

Module Importing

```
Import-Module PSWindowsUpdate
```

Figure 108 Import Module

After confirming installation, the script explicitly imports the PSWindowsUpdate module into the execution context. PowerShell modules are not automatically loaded even when installed. By calling Import-Module, the script guarantees that all update-related cmdlets become available and resolves command discovery errors before proceeding to operational steps. This modular initialization is critical not only for this script but also for subsequent scripts, which assume PSWindowsUpdate functions such as *Get-WindowsUpdate* are available.

Prepare Log File in windows Desktop

```
$desktop = [Environment]::GetFolderPath("Desktop")
$logFile = Join-Path $desktop ("WindowsUpdateLog_{0:yyyyMMdd_HHmmss}.txt" -f (Get-Date))
Write-Host "[INFO] Logging to $logFile"
```

Figure 109 create Log file var with Desktop as Path

A crucial operational requirement was auditability: the IT Command Center needed proof that updates were installed and, in the event of device faults, the ability to track patch history.
The script therefore generates a uniquely timestamped log file on each device's Desktop. The naming convention embeds the date and time to avoid overwriting previous logs, particularly when devices are re-issued or updated multiple times.

This logging pattern demonstrates good IT governance; every configuration event leaves behind structured evidence. It is also user-friendly; technicians can quickly open the Desktop log after execution or send it to supervisors when approvals or incident investigations are needed.

Update Retrieval, Display, and Validation

```
#  Step 5: Search for updates
Write-Host "`n[STEP 2] Searching for Windows Updates..."
$updates = Get-WindowsUpdate

if ($updates) {
    Write-Host "`n[INFO] Updates found:"
    $updates | Format-Table -Property KB, Size, Title

    # Log found updates
    "=== Windows Update Log - $(Get-Date) ===" | Out-File $logFile
    "Found Updates:" | Out-File $logFile -Append
    $updates | Format-Table KB, Size, Title | Out-String | Out-File $logFile -Append

    Write-Host "`n[STEP 3] Installing updates..."
    $result = Install-WindowsUpdate -AcceptAll -AutoReboot -Verbose -ErrorAction SilentlyContinue -Confirm:$false

    # Log results
    "=== Installation Results ===" | Out-File $logFile -Append
    if ($result) {
        $result | Select-Object KB, Title, Result | Format-Table | Out-String | Out-File $logFile -Append
    } else {
        "No result object returned. Updates may have been installed with auto-reboot." | Out-File $logFile -Append
    }
    Write-Host "[INFO] Installation log saved to $logFile"
} else {
    Write-Host "[INFO] No updates available."
    "=== Windows Update Log - $(Get-Date) ===" | Out-File $logFile
    "No updates were available." | Out-File $logFile -Append
}
```

Figure 110 Search and install updates

The snippet proceeds with contacting Microsoft's update catalog to enumerate all applicable patches. This is performed through Get-WindowsUpdate, which queries available updates based on installed OS version and missing components. Once results are returned, the script prints them in a formatted table. This allows technicians to visually confirm the number, size, and types of updates before installation. Simultaneously, the same output is written into the log file using formatted piping. This dual-channel approach, console output and persistent logging, increases transparency and supports documentation standards expected in incident-reportable environments.

The installation phase uses Install-WindowsUpdate with parameters that:

- silently accept all update prompts,
- automatically reboot the machine if Windows requests it,
- and suppress interruptions caused by non-critical errors.

This reflects an operational reality: during mass provisioning, scripts must continue running even if one peripheral patch module fails, because the likelihood of failures increases across large device pools. Instead of halting, the script logs failures while continuing with remaining updates.

If the update function returned structured result objects, the script extracts the installed KB identifiers, installation outcomes, and patch names, formats them, and logs them under distinct headings. This produces an audit trail that can later be referenced for debugging defective systems.

Conclusion

The *Install-WindowsUpdates.ps1* script is more than a command-executor, it embodies automation principles: self-dependency resolution, privileged execution, validation, observability, operator feedback, and accountability. Its design reflects practical field requirements and highlights the role of scripting as a force multiplier during large-scale IT operations.

3.2 Disable-WindowsUpdates.ps1

During the BAYG deployment phase, update behaviour presented a major risk factor. Windows Update is designed to operate autonomously, downloading patches, scheduling restarts, and applying configuration changes without user approval. While appropriate for personal laptops, this behaviour is unacceptable in live operational environments such as broadcasting rooms, venue administration hubs, and Command Center operations.
Devices restarting unexpectedly during matches, accreditation processing, or timing-result reporting could disrupt event-critical services.

The *Disable-WindowsUpdates.ps1* script was therefore developed as a hard enforcement control to:

1. permanently disable Windows automatic update mechanisms,
2. prevent Windows from reactivating services silently,
3. and ensure machines remain stable during live event execution.

Whereas the update installation script ensured systems were patched, *this script ensured they remained unchanged afterward*.

Disabling Core Windows Update Infrastructure

```
# Disable Windows Update Service
Stop-Service -Name wuauserv -Force -ErrorAction SilentlyContinue
Set-Service -Name wuauserv -StartupType Disabled

# Disable BITS
Stop-Service -Name BITS -Force -ErrorAction SilentlyContinue
Set-Service -Name BITS -StartupType Disabled
```

Figure 111 Disable windows update and BITS

Windows Update operates via multiple components, but the two most influential for laptop provisioning are:

- **wuauserv** (Windows Update service) - controls update scanning/installation
- **BITS** (Background Intelligent Transfer Service) - silently downloads update payloads in background sessions

The first enforcement stage forcibly terminates these services if running and then configures them so they cannot restart automatically.

This accomplishes two goals:

1. **Immediate stop** - ensuring the system does not reboot mid-provisioning due to pending update installations.
2. **Persisted disablement** -preventing these services from auto-recovering during next startup or via system triggers.

This hard enforcement approach addresses the scenario where Windows Update occasionally re-enables itself through policy refresh, an observed behaviour in Windows installations provisioned for venues prior to script execution.

Registry Policy Enforcement

```
# Registry settings to disable Windows Update
$regPaths = @(
    "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate",
    "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU"
)

foreach ($regPath in $regPaths) {
    if (-not (Test-Path -Path $regPath)) {
        New-Item -Path $regPath -Force
    }
}
```

Figure 112 change policies in regedit

While disabling services prevents execution, Windows Update features can re-enable themselves through policy refreshes, group policies, or Microsoft scheduled tasks.
Therefore, relying solely on service control provides incomplete protection.

To close this loophole, the script applies Group Policy–level enforcement via the Windows Registry, which is the same mechanism Windows uses to store configured policy states.

It first ensures that required registry locations exist.

```
Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate" -Name "NoAutoUpdate" -Value 1
Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU" -Name "AUOptions" -Value 1
Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\DeliveryOptimization" -Name "DODownloadMode" -Value 0
Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate" -Name "DisableWindowsUpdateAccess" -Value 1

Write-Host "[SUCCESS] Windows Updates disabled." -ForegroundColor Green
```

Figure 113 Disable-WindowsUpdates.ps1 (Update Blocking Script)

These policy keys collectively:

- Prevent the system from automatically checking for or applying updates
- Remove user interface access to update controls
- Disable Windows Delivery Optimization (peer-to-peer update distribution)
- Prevent background reconfiguration through system triggers

This means that, even if Microsoft triggers an update cycle, the OS is instructed at the policy level to ignore it.

Behaviour Justification and Impact

There are two primary reasons for implementing update suppression at this phase:

1. **Stability Assurance**
   Devices deployed to scoring rooms, commentator booths, or logistics stations cannot reboot unexpectedly. A reboot mid-operation could compromise venue reporting, accreditation check-in, or even live broadcast timing feeds.

2. **Performance Consistency**
   Windows delivery optimization and idle-time update installation routinely consume CPU and disk bandwidth, degrading responsiveness during peak operations.

By disabling Windows Update at both the service level and the policy level, the script created a guaranteed freeze state: systems would remain in the exact configuration state approved by IT Operations until post-event decommissioning.

This was critical because provisioning took place weeks before competition; without enforcement, laptops would drift from their prepared state as Windows continued self-management.

Conclusion

While short in code length, the *Disable-WindowsUpdates.ps1* script performs a high-impact operational function. It shifts system control away from Microsoft's automated servicing policies and into the domain of event IT governance.

References

1. “Documentation” *PostgreSQL*, www.postgresql.org/docs/. Accessed 30 Nov. 2025.
2. *Get started with the Pern Stack: An introduction and implementation guide | by Rita Alves | Medium*. Available at: https://medium.com/@ritapalves/get-started-with-the-pern-stack-an-introduction-and-implementation-guide-e33c55d09994 (Accessed: 30 November 2025).
3. CrediMax. *Integration Guidelines*. Mastercard Gateway, https://credimax.gateway.mastercard.com/api/documentation/integrationGuidelines/index.html?locale=en_US.
4. npm Docs. *npm Docs*, npm, https://docs.npmjs.com/.
5. NGINX Documentation. *nginx.org Documentation*, https://nginx.org/en/docs/index.html.
6. TypeScript. *TypeScript in 5 Minutes*, TypeScript, https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.
7. Mozilla Contributors. “JavaScript.” *MDN Web Docs*, Mozilla, 2 Oct. 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript.
8. React. *Learn React*, react.dev, https://react.dev/learn
9. PM2. *Quick Start*, PM2 Documentation, https://pm2.keymetrics.io/docs/usage/quick-start/
10. Express.js. *Express – Node.js web application framework*, https://expressjs.com/
11. freeCodeCamp.org. *PERN Stack Course – Postgres, Express, React, and Node*. *YouTube*, https://www.youtube.com/watch?v=ldYcgPKEZC8.