Project of AAA Furnitures

Software Requirement Specification

Version
1.0

Status
DRAFT adapted from IEEE SRS Template

Proposed by:
Andreandhiki Riyanta Putra
Andrian Danar Perdana
M. Argya Vityasy

Instructor
Khabib Mustofa

March 5, 2025

Contents

Revision History

Name	Date	Reason For Changes	Version
21	22	23	24
31	32	33	34

CONTENTS

CHAPTER 1
Introduction

1.1 PURPOSE

The purpose of this document is to present a detailed description of the functional and non-functional requirements for the Furniture Selling Web Based Application. It will also explain the app constraints, interface, and interactions with each services. This document is intended for both the stakeholders and the developers for a reference of developing the first version of the application.

1.2 PROJECT SCOPE

AAA Furnitures is a web-based application that allows the customers of AAA Furnitures to purchase furnitures online. The application will allow the customers to browse the available furnitures, add them to the cart, make secure payments, and track the delivery of the purchased items. The application will also allow the admin to manage the products displayed on the website and the orders made by the customers. The system will consist of multiple microservices, including the auth service, product service, the order service, the payment service, and the delivery service.

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Term	Definition
User	A registered customer who can browse, add to cart, and place orders.
Admin	A user with special privileges to manage products, orders, and the contact for users.
Cart	A temporary collection of items selected by a user for purchase.
Order	A confirmed request for purchasing one or more items.
JWT	JSON Web Token, used for authentication and authorization. Used for the user login, logout[1]
API	Application Programming Interface, allows different services to communicate with each other.[2]
REST	Representational State Transfer, an architectural style for designing APIs that use stateless communication over HTTP.[3]
Session	A temporary authentication state that maintains user login status.
SQL	Structured Query Language, used for managing relational databases by defining, querying, and modifying data.[4]
Microservice	An architectural style that structures an application as a collection of small, independent, and loosely coupled services.[5]
Message Queueing	A communication method where messages are sent and stored in a queue, ensuring asynchronous processing between microservices such as order, delivery, and payment services.[6]

Table 1.1: Definitions, Acronyms, and Abbreviations

1.4 REFERENCES

[1] M. B. Jones, J. Bradley, and N. Sakimura, *JSON Web Token (JWT)*, RFC 7519, May 2015. DOI: 10.17487/RFC7519. [Online]. Available: https://www.rfc-editor.org/info/rfc7519.

[2] J. Ofoeda, R. Boateng, and J. Effah, "Application programming interface (api) research: A review of the past to inform the future," *International Journal of Enterprise Information Systems*, vol. 15, pp. 76–95, Jul. 2019. DOI: 10.4018/IJEIS.2019070105.

[3] N. Patel, "Representational state transfer (rest) application program interface (api)," Jun. 2018.

[4] Y. Silva, I. Almeida, and M. Queiroz, "Sql: From traditional databases to big data," Feb. 2016, pp. 413–418. DOI: 10.1145/2839509.2844560.

[5] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li, "Microservices: Architecture, container, and challenges," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2020, pp. 629–635. DOI: 10.1109/QRS-C51114.2020.00107.

[6] S. Bouchenak and N. de Palma, “Message queuing systems,” in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 1716–1717, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_1548. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_1548.

CHAPTER 2

Overall Description

2.1 PRODUCT PERSPECTIVE

This product is a new e-commerce platform designed as a microservices-based system. It aims to provide a comprehensive online shopping experience with user authentication, product browsing, order management, payment processing, and delivery tracking capabilities. The system is self-contained but designed with a modular architecture to allow for future expansion and integration with third-party services such as payment gateways and shipping providers. The microservices architecture ensures that each component can be developed, deployed, and scaled independently. The following figure **??** illustrates the high-level architecture of the system:

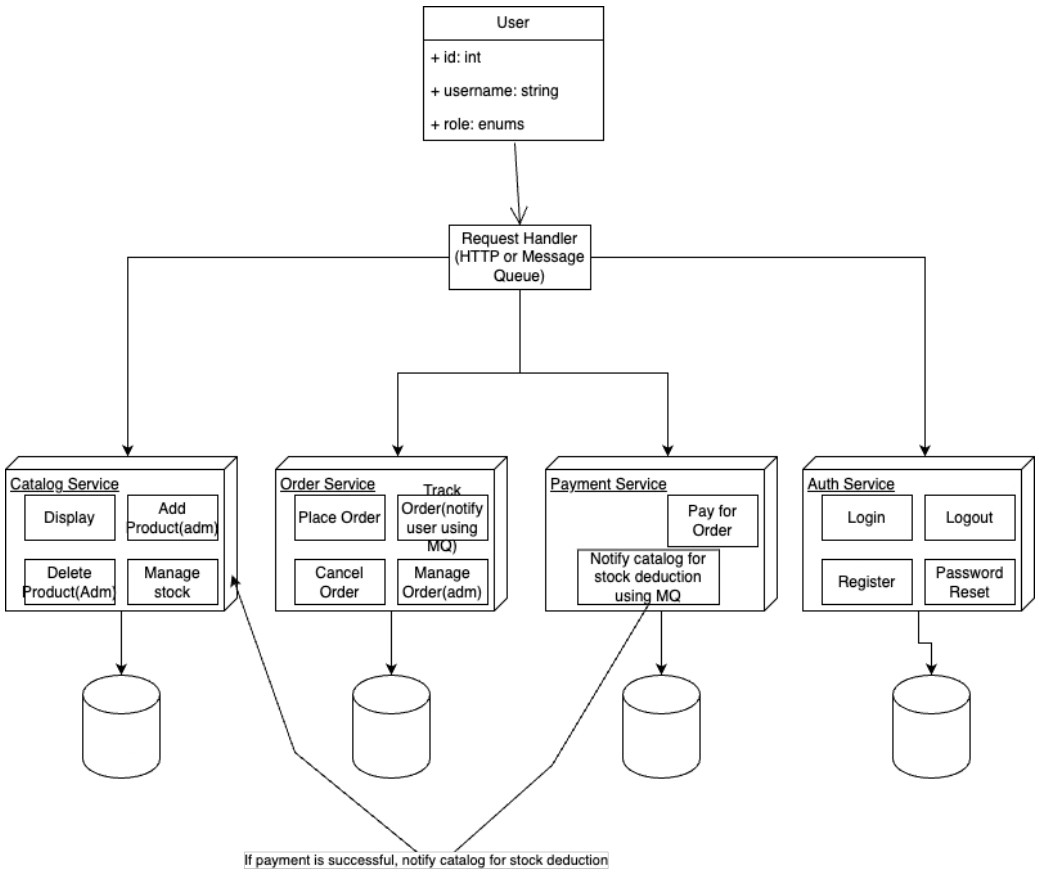

Figure 2.1: System Architecture Diagram

2.2 PRODUCT FUNCTIONS

The e-commerce platform will perform the following major functions:

- User Authentication and Account Management
  - Register new user accounts
  - Login and logout functionality
  - Password reset capabilities
- Product Catalog Management
  - Browse and display product information
  - Add new products (administrator function)
  - Delete products (administrator function)
  - Manage product inventory
- Order Processing
  - Add products to shopping cart
  - Place and confirm orders
  - Cancel existing orders
  - Track order status and delivery
- Payment Processing
  - Process secure payments for orders
  - Notify catalog service for inventory updates
- Delivery Management
  - Arrange delivery for completed orders
  - Track delivery status

Figure **??** illustrates the use case for this application:

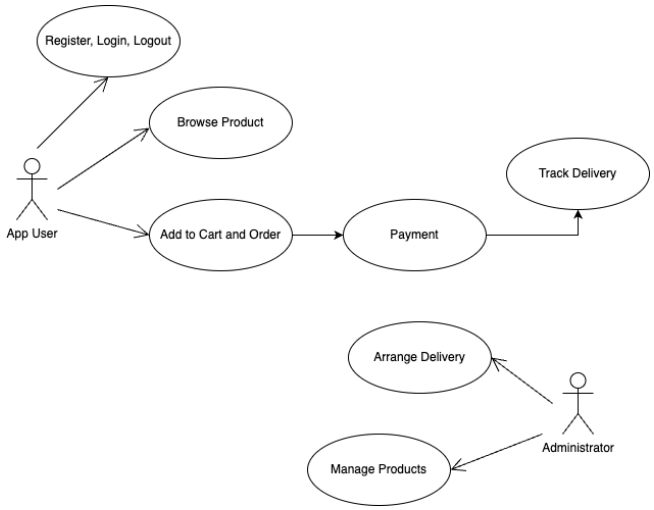

Figure 2.2: Use Case Diagram

2.3 USER CLASSES AND CHARACTERISTICS

The system will serve the following user classes:

- Regular App Users
  - Frequency: Regular, potentially daily use
  - Functions: Browse products, manage cart, place orders, track deliveries
  - Technical expertise: Minimal, should be able to navigate standard e-commerce interfaces
  - Priority: High - primary user class
- Administrators
  - Frequency: Regular, primarily during business hours
  - Functions: Manage product catalog, arrange deliveries, handle order issues
  - Technical expertise: Moderate, trained on system administration
  - Security level: High, with access to user data and system configuration
  - Priority: Medium - essential for system operation but smaller user base
- Guest Users
  - Frequency: Occasional to one-time use
  - Functions: Browse products only, limited functionality until registration
  - Technical expertise: Minimal
  - Priority: Low - encourage conversion to registered users

2.4 OPERATING ENVIRONMENT

The software will operate in the following environment:

- Server Environment
  - Containerized microservices using Docker
  - Kubernetes for orchestration
  - Linux-based operating systems
  - Scalable cloud infrastructure (AWS, Google Cloud, or Azure)
- Client Environment
  - Web browser support: Chrome, Firefox, Safari, Edge (latest versions)
  - Responsive design for various screen sizes
- Database Environment
  - Dedicated database for each microse

  - Support for SQL and NoSQL databases depending on service requirements
- Network Environment
  - HTTP/HTTPS protocols for client-server communication
  - Message Queue (MQ) for asynchronous inter-service communication

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

The following constraints will affect the design and implementation of the system:

- Architectural Constraints
  - Microservices architecture as depicted in the diagram
  - RESTful API design principles for service interfaces
  - Message Queue for asynchronous communication between services
- Security Constraints
  - Compliance with PCI DSS for payment processing
  - Secure user authentication and authorization
  - Data encryption for sensitive information
  - Safe payment for user
  - Regular security audits and penetration testing
- Integration Constraints
  - Compatible with standard payment gateways
  - API-based integration with shipping and delivery services
- Performance Constraints
  - Response time for user interactions under 2 seconds
  - System capable of handling at least 1000 concurrent users
  - Scalability to accommodate peak shopping seasons
- Development Constraints
  - Coding standards and style guides for consistency
  - Comprehensive unit and integration testing
  - CI/CD pipeline for automated testing and deployment

2.6 ASSUMPTIONS AND DEPENDENCIES

The following assumptions and dependencies apply to the project:

- Assumptions
  - Users have access to stable internet connections
  - Peak usage will occur during promotional events and holidays
  - Most users will access the system via mobile devices
  - Inventory data will be updated in near real-time
- Dependencies
  - Reliable third-party payment processing services
  - Shipping and delivery partner APIs
  - Cloud infrastructure provider's uptime and service level agreements
  - Message Queue service for inter-service communication
  - Database management systems for each service's data storage

  x
- Risks
  - Integration challenges with third-party services
  - Scalability issues during peak usage periods
  - Security vulnerabilities in payment processing
  - Data consistency across microservices

CHAPTER 3

External Interface Requirements

3.1 USER INTERFACES

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

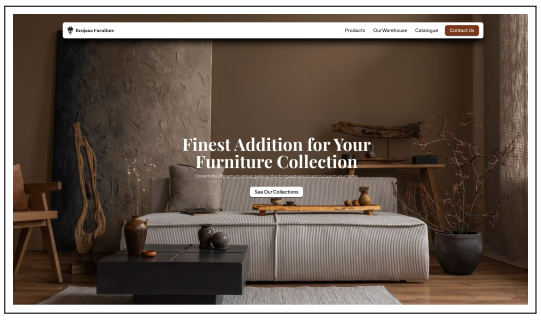

Figure 3.1: Hero Section

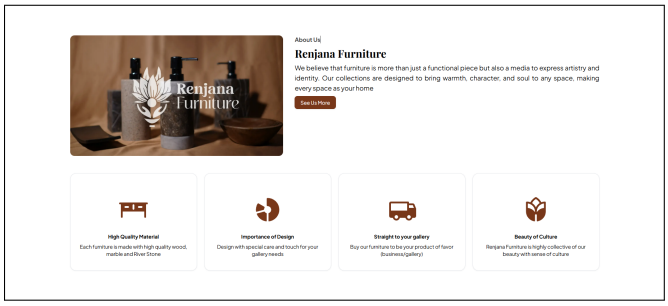

Figure 3.2: About Section

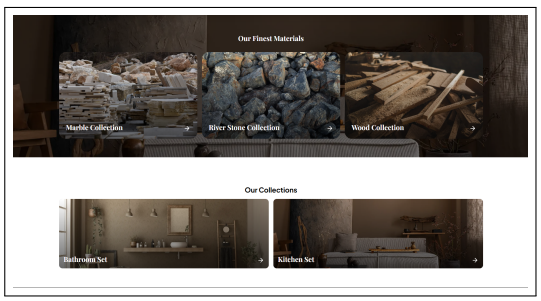

Figure 3.3: Hero Section

Figure 3.4: About Section

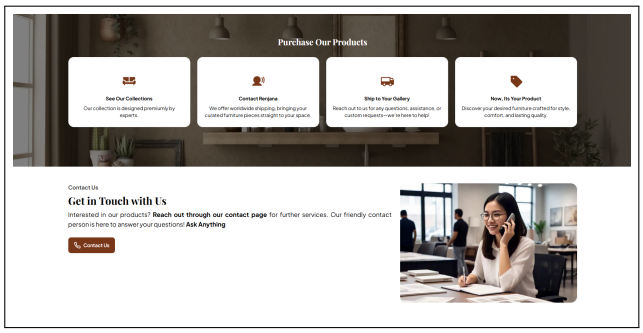

Figure 3.5: Hero Section

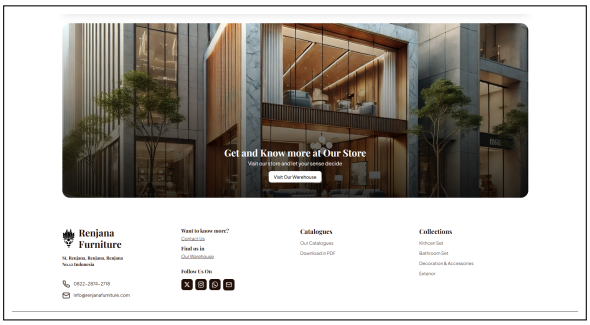

Figure 3.6: About Section

3.2 HARDWARE INTERFACES

The catalog website and the admin dashboard do not depend on specific hardware components, and therefore, no direct hardware interfaces are required. All product-related data, including images, descriptions, and other relevant details, are managed within the web platform. Payment transactions are facilitated through direct communication via WhatsApp rather than a dedicated payment processing hardware.

The database server connection is handled by the underlying infrastructure provided by the hosting service, ensuring reliable data storage and retrieval for both visitors and administrators. This approach abstracts hardware dependencies, allowing the system to operate efficiently across various environments.

3.3 SOFTWARE INTERFACES

The catalog website interacts with various software components to ensure seamless functionality and efficient data management. It is developed using Next.js as the primary frontend framework, with Tailwind CSS for styling and responsive design. The backend is managed through Payload CMS, which facilitates content organization and retrieval.

The system utilizes Supabase (Vercel Postgres) as its primary database for storing product details, user interactions, and other relevant data. Additionally, Supabase storage is used to handle and serve product images, ensuring optimized retrieval and delivery. Communication between the frontend and backend occurs via API requests, allowing structured data exchange and synchronization.

For transactions, the website does not include an integrated payment gateway but instead facilitates purchases through direct WhatsApp messaging. Furthermore, third-party libraries and services are integrated for image optimization, SEO enhancements, and performance monitoring to improve user experience and website visibility.

Data sharing between software components follows an API-driven approach, ensuring security, consistency, and scalability. If required, additional implementation constraints, such as authentication mechanisms and rate limiting, can be applied to protect data integrity.

3.4 COMMUNICATIONS INTERFACES

The catalog website relies on standard web communication protocols to ensure seamless data exchange between the frontend, backend, and external services. The system primarily uses

HTTPS for secure communication and is accessible through modern web browsers.
Data transactions between the frontend and backend are managed through GraphQL API provided by Payload CMS, allowing efficient retrieval and management of product information stored in Supabase (Vercel Postgres).
For customer inquiries via the Contact Us form, the system integrates with Google Sheets API to store submitted form data in a centralized Google Sheet. Additionally, Nodemailer is used to send email notifications based on form submissions, ensuring that administrators are promptly informed of customer inquiries.
To enhance the user experience, the website also integrates Google Maps API, which provides an interactive map displaying the warehouse location. This allows users to easily locate the company's physical store or storage facility.
Since transactions are conducted outside the platform via WhatsApp messaging, no payment gateway integration is required. WhatsApp's built-in encryption ensures secure communication between customers and the business.
To maintain data integrity and security, all API communications follow industry best practices, including authentication mechanisms, rate limiting, and encrypted data transmission where applicable.

CHAPTER 4

System Features

4.1 PRODUCT MANAGEMENT

4.1.1 Description and Priority

This feature enables administrators to efficiently manage the product catalog by adding, editing, and deleting products. Each product entry includes essential details such as name, description, availability status, dimensions (X, Y, Z), category, and optional pricing. Additionally, product images are stored securely in Supabase (Vercel Postgres). Given its critical role in maintaining the catalog, this feature is assigned high priority.

4.1.2 Stimulus/Response Sequences

1. The admin logs into the dashboard or admin panel.
2. The admin selects Add Product, fills in the product form, and submits it.
3. The system stores the data in the database via Payload CMS.
4. The admin selects Edit Product, modifies the details, and saves the changes.
5. The system updates the product data in the database.
6. The admin selects Delete Product, and the system removes the product from the database.

4.1.3 Functional Requirements

- REQ-1: The system must provide an admin dashboard to manage products.
- REQ-2: The system must support CRUD (Create, Read, Update, Delete) operations for products.
- REQ-3: Product images must be stored and retrieved from Supabase (Vercel Postgres).
- REQ-4: The system must display error messages for invalid inputs.

4.2 PRODUCT CATALOGUE DISPLAY

4.2.1 Description and Priority

This feature ensures that products in the catalog are displayed in an organized manner based on predefined collections. The catalog is structured into two main categories:

1. Materials Collection: Includes products categorized by material, such as Wood, Marble, and River Stone.
2. Other Categories: Groups products based on their function or placement, including **Kitchen Set, Bathroom Set, Decoration Accessories, and Exterior.

This feature is high priority, as it directly impacts the browsing experience and usability of the catalog for customers.

4.2.2 Stimulus/Response Sequences

1. User Action: Opens the catalog landing page.
    - System Response: Displays collections section and navigation bar with various predefined collections with a categorized product listing.
2. User Action: Selects a collection (e.g., ”Wood” or ”Kitchen Set”).
    - System Response: Filters and presents products from the selected collection.
3. User Action: Clicks on a product to view its details.
    - System Response: Displays the product’s name, description, dimensions, availability status, images, and optional price.

4.2.3 Functional Requirements

- REQ-5: The system must categorize products into Materials Collection and Other Categories.
- REQ-6: The catalog page must display collections in a structured manner.
- REQ-7: Users must be able to filter products by collection.
- REQ-8: The system must provide a detailed product view when a user selects a product.

4.3 CONTACT FORM SUBMISSION

4.3.1 Description and Priority

This feature allows users to submit inquiries or information requests via the contact form. Data is stored in Google Sheets API, and the admin receives email notifications via Nodemailer. This feature has a medium priority.

4.3.2 Stimulus/Response Sequences

1. User Action: Fills out the contact form and clicks Submit.
    - System Response: Stores the data in Google Sheets API.
2. System Action: Sends an email notification to the admin using Nodemailer.
3. User Action: Receives a confirmation message indicating that the inquiry was successfully submitted.

4.3.3 Functional Requirements

- REQ-8: The contact form must store submissions in Google Sheets API.
- REQ-10: The system must send email notifications to the admin via Nodemailer.
- REQ-11: The system must display a confirmation message after form submission.

4.4 WAREHOUSE LOCATION DISPLAY

4.4.1 Description and Priority

This feature displays the warehouse/store location using Google Maps API, allowing users to easily find the business's physical address. This feature has a low priority, but it enhances the user experience.

4.4.2 Stimulus/Response Sequences

1. User Action: Opens the warehouse location page.
    - System Response: Displays the warehouse address along with a map preview.
2. User Action: Clicks the CTA button to open the location.
    - System Response: Redirects the user to Google Maps, opening the coordinates in the app or web version.

4.4.3 Functional Requirements

- REQ-12: The system must display the warehouse location on a map.
- REQ-13: The system must provide a CTA button that allows users to open the location in Google Maps.
- REQ-14: Clicking the CTA button must redirect users to Google Maps with the correct coordinates.

CHAPTER 5

Other Nonfunctional Requirements

5.1 PERFORMANCE REQUIREMENTS

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

5.2 SAFETY REQUIREMENTS

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 SECURITY REQUIREMENTS

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

5.4 SOFTWARE QUALITY ATTRIBUTES

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

5.5 BUSINESS RULES

<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>

CHAPTER 6

Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

CHAPTER 7
Appendices

7.1 APPENDIX A: GLOSSARY

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

7.2 APPENDIX B: ANALYSIS MODELS

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

7.3 APPENDIX C: TO BE DETERMINED LIST

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>