Software Requirement Specification Document

For

ShellPG-Swift: An Online Gas Booking System

Submitted By:

Shivang Dubey

Atishi Jain

Priti Sah

Naman Mittal

Priyanshu Tiwari

Batch: SE-T&S (Room 209)

Contents

Introduction

In today's fast-paced world, convenience and efficiency are paramount. Recognizing the evolving needs of our valued customers, we are proud to introduce "ShellPG-Swift," an innovative online LPG (Liquefied Petroleum Gas) booking system. This System Requirements Specification (SRS) document outlines the fundamental aspects and objectives of this project, setting the stage for its development and deployment.

1. Purpose of the Application

The primary purpose of this SRS is to provide a comprehensive overview of the ShellPG-Swift project. It serves as a crucial document for various stakeholders, including project managers, developers, testers, and end-users. The SRS outlines the project's goals, functionality, and constraints, ensuring everyone involved understands the scope and objectives of this online LPG booking system.

2. Scope of the Application

ShellPG-Swift aims to revolutionize the way our customers interact with us by offering a seamless online platform for booking LPG gas. The system will encompass the following key features and functionalities:

a. **User Registration and Authentication**
Customers can create accounts, log in securely, and manage their profiles.

b. **LPG Booking**
Users can place, modify, or cancel LPG gas booking orders with ease.

c. **Record Maintenance**
Maintaining gas booking records for data analysis and order history safekeep.

d. **Payment Processing**
Secure online payment options for booking charges, enhancing user convenience.

e. **User Support**
Access to FAQs for assistance.

3. Overview

The following sections of this SRS will delve deeper into the technical and functional aspects of ShellPG-Swift. It will include details on the Functional Requirements, Non-Functional Requirements. Furthermore, it will outline the software and hardware requirements, the use case diagrams, sequence diagram to ensure the successful development and deployment of this cutting-edge online LPG booking system.

System Overview

1. Application Flow

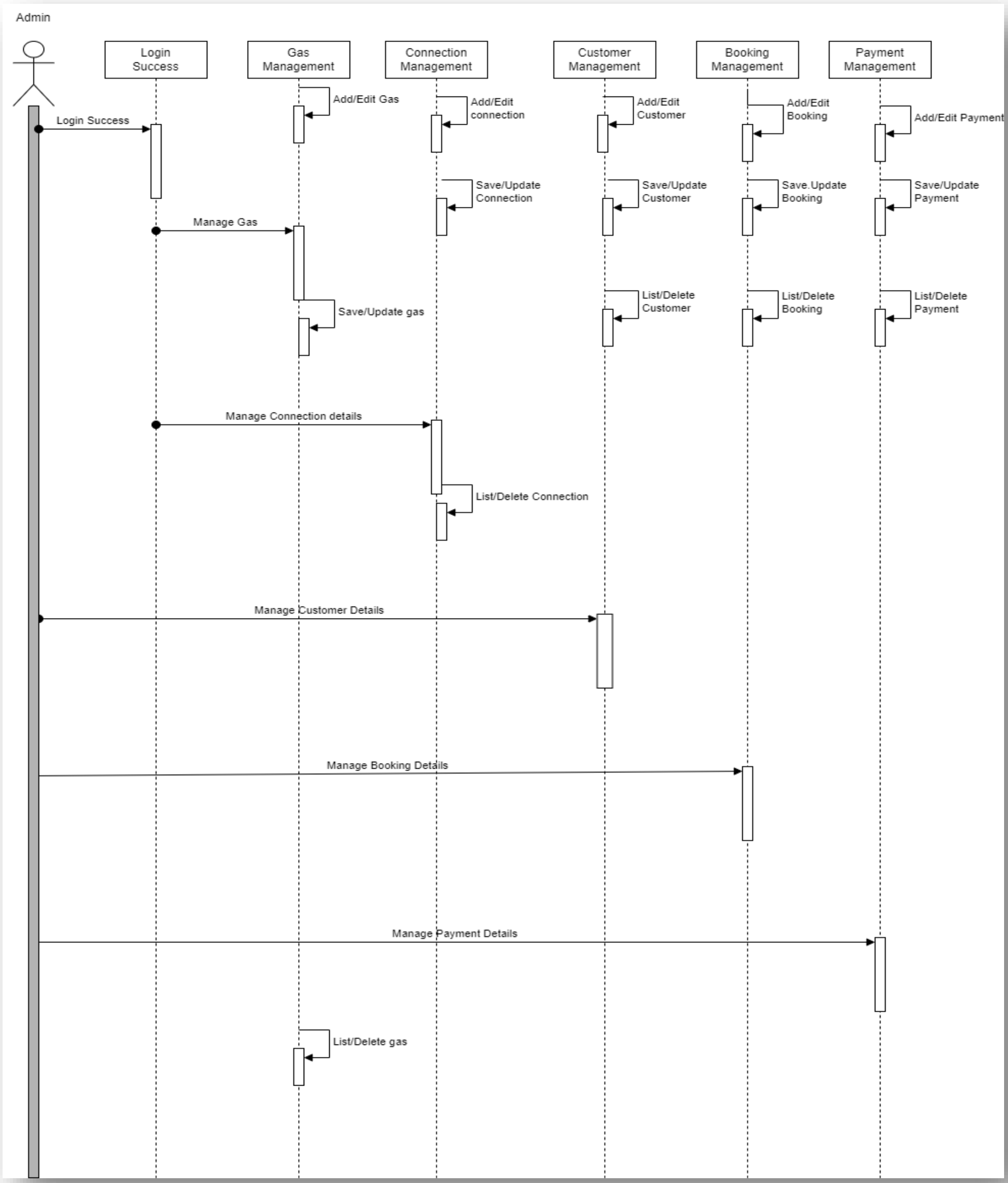

Figure 1: Sequence Diagram for ShellPG-Swift

2. User Interface

For a responsive and user-friendly web application, we will be using a mobile-first design approach with a modular and clean UI. The UI will be using the following structure:

- Header:
    - The header should contain the application logo or branding.
    - Include a navigation menu for easy access to different sections of the application.
    - Display user account information and options for login/logout.
- Home Dashboard:
    - Upon login, users are directed to a home dashboard.
    - The dashboard provides an overview of their current LPG gas bookings and any important notifications.
    - Include quick action buttons for placing new orders or tracking existing ones.
- LPG Booking:
    - When users initiate a new LPG gas booking, they should be presented with a form.
    - The form collects delivery details, such as address and preferred delivery time, and allows users to specify the quantity of gas they need.
    - Provide clear calls to action for submitting the order.
- Order Management:
    - Users should have a screen for managing their existing orders.
    - This screen displays a list of current and past orders with details like order ID, delivery status, and date.
    - Users can select orders to modify or cancel them.
- Delivery Tracking:
    - For tracking LPG gas deliveries, generate a unique tracking ID.
    - Include estimated delivery times and delivery agent contact information.
- Payment Processing:
    - When users proceed to make a payment, display a secure payment page.
    - Offer multiple payment options (e.g., credit card, online banking) and ensure a smooth and secure payment process.
    - Provide order summaries and payment confirmation.
- User Profile:
    - Users should have a profile section where they can update their personal information, contact details, and preferences.
    - Allow users to change their password and manage notification preferences.
- User Support and FAQs:
    - Create a support center that offers FAQs, help guides, and contact options for customer support.

- Users can access this section to find answers to common questions or request assistance.

3. Dependencies

a. Software Dependencies

Software dependencies refer to the specific software components or libraries that the ShellPG-Swift system relies upon to function effectively. These dependencies are integral to the software's operation, and any changes or updates to them must be carefully managed.

- *React Framework and Libraries:*
  The React frontend of ShellPG-Swift depends on the React framework itself and various libraries used for user interface components and state management. React must be maintained and updated to ensure compatibility with the latest features and security patches.

- *ASP.NET and .NET Libraries:*
  ASP.NET serves as the middleware layer and depends on the .NET framework and associated libraries. This includes libraries for RESTful API development, authentication, and database interactions. Updates to .NET and related libraries should be tracked and applied as necessary.

- *PostgreSQL Database Server:*
  The PostgreSQL database is a critical component of the system. The database software, including its core engine and any extensions or modules, must be maintained and updated to ensure data integrity, performance, and security.

b. Hardware Dependencies

Hardware dependencies encompass the physical components and infrastructure required to host and run the ShellPG-Swift system. These dependencies ensure that the system has the necessary hardware resources for efficient operation.

- *Azure Stack Infrastructure:*
  The ShellPG-Swift system is hosted on Azure Stack infrastructure. This includes servers, storage, networking equipment, and virtualization technologies provided by Azure Stack. The system depends on the availability, scalability, and reliability of these hardware resources.

- *Network Infrastructure:*
  The network infrastructure, including switches, routers, and firewalls, is essential for system communication and data transfer. It must be properly configured to support the system's requirements, including security and data traffic.

- *Server Hardware:*
  The physical servers within the Azure Stack environment host the ASP.NET middleware and other components of the system. These servers must meet performance and capacity requirements to ensure smooth operation.

- *Storage Hardware:*
  Adequate storage solutions, including disk arrays or cloud-based storage, are essential for data retention and retrieval. Proper management of storage resources is critical to prevent data loss or performance degradation.

- *Load Balancers and Redundancy Mechanisms:*
  Load balancers and redundancy mechanisms ensure high availability and fault tolerance. These hardware components distribute user requests across multiple servers and facilitate failover in case of hardware failures.

4. Components

a. React Frontend:

The React frontend remains the user-facing part of the system, providing an intuitive interface for customers to interact with ShellPG-Swift.

b. ASP.NET Middleware:

ASP.NET serves as the middleware layer, connecting the React frontend with the Azure Stack environment. It processes user requests, manages authentication, and implements business logic.

c. Azure Stack Infrastructure:

Azure Stack forms the foundation of the system's cloud-based infrastructure. It hosts the ASP.NET middleware, providing scalability, flexibility, and a secure environment.

d. PostgreSQL Database:

PostgreSQL continues to function as the database server, hosted within the Azure Stack environment. It securely stores and manages user data, bookings, and system-related information.

Functional Requirements

1. User Registration and Authentication

Users should be able to register new accounts by providing necessary details. Registered users can log in securely to access the system's features.

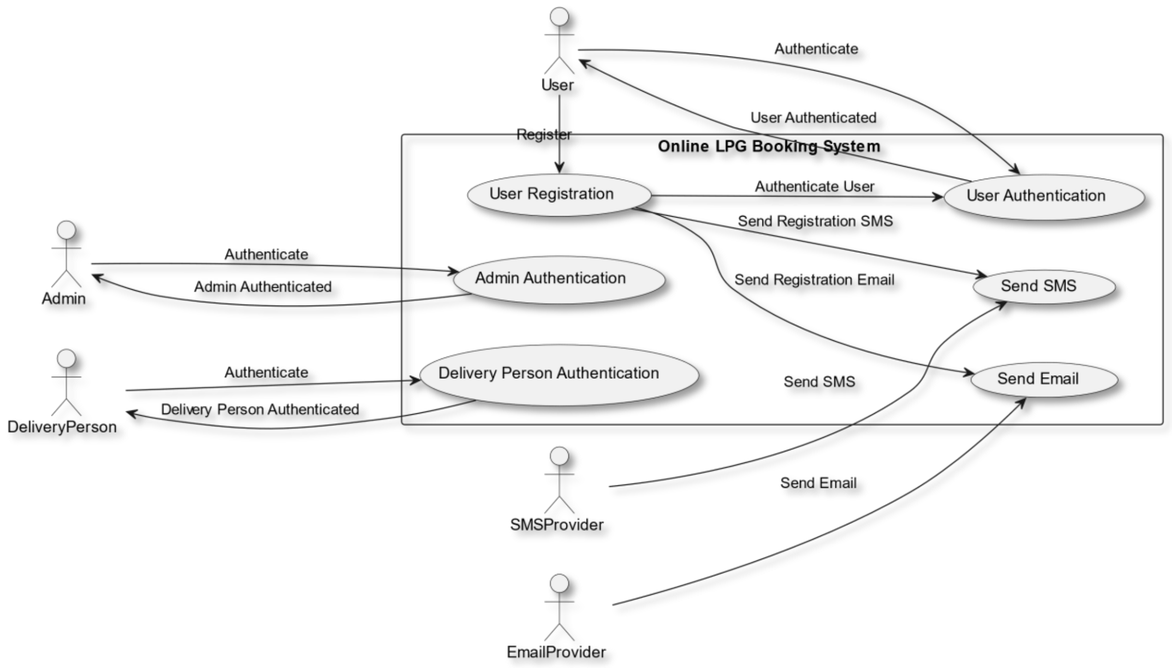

Figure 2: Use Case Diagram for User Registration and Authentication

2. LPG Booking

Users should have the ability to place new LPG gas booking orders, specifying delivery details and quantities.

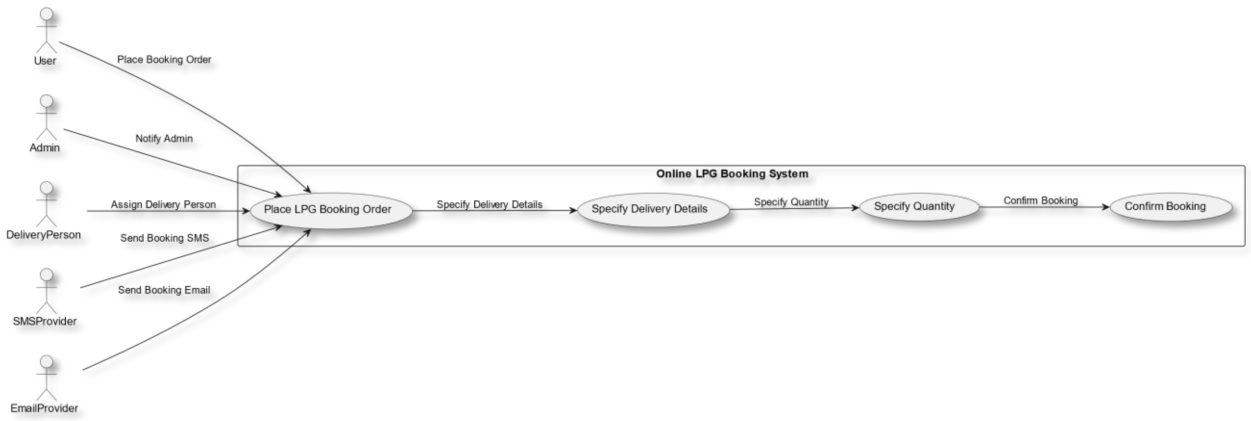

Figure 3: Use Case Diagram for LPG Booking

3. Order Modification and Cancellation

Users should be able to modify or cancel their existing LPG gas booking orders within a specified timeframe.

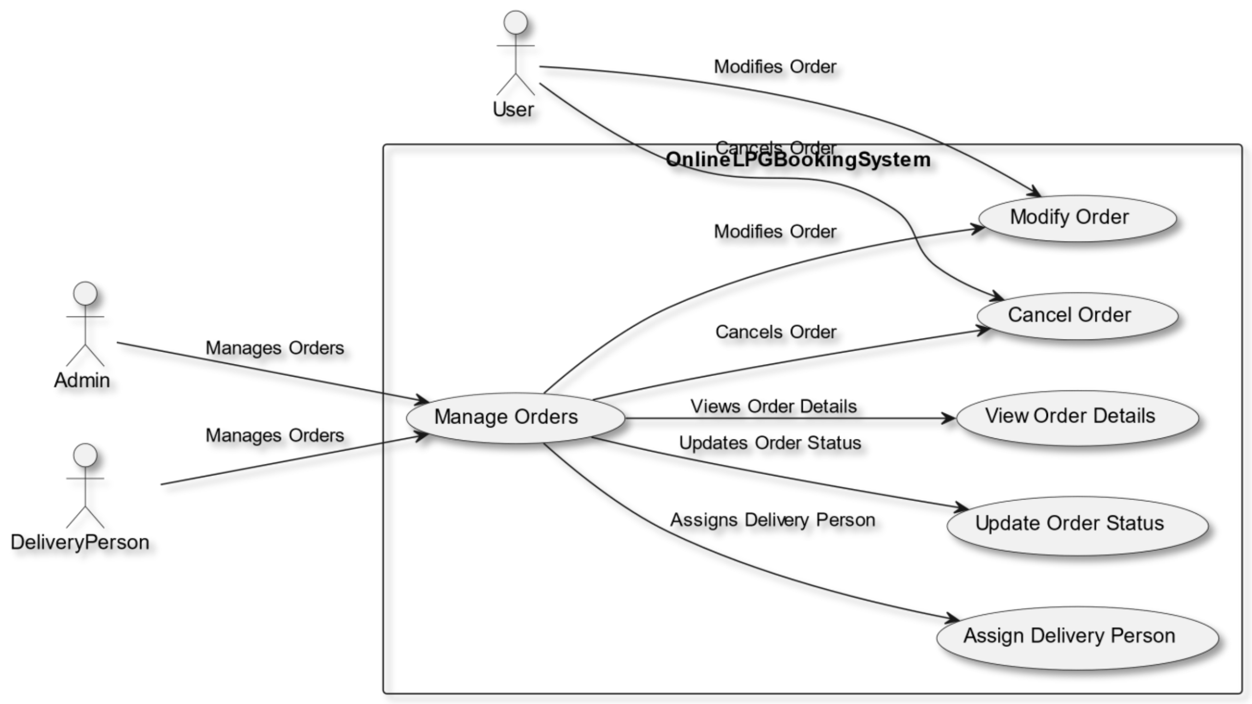

Figure 4: Use Case Diagram for Order Modification and Cancellation

4. User Support

Users should have access to customer support channels and frequently asked questions (FAQs) to seek assistance and find answers to common queries.

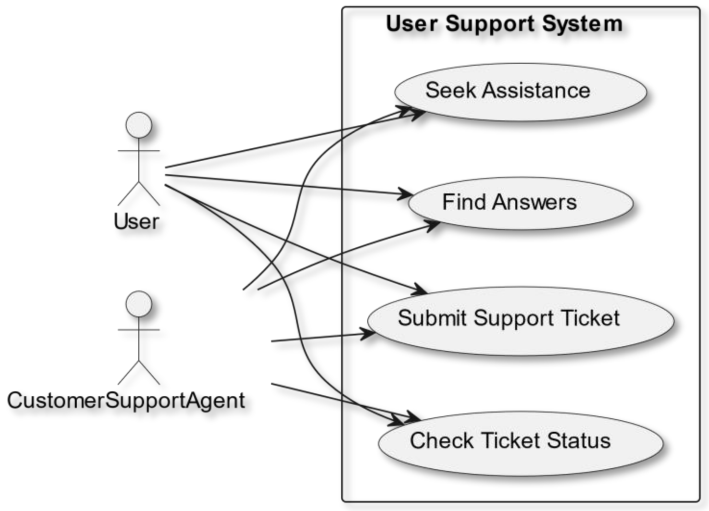

Figure 5: Use Case Diagram for User Support

Non-Functional Requirements

1. Performance:
   a. Response Time
   The system should respond to user actions within an acceptable time frame. For instance, LPG booking requests should be processed in under 60 seconds.

   b. Scalability
   The system should be capable of handling increased user loads without a significant decrease in performance. Scalability should be achieved through horizontal scaling and efficient resource allocation within the Azure Stack environment.

   c. Concurrency
   The system should support a defined number of concurrent users performing various actions without degradation in performance.

2. Security:
   a. Data Encryption
   All sensitive user data, including personal information and payment details, must be encrypted during transmission (using SSL/TLS) and when stored in the database.

   b. Authentication and Authorization
   User authentication should be secure and implemented according to best practices. Authorization mechanisms should restrict access to sensitive features based on user roles.

   c. Data Access Control
   The system should enforce access control to ensure that users can only access and modify their own data or data for which they have proper permissions.

   d. Audit Logging
   The system should log all significant security-related events, including login attempts, access to sensitive data, and unauthorized access attempts.

3. Usability:
   a. User-Friendly Interface
   The React frontend should provide an intuitive and user-friendly interface, with clear navigation, informative error messages, and accessibility features.

   b. Consistency
   Ensure consistency in design, layout, and behaviour across different sections of the user app for a cohesive user experience.

   c. Mobile Responsiveness
   The user app should be responsive and functional on various devices and screen sizes, including mobile phones and tablets.

4. Reliability:
    a. System Uptime
    The system should aim for high availability, with minimal downtime for maintenance or unexpected failures. Define a target uptime percentage (e.g., 99.9%).

    b. Fault Tolerance
    The system should have mechanisms in place to handle failures gracefully, including backup servers, automatic failover, and data recovery procedures.

5. Compatibility:
    a. Browser Compatibility:
    The React frontend should be compatible with modern web browsers, including Chrome, Firefox, Safari, and Edge.

    b. Mobile Compatibility:
    Ensure compatibility with popular mobile operating systems (e.g., iOS and Android) and browsers (e.g., Chrome Mobile, Safari).

6. Regulatory Compliance:
    a. Data Privacy:
    Ensure compliance with data privacy regulations such as GDPR (General Data Protection Regulation) or relevant local data protection laws.

    b. Payment Security:
    Since we are handling payments, the application needs to comply with PCI DSS (Payment Card Industry Data Security Standard) requirements.

7. Performance Testing:
    a. Load Testing:
    Conduct load testing to ensure the system performs well under heavy user loads, identifying bottlenecks and optimizing resource allocation as needed.

    b. Stress Testing:
    Test the system's resilience by simulating high-stress scenarios, ensuring it gracefully handles extreme conditions.

Current Scope

The initial scope of the ShellPG Swift application is limited to essential features and local deployment for tracking LPG gas deliveries. This phase serves as the foundation for future enhancements and broader functionality. The current scope includes the following key aspects:

1. Local Deployment:
    - The ShellPG Swift application will be deployed exclusively on a local server or development environment. This localized deployment allows for controlled testing and development.

2. User Registration and Authentication:
   - Users can register new accounts by providing necessary details.
   - Registered users can securely log in to the system using their credentials.
3. Delivery Tracking System:
   - The primary focus of the application is to generate a tracking ID for LPG gas deliveries.
   - Users, upon placing an order, will receive a unique tracking ID for each order they make.
4. User Profile Management:
   - Users can access and update their profiles, including personal information and contact details.
5. Basic User Interface:
   - The user interface (UI) will feature a simple, clean, and intuitive design.
   - The UI will be responsive, ensuring usability on various devices and screen sizes.
6. Notification Alerts:
   - Users will receive basic notification alerts related to order confirmations, delivery updates, and payment receipts.
7. Limited Functionalities:
   - The application's functionalities are deliberately limited to core features, emphasizing tracking and user management.
8. Data Security:
   - The application will prioritize data security, including encrypted transmission of user data and secure storage of user information.
9. No External Integrations:
   - External integrations, such as payment gateways or external APIs, will not be part of the current scope.
10. Usability and Accessibility:
    - The application will strive for basic usability and accessibility, providing alt text for images and basic keyboard navigation.

Purpose of the Current Scope:

The current scope of the ShellPG Swift application is deliberately narrow, focusing on the foundational aspects of user registration, order tracking, and user profile management. This approach allows for the following:

- **Controlled Development**: By limiting the scope, development efforts can be concentrated on essential features, ensuring their stability and quality.
- **Testing and Validation**: The local deployment allows for thorough testing and validation of core functionalities before expanding the application's reach.
- **Iterative Development**: Future phases of development can build upon this foundation, gradually introducing additional features, external integrations, and scalability measures.