# **Table of Contents**

**Introduction**

1. Purpose

1.1 Project Scope

1.2 Intended Audience and Reading Suggestions

1.3 Definitions and Acronyms

1.4 Overview

**Overall Description**

2. Product Perspective

2.1 Product Features

2.2 User Classes and Characteristics

2.3 Operating Environment

2.4 Assumptions and Dependencies

**Specific Requirement**

3. Functional Requirement

3.1 CMS Features

3.2 CSE Features

3.3 Admin & Role Dashboard

3.4 Subscription & Billing

3.5 Support System

**NON-Functional Requirement**

4.1 High Availability and Fault Tolerance

4.2 Responsive and Accessible UI

4.3 Scalable Architecture

4.4 Secure Data Handling

4.5 Performance Optimization for API Response Time

4.6 High Request Handling Capacity

4.7 Uptime Guarantee

**System Components**

5. UI/UX Design

5.1 Infrastructure & DevOps

5.2 Database Design

5.3 API Development (CSE)

5.4 Booking & Cancellation

5.5 Customer Management

5.6 Subscription & Payment

5.7 Static & Dynamic Content Enrichment

5.8 Customer Support

5.9 Reports & Finance

**API Specifications**

6.1 Search API (CSE)

6.2 Booking API

6.3 Cancellation API

**7. Performance Requirements**

**8. Security & Compliance**

**9. Backup & Disaster Recovery**

**Reporting & Analytics**

10.1 API Usage Tracking By Customer

10.2 Revenue Breakdown: Booking and Subscription

10.3 Exportable Dashboards and Financial Reports

**Supports and Maintenance**

11.1 In-App Ticketing and Live Support

11.2 Auto-Routing To Relevant Teams

11.3 SLA Timers and Ticket Priority Tracking

**12. Uses of SRS Document**

**FAQs On SRS Format**

13.1 Why is SRS Important?

13.2 What Should be included in SRS?

**Software Requirements Specification (SRS)**

# **Introduction**

## **1. Purpose**

This document provides a complete description of the Traviox Platform. It outlines all functional and non-functional requirements for development and implementation.. It details the functionalities, features, and performance expectations of the system, serving as a guide for development, testing, and implementation. It is intended for developers, testers, project managers, and stakeholders involved in the project.

### **1.1 Project Scope**

The Traviox platform will manage both static hotel data (CMS) and dynamic hotel data (CSE) from providers like Amadeus. It will support search, booking, cancellation, subscription, reporting, and support features.

### **1.2 Intended Audience and Reading Suggestions**

* **Developers:** To understand system functionality.
* **QA Team:** To verify features and functionality.
* **Business Analysts**: To ensure the system meets business requirements.
* **Project Managers:** To track module implementation.

### **1.3 Definitions, Acronyms, and Abbreviations**

* **CMS:** Content Management System
* **CSE:** Centralized Search Engine
* **MFA:** Multi-Factor Authentication
* **RBAC:** Role-Based Access Control
* **SLA:** Service Level Agreement
* **API:** Application Programming Interface
* **CI/CD:** Continuous Integration/Continuous Deployment

### **1.4 Overview**

This SRS covers modules, features, and technical expectations for the platform, including user management, subscription handling, customer support, and analytics.

# **Overall Description**

## **2. Product Perspective**

Traviox is a **modular backend system** that acts as a bridge between hotel data sources and front-end systems (B2B/B2C platforms, Website and mobile apps, third-party clients). It integrates static hotel content (CMS) with real-time availability and pricing (CSE) via APIs.

Traviox is a **new, independent product**, but it is built to integrate with:

* External hotel data providers **(e.g., Amadeus)**
* Telr payment gateway
* Admin dashboards and third-party UIs
* Customer CRMs and analytics tools

### **2.1 Product Features**

Key features of the product include:

* **CMS Module**: Manage static hotel metadata (hotels, rooms, amenities, images).
* **CSE Module**: Real-time rate and availability search via API.  
   Booking and cancellation workflow.
* **API Gateway**: Unified interface for search, booking, and content retrieval  
   Load-balanced, high-concurrency support.
* **Admin & Customer Dashboards**: Role-based access, real-time metrics, analytics, ticketing.
* **DevOps & Infrastructure**: CI/CD pipelines, failover, disaster recovery, AWS - S3-Bucket hosting.

### **2.2 User Classes and Characteristics**

* **Admin:** Full access to system configuration, user roles, CMS data, and monitoring dashboards.
* **B2B Partner:** Third-party businesses that use the API to integrate hotel content into their platforms.
* **B2C End Users:** Customers using web/mobile interfaces powered by theTraviox to search and book hotels.
* **Developers:** Internal or partner developers who consume APIs or build on top of the Traviox.
* **Support Agents:** Handle Tickets, troubleshoot issues, and ensure SLAs are met.

### **2.3 Operating Environment**

* **Cloud Stack**: AWS - S3-Bucket, EC2, Nginx / Gunicorn
* **Frontend Tools**: React.js / HTML5 / CSS3 / JavaScript **(ES6+) /**

Axios **(Fetch Api (For HTTP Requests)) /** React Router / Reduce Context API

* **Backend**: Python 3 / Django Rest Framework / Django ORM
* **Databases**: PostgreSQL
* **Other Tools & Service:**  BitBucket / CI/CD (Bitbucket Actions / Postman (API Testing)

### **2.4 Assumptions and Dependencies**

* Amadeus API will be fully available for integration and testing.
* Partner clients will conform to provided API specs and rate limits.
* Network latency and uptime are assumed to be optimal in AWS.
* End users will access the system via partner UIs or APIs, not directly.
* A stable payment gateway (Telr) will support all required operations.
* Support teams and SLA enforcement tools will be available for deployment and customer service.

# **Specific Requirements**

## **3. Functional Requirements**

### **3.1 CMS Features**

* Upload and manage static hotel metadata.
* Bulk upload of rooms and amenities.
* Image optimization and metadata validation.

### **3.2 CSE Features**

* Real-time search API for hotel pricing and availability.
* Booking and cancellation APIs.
* Multi-provider integration framework.

### **3.3 Admin & Role Dashboards**

* Manage users, roles, API access.
* Monitor customer usage, renewals.

### **3.4 Subscription & Billing**

The Traviox supports a **tiered subscription model** to manage customer access and billing for CMS and CSE API services. Each tier defines the level of access, usage limits, and available features. Customers can choose between **monthly** and **yearly** billing cycles.

### **3.5 Support System**

* Ticketing and live chat support.
* SLA and assignment tracking.

## **Non-Functional Requirements**

The **Non-Functional Requirements** define the quality attributes of the system, including its performance, reliability, scalability, and security. These ensure the system operates efficiently under expected workloads and provides a seamless experience for users and stakeholders.

### **4.1 High Availability and Fault Tolerance**

* The system must be designed to ensure **continuous availability**, even in the event of hardware failure or network outages.
* **Redundant components** (e.g., servers, databases) should be deployed across multiple availability zones.
* Automatic **failover mechanisms** must be in place to switch to backup services with minimal downtime.
* Regular **backups** and **disaster recovery plans** should be maintained and tested.

### **4.2 Responsive and Accessible UI**

* The user interface must be fully **responsive**, adapting seamlessly to different screen sizes and devices (mobile, tablet, desktop).
* It should comply with **WCAG 2.1 accessibility standards** to support users with disabilities.
* Interactive elements must have **keyboard navigation**, **screen reader support**, and sufficient **color contrast** for readability.

### **4.3 Scalable Architecture**

* The system must be able to **scale horizontally** and **vertically** to handle increased workloads and user traffic without performance degradation.
* Support for **auto-scaling** in cloud environments (e.g, AWS) must be included.
* Microservices-based architecture is preferred to independently scale services as needed.

### **4.4 Secure Data Handling**

* All data must be transmitted over **HTTPS** with TLS 1.2 or higher encryption.
* Sensitive information (e.g., passwords, tokens) must be **encrypted at rest** and **in transit**.
* The system must follow **OWASP security best practices** to prevent common vulnerabilities such as XSS, SQL injection, and CSRF.
* **Role-based access control (RBAC)** and **audit logging** must be implemented to ensure secure access and activity tracking.

### **4.5 Performance Optimization for API Response Time**

The system must ensure an **average API response time of less than 2 seconds** under normal operating conditions.

* Performance optimizations should include:  
  + **Efficient database indexing**
  + **Caching strategies** (e.g., Redis)
  + **Asynchronous processing** where appropriate
  + **Load testing and profiling** to identify bottlenecks

### **4.6 High Request Handling Capacity**

* The system must support **a minimum of 10,000 API requests per second**.
* Load balancing and rate limiting should be configured to distribute traffic efficiently and protect against abuse.
* A robust **queuing mechanism** (e.g., Kafka, RabbitMQ) may be used to handle bursts in traffic without data loss.

### **4.7 Uptime Guarantee**

* The system must provide a **99.9% uptime guarantee**, ensuring maximum availability to end users.
* Uptime is to be monitored via automated tools and reported monthly.
* Scheduled maintenance windows should be communicated in advance and kept outside of peak usage hours.

# **System Components**

## UI/UX Design

* Design responsive wireframes in Figma for all CMS & CSE pages
* Define design elements (fonts, colors, UI components)
* Create interactive prototypes for user testing
* Handoff approved designs to the development team

### **5.1 Infrastructure & DevOps**

* Configure CI/CD pipelines for seamless deployment
* Implement API gateway, load balancing, and sandbox environments
* Enable monitoring tools, compliance checks, and system alerts

### **5.2 Database Design**

* CMS DB: Store static hotel data (rooms, images, amenities)
* CSE DB: Store cache of real-time rates and availability
* Use normalized schema with indexing and failover support
* Support bulk uploads and data validation workflows.

### **5.3 API Development (CSE)**

* Search API to query hotels by location, dates, guest count, and filters
* Integrate with Amadeus (initially), supporting future providers
* Return negotiated, loyalty-based, and standard pricing
* Optimize for high-load traffic (10,000 req/sec)

### **5.4 Booking & Cancellation**

* Confirm bookings through third-party providers
* Handle cancellations and refunds based on policy
* Automate email confirmations and transaction status updates
* Apply VAT, markup, and pricing rules

### **5.5 Customer Management**

* Unified login system across CMS & CSE
* Role-based dashboards for Admin, B2B, B2C
* Track subscription renewals, activity, and API usage
* Enable access to sandbox or live APIs based on role

### **5.6 Subscription & Payment**

* Tiered plans for CMS and CSE access
* Integration with Telr for payment processing
* Generate invoices and maintain payment history
* Automated renewal reminders and subscription controls

### **5.7 Static & Dynamic Content Enrichment**

* Merge CMS static data with CSE real-time data in API responses
* Ensure complete hotel info is sent: room types, images, pricing
* Optimize image loading and metadata validation

### **5.8 Customer Support**

* Centralized ticketing system for issues and feature requests
* Real-time chat support for faster resolution
* Automate ticket assignment and SLA tracking
* Dashboard for customer support analytics

### **5.9 Reports & Finance**

* Dashboard showing revenue from bookings and subscriptions
* Exportable reports for B2B/B2C usage
* Track payment logs, invoice history, and VAT reports securely

## **API Specifications**

This section defines the core APIs exposed by the system, primarily focusing on hotel search, booking, and cancellation functionalities. These APIs enable seamless integration between the frontend, CMS, and external providers such as Amadeus.

### **6.1 Search API (CSE)**

**Purpose**:  
To fetch available hotel listings based on user-defined criteria such as travel dates, guest preferences, and rating filters.

**Filters/Parameters**:

* **check\_in\_date** & **check\_out\_date**: Specify the start and end date of the stay.
* **star\_rating:** Filter hotels by their rating (e.g., 3-star, 4-star, 5-star).
* **guest\_count:** Number of guests for the booking (used to determine room suitability and pricing).
* Optional: **location, hotel\_name, price\_range, amenities.**

**Data Source**:

* The primary data source is the **Amadeus API**, which provides real-time hotel availability, pricing, and content.
* Additional providers such as **Leonardo**, **Vervotech**, or direct hotel integrations may be added in the future to enhance the inventory and pricing accuracy.

**Response**:

A list of hotel results including:

* Hotel name, address, star rating
* Room availability and prices
* Amenities and cancellation policies
* Hotel images and descriptions (via static content or 3rd-party content sources)

### **6.2 Booking API**

**Purpose**:  
To confirm and finalize a hotel reservation using guest and payment details.

**Input Parameters**:

* **guest\_info:** Full name, contact number, email, nationality, etc.
* **room\_type:** Selected room category from search results.
* **payment\_details:** Card information, billing address, payment gateway reference.
* **check\_in\_date, check\_out\_date, guest\_count:** To match availability and pricing.
* **special\_requests:** Optional notes for the hotel (e.g., late check-in, bed preference).

**Process**:

* The system validates availability with the provider (e.g., Amadeus).
* Upon successful validation, payment is processed and booking is created.

**Response**:

* **booking\_id:** Unique identifier for the confirmed reservation.
* **confirmation\_code:** Provider’s booking reference for customer use.
* **status:** Confirmation status (e.g., Confirmed, Pending, Failed).
* **invoice\_url:**  Link to downloadable invoice or booking summary.

### **6.3 Cancellation API**

**Purpose**:  
 To cancel an existing hotel booking and return applicable refund and policy information.

**Input Parameters**:

**booking\_id:** Unique ID generated during the booking process that identifies the reservation to be canceled.

**Process:**

* The system verifies the booking ID and retrieves cancellation eligibility based on the provider’s rules.
* If permitted, a cancellation request is sent to the provider (e.g., Amadeus).
* Refund (if applicable) is calculated and initiated through the original payment method.

**Response:**

* **cancellation\_status:** Indicates whether the cancellation was successful or failed.
* **refund\_amount:** The total refundable amount (if any).
* **cancellation\_policy:** Describes the penalty, refund conditions, and cutoff times.
* **refund\_status:** Indicates whether the refund has been processed, is pending, or not applicable.

## **7. Performance Requirements**

* The system must ensure API response times are under **2 seconds for 95%** of all requests to maintain a smooth user experience.
* It must be capable of handling **10,000 concurrent API requests per second** without degradation in performance.
* All dashboard pages should load in **under 3 seconds**, even during peak usage periods.

## **8. Security & Compliance**

**OAuth2 & JWT Tokens**: All API access is secured using OAuth2 and JWT tokens for authentication and session management.  
  
**GDPR Compliance**: User data is handled according to GDPR standards, including user consent, data access, and deletion rights.  
  
**Secure Payments via Telr**: Payments are processed through Telr using PCI-DSS compliance, 3D Secure, and no card data is stored on the platform.

**HTTPS & Input Validation**: All data is transmitted over HTTPS. Input is validated to prevent XSS, SQL injection, etc.

**Audit Logs**: All critical actions are logged with timestamps and user details for traceability and security audits.

## **9. Backup & Disaster Recovery**

* Daily incremental & weekly full backups
* Rollback support for critical failures
* Alerts for backup failures
* Documented disaster recovery procedures

## **Reporting & Analytics**

The **Reporting & Analytics** module provides critical insights into platform performance, customer engagement, and financial health. It enables administrators and stakeholders to monitor API usage, track revenue streams, and generate exportable reports for informed decision-making.

### **10.1 API Usage Tracking by Customer**

**Customer-Level Tracking**: The system monitors API consumption on a per-customer basis, allowing visibility into which clients are making the most requests and what services they are using.

* **Usage Metrics**:  
  + Total number of API calls per customer
  + Most frequently accessed endpoints
  + Peak usage times and volumes
  + Error rates per customer

**Alerts & Throttling**: Optionally, admins can set usage limits or thresholds and receive alerts when customers approach or exceed predefined limits.

**Visualization**: Interactive graphs and charts display API usage trends over selectable time ranges (daily, weekly, monthly).

### **10.2 Revenue Breakdown: Bookings and Subscriptions**

* **Categorized Revenue Reporting**:  
  + **Bookings**: Revenue generated from hotel, limousine, restaurant, and other travel-related bookings.
  + **Subscriptions**: Recurring revenue from customers subscribed to premium services or platform features.

* **Filters and Sorting**:  
  + View revenue by service type, region, date range, or customer segment.
  + Compare monthly/quarterly growth and trends.
* **Tax and VAT Handling**: Reports include tax/VAT components for accurate financial analysis and compliance.

### **10.3 Exportable Dashboards and Financial Reports**

* **Customizable Dashboards**:  
  + Visualize KPIs such as total revenue, customer growth, booking volume, and subscription churn rate.
  + Admins can customize widgets to track metrics relevant to their department (finance, operations, support).
* **Export Options**:  
  + Reports and dashboards can be exported in Excel (.xlsx), PDF, and CSV formats.
  + Scheduled exports via email or secure download links.
* **Access Control**:  
  + Role-based access ensures that sensitive financial data is only visible to authorized personnel.
  + Audit logs record who accessed or exported specific reports.

## **Support & Maintenance**

The **Support & Maintenance** module is designed to ensure continuous operational stability and quick resolution of user issues. It includes built-in tools for ticketing, live support, and automated escalation to relevant support teams. The primary features include:

### **11.1 In-App Ticketing and Live Support**

**Integrated Support System**: Users can create support tickets directly from within the application interface. The ticketing system captures necessary information such as issue description, category, affected module, severity, and user contact details.

**Live Chat Support**: A real-time live chat feature is embedded within the platform to allow users to connect instantly with a support representative during business hours.

**Ticket Reference ID**: Every ticket is assigned a unique reference ID, which helps in tracking the progress and history of the issue.

**Attachment Support**: Users can upload screenshots or documents to provide more context for the issue.

### **11.2 Auto-Routing to Relevant Teams**

**Smart Routing Engine**: The system automatically routes tickets to the appropriate support team (e.g., Technical, Billing, Admin) based on predefined rules, such as keyword matching, category selection, or past history.

**Role-Based Escalation**: If a ticket remains unresolved beyond a certain threshold, it will be escalated to a higher-level technician or team lead automatically.

**Team Notification**: Assigned team members receive real-time notifications (via email or in-app) when a ticket is routed to them.

### **11.3 SLA Timers and Ticket Priority Tracking**

**Service Level Agreements (SLAs):** Each ticket is associated with an SLA based on the priority level (e.g., High, Medium, Low). The system tracks these timers and highlights tickets nearing or breaching their SLA deadlines.

* **Priority Matrix:**  
  + High Priority: Response within 1 hour, resolution within 4 hours.
  + Medium Priority: Response within 4 hours, resolution within 24 hours.
  + Low Priority: Response within 12 hours, resolution within 72 hours.

**Real-Time Dashboard:** A dedicated dashboard provides analytics on open tickets, resolved issues, SLA compliance, and average resolution time.

**Reminders & Escalations:** Automatic reminders are sent to support staff for approaching SLA breaches, and escalation protocols are triggered if a ticket remains unattended beyond SLA limits.

## **Uses of SRS Document**

* Development & Implementation.
* Testing & Quality Assurance.
* Future Enhancements.

## **FAQs on SRS Format**

### **13.1 Why is SRS important?**

It defines clear system requirements, clearly defining the scope, functionalities, and constraints, ensuring all stakeholders (developers, testers, clients) have a unified understanding of what needs to be built, minimizing misunderstandings and potential project deviations throughout the development process

### **13.2 What should be included in SRS?**

A software requirement specifications (SRS) document lists the requirements, expectations, design, and standards for a future project. These include the high-level business requirements dictating the goal of the project, end-user requirements and needs, and the product's functionality in technical terms.

|  |
| --- |

|  |
| --- |