1. Hotel Management System

Problem Statement:

Many small to medium-sized hotels face significant operational inefficiencies due to manual or fragmented processes for reservations, check-in/out, room management, billing. This leads to many problems like slow service, increased errors and ultimately reduced guest satisfaction & revenue An integrated, user-friendly Hotel Management System (HMS) is needed to automate these core functions and enhance guest experience.

SRS Document:

1. Introduction

1.1 Purpose
Documents all requirements for the Hotel Management system to guide development, testing and stakeholder understanding.

1.2 Scope
Web based HMS for core hotel operations such as reservations, check in/out, room allocation and billing etc. supports front desk and administration roles. Initial focus is internal operations. Future Integrations (payment gateways, OTAs) are not in this release

1.3 overview

HMS is a web application with a centralized database. It automates various tasks such as reservations, check in/out and invoicing. Housekeeping updates room status. Aims to improve efficiency and accuracy

2. General Description

2.1 Product Perspective

standalone web based system replacing manual processes. Centralized platform for room operations, no specific hardware beyond standard computers

2.2 Product Functions

Reservation management, check in/out, room status/allocation, billing, guest profiles, user role management, housekeeping tasks

2.3 User characteristics

Frontdesk : Fast, user-friendly interface for guest interactions

Housekeeping : simple, mobile friendly interface for status updates

Administrators : comprehensive access for configuration and reporting

2.4 General Constraints

Python backend, React frontend, SQL database. Accessible via standard web browsers, strict security for guest data. Scalable & multiple users

3. Functional Requirements
- The system shall allow customers to search and book rooms online
- The system shall display room-availability in real time
- It shall allow staff to check-in & check-out guests
- It shall generate e-invoices & support multiple payment modes
- It shall allow administrators to manage room types, rates & services
- It shall maintain customer records & history
- It shall send email/SMS notifications for booking, cancellations and reminders
- It shall allow managers to generate reports
- It shall allow staff to log maintenance & housekeeping tasks

4. Interface Requirements

4.1 User Interfaces
Customer Interface: Online booking portal with interactive search filters
Staff Interface: Dashboard for book-in, check in/out and billing
Admin Interface: User management, system settings, reporting

4.2 Hardware Interfaces
Barcode/QR scanner for guest check in/out
POS terminals for payment

4.3 Software Interfaces
Integration with third party payment APIs
Email/SMS gateways

5. Performance Requirements
system shall support upto 500 concurrent users
It shall display room availability results within 2 seconds
It shall achieve 99.5% uptime
It shall process bookings & payments within 5 seconds

6. Design constraints
Must comply with GDPR & data privacy regulations
Must follow OWASP
Must support both desktop & mobile browsers
Must be scalable

7. Non Functional Attributes
Security : Data encryption for customer & payment data
Reliability : system should be reliable
Usability : simple User Interface
Maintainability : Modular code structure
Scalability : should be able to accomodate change

8. Preliminary schedule and Budget

8.1 Schedule
Month 1 : Requirement gathering & design
Month 2 : Database & backend development
Month 3 : Frontend development
Month 4 : Integration of payment & notification modules
Month 5 : Testing
Month 6 : Deployment & Staff Training

8.2 Budget
Development Team (5 members, 6 months) : $60,000
Cloud Hosting & infrastructure : $5,000

Licenses & Tools : $ 3,000
Testing & QA : $ 7,000
Training & Maintenance : $ 5,000
Total : $ 80,000

2. Credit Card Processing

Problem statement
Small to medium businesses face inefficient, insecure, costly credit card processing due to manual or fragmented systems. This results in high fees, errors, delayed reconciliation and non compliance. A secure, efficient and integrated CCPS is needed to streamline transactions, cut costs, boost security and provide real-time financial reporting.

SRS Document :

1. Introduction

1.1 Purpose: To define all requirements for the CCPS, guiding development and ensuring common understanding.

1.2 Scope: A secure, web based system for credit/debit card transactions. Integrates with third party payment gateways. Provides transaction history & basic merchant reporting. No physical POS or direct bank integration

1.3 Overview: CCPS medistes between merchants & payment gateways, securely processing transactions via web UI/API, storing non sensitive data & providing oversight

2. General Description

2.1 Product Perspective
Standalone web app / API service. Integrates with existing merchant systems but doesn't manage inventory / customers

2.2 Product Functions
Transaction processing (auth, capture, void, refund) gateway integration, secure tokenization transaction history, basic reporting.

2.3 User Characteristics
Merchants: Need easy processing, void / refund, transaction viewing
Administrators: system config, user management, detailed reporting
Developers: clear API documentation for integration

2.4 General constraints
Django backend, React frontend, SQL database. Must be PCI DSS compliant. Integrates via established gateway APIs. High scalability for transaction volumes

3. Functional Requirements
- User Authentication & Authorization: secure log in, role-based access
- Merchant Onboarding & Management: Admin can create / manage merchant accounts, configure gateway / currency
- Transaction processing: Auth, Capture, void unsettled, Refund settled transactions. Tokenize card data

- Payment gateway Integration : support integration with > 2 gateways via APIs
- Transaction & Search history : Search transaction by date, amount, status, last 4 digits, reference
- Reporting : Generate daily/weekly/monthly transitions by summaries. Export CSV

4. Interface Requirements

4.1 User Interfaces :
Web Application : Responsive dashboard, transaction forms, history/reports views

4.2 Hardware Interfaces :
None directly, relies on merchant's existing infrastructure (computer, internet)

4.3 Software Interfaces :
Payment Gateway APIs : Integration with chosen gateway APIs
Merchant system APIs : CCPS's Restful API for external systems

4.4 Communications Interfaces
Network : All communication via HTTPS
Data : JSON for API

5. Performance Requirements :
Response Time : Auth/capture < 1s ; Void/Refund < 2s ; History search < 3s (1 month data)
Transaction volume : 100 transactions/second with 99.99% success

Scalability : support 1000 concurrent API requests
Availability : 99.9% uptime
Data consistency : Real time transaction status sync with gateway

6. Design Constraints :
Development : Linux, standard IDEs.
Languages : Python (Django) backend, JavaScript (React) frontend
Security & Database : SQL, strong encryption, PCI DSS compliance
Open source & Cloud : Deployed on reputable cloud platform in a secure region, well maintained components

7. Non-Functional
Security : adherence to regulations, data encryption, role based access, regular vulnerability testing.
Reliability : automated recovery & error handling
Maintainability : Modular design, high testability
Portability : cross browser compatibility
Usability : Intuitive & efficient interfaces for users.

8. Preliminary schedule and Budget :
8.1 Schedule

Phase	Durtion (weeks)
Planning & Requirements	4
Detailed Design	5
Backend Dev	10
Frontend Dev	8
Security & PCI Prep	4
Integration & testing	6
Deployment & Launch	2
Total Project Durtion	~ 37 weeks

8.2 Preliminary Budget

Personnel Costs : $75,000
Infra & Tools : $5,000
PCI DSS certification : $10,000
Contingency (15%) : $13,500
Total Budget : $103,500

3. Library Management System :

Problem statement :
Many libraries struggle with inefficient manual or outdated systems for managing books, patrons, circulation. This leads to slow service, misplaced items, tracking issues & low patron satisfaction. A modern automated Library Management System (LMS) is needed to streamline operations and improve the overall library experience.

SRS Document :

1. Introduction

1.1 Purpose :
To automate library operations like book management, member management, borrowing/returns and fines

1.2 Scope :
Web based system with roles : Admin, Librarian, member. Provides book search, reports & secure access

1.3 Overview :
Replaces manual records with a centralized database & user friendly user interface

2. General Description

2.1 Product Perspective
A standalone web application for small to medium sized libraries, replacing manual systems

2.2 Product Functions
Includes cataloging, patron registration, circulation, fine management, search & browse, resource reservation & basic reporting

2.3 User characteristics
Admin : manage system
Librarian : manage books / members
Member : search & borrow books

2.4 General Constraints
Web based Django for backend, React for frontend, SQL database and strong security for patron data & scalability

3. Functional Requirements :
- User and Role Management - Secure login, Admin / Librarian / Member roles, access control
- Catalog Management : Add / edit / delete resources, manage details, update availability
- Circulation management - Issue, Return, renew items, automatically track overdue status and calculate fines
- Search / browse and reserve : Members can search resources by various criteria and reserve items, system notifies when available
- Reporting - Generate reports
- Fine management - Apply, collect & waive fines and record payments.

4. Interface Requirements
UI : simple web interface, dashboards, search bar
Hardware : server (8 GB RAM, Android 500 GB storage).
Software : My SQL database, backend python / Django, frontend react

5. Performance Requirements
Support 100 concurrent users
Book search response < 2 seconds
Handle 50,000+ records
Availability - 99% uptime

6. Design Constraints
web based, SQL compliance, cross browser support
Restricted access for sensitive data information

7. Non Functional Attributes
Reliability : 99% uptime
Security : Encrypted login, role based access
Usability : Easy to interact user - Interface
Maintainability : Modular design
Portability : Runs on MacOS / Windows / Linux

8. Preliminary schedule and Budget :
8.1 Schedule (6 months)
Requirement Analysis - 3 weeks
Design - 4 weeks
Development - 10 weeks
Testing - 4 weeks
Deployment - 2 weeks

8.2 Budget (£30,000)

Hardware : $ 5,000
software : $ 3,000
development : $ 12,000
Testing : $ 5,000
Training : $ 3,000
Maintenance : $ 2,000 / year

4. Stock Maintenance System

Problem Statement :
Manual / outdated inventory management leads to inaccurate inventory, stockouts / overstock, inefficient operations, poor decision making. A secure, efficient and real time stock Maintenance system (SMS) is needed to optimize inventory streamline movements and provide actionable insights

SRS Document :

1. Introduction

1.1 Purpose :
Automate the management of stock in warehouse / shops, including item entry, updates, tracking and reporting

1.2 Scope : The system will track stock levels generate alerts for low inventory, manage supplier & purchase making details & provide reports for decision making. Users are Admin, store Manager and Staff

1.3 Overview : Web - based system with a centralized database ensuring accurate, real time inventory management

2. General Description

2.1 Users :
Admin - configure system, manage users
Store Manager - Manage inventary, suppliers & reports
staff - Update stock Out/in entries

2.2 Dependencies : stable, regular database backups, user authentication

3. Functional Requirements

Stock Management : Add/update/delete items, record stock in/out

supplier Management : Maintain supplier details, purchase orders

Alerts : Notify when stock falls below threshold

Reports : Daily/weekly stock summary, shortage alerts

Security : Role-based access for admin, manager and staff

4. Interface Requirements :

UI : simple web dashboard, search bar, alerts panel

Hardware - Server (8 GB RAM, quad core, 500 GB storage)

software : MySQL / DB, backend (python Django), frontend (React js)

6. Design constraints

Web based, SQL - compliant, cross browser support

Access restricted to authorized roles only

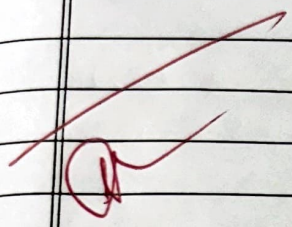
7. Non Functional Attributes
- Reliability : 99% uptime
- Security : Encrypted login, restricted access
- Usability : Easy interface for staff
- Maintainability : modular, upgradable
- Portability : works on windows / mac / Linux

8. Preliminary Schedule & Budget

Schedule (5 months)
Analysis : 2 weeks
Design : 3 weeks
Implementation : 8 weeks
Testing : 3 weeks
Deployment : 2 weeks

Budget ($25,000)
Hardware - $4,000

Software - $ 3,000
Development - $ 10,000
Testing - $ 4,000
Training - $ 2,000
Maintenance - $ 2,000 / year



Passport Automation System

Problem Statement :
The manual passport application process is slow, error-prone, and lacks transparency. Applicants face delays & difficulty tracking their status, while officers handle heavy paperwork & risk errors. A PAS is needed to digitize applications, document verification, status tracking, approvals ensuring fast, secure & efficient processing.

SRS Document

1. Introduction

Purpose : To automate the process of passport application, verification and issuance, reducing manual effort & ensuring faster processing

Scope : Web-based where applicants can apply online, upload documents, schedule appointments, track status & receive notifications. officials can verify applications, approve/reject them & generate reports. Roles include Admin, Passport officer, and Applicant

overview: Centralized, secure system integrated with government databases

to improve efficiency, transparency & accuracy.

2. General Description

- Users:
- Admin: Manage officers, System settings
- Passport officer: verify documents, process applicants
- Applicant: Apply, track, and download status updates

- Dependencies: Internet connectivity, Secure authentication, integration with national ID, police database

3. Functional Requirements

Application Management: Submit passport applications, upload documents, schedule appointments

Verification: officers verify documents, conduct background checks, approve/reject applications

Status Tracking: Applicants track their application progress online

Notifications: Automated SMS/Email updates for status changes

Report: Generate reports on applications, approvals, rejections

4. Interface Requirements

UI : Web portal & mobile-friendly interface, status dashboard

Hardware : central server with backups

Software : Secure web server, RDBMS (MySQL), backend (Java), frontend (HTML/CSS/JS)

5. Performance Requirements
support 500+ concurrent users
Application submission response < 3 seconds
Handle 1,000,000+ application records

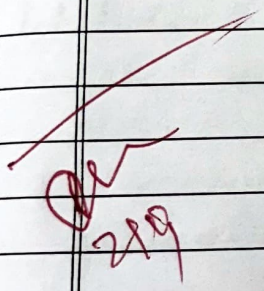
6. Design Constraints
Must comply with government security policies
only authorized users can approve/reject applications
Integration with national ID & police verification systems

7. Non-Functional Attributes
- Security : End to end encryption, role based access
- Reliability : 99.5% uptime
- Usability : simple interface for citizens
- Maintainability : scalable for future modules
- Portability : Accessible via web and mobile devices

8. Preliminary Schedule & Budget

Schedule (8 months)
Analysis : 1 month
Design : 1 month
Implementation : 3 months
Testing : 2 months
Deployement : 1 month

Budget (~ $50,000)
Infrastructure : $15,000
Development : $20,000
Testing : $7,000
Training & Docs : $5,000
Maintenance : $3,000 / year



classmate
Date
Page

Mini Project

Problem statement :
online learners face information overload from PDF's, notes and textbooks Lack of personalized study tools limits engagement & retention, existing e-learning platforms don't integrate summarization, flashcards, semantic search & voice interaction in one ecosystem. Need for an AI - powered assistant that makes studying faster, interactive & adaptive.

Introduction

1.1 Purpose of the Document :
This document defines requirements for developing an AI based personalized learning platform, guiding developers, stakeholders and testers

1.2 Scope : A scalable system for students, educators, & professionals that offers automatic summarization, flashcards, MCQ generation, semantic search, voice interaction & concept mapping

1.3 Overview : Uses advanced AI / ML to deliver adaptive learning via web and voice interfaces

2. General Description
The platform tackles information overload by providing an integrated ecosystem with summarization, quizzes and accessibility features, designed for web and mobile deployement, cloud-ready and scalable

3. Functional Requirements
- Summarization : Generate concise summaries from PDF/TXT
- Flashcards & MCQs : Auto generate quizzes using NLP
- Semantic Search : Personalized results using embeddings & knowledge graphs
- Voice Interaction : speech to text navigation & Q&A
- Concept Mapping : Visualize topic relationships
- User Profiling : Customised learning paths based on history & preferences

4. Interface Requirements
- Summaries / search within 5 sec
- Quiz generation ≤ 10 sec per document
- 500+ concurrent users supported
- Voice accuracy ≥ 90% in quiet settings

5. Performance Requirements

- Web UI : Upload / view materials, summaries, quizzes, maps
- Voice Interface : For navigation and Q&A
- API Layer : Rest APIs for integration
- Admin Dashboard : Manage users, resources and analytics

6. Design Constraints

- Cloud ready microservices architecture
- ML models retrainable with new data
- GDPR compliant handling of student info
- Robust voice components

7. Non Functional Requirements

- Security & Privacy : Encrypted, access-controlled
- Maintainability : Modular, documented code
- Usability : Accessible UX for disabled users
- Reliability : 99.5% uptime
- Scalability : Future growth support

8. Schedule and Budget

Schedule :

Requirement Analysis : 2 weeks
Design & Architecture : 2 weeks
Model Integration & development : 4 weeks
Testing : 2 weeks
Deployement & User Feedback : 2 weeks

Budget Estimate

Cloud Hosting & Compute : 500 $ / month
ML Model Training & API usage : $700
Development Team (4 members) : $4000
Miscellaneous : $ 500

Total estimated budget : $ 5700 for initial launch & 6 months operation