# System Design Document (SDD)  
  
## 1. Introduction  
  
This System Design Document (SDD) provides a detailed description of the design for the new system. It outlines the system's architecture, components, interfaces, data model, and security considerations. This document serves as a blueprint for the development team and stakeholders, ensuring a common understanding of the system's requirements and design.  
  
## 2. Requirements  
  
### 2.1 Functional Requirements  
  
#### Must Have  
  
\* \*\*FR001: User Authentication:\*\* The system shall provide user authentication functionality, allowing users to log in with a username and password.  
 \* Description: The system must verify user credentials against a stored database of authorized users.  
\* \*\*FR002: User Registration:\*\* The system shall allow new users to register by providing necessary information (e.g., name, email, password).  
 \* Description: The system must validate user input during registration and ensure password security.  
\* \*\*FR003: Data Input:\*\* The system shall provide interfaces for users to input data related to customer orders, product information.  
 \* Description: Input interfaces should include appropriate validation to ensure data accuracy.  
\* \*\*FR004: Data Storage:\*\* The system shall store all entered data securely in a database.  
 \* Description: The database design should accommodate the expected data volume and growth.  
\* \*\*FR005: Data Retrieval:\*\* The system shall allow users to retrieve stored data based on various search criteria.  
 \* Description: The system should provide efficient search capabilities.  
\* \*\*FR008: Role-Based Access Control:\*\* The system shall implement role-based access control to restrict access to certain functionalities based on user roles.  
 \* Description: Different user roles (e.g., administrator, manager, user) should have different permissions.  
  
#### Should Have  
  
\* \*\*FR006: Data Modification:\*\* The system shall allow authorized users to modify existing data.  
 \* Description: Data modification should be tracked with audit logs.  
\* \*\*FR007: Reporting:\*\* The system shall generate reports based on stored data.  
 \* Description: Reports should be customizable and exportable in various formats (e.g., PDF, CSV).  
\* \*\*FR009: Workflow Management:\*\* The system shall support order processing, document approval.  
 \* Description: The system should guide users through the defined workflow steps.  
  
#### Could Have  
  
\* \*\*FR010: Notifications:\*\* The system shall send notifications to users based on specific events.  
 \* Description: Notifications can be sent via email, SMS, or in-app messages.  
  
#### Won't Have  
\* None  
  
### 2.2 Non-Functional Requirements  
  
#### Must Have  
  
\* \*\*NF002: Security:\*\* The system shall protect sensitive data from unauthorized access and modification.  
 \* Description: Security measures should include encryption, access controls, and regular security audits.  
\* \*\*NF004: Reliability:\*\* The system shall be available 99.9% of the time.  
 \* Description: The system should be designed for high availability and fault tolerance.  
\* \*\*NF009: Data Integrity:\*\* The system shall ensure the accuracy and consistency of data.  
 \* Description: Data validation and error handling should be implemented.  
\* \*\*NF010: Compliance:\*\* The system shall comply with relevant industry regulations and standards.  
 \* Description: The system should be designed to meet legal and regulatory requirements.  
  
#### Should Have  
  
\* \*\*NF001: Performance:\*\* The system shall respond to user requests within 2 seconds under normal load conditions.  
 \* Description: System performance should be optimized for speed and efficiency.  
\* \*\*NF003: Usability:\*\* The system shall be easy to use and understand by users with varying levels of technical expertise.  
 \* Description: The user interface should be intuitive and user-friendly.  
\* \*\*NF005: Scalability:\*\* The system shall be able to handle increasing user loads and data volumes without significant performance degradation.  
 \* Description: The system architecture should be scalable to accommodate future growth.  
  
#### Could Have  
  
\* \*\*NF006: Maintainability:\*\* The system shall be designed for easy maintenance and modification.  
 \* Description: The codebase should be well-structured and documented.  
\* \*\*NF007: Portability:\*\* The system shall be able to run on different platforms (e.g., Windows, Linux, macOS).  
 \* Description: The system should be platform-independent.  
\* \*\*NF008: Availability:\*\* The system shall be available during specified business hours.  
 \* Description: Maintenance windows should be scheduled outside of these hours.  
  
#### Won't Have  
\* None  
  
## 3. System Architecture  
  
### 3.1 Component Diagram  
  
```mermaid  
%% Component Diagram  
  
classDiagram  
 class User {  
 +Username  
 +Password  
 +Email  
 +Name  
 }  
  
 class WebApplication {  
 +UI  
 +Authentication()  
 +DataInput()  
 +DataRetrieval()  
 +RoleBasedAccessControl()  
 }  
  
 class AuthenticationService {  
 +Authenticate(Username, Password)  
 +RegisterUser(User)  
 +Authorize(User, Permission)  
 }  
  
 class DataService {  
 +StoreData(Data)  
 +RetrieveData(Criteria)  
 +ModifyData(Data)  
 }  
  
 class Database {  
 +UsersTable  
 +DataStore  
 }  
   
 class RBACService {  
 +AssignRole(User, Role)  
 +CheckPermission(User, Permission)  
 }  
  
 WebApplication -- User : Uses  
 WebApplication -- AuthenticationService : Uses  
 WebApplication -- DataService : Uses  
 WebApplication -- RBACService : Uses  
 AuthenticationService -- Database : Reads/Writes  
 DataService -- Database : Reads/Writes  
 RBACService -- Database : Reads/Writes

### 3.2 Data Flow Diagram

%% Data Flow Diagram  
  
graph LR  
 A[User] --> B(Web Application);  
 B --> C{Authentication Service};  
 C --> D((Database));  
 C --> B;  
 B --> E{Data Service};  
 E --> D;  
 D --> E;  
 E --> B;  
 B --> F{RBAC Service};  
 F --> D;  
 D --> F;  
 F --> B;  
 B --> A;  
 style D fill:#f9f,stroke:#333,stroke-width:2px

## 4. Data Model

### 4.1 Entity Relationship Diagram (ERD)

erDiagram  
 User {  
 int userID PK  
 string username  
 string password  
 string email  
 string name  
 string registrationDate  
 }  
  
 Role {  
 int roleID PK  
 string roleName  
 string description  
 }  
  
 Permission {  
 int permissionID PK  
 string permissionName  
 string description  
 }  
  
 Data {  
 int dataID PK  
 string dataContent  
 string dataType  
 string dateCreated  
 int userID FK  
 }  
  
 AuditLog {  
 int logID PK  
 string eventType  
 string timestamp  
 string description  
 int userID FK  
 int dataID FK  
 }  
  
 Report {  
 int reportID PK  
 string reportName  
 string reportType  
 string dateGenerated  
 int userID FK  
 }  
   
 Workflow {  
 int workflowID PK  
 string workflowName  
 string description  
 string status  
 int userID FK  
 }  
   
 Notification {  
 int notificationID PK  
 string notificationType  
 string message  
 string timestamp  
 int userID FK  
 }  
  
 User ||--o{ Data : has  
 User ||--o{ AuditLog : performed  
 User ||--o{ Report : generated  
 User ||--o{ Workflow : initiated  
 User ||--o{ Notification : received  
  
 Role ||--o{ User : assigned  
 Role ||--o{ Permission : has  
  
 Permission ||--o{ Role : granted  
  
 Data ||--o{ AuditLog : associated

## 5. API Specification

This document defines the API endpoints, request/response formats, and security protocols for the system.

### 5.1 Authentication

| Endpoint | Method | Description | Request Body | Response (Success) | Response (Error) | Security |
| --- | --- | --- | --- | --- | --- | --- |
| /api/auth/login | POST | Authenticates a user and returns a JWT token. | json { "username": "string", "password": "string" } | json { "token": "string", "expiration": "datetime" } | json { "error": "string", "message": "string" } | Basic Auth |
| /api/auth/register | POST | Registers a new user. | json { "username": "string", "password": "string", "email": "string", "name": "string" } | json { "message": "User registered successfully", "userID": "int"} | json { "error": "string", "message": "string" } | None |
| /api/auth/refresh | POST | Refreshes the JWT token. | json { "refreshToken": "string" } | json { "token": "string", "refreshToken": "string", "expiration": "datetime" } | json { "error": "string", "message": "string" } | JWT |

**Security:**

* Authentication is performed using Basic Authentication for the /api/auth/login endpoint.
* Subsequent requests require a valid JWT (JSON Web Token) in the Authorization header: Bearer <token>.
* Consider using refresh tokens with short JWT expiry times to enhance security.

### 5.2 Data Input (Example: Customer Orders)

| Endpoint | Method | Description | Request Body | Response (Success) | Response (Error) | Security |
| --- | --- | --- | --- | --- | --- | --- |
| /api/orders | POST | Creates a new customer order. | json { "customerID": "int", "orderDate": "date", "items": [ { "productID": "int", "quantity": "int", "price": "decimal" } ], "shippingAddress": "string", "billingAddress": "string" } | json { "orderID": "int", "message": "Order created successfully" } | json { "error": "string", "message": "string" } | JWT |

**Data Validation:**

* The API should perform validation on all input data, including data types, required fields, and format.
* Return appropriate error messages to the client if validation fails.

### 5.3 Data Retrieval (Example: Customer Orders)

| Endpoint | Method | Description | Query Parameters | Response (Success) | Response (Error) | Security |
| --- | --- | --- | --- | --- | --- | --- |
| /api/orders/{orderID} | GET | Retrieves a specific order by its ID. | None | json { "orderID": "int", "customerID": "int", "orderDate": "date", "items": [ { "productID": "int", "quantity": "int", "price": "decimal" } ], "shippingAddress": "string", "billingAddress": "string" } | json { "error": "string", "message": "string" } | JWT |
| /api/orders | GET | Retrieves a list of orders based on search criteria. | customerID: “int”, orderDate: “date”, status: “string” | json [ { "orderID": "int", "customerID": "int", "orderDate": "date", "status": "string" }, ... ] | json { "error": "string", "message": "string" } | JWT |

**Filtering and Pagination:**

* Implement filtering using query parameters to allow users to search for specific data.
* Use pagination for large datasets to improve performance and user experience. Include limit and offset parameters.

### 5.4 Data Modification (Example: Customer Orders)

| Endpoint | Method | Description | Request Body | Response (Success) | Response (Error) | Security |
| --- | --- | --- | --- | --- | --- | --- |
| /api/orders/{orderID} | PUT | Updates an existing order. | json { "customerID": "int", "orderDate": "date", "items": [ { "productID": "int", "quantity": "int", "price": "decimal" } ], "shippingAddress": "string", "billingAddress": "string" } | json { "orderID": "int", "message": "Order updated successfully" } | json { "error": "string", "message": "string" } | JWT |
| /api/orders/{orderID} | DELETE | Deletes an order. | None | json { "message": "Order deleted successfully" } | json { "error": "string", "message": "string" } | JWT |

**Audit Logging:**

* Implement audit logging for all data modification operations. Store user ID, timestamp, and changes made.

### 5.5 Role-Based Access Control

* The API should use JWT claims to verify user roles and permissions.
* Example: A user with the “administrator” role can access all endpoints, while a user with the “user” role can only access a subset of endpoints.
* Implement middleware to check user roles before granting access to protected resources.

**Example:**

// Middleware to check for admin role  
function requireAdmin(req, res, next) {  
 const userRole = req.user.role; // Assuming user role is in the JWT payload  
 if (userRole === 'administrator') {  
 next(); // Allow access  
 } else {  
 res.status(403).json({ error: 'Forbidden', message: 'Insufficient permissions' });  
 }  
}  
  
// Apply middleware to specific routes  
app.put('/api/admin/settings', requireAdmin, updateSettings);

### 5.6 General API Guidelines

* **Use HTTPS:** All API communication should be encrypted using HTTPS.
* **Error Handling:** Return meaningful error messages in a consistent format. Use appropriate HTTP status codes.
* **Versioning:** Implement API versioning (e.g., /api/v1/orders) to allow for future changes without breaking existing clients.
* **Rate Limiting:** Implement rate limiting to prevent abuse and ensure API availability.
* **Input Sanitization:** Sanitize all input data to prevent injection attacks.
* **CORS:** Configure CORS (Cross-Origin Resource Sharing) appropriately to allow access from authorized domains.
* **Data serialization:** All the data should be serialized using JSON format for consistency.
* **Statelessness:** API should follow statelessness for scalability.

This specification provides a foundation for building a secure and well-defined API. Specific details may need to be adjusted based on the specific requirements of the application. Remember to comply with all relevant industry regulations and standards.

## 6. Security Considerations

graph LR  
 subgraph User  
 A[User]  
 end  
  
 subgraph Web Application  
 B[Web Application]  
 C[Authentication Service]  
 D[Data Service]  
 E[RBAC Service]  
 F[Reporting Service]  
 G[Notification Service]  
 end  
  
 subgraph Database  
 H[Database]  
 end  
  
 A --> B  
 B --> C  
 B --> D  
 B --> E  
 B --> F  
 B --> G  
 C --> H  
 D --> H  
 E --> H  
 F --> H  
 G --> H  
  
 subgraph Threats  
 T1[SQL Injection]  
 T2[Cross-Site Scripting (XSS)]  
 T3[Broken Authentication]  
 T4[Sensitive Data Exposure]  
 T5[Insufficient Access Control]  
 T6[Denial of Service (DoS)]  
 T7[Data Breach]  
 T8[Compliance Violation]  
 end  
  
 T1 --> H  
 T2 --> B  
 T3 --> C  
 T4 --> H  
 T5 --> E  
 T6 --> B  
 T7 --> H  
 T8 --> B  
  
 style Threats fill:#f9f,stroke:#333,stroke-width:2px  
 style Database fill:#ccf,stroke:#333,stroke-width:2px  
 style "Web Application" fill:#ddf,stroke:#333,stroke-width:2px  
 style User fill:#eef,stroke:#333,stroke-width:2px

### 6.1 Role-Based Access Control (RBAC) Schema

| Role | Description | Permissions |
| --- | --- | --- |
| Administrator | Full access to the system. | Create, Read, Update, Delete (CRUD) on all data; Manage users and roles; Configure system settings; Generate all reports. |
| Manager | Manage data within their scope; generate reports. | Read, Update, and Create (RUC) on data within their department; Generate reports related to their department; Approve workflow requests. |
| User | Limited access to view and interact with data. | Read data; Create new data entries; Initiate workflows; Receive notifications. |
| Guest | Read-only access to specific public data. | Read-only access to a limited set of public data. |
| Auditor | Read-only access to audit logs and system activity. | Read-only access to audit logs and system activity for compliance and security monitoring purposes. |
| Support | Read-only access for troubleshooting system usage | Access to read-only data, ability to create support tickets. |

### 6.2 Encryption Strategy

1. **Data at Rest:**
   * **Database Encryption:** Implement Transparent Data Encryption (TDE) at the database level. All data stored in the database, including user credentials, customer data, and application data, will be encrypted.
   * **File System Encryption:** Encrypt any files stored on the file system, such as backups and logs, using AES-256 encryption.
2. **Data in Transit:**
   * **HTTPS/TLS:** All communication between the client and the server, and between internal services, must be encrypted using HTTPS/TLS 1.3 or higher. Use strong cipher suites and regularly update certificates.
   * **VPN:** If connecting to the system from outside the network, use a VPN with strong encryption protocols.
3. **Key Management:**
   * **Hardware Security Modules (HSM):** Use HSMs to store and manage encryption keys securely. HSMs provide a tamper-proof environment for key storage and cryptographic operations.
   * **Key Rotation:** Regularly rotate encryption keys to minimize the impact of a potential key compromise.
   * **Access Control:** Implement strict access control policies to limit access to encryption keys. Only authorized personnel should have access to key management functions.
4. **Specific Data Encryption:**
   * **Password Hashing:** Store user passwords using a strong adaptive hashing algorithm such as Argon2id or bcrypt. Use a unique salt for each password.
   * **Sensitive Data Fields:** Encrypt sensitive data fields, such as credit card numbers and social security numbers, using field-level encryption.
5. **Compliance Considerations:**
   * **GDPR:** Ensure that encryption methods comply with GDPR requirements for protecting personal data.
   * **HIPAA:** If handling protected health information (PHI), ensure that encryption methods comply with HIPAA Security Rule requirements.
   * **PCI DSS:** If processing credit card data, ensure that encryption methods comply with PCI DSS requirements.
6. **Regular Audits and Testing:**
   * **Security Audits:** Conduct regular security audits to identify potential vulnerabilities in the encryption implementation.
   * **Penetration Testing:** Perform penetration testing to simulate real-world attacks and assess the effectiveness of the encryption strategy.
   * **Key Management Audits:** Audit key management practices to ensure that keys are stored and managed securely.

This comprehensive encryption strategy ensures the confidentiality and integrity of sensitive data both at rest and in transit, while also meeting relevant compliance requirements.

## 7. Traceability Matrix

| Requirement ID | Description | Component(s) Affected | API Endpoint(s) | Test Case(s) | Status |
| --- | --- | --- | --- | --- | --- |
| FR001 | User Authentication | WebApplication, AuthenticationService, Database | /api/auth/login, /api/auth/register, /api/auth/refresh | TC001, TC002 | Implemented |
| FR002 | User Registration | WebApplication, AuthenticationService, Database | /api/auth/register | TC003, TC004 | Implemented |
| FR003 | Data Input | WebApplication, DataService, Database | /api/orders | TC005, TC006 | Implemented |
| FR004 | Data Storage | DataService, Database | /api/orders | TC007 | Implemented |
| FR005 | Data Retrieval | WebApplication, DataService, Database | /api/orders/{orderID}, /api/orders | TC008, TC009 | Implemented |
| FR006 | Data Modification | WebApplication, DataService, Database | /api/orders/{orderID} | TC010, TC011 | Should Have |
| FR007 | Reporting | WebApplication, ReportingService, Database | /api/reports | TC012, TC013 | Should Have |
| FR008 | Role-Based Access Control | WebApplication, RBACService, Database | N/A | TC014, TC015 | Implemented |
| FR009 | Workflow Management | WebApplication, Database | /api/workflows | TC016, TC017 | Should Have |
| FR010 | Notifications | WebApplication, NotificationService, Database | /api/notifications | TC018, TC019 | Could Have |
| NF001 | Performance | WebApplication, AuthenticationService, DataService, Database | All | TP001 | Should Have |
| NF002 | Security | All | All | TS001, TS002 | Implemented |
| NF003 | Usability | WebApplication | All UI elements | TU001 | Should Have |
| NF004 | Reliability | All Infrastructure | N/A | TR001 | Implemented |
| NF005 | Scalability | All | All | TS002 | Should Have |
| NF009 | Data Integrity | DataService, Database | All Data related | TDI01 | Implemented |
| NF010 | Compliance | All | N/A | TCO01 | Implemented |

```