**Problem Statement:**

**Stock Management System -** 📌 **Features:**

* **Product & Order Management**
* **Event-driven stock updates (RabbitMQ/Kafka)**
* **Redis caching for fast product retrieval**
* **Background tasks for low-stock alerts**

**Core Features –**

**1. Product Management**

* Add, update, delete products
* Retrieve product details (with caching)
* Maintain stock levels

**2. Order Management**

* Place an order: Stock is **checked** before confirming
* Cancel an order: Stock is **restored**
* Event-driven notifications on order placement

**3. Caching with Redis**

* Store frequently accessed product & stock data in **Redis**
* **Cache invalidation** when stock is updated

**4. Event-Driven Stock Updates**

* Order placed → **Stock reduced** → Event published
* If stock goes below a threshold → **Low stock alert event**

**5. Background Tasks**

* **Low stock notifications** (Trigger email/SMS alerts)
* **Daily stock summary** report generation

**System Architecture Design:**

# **Tables –**

**1)**

|  |  |  |
| --- | --- | --- |
| **Products** | | |
| **Fields** | **Data Type** | **Methods** |
| ProductId 🡪 PK | int | AddProduct() |
| Name | string | UpdateProduct() |
| Description | string | DeleteProduct() |
| Price | decimal | GetProductById() |
| StockQuantity | int |  |
| LastUpdated | DateTime |  |

**2)**

|  |  |  |
| --- | --- | --- |
| **Order** | | |
| **Fields** | **Data Type** | **Methods** |
| OrderId 🡪 PK | int | PlaceOrder() |
| CustomerId 🡪 FK(Customers) | int | CancelOrder() |
| ProductId 🡪 FK(Products) | int | GetOrderById() |
| Quantity | int | GetOrderByCustomerId() |
| TotalPrice | decimal | GetAllOrders() |
| OrderDate | DateTime |  |
| Status | string |  |

**3) (Optional)**

|  |  |  |
| --- | --- | --- |
| **Order Details** | | |
| **Fields** | **Data Type** | **Methods** |
| OrderDetailId 🡪 PK | int | AddOrderDetail() |
| OrderId 🡪 FK(Order) | int | GetOrderDetailsByOrderId() |
| ProductId 🡪 FK(Products) | int |  |
| Quantity | int |  |
| Price | decimal |  |

**4)**

|  |  |  |
| --- | --- | --- |
| **Stock Event** | | |
| **Fields** | **Data Type** | **Methods** |
| EventId 🡪 PK | int | PublishEvent() |
| ProductId 🡪 FK | int | HandleEvent() |
| EventType | string |  |
| TimeStamp | DateTime |  |

**5)**

|  |  |  |
| --- | --- | --- |
| **Customer** | | |
| **Fields** | **Data Type** | **Methods** |
| CustomerId 🡪 PK | int | AddCustomer() |
| FirstName | string | UpdateCustomer() |
| LastName | string | DeleteCustomer() |
| Email | string | GetCustomerById() |
| PhoneNumber | string | GetAllCustomers() |

**6)**

|  |  |  |
| --- | --- | --- |
| **Low Stock Alert** | | |
| **Fields** | **Data Type** | **Methods** |
| AlertId 🡪 PK | int | CheckLowStock() |
| ProductId 🡪 FK | int | TriggerEvent() |
| Threshold | int |  |
| AlertDate | DateTime |  |

**# Tech Stack**

* **Backend**: C# with ASP.NET Core + Entity Framework
* **Frontend**: React
* **Database**: SQL Server
* **Caching**: Redis
* **Messaging**: Kafka

**# Estimated Time Plan:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Task** | **Sub-Task** | **Time required** |
| 1) | **Planning and requirement gathering** | i). Identifying necessary technologies | 1 days |
| ii). Designing system architecture |
| 2) | **Database Design and Table Creation** | i). Designing database schema | 2 days |
| ii). Creating tables |
| iii). Defining relationships (foreign key) |
| iv). Setting up database environment (SQL server) |
| 3) | **CRUD operations implementations** | i). Implementing CRUD for each class | 3 days |
| ii). Developing service methods for CRUD operations |
| iii). Implementing Data access layer using ADO.NET for database connectivity |
| 4) | **Event Driven Stock Updates** | i). Installing Kafka | 1 week |
| ii). Integrating Kafka for event-driven updates |
| iii). Creating event listeners |
| iv). Implementing stock updates |
| v). Writing unit tests |
| 5) | **Redis Caching for Performance Optimization** | i). Installing Redis | 1 week |
| ii). Set up Redis for caching stock quantities and product details |
| iii). Implementing caching logic in service methods for fetching product data |
| iv). Test cache invalidation strategies |
| 6) | **Background Task for Low-Stock Alerts** | i). Designing and implementing background tasks | 5 days |
| ii). Set up job scheduler |
| iii). Implementing low stock alerts |
| 7) | **Integration Testing and Debugging** | i). Integrating all components | 1 week |
| ii). Testing entire system workflow |
| iii). Debug and resolve bugs |
| 8) | **Project Overview** |  |  |