

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES, CHENNAI – 602105**

**CAPSTONE PROJECT REPORT**

**TITLE**

**Designing a Cloud-based Inventory Management System for Azura's E-commerce Platform on AWS**

***Submitted to***

**SAVEETHA SCHOOL OF ENGINEERING**

***By***

**J. Siva Sankar Reddy (192211356)**

***Guided by***

## Dr. J. Chenni Kumaran

**1. Problem Statement:**

* **Current System Analysis:** Evaluate the existing inventory management system, identifying its limitations and areas for improvement.
* Identify the challenges such as lack of real-time tracking, difficulty in managing stock levels across multiple locations, and inefficiencies in order processing.

**2.Proposed Design Work:**

***Identifying Key Components***

* **Inventory Database:** Evaluate and choose between Amazon RDS (relational) and DynamoDB (NoSQL) based on data requirements.
* **Backend Services:** Utilize AWS Lambda for scalability and cost efficiency.
* **Frontend Interface:** Select frameworks that offer responsiveness and ease of maintenance (React.js, Angular).
* **Authentication and Authorization:** AWS Cognito for secure user management.
* **Notifications:** Use AWS SNS and possibly integrate with messaging services like Slack or email for alerts.

### Functionality

* **Real-time Inventory Tracking:** Implement Web Sockets or AWS AppSync for real-time updates.
* **Order Management:** Integrate with payment gateways and shipping services.
* **Stock Alerts:** Allow customizable alert settings for different products.
* **Reporting**: Use Amazon Quick Sight for advanced data visualization and analytics.

**Architectural Design:**

* **Microservices Architecture:** Detail the microservices and their responsibilities.
* **Data Flow:** Include a data flow diagram to visualize interactions between components.
* **Security:** Incorporate AWS IAM roles, policies, and KMS for encryption.
* **Scalability:** Design for horizontal scaling using AWS Auto Scaling groups.

**3.GUI Design:**

**Layout:**

* **Responsive Design:** Ensure the interface is fully responsive and adapts to different screen sizes, including mobile devices.
* **Intuitive Navigation:** Design a navigation structure that is easy to understand and allows users to quickly find the information they need.
* **Dashboard Customization:** Allow users to customize their dashboard layout according to their preferences and needs.
* **Consistency :** Maintain consistency in design elements such as buttons, fonts, and colors across the entire application.
* **Accessibility (2 Marks):** Implement accessibility features to ensure the application is usable by people with disabilities, following ARIA standards and WCAG guidelines.

**User-Friendly:**

* **Onboarding Tutorials:** Provide guided tours or walkthroughs for new users to help them get started quickly.
* **Help and Support:** Integrate a help center, FAQs, or chat support within the application for easy access to assistance.
* **Search and Filter Options:** Implement advanced search and filter functionalities to allow users to quickly find specific products or orders.
* **User Feedback:** Include features for users to provide feedback or report issues directly from the application.
* **Error Handling:** Display user-friendly error messages and provide clear instructions on how to resolve issues.

**Color Selection:**

* **Brand Consistency:** Use a color scheme that aligns with Azura’s branding guidelines to ensure brand consistency.
* **Contrast and Readability:** Ensure sufficient contrast between text and background colors to enhance readability and comply with accessibility standards.
* **Visual Hierarchy:** Use colors strategically to create a visual hierarchy that guides users’ attention to important elements.
* **User Preferences:** Provide options for users to switch between different color themes (e.g., light and dark modes) to cater to their preferences.

**4.Program / Coding:**

**Language Selection*:***

* **Appropriate Language for Backend:** Choose a language that is efficient, scalable, and well-supported for backend services (e.g., Python, Node.js).
* **Appropriate Language for Frontend:** Choose a language/framework that provides a responsive and interactive user experience (e.g., React.js, Angular).
* **Integration Capabilities:** Ensure the selected languages can easily integrate with AWS services.

**Algorithm/Program:**

* **Efficiency of Algorithms:** Implement efficient algorithms for inventory management, ensuring minimal latency and resource usage.
* **Code Readability and Maintainability:** Ensure code is clean, well-documented, and follows best practices for readability and maintainability.
* **Modular Design:** Structure the code into modules to enhance reusability and ease of maintenance.
* **Error Handling and Logging:** Implement comprehensive error handling and logging mechanisms to capture and manage errors effectively.
* **Use of Libraries and Frameworks:** Utilize reliable libraries and frameworks to accelerate development and maintain code quality. **Program**

import json import boto3 dynamodb = boto3.resource('dynamodb') table = dynamodb.Table('InventoryTable') def lambda\_handler(event, context):

action = event['action'] if action == 'add\_stock': return add\_stock(event) elif action == 'remove\_stock': return remove\_stock(event)

else:

return {

'statusCode': 400,

'body': json.dumps('Invalid action'), } def add\_stock(event):

product\_id = event['product\_id']

quantity = int(event['quantity'])

# Update DynamoDB item for adding stock response = table.update\_item(

Key={'product\_id': product\_id},

UpdateExpression='ADD stock :quantity',

ExpressionAttributeValues={':quantity': quantity},

ReturnValues='UPDATED\_NEW'

)

return {

'statusCode': 200,

'body': json.dumps(f'Added {quantity} stock to product {product\_id}')

}

def remove\_stock(event):

product\_id = event['product\_id'] quantity = int(event['quantity'])

# Update DynamoDB item for removing stock response = table.update\_item(

Key={'product\_id': product\_id},

UpdateExpression='ADD stock :quantity',

ConditionExpression='stock >= :quantity',

ExpressionAttributeValues={':quantity': -quantity},

ReturnValues='UPDATED\_NEW'

) return {

'statusCode': 200,

'body': json.dumps(f'Removed {quantity} stock from product {product\_id}')

}

**Execution:**

* **Automated Testing (2 Marks):** Implement unit tests, integration tests, and end-to-end tests to ensure code quality.
* **Continuous Integration/Continuous Deployment (CI/CD) (2 Marks):** Set up CI/CD pipelines using tools like AWS CodePipeline for automated deployment and testing.
* **Monitoring and Debugging (1 Mark):** Use AWS CloudWatch for real-time monitoring and debugging to maintain system health and performance.

**5. Implementation:**

**Connecting the Components:**

* **API Integration:** Ensure seamless communication between frontend and backend components using RESTful APIs or Graph sQL.
* **Data Synchronization:** Implement data synchronization mechanisms to ensure consistency across different components.
* **Inter-service Communication:** Use AWS SQS or AWS SNS for inter-service communication to handle asynchronous tasks and notifications.
* **Error Handling and Resilience:** Design robust error handling and fallback mechanisms to maintain system resilience.

**Cloud Deployment:**

* **Infrastructure as Code:** Use AWS CloudFormation or Terraform to manage infrastructure as code, ensuring repeatable and scalable deployments.
* **Continuous Deployment:** Set up continuous deployment pipelines using AWS Code Pipeline and Code Deploy to automate the deployment process.
* **Security Best Practices:** Implement security best practices, including AWS IAM roles, VPCs, security groups, and encryption for data at rest and in transit.
* **Scalability:** Design for horizontal scalability using AWS Auto Scaling groups and load balancers to handle varying loads.
* **Monitoring and Logging:** Utilize AWS CloudWatch for real-time monitoring and logging to track performance and identify issues.

**Project Testing:**

* **Unit Testing (1 Mark):** Implement unit tests to verify the functionality of individual components.
* **Integration Testing (1 Mark):** Conduct integration tests to ensure different components work together as expected.
* **End-to-End Testing (1 Mark):** Perform end-to-end tests to validate the entire system’s workflow from start to finish.
* **User Acceptance Testing (1 Mark):** Conduct UAT to gather feedback from actual users and ensure the system meets their needs.
* **Performance Testing (1 Mark):** Use tools like JMeter or AWS Load Balancer to conduct load and stress testing, ensuring the system can handle expected traffic.

**6.Performance Evaluation:**

**Load Testing:**

* **Tool Selection:** Choose appropriate tools like Apache JMeter or AWS Load Balancer for conducting load testing.
* **Scalability Testing:** Test the system’s ability to handle increased load by simulating concurrent user interactions and transactions.
* **Results Analysis:** Analyze load test results to identify performance bottlenecks and optimize system scalability.

**Response Time:**

* **Benchmarking:** Establish baseline response time metrics under normal operating conditions.
* **Optimization:** Optimize backend algorithms, database queries, and frontend rendering to improve response time.

**Resource Utilization*:***

* **Monitoring:** Implement AWS CloudWatch or similar tools to monitor resource utilization metrics such as CPU, memory, and storage.
* **Optimization Strategies:** Develop strategies to optimize resource allocation and reduce operational costs.

**High Availability:**

* **Redundancy and Failover:** Ensure high availability through redundancy and failover mechanisms such as multi-AZ deployments and AWS Elastic Load Balancing.
* **Fault Tolerance:** Design fault-tolerant architecture to minimize downtime and maintain system availability during failures.

**Security and Compliance:**

* **Security Testing:** Conduct security testing using tools like AWS Inspector or third-party penetration testing services to identify and remediate security vulnerabilities.

**7.Conclusion:**

This capstone project successfully addressed the challenge of designing and implementing a cloud-based inventory management system tailored for Azura's e-commerce platform on AWS. The project aimed to enhance inventory tracking, streamline order management, and optimize overall system performance. Throughout the development process, significant achievements were realized, including real-time inventory updates and improved user experience through an intuitive interface. Challenges such as integrating AWS services effectively were overcome using agile methodologies and iterative development. Looking ahead, future enhancements could include advanced analytics for predictive inventory management and multi-region deployment to enhance system resilience. This project not only met its objectives but also paved the way for continuous improvement and innovation in managing Azura's inventory effectively within a dynamic cloud environment.