

# **PROJECT PROPOSAL: CAMPUSBITES A High-Availability Canteen Pre-order System Deployed on AWS**

## **GROUP MEMBERS:**

- Muhammad Samer Nisar (BSSE23113)
- Ahmad Umar Khan (BSSE23008)
- Mirza Muhammad Rehan (BSSE23021)

## **1. EXECUTIVE SUMMARY:**

"Campus Bites" is a web-based application designed to streamline the food ordering process in university cafeterias. By leveraging a 3-tier serverless container architecture, the system addresses the inefficiencies of manual ordering during peak break times. This document outlines the business logic, problem statement, and the detailed technical deployment strategy using Amazon Elastic Container Service (ECS), AWS Fargate, and Amazon ECR.

## **2. PROBLEM STATEMENT:**

The university canteen currently operates on a "first-come, first-served" manual model. During short break intervals (15–30 minutes), this results in:

- **Severe Congestion:** Physical queues often exceed the canteen's capacity, causing students to abandon lines and lose break time.
- **Operational Inefficiency:** Staff spend valuable time manually writing down orders and managing cash flow instead of focusing on food preparation.
- **Lack of Accountability:** Without a verified identity system, "prank" orders are occasionally placed, leading to food wastage and revenue loss.

## **3. SOFTWARE REQUIREMENTS**

### **3.1 Functional Requirements**

- **User Authentication:** Mandatory login via Student/Staff ID to verify identity and prevent fake orders.
- **Menu Browsing:** Authenticated users must be able to browse a dynamic menu with real-time availability.
- **Order Management:** Students can place orders which are initially saved with a PENDING status.

- **Kitchen Dashboard:** A real-time interface for canteen staff to view incoming orders and manage the queue.
- **Order Status Tracking:** Staff can update status to `READY_FOR_PICKUP`, triggering a notification to the student.
- **Identity Verification:** Automatic capture of User ID from the session to ensure accountability.

### 3.2 Non-Functional Requirements

- **High Availability:** The system must be deployed across at least two Availability Zones (AZs) to ensure service continuity.
- **Scalability:** Must utilize Service Auto Scaling to handle high demand during 15-30 minute peak break windows.
- **Security:** Use of private subnets for application and data layers, with secrets managed by AWS Secrets Manager.
- **Maintainability:** Adoption of a containerized approach (ECS Fargate) to eliminate server management and patching overhead.

## 4. AWS CLOUD ARCHITECTURE:

The architecture is built on a serverless container model to provide maximum reliability with minimum operational effort.

### 4.1 Service Breakdown

- **Amazon ECR (Elastic Container Registry):** Secure storage for the Docker images of our React.js frontend and Node.js backend.
- **Amazon ECS Cluster:** A logical grouping of tasks or services that provides the infrastructure boundary for our containerized environment.
- **Amazon ECS (Elastic Container Service):** Orchestrates the container lifecycle, ensuring tasks are running across multiple AZs.
- **AWS Fargate:** The serverless compute engine for ECS, removing the need to manage EC2 instances.
- **Bastion Host (Amazon EC2):** A secure jump server located in the Public Subnet, used by administrators to access the private PostgreSQL RDS instance for maintenance and database management.
- **Application Load Balancer (ALB):** Distributes incoming traffic across the active Fargate tasks.

- Service Auto Scaling: Dynamically scales the number of running tasks based on CPU and memory utilization metrics.
- Internet Gateway: Provides entry/exit for public traffic to the ALB and NAT Gateway.
- NAT Gateway: Enables private tasks to fetch updates (e.g., npm install) without direct exposure to the internet.
- Amazon RDS: Managed relational database (PostgreSQL) for user data and order history.
- AWS Secrets Manager: Centralized vault for database credentials and API keys.

## 5. DEPLOYMENT STRATEGY:

We employ a "Defense in Depth" strategy by isolating layers into specific subnets.

**5.1 The Web Tier (Frontend - Public Subnet) Deployment:** The React.js user interface is containerized and deployed as Fargate tasks. These tasks are managed by an ECS Service residing within the ECS Cluster, located in the Public Subnet for internet accessibility. **Logic:** This tier handles the presentation layer and user interface. It is the primary entry point for users via the ALB and forwards requests to the backend App Tier.

**5.2 The App Tier (Backend - Private Subnet) Deployment:** The Node.js business logic is containerized and deployed as Fargate tasks. These tasks are managed by an ECS Service residing within our ECS Cluster. **Isolation:** Located in the Private Subnet, these tasks have no public IP addresses. They accept inbound traffic only from the ALB Security Group to ensure the API layer is not exposed.

**5.3 The Data Tier (Database - Private Subnet) Deployment:** Managed Amazon RDS instance (PostgreSQL) hosted in an isolated subnet. **Security:** The Database Security Group is restricted to allow inbound traffic only from the App Tier's ECS Task Security Group.

**5.4 Security & Secrets Management IAM Roles for Tasks:** Each ECS task is assigned a specific IAM Role to retrieve secrets at runtime. **Credential Flow:** On startup, the backend container queries AWS Secrets Manager for the DB password, ensuring no plaintext data is stored in the Docker image.

**5.5 Administrative Access** Direct access to the private database tier is blocked for security. For maintenance and PostgreSQL management:

- **Provisioning:** A Bastion Host (EC2 instance) is provisioned in the Public Subnet.
- **Access Control:** The Bastion Host accepts SSH connections only from whitelisted administrative IP addresses.

- Database Management: Administrators first SSH into the Bastion Host and use it as a secure bridge to manage the PostgreSQL RDS instance in the Private Data Subnet.

## 6. CONCLUSION:

The transition from EC2 to an ECS Fargate architecture ensures that "CampusBites" is modern, scalable, and highly secure. By utilizing serverless containers across both the Web and App tiers and protecting data access through a dedicated Bastion Host, we minimize infrastructure management while providing a robust platform that guarantees university students can order their meals efficiently during their limited break time.

## 7. ARCHITECTURE DIAGRAM:

