**Project Charter: Smart Restaurant Ordering Web App**

**1. Project Title**

**Smart Restaurant / Innovative Food Ordering Web App**

**2. Project Purpose / Justification**

The purpose of this project is to design and implement a **cloud-based, multi-tier web application** that allows users to browse restaurant menus, place orders, and receive AI-based dish recommendations. The system will demonstrate **full-stack development** and **cloud architecture skills**, including S3, CloudFront, EC2, RDS, and Auto Scaling.

This project will:

* Provide a **real-life, interactive application** for users.
* Showcase **modern cloud and DevOps practices**.
* Serve as a **portfolio-ready project** demonstrating practical skills in AWS and full-stack development.

**3. Project Objectives**

1. Develop a **frontend hosted on S3 + CloudFront** with a visually attractive menu interface.
2. Implement a **backend API on EC2** to handle orders, menu updates, and recommendations.
3. Store and manage **menu items, orders, and user data in RDS**.
4. Enable **cash payments** and a **sandbox payment simulation**.
5. Implement **auto scaling with load balancing** to handle variable traffic.
6. Provide **AI-based dish recommendations** based on user order history.

**4. Project Scope**

**In Scope:**

* Multi-tier architecture: frontend, backend, database
* Hosting frontend on S3 + CloudFront
* Backend APIs on EC2 instances
* Database using RDS
* Auto Scaling and Load Balancer configuration
* Basic order management and recommendation system
* Cash and simulated payment functionality

**Out of Scope:**

* Real credit card processing (for security reasons)
* Mobile app version (only responsive web design)
* Integration with external delivery services

**5. Deliverables**

1. **Frontend:** Static website with animated menu categories, order form, and recommendation section.
2. **Backend:** EC2-hosted API for managing menu, orders, and recommendations.
3. **Database:** RDS instance storing menu items, orders, and customer profiles.
4. **Cloud Infrastructure:**
   * S3 + CloudFront for frontend
   * EC2 for backend
   * Auto Scaling group with Load Balancer
5. **Documentation:**
   * Project report
   * Deployment guide
   * GitHub repository link

**6. Milestones**

| **Milestone** | **Date / Duration** |
| --- | --- |
| Project Planning & Charter Approval | Day 1 |
| Frontend Development (S3 + CloudFront) | Day 2–5 |
| Backend API Development (EC2) | Day 6–10 |
| Database Setup (RDS) | Day 6–10 |
| Integration & Testing | Day 11–13 |
| Auto Scaling & Load Balancer Setup | Day 14 |
| Final Testing & Deployment | Day 15 |
| Documentation & Project Submission | Day 16 |

**7. Team Roles**

* **Project Owner / Developer:** Babi (Full Stack & Cloud)
* **Advisor / Mentor (Optional):** Guidance on AWS best practices

**8. Risks & Mitigation**

| **Risk** | **Mitigation** |
| --- | --- |
| Misconfiguration of AWS services | Follow AWS documentation and best practices |
| Overcomplicating AI recommendations | Start with simple logic based on previous orders |
| Cost overrun on AWS resources | Use Free Tier and terminate unused instances |
| Security risks with payment data | Only simulate payment in sandbox / cash on delivery |

**9. Success Criteria**

* Fully functional multi-tier web app deployed on AWS
* Frontend accessible globally via CloudFront
* Backend correctly processes orders and recommendations
* Orders stored in RDS accurately
* Auto Scaling responds to traffic changes
* Safe and portfolio-ready demonstration of cash and simulated payments