Functional and Nonfunctional Requirements

Carolina Wings Web Application - Requirements Document

Functional Requirements

---

1. User Management

- Users can create an account using email and password with verification code upon creation.
- Users can log in and log out securely.
- Roles:
    - Root user (me)
        - do everything access everything
    - Admin (General Manager)
        - View and access orders/history of orders
        - Admin view with the ability to make changes to the menu
        - able to check logs and total sales
    - Customer
        - Can create orders
        - View menu
        - Create account and login
        - Sign up for weekly newsletter
        - Password reset functionality via email.

2. Menu & Item Browsing

- Menu items are categorized based on what submenu they belong to
    - Wings
    - Ribs
    - Sandwiches
    - Appetizers
    - Wraps

    - Salads
    - Burgers
    - Side items
    - Premium side items
- Each item displays: name, short description, price, and image and what subcategory it is or type
- Menu is fetched from backed (DB)

3. Cart & Order Management

- Users can add/remove items from the cart, as well as add special requests or notes about specific items that will display on the ticket
    - E.g. allergies or memos
- Users can update item quantities in real time.
- Cart displays a live summary with total cost plus credit card surcharge.

4. Payment Processing

- Users can checkout using:
    - Credit/Debit Cards (Toast most likely or Stripe)
    - Apple Pay
    - Google Pay
    - PayPal
- Secure and tokenized payments (sent to payment processor)
    - If its through third party (PayPal/Apple Pay, that will be handled on their side)
- Successful checkout/confirmed payment generates an order that is then formatted correctly and sent to the correct restaurant where it will print.

5. Order Handling

- Orders are sent to a backend endpoint.
- Orders are printed at the restaurant via ticket printers.
- Admin can view order status: Pending, In Progress, Completed.

6. Availability Logic

- Orders can only be placed during open hours.
- Users are informed if ordering is currently unavailable.

7. Admin Dashboard

- Admin can:
  - View and manage orders
  - Edit menu items (Add, Remove, Update)
  - Set restaurant open hours

##Non-Functional Requirements

1. Performance

- Support 20–30 concurrent users at any given time
- Menu/cart interactions under 500ms latency.

2. Scalability

- Backend and database scalable using AWS services, but cost optimized
- Frontend assets served S3 as they will not change much.

3. Security

- HTTPS for all data transmission****
- JWT-based authentication (tokens stored securely).
- Input validation and sanitation on all APIs and forms.
- Use Stripe/PayPal/Toast to offload PCI compliance for payments.
- Role-based access control (RBAC) for admin routes and root routes
  - also to distinguish customer routes from admin routes as customers should not have access to any admin routes

4. Offline Access

- Site shell and menu cached with S3 to ensure offline access
  - serve static website after hours unless Admin account logs in
    - if they do send lambda function to boot a t2.micro instance for anything the ADMIN may need
- Display notification when offline using SNS and send that to all IAM users within ADMIN group
- Stop instance to disable ordering after hours.

5. Availability & Reliability

- 99% uptime goal
- Use AWS CloudWatch and alerting for downtime/errors.
- SNS service when there is downtime, so that way admins are notified service is down

6. Maintainability

- React, Spring Boot, PostgreSQL as tech stack
- Code adheres to SOLID principles and is well-documented.
- RESTful API design.

7. Deployment

- CI/CD with GitHub Actions or AWS CodePipeline.
- Containerized app with Docker
- Orchestration with K8 or just a simple Docker compose
- Hosted on AWS (Frontend: S3, Backend: ECS or EC2) with WAF, Internet Gateway, and consolidated private and public subnet with secure traffic only being handled in the private subnet, and the public subnet being public facing.

Tech Stack

- **Frontend:** React
- **Backend:** Spring Boot
- **Database:** PostgreSQL
- **Auth:** JWT
- **Hosting:** AWS (EC2/ECS, S3)
- **Payments:** Stripe, Apple Pay, Google Pay, PayPal
- **Payment Processor**: Toast or other third party (Heartland is currently in use)