College Mess Feedback Portal - Backend Specification

Author: Manus AI

Date: January 2024

Version: 1.0

Executive Summary

This document outlines the comprehensive backend architecture for the College Mess Feedback Portal, designed to provide secure admin-only access to poll results, implement one-time voting restrictions, and enable automatic data export to Excel format. The backend will be built using Flask with SQLite database integration, providing a robust foundation for managing student feedback, complaints, and menu voting data.

The system addresses critical requirements including data persistence, user session management, administrative controls, and automated reporting capabilities. By implementing proper authentication mechanisms and vote tracking, the platform ensures data integrity while maintaining user anonymity where appropriate.

System Architecture Overview

The backend architecture follows a traditional three-tier model consisting of presentation layer (existing frontend), business logic layer (Flask API), and data access layer (SQLite database). This architecture provides clear separation of concerns while maintaining simplicity for deployment and maintenance.

The Flask application serves as the central hub for all data operations, implementing RESTful API endpoints that handle CRUD operations for votes, feedback, and complaints. The system incorporates session management to prevent duplicate voting while maintaining user privacy through IP-based tracking rather than personal identification.

Administrative functionality is secured through token-based authentication, ensuring that sensitive data access is restricted to authorized personnel only. The system includes comprehensive logging and audit trails to track all administrative actions and data modifications.

Database Schema Design

The database schema is designed to efficiently store and retrieve voting data, feedback submissions, and complaint records while maintaining referential integrity and supporting

complex queries for administrative reporting.

Tables Structure

Votes Table

The votes table serves as the primary repository for all menu voting data, capturing essential information about user preferences across different meals and days.

SQL

```
CREATE TABLE votes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    day VARCHAR(20) NOT NULL,
    meal VARCHAR(20) NOT NULL,
    dish VARCHAR(100) NOT NULL,
    user_identifier VARCHAR(100) NOT NULL,
    ip_address VARCHAR(45) NOT NULL,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    session_id VARCHAR(100),
    UNIQUE(day, meal, user_identifier)
);
```

The unique constraint on (day, meal, user_identifier) ensures that each user can vote only once per meal per day, implementing the one-time voting requirement. The user_identifier field combines IP address and browser fingerprinting to create a unique identifier without storing personal information.

Feedback Table

The feedback table stores anonymous user feedback with rating information and categorization for administrative analysis.

SQL

```
CREATE TABLE feedback (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    feedback_type VARCHAR(50) NOT NULL,
    message TEXT NOT NULL,
    rating INTEGER CHECK(rating >= 1 AND rating <= 5),
    ip_address VARCHAR(45) NOT NULL,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    session_id VARCHAR(100)
);
```

The rating field includes a check constraint to ensure valid star ratings between 1 and 5, maintaining data quality for analytical purposes.

Complaints Table

The complaints table manages user complaints with urgency levels and categorization for prioritized administrative response.

```
SQL

CREATE TABLE complaints (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    category VARCHAR(50) NOT NULL,
    message TEXT NOT NULL,
    urgency VARCHAR(20) NOT NULL,
    ip_address VARCHAR(45) NOT NULL,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    session_id VARCHAR(100),
    status VARCHAR(20) DEFAULT 'pending'
);
```

The status field allows administrators to track complaint resolution progress, supporting workflow management for customer service operations.

Menu Suggestions Table

The menu suggestions table captures user-generated menu ideas and dietary preferences for menu planning purposes.

```
SQL

CREATE TABLE menu_suggestions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    dish_name VARCHAR(100) NOT NULL,
    meal_type VARCHAR(20) NOT NULL,
    ingredients TEXT,
    description TEXT,
    ip_address VARCHAR(45) NOT NULL,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    session_id VARCHAR(100)
);
```

Admin Users Table

The admin users table manages administrative access credentials and permissions for

secure data access.

```SQL
CREATE TABLE admin_users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    email VARCHAR(100),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    last_login DATETIME,
    is_active BOOLEAN DEFAULT TRUE
);
```

Admin Sessions Table

The admin sessions table tracks active administrative sessions for security and audit purposes.

```SQL
CREATE TABLE admin_sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    admin_id INTEGER NOT NULL,
    token VARCHAR(255) UNIQUE NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    expires_at DATETIME NOT NULL,
    ip_address VARCHAR(45) NOT NULL,
    FOREIGN KEY (admin_id) REFERENCES admin_users (id)
);
```

API Endpoints Specification

The API follows RESTful conventions with clear endpoint naming and appropriate HTTP methods for different operations. All endpoints return JSON responses with consistent error handling and status codes.

Public Endpoints (No Authentication Required)

POST /api/vote

Submits a new vote for a specific meal and day combination.

Request Body:

JSON

```
{
    "day": "monday",
    "meal": "breakfast",
    "dish": "Idli Sambar"
}
```

Response:

JSON

```
{
    "success": true,
    "message": "Vote submitted successfully",
    "vote_id": 123
}
```

Error Responses:

- 400: Invalid input data or duplicate vote attempt
- 429: Rate limiting exceeded
- 500: Internal server error

POST /api/feedback

Submits anonymous feedback with rating information.

Request Body:

JSON

```
{
    "feedback_type": "food_quality",
    "message": "The food was excellent today!",
    "rating": 5
}
```

Response:

JSON

```
{
    "success": true,
    "message": "Feedback submitted successfully",
```

```
    "feedback_id": 456
}
```

POST /api/complaint

Submits an anonymous complaint with urgency level.

Request Body:

JSON

```
{
    "category": "food_quality",
    "message": "The rice was undercooked",
    "urgency": "medium"
}
```

Response:

JSON

```
{
    "success": true,
    "message": "Complaint submitted successfully",
    "complaint_id": 789
}
```

POST /api/menu-suggestion

Submits a menu suggestion with dish details.

Request Body:

JSON

```
{
    "dish_name": "Masala Dosa",
    "meal_type": "breakfast",
    "ingredients": "Rice, lentils, potatoes, spices",
    "description": "Crispy dosa with spiced potato filling"
}
```

Administrative Endpoints (Authentication Required)

POST /api/admin/login

Authenticates administrative users and returns access token.

Request Body:

JSON

```
{
    "username": "admin",
    "password": "secure_password"
}
```

Response:

JSON

```
{
    "success": true,
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "expires_in": 3600
}
```

GET /api/admin/votes

Retrieves all voting data with filtering and pagination options.

Query Parameters:

- `page` : Page number (default: 1)
- `limit` : Items per page (default: 50)
- `day` : Filter by specific day
- `meal` : Filter by specific meal
- `start_date` : Filter by start date
- `end_date` : Filter by end date

Response:

JSON

```
{
    "success": true,
    "data": [
        {
            "id": 1,
            "day": "monday",
            "meal": "breakfast",
```

```
            "dish": "Idli Sambar",
            "timestamp": "2024-01-15T08:30:00Z",
            "ip_address": "192.168.1.100"
        }
    ],
    "pagination": {
        "page": 1,
        "limit": 50,
        "total": 150,
        "pages": 3
    }
}
```

GET /api/admin/feedback

Retrieves all feedback submissions with filtering capabilities.

GET /api/admin/complaints

Retrieves all complaints with status filtering and urgency sorting.

GET /api/admin/menu-suggestions

Retrieves all menu suggestions with meal type filtering.

GET /api/admin/export/excel

Generates and downloads Excel file containing all data.

Query Parameters:

- `type` : Data type to export (votes, feedback, complaints, suggestions, all)
- `start_date` : Start date for data range
- `end_date` : End date for data range

Response: Binary Excel file download

GET /api/admin/dashboard

Retrieves summary statistics for administrative dashboard.

Response:

JSON

```
{
    "success": true,
    "stats": {
```

```
        "total_votes": 1250,
        "total_feedback": 89,
        "total_complaints": 23,
        "total_suggestions": 45,
        "votes_today": 67,
        "popular_dishes": [
            {"dish": "Biryani", "votes": 89},
            {"dish": "Dal Rice", "votes": 76}
        ]
    }
}
```

Authentication and Security

The system implements robust security measures to protect administrative access and ensure data integrity. Authentication is handled through JWT (JSON Web Tokens) with configurable expiration times and automatic session management.

Token-Based Authentication

Administrative access utilizes JWT tokens with the following characteristics:

- 1-hour default expiration time
- Secure random secret key generation
- IP address binding for additional security
- Automatic token refresh capabilities

Password Security

Administrative passwords are secured using industry-standard hashing algorithms:

- bcrypt hashing with configurable salt rounds
- Minimum password complexity requirements
- Password change enforcement policies
- Account lockout after failed attempts

Rate Limiting

The system implements rate limiting to prevent abuse and ensure fair usage:

- Vote submissions: 10 per hour per IP address
- Feedback submissions: 5 per hour per IP address

- Complaint submissions: 3 per hour per IP address
- Admin login attempts: 5 per 15 minutes per IP address

One-Time Voting Implementation

The one-time voting mechanism combines multiple identification methods to prevent duplicate voting while maintaining user anonymity. The system creates a unique identifier for each user session without storing personally identifiable information.

User Identification Strategy

The system generates a unique identifier using the following components:

- IP address (primary identifier)
- Browser fingerprinting (User-Agent, screen resolution, timezone)
- Session timestamp (for daily vote resets)
- Cryptographic hash of combined identifiers

Vote Validation Process

Each vote submission undergoes the following validation steps:

1. Generate user identifier from request metadata
2. Check existing votes for the same day/meal/user combination
3. Validate vote data (day, meal, dish selection)
4. Store vote with timestamp and identifier
5. Return success confirmation or duplicate vote error

Session Management

The system maintains session state through secure cookies and server-side session storage:

- Secure HTTP-only cookies for session identification
- Server-side session data storage in SQLite
- Automatic session cleanup for expired sessions
- Cross-site request forgery (CSRF) protection

Excel Export Functionality

The Excel export feature provides comprehensive data export capabilities for administrative analysis and reporting. The system generates formatted Excel files with multiple worksheets for different data types.

Export Features

The Excel export includes the following capabilities:

- Multiple worksheet support (votes, feedback, complaints, suggestions)
- Formatted headers and data styling
- Automatic column sizing and data formatting
- Summary statistics and charts
- Date range filtering options
- Custom export templates

File Generation Process

Excel file generation follows these steps:

1. Query database based on export parameters
2. Process and format data for Excel output
3. Create workbook with multiple worksheets
4. Apply formatting and styling
5. Generate summary statistics
6. Create downloadable file response

Export Formats

The system supports multiple export formats:

- Excel (.xlsx) - Primary format with full formatting
- CSV (.csv) - Simple comma-separated values
- JSON (.json) - Structured data for API integration
- PDF (.pdf) - Formatted reports for printing

Error Handling and Logging

Comprehensive error handling and logging ensure system reliability and facilitate troubleshooting. The system implements structured logging with different severity levels and automatic error reporting.

Error Response Format

All API endpoints return consistent error responses:

```JSON
{
    "success": false,
    "error": {
        "code": "DUPLICATE_VOTE",
        "message": "You have already voted for this meal",
        "details": {
            "day": "monday",
            "meal": "breakfast"
        }
    }
}
```

Logging Configuration

The system implements comprehensive logging:

- Application logs: INFO level for normal operations
- Error logs: ERROR level for system failures
- Security logs: WARNING level for authentication failures
- Audit logs: INFO level for administrative actions
- Performance logs: DEBUG level for optimization

Deployment Considerations

The backend is designed for easy deployment across different environments with minimal configuration requirements. The system supports both development and production deployment scenarios.

Environment Configuration

The application uses environment variables for configuration:

- `DATABASE_URL` : SQLite database file path
- `SECRET_KEY` : JWT token signing key
- `ADMIN_USERNAME` : Default admin username
- `ADMIN_PASSWORD` : Default admin password

- `CORS_ORIGINS` : Allowed CORS origins
- `RATE_LIMIT_STORAGE` : Rate limiting storage backend

Production Deployment

For production deployment, the system requires:

- Python 3.8+ runtime environment
- SQLite 3.x database engine
- Reverse proxy (nginx recommended)
- SSL certificate for HTTPS
- Process manager (systemd or supervisor)
- Log rotation and monitoring

This comprehensive specification provides the foundation for implementing a robust backend system that meets all requirements for secure data management, one-time voting enforcement, and administrative data access with Excel export capabilities.