Online Cafe Ordering System - MCO 1

Product Requirements Document

### Technology stack

# Python (Django framework)

* SQLite (Database)
* Bootstrap (CSS framework)

# Project overview

A web-based cafe ordering system demonstrating practical implementation of encryption and authentication techniques for securing user data. This project prioritizes backend security while maintaining functional UI/UX design.

## **1.1 Academic objectives**

* Demonstrate secure password hashing (Argon2)
* Implement symmetric encryption for PII (AES-256)
* Showcase secure session management
* Document and present security implementation

## **1.2 Project scope**

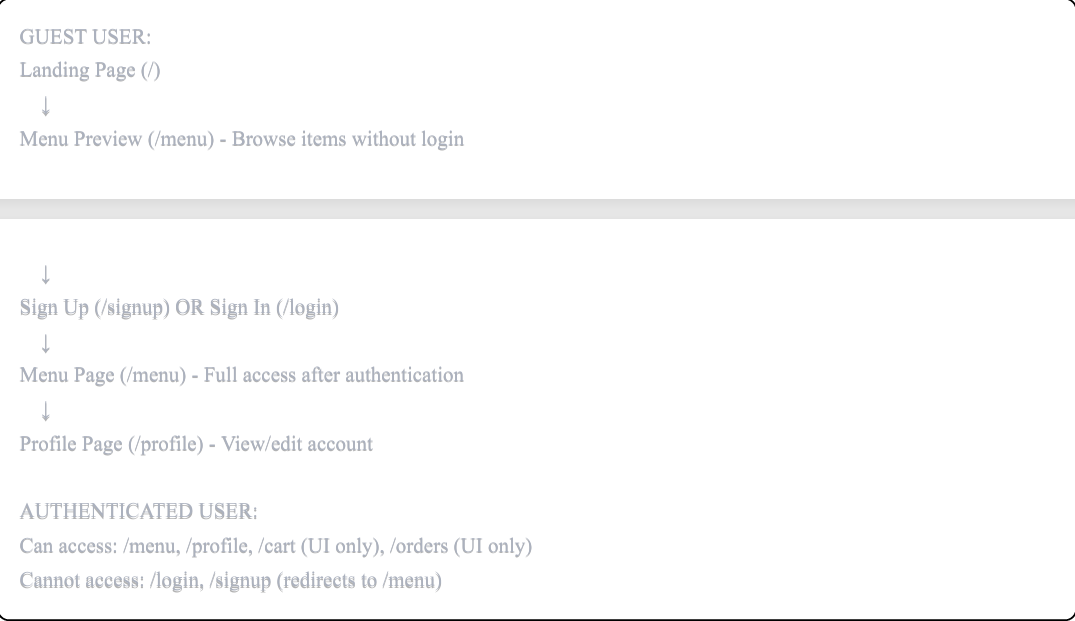
**MCO 1 (Current Phase):** Full backend + frontend implementation for

authentication and profile management, with UI-only designs for ordering features.

**MCO 2 (Future Phase):** Full implementation of cart, checkout, payment, and order management with encrypted transaction data

# System workflow

## **2.1 User journey**



## **2.2 Authentication flow**

**Sign Up Process:**

1. User submits: Username, Email, Password

2. Backend validates password strength

3. Password → Argon2 hash

4. Email → AES-256 encryption

5. Store in database

6. Create session

7. Redirect to /menu

**Sign In Process:**

1. User submits: Username, Password

2. Backend checks rate limit (5 attempts/15 min)

3. Verify credentials

4. Check account not locked

5. Create encrypted session

6. Log login event

7. Redirect to /menu

**Profile Management:**

1. View decrypted user data

2. Edit username (uniqueness check)

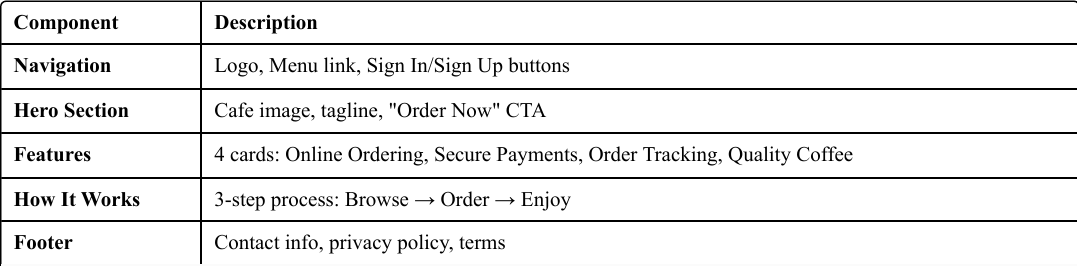
3. Change password (re-hash)

4. Delete account (permanent removal

# Feature specifications

## **3.1 Landing page (Full implementation)**

Introduce cafe and drive signups



## **3.2 Sign up page (Full implementation)**

Demonstrate encryption at point of data collection.

**Form Fields**

* Username (plain text, unique)
* Email (validated format)
* Password (strength validated)
* Confirm Password (match verification)

**Password Validation Rules**

* Minimum 8 characters
* At least 1 uppercase letter
* At least 1 number
* At least 1 special character (!@#$%^&\*)

**Security Features**

* CSRF token protection
* Server-side validation
* Email format verification
* Rate limiting (5 signups/hour per IP)
* Real-time password strength indicator

**Backend Process**

1. Validate input fields
2. Check username uniqueness
3. Verify email format and uniqueness
4. Confirm passwords match
5. Hash password with Argon2
6. Encrypt email with AES-256
7. Save user to database
8. Create session
9. Redirect to /menu

**Error Handling**

* "Username already exists"
* "Email already registered"
* "Passwords don't match"
* "Password too weak"
* Inline error messages (non-blocking)

**Database Storage**

* Username: Plain text (indexed for auth)
* Password: Argon2 hashed (irreversible)
* Email: AES-256 encrypted (reversible)

## **3.3 Sign in page (Full implementation)**

Demonstrate brute-force protection and secure session creation

**Form Fields**

* Username (login identifier)
* Password
* Remember Me (optional, extends session to 2 weeks)

**Security Features**

* Rate limiting (5 attempts per 15 minutes per IP)
* Account lockout after 5 failed attempts (1 hour duration)
* CSRF protection
* Session encryption
* Login event logging
* Generic error messages (prevents user enumeration)

**Backend Process**

1. Validate input
2. Check IP-based rate limit
3. Retrieve user by username
4. Check if account locked
5. Verify password against hash
6. Create encrypted session
7. Log login event (timestamp, IP, success/failure)
8. Redirect to /menu

**Error Messages Generic:**

* "Invalid username or password" (security best practice)
* Lockout: "Account locked due to multiple failed attempts. Try again in X minutes." Rate limit: "Too many login attempts. Please try again later.

## **3.4 Profile page (Full implementation)**

Demonstrate data encryption and account management

**Profile Information**

* Display Username (editable)
* Email (decrypted from database, display only)
* Account created date
* Last login timestamp

**Change Username**

* Input field with validation
* Uniqueness check against database
* Update confirmation message
* Immediate UI update

**Change Password**

* Current password verification (security check)
* New password with strength validator
* Confirm new password field
* Re-hash with Argon2 on save Invalidate existing sessions (logout all devices)
* Success message with redirect

**Delete Account**

* Confirmation modal with warning message
* Password verification required
* Permanent deletion from database (GDPR compliance)
* Session invalidation
* Redirect to landing page with confirmation

**Security Information Panel**

* Last login: Date and time Account age:
* Days since creation Security tip: "Use a unique password for this account"
* Access Control User can only view/edit their own profile Middleware checks authentication
* Redirect to /login if not authenticated

## **3.5 Menu display (Full implementation)**

Browse available cafe items

**Menu Item Data**

* Item name
* Description
* Price (formatted as currency)
* Category (Coffee, Tea, Pastries, Snacks, Food)
* Image Availability status (in stock/out of stock)

**Display Features**

* Bootstrap card grid layout (responsive)
* Category filter tabs (All, Coffee, Tea, Pastries, Snacks, Food)
* Client-side filtering (JavaScript, no page reload)
* "Add to Cart" button (visible, frontend handler only for MCO 1)
* Out of stock indicator (grayed out card

## **3.6 Shopping cart (UI only)**

Design demonstration for future implementation

**UI Components**

* Cart item list (mock data)
* Item image, name, price per unit
* Quantity controls (+/- buttons)
* Remove item button (×)
* Subtotal calculation
* Tax calculation (8%)
* Total amount
* "Proceed to Checkout" button
* Empty cart state: "Your cart is empty"

**Frontend Interactions (JavaScript Only)**

* Update quantity → recalculate totals
* Remove item → update display
* All calculations client-side
* Navigate to checkout page

**Note**: Backend cart storage with encrypted data will be implemented in MCO 2

## **3.7 Checkout (UI only)**

Design demonstration for future implementation

**UI Components**

* Order type selection (Radio buttons: Pick Up / Dine In)
* Contact information (pre-filled if logged in)
* Delivery instructions (textarea)
* Order summary card (items, quantities, total)
* "Continue to Payment" button

**Frontend Interactions**

* Order type selection updates
* UI Form validation (client-side)
* Navigate to payment page

**Note**: Backend order creation with encrypted customer data will be implemented in MCO 2

## **3.8 Payment (UI only)**

Design demonstration for future implementation

**UI Components**

* Payment method tabs (Credit Card, GCash, Cash on Pickup)
* Card number input (formatted: XXXX XXXX XXXX XXXX)
* Expiry date (MM/YY format) CVV input (3-4 digits)
* Cardholder name
* Order total display
* "Place Order" button
* Disclaimer banner: "UI Demonstration - No real payment processing"

**Frontend Interactions**

* Auto-format card number (spaces every 4 digits)
* Tab switching between payment methods
* Form validation (client-side)
* Show success alert on submit

**Note**: Backend payment encryption with AES-256-GCM will be implemented in MCO 2.

## **3.9 Order confirmation (UI only)**

Design demonstration for future implementation

**UI Components**

* Success checkmark icon
* "Order Confirmed!" heading
* Mock order number (e.g., #ORD-20250001)
* Order details (type, pickup time, total)
* Items list with quantities
* Confirmation message: "We've sent a confirmation to your email"
* "View Order History" button
* "Order Again" button

**Note**: Backend order creation and email confirmation will be implemented in MCO 2.

## **3.10 Order history (UI only)**

Design demonstration for future implementation

**UI Components**

* Order cards with mock data
* Order number, date, status badge
* Item summary (e.g., "3 items")
* Total amount
* "View Details" and "Reorder" buttons
* Empty state: "No orders yet - Start browsing our menu!"

**Status Badges**

* Pending: Orange badge
* Preparing: Blue badge
* Ready: Green badge
* Completed: Gray badge

**Note**: Backend will query real orders from encrypted database in MCO 2

# 4. Testing

## **4.1 Manual test cases**

**Authentication test**

* Sign up with weak password → Rejected with helpful message
* Sign up with strong password → Success, redirect to menu
* Sign up with existing username → Error: "Username already exists"
* Sign up with existing email → Error: "Email already registered"
* Login with correct credentials → Success, redirect to menu
* Login with wrong password → Error: "Invalid username or password"
* Login with wrong password 6 times → Account locked for 1 hour
* Login after timeout → Auto-logout, redirect to login

**Profile Tests**

* View profile → Shows username and decrypted email
* Change username → Success, immediate update
* Change username to existing one → Error: "Username already taken"
* Change password with wrong current password → Error
* Change password with correct current password → Success
* Delete account → Confirmation modal, account removed from DB

**Security Tests**

* Access /profile without login → Redirect to /login
* Submit form without CSRF token → 403 Forbidden
* Attempt SQL injection in login → Query fails safely
* Attempt XSS in profile name → Script tags escape

## **4.2 Database Verification**

Using DB browser for SQLite

* Check password field → Shows Argon2 hash (starts with $argon2id$)
* Check email field → Shows encrypted string (gibberish)
* Check LoginAttempt table → Shows logged events with IPs
* Delete user → Record removed completely

# 5. Success criteria

## **5.1 Functional requirements**

* User can sign up with encrypted data storage
* User can log in with rate limiting protection
* User can view profile with decrypted data
* User can change username and password
* User can delete account permanently
* Menu displays items from database
* All UI pages are responsive (mobile, tablet, desktop)

## **5.2 Security requirements**

* Passwords are hashed with Argon2 (visible in database)
* Emails are encrypted with AES-256 (visible in database)
* Rate limiting prevents brute force (testable)
* Sessions are secure (verifiable in DevTools)
* CSRF protection is active (testable)
* SQL injection is prevented (testable)
* XSS is prevented (testable)