CSE 454: Systems Analysis and Design

Course Project

Titanium OS

Advanced Restaurant Management System

Project Report Submitted by:

Abdelrahman ElBatanouny - 232403

Omar Sameh Mohamed Ali - 235153

Mohamed Raed Atef - 234197

Mahmoud Mohamed - 231437

Department of Computer Science

MSA University & Greenwich University

December 12, 2025

Abstract

Titanium OS is a comprehensive, desktop-based Restaurant Management System designed to bridge the gap between Point of Sale (POS) operations and back-end financial treasury management. Built using Python (Tkinter/TTKBootstrap) and MySQL, the system automates order processing, inventory tracking, kitchen workflow management, and financial reporting. Key innovations include a real-time "Wallet" system that tracks net cash flow, automated tax and service charge pooling, and a robust stock control system that deducts inventory upon ordering and restores it upon cancellation. This report details the system's architecture, database design, and implementation strategies.

Contents

Chapter 1

Introduction

1.1 Project Overview

Titanium OS is a desktop-based Restaurant Management System providing a unified interface for Waiters, Chefs, and Administrators. Built with Python (Tkinter/TTKBootstrap) and MySQL (XAMPP), it features a financial "Wallet" system that tracks all transactions in real-time. The system includes five main modules: Menu View, Order Processing, Kitchen Queue, Status/Payment, and Admin Dashboard.

Key features: real-time inventory tracking, integrated wallet system with tax/service/tips pools, automated receipt generation, profit margin calculator, and password-protected admin access.

Repository: https://github.com/Batanounyy/TitaniumOS

1.2 Objectives

- **Automation:** Eliminate manual calculation errors in billing, tax, and service charges.
- **Inventory Control:** Automatically deduct stock when items are ordered and restore stock if orders are cancelled.
- **Kitchen Synchronization:** Provide a digital display for the kitchen to view active orders and mark them as ready.
- **Financial Integrity:** Maintain a secure "Wallet" that tracks cash on hand, separate pools for Taxes and Service charges, and employee tip collection.
- **Customization:** Allow admins to dynamically add tables, change UI themes, and adjust financial rates.

1.3 Scope

The system targets small to medium-sized restaurants requiring an offline, robust management solution. It covers the complete order lifecycle: table selection, menu browsing, kitchen preparation, billing, and financial auditing. In-scope: order management, inventory control, financial tracking, menu CRUD, table management, receipt generation, and data export. Out-of-scope: multi-user login, cloud sync, payment gateways, and employee scheduling.

Chapter 2

System Analysis

2.1 Functional Requirements

1. **Order Management:** Table selection, cart management with stock validation, automatic stock deduction, timestamped orders sent to kitchen.
2. **Kitchen Operations:** View active orders with elapsed time, mark as Ready or Cancel (auto-restores stock).
3. **Billing & Payment:** Automated tax/service calculation, discount/tip support, receipt generation, wallet credit, pool tracking.
4. **Administration:** Wallet management with pools (tax/service/tips), inventory control, menu CRUD, profit margin calculator, order history, CSV export, customizable settings (theme, rates, tables).

2.2 Feasibility Study

Category	**Analysis**
Technical	Python and MySQL are industry standards. The use of `ttkbootstrap` ensures a modern UI on standard hardware.
Economic	Zero licensing costs (Open Source stack). Low maintenance requirements.
Operational	The UI is designed for touch-screens or mouse input, minimizing staff training time.

Table 2.1: Feasibility Analysis

Chapter 3

System Design

3.1 Use Case Diagram

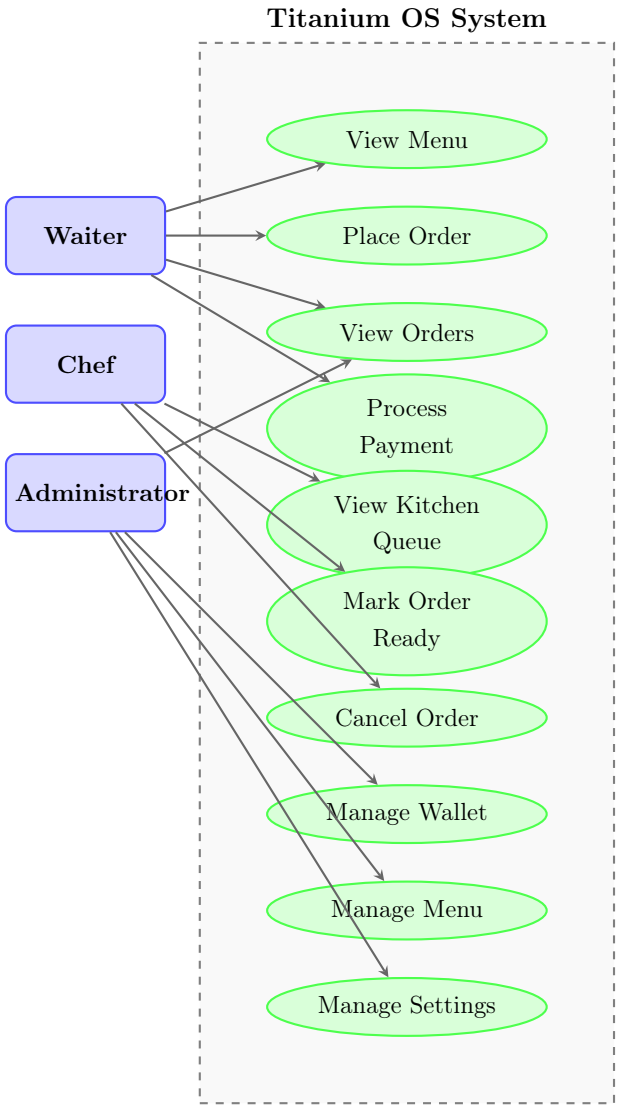

Figure 3.1: Use Case Diagram - Shows interactions between actors (Waiter, Chef, Administrator) and system use cases

3.1.1 Use Case Description

The Use Case Diagram identifies three main actors and their interactions:

Actors:

- **Waiter:** Places orders, views menu/orders, processes payments
- **Chef:** Views kitchen queue, marks orders ready, cancels orders
- **Administrator:** Manages wallet, menu items, settings, and views all orders

Key Use Cases:

- **Order Management:** View Menu, Place Order, View Orders, Process Payment
- **Kitchen Operations:** View Kitchen Queue, Mark Order Ready, Cancel Order
- **Administration:** Manage Wallet, Manage Menu, Manage Settings

3.2 Entity Relationship Diagram (ERD)

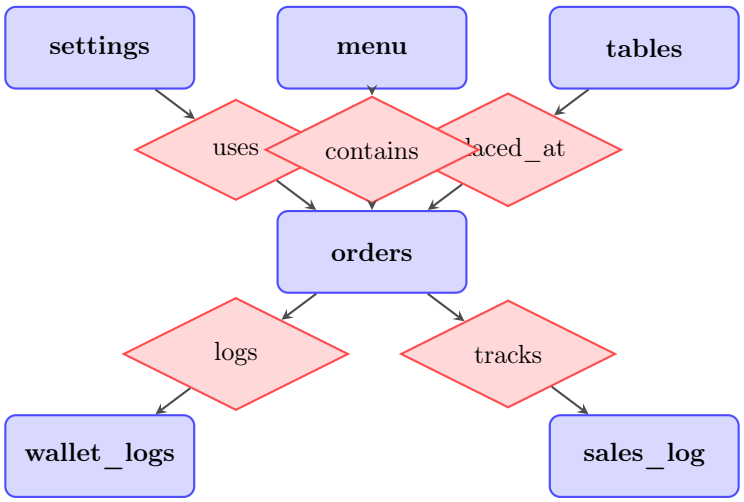

Figure 3.2: Entity Relationship Diagram (ERD) - Shows relationships between database entities

3.2.1 ERD Description

The Entity Relationship Diagram illustrates the database structure and relationships:

- **settings** - Stores system configuration (tax rates, wallet balance, themes, etc.)
- **menu** - Contains menu items with pricing and stock information
- **tables** - Manages restaurant table status and occupancy

- **orders** - Central transaction entity linking tables, menu items, and financial data
- **wallet_logs** - Records all financial transactions for audit purposes
- **sales_log** - Tracks sales analytics and profit calculations

Key Relationships:

- Orders **uses** Settings (for tax/service rates, wallet balance)
- Orders **contains** Menu items (items ordered)
- Orders **placed_at** Tables (table assignment)
- Orders **logs** to Wallet_logs (transaction recording)
- Orders **tracks** in Sales_log (analytics data)

3.3 Database Schema

The system uses MySQL database (`titanium2`) with 6 normalized tables:

- **settings:** Key-value store (app_name, admin_pass, theme, wallet, tax_pool, service_pool, tips_pool, tax_rate, service_rate)
- **menu:** id, name, price, cost, stock
- **tables:** id, table_num (UNIQUE), status, occupied_at
- **orders:** id, table_num, items (TEXT), subtotal, tax, service, total, discount, tip, status, created_at
- **wallet_logs:** id, type, amount, description, date (audit trail)
- **sales_log:** id, item_name, price, cost, date_sold (analytics)

3.4 Data Flow Diagram (DFD) - Level 0

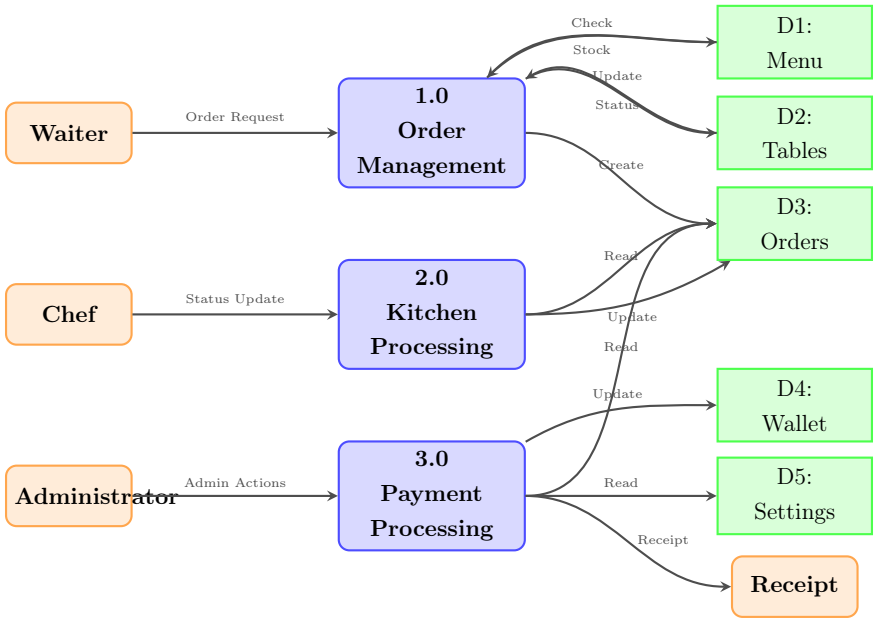

Figure 3.3: Data Flow Diagram (DFD) - Level 0 showing data flows between external entities, processes, and data stores

3.4.1 DFD Description

The Level 0 Data Flow Diagram shows the system's main processes and data flows:

Processes:

1. **Order Management (1.0):**
   - Receives order requests from Waiter
   - Validates stock availability (D1: Menu)
   - Updates table status (D2: Tables)
   - Creates order record (D3: Orders)
2. **Kitchen Processing (2.0):**
   - Receives status updates from Chef
   - Reads pending orders (D3: Orders)
   - Updates order status (Ready/Cancelled)
3. **Payment Processing (3.0):**
   - Receives admin actions for payment
   - Reads order details (D3: Orders)

- Updates wallet balance (D4: Wallet)
- Reads configuration (D5: Settings)
- Generates receipt output

Data Stores:

- **D1: Menu** - Menu items and inventory
- **D2: Tables** - Table status and occupancy
- **D3: Orders** - All order records
- **D4: Wallet** - Financial transaction logs
- **D5: Settings** - System configuration

3.5 Wireframes

This section presents the initial wireframes and screen mockups that guided the user interface design of Titanium OS.

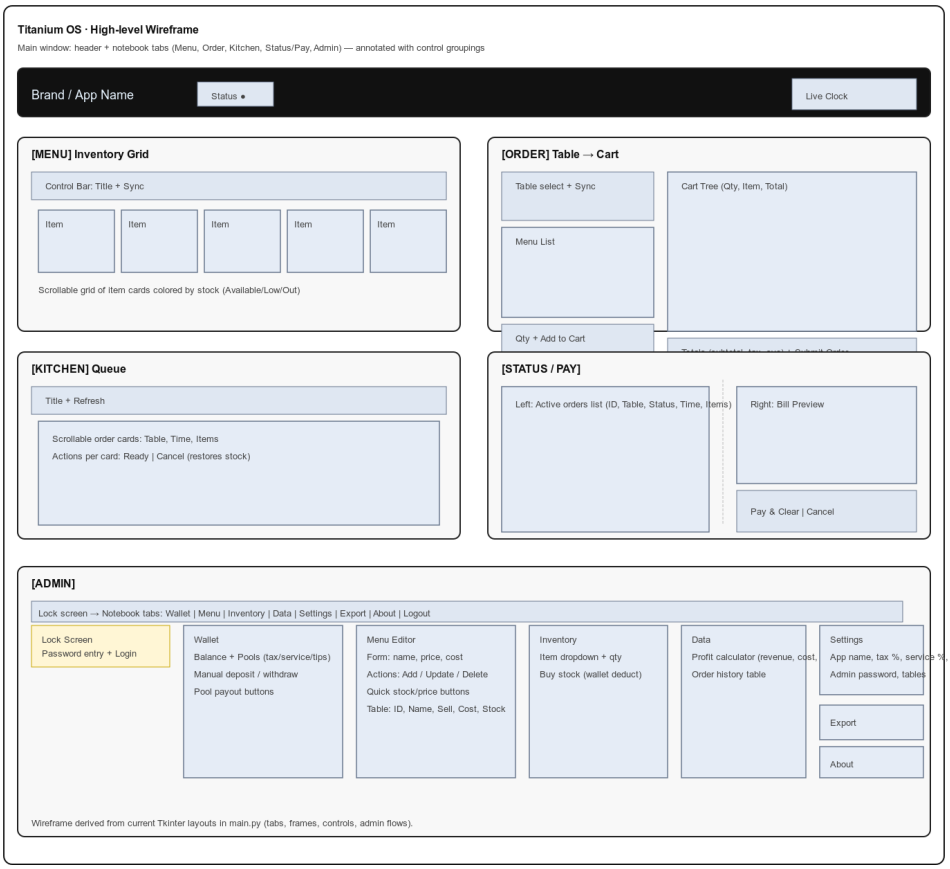

Figure 3.4: Wireframes - Initial UI layout designs for Titanium OS

3.5.1 UI Design Overview

The wireframes and mockups illustrate the system's user interface design:

- **Tab-based Navigation:** Five main tabs (Menu, Order, Kitchen, Status, Admin) for easy access
- **Modern Dark Theme:** Futuristic design with multiple theme options
- **Card-based Layout:** Visual cards for menu items and orders
- **Real-time Updates:** Live data refresh across all interface components
- **Responsive Design:** Optimized for both touch-screen and mouse input

Chapter 4

Technology Stack & Architecture

4.1 Technology Stack

- **Language:** Python 3.x with Tkinter/TTKBootstrap (6 dark themes)
- **Database:** MySQL 8.0 via XAMPP
- **Libraries:** ttkbootstrap, mysql-connector-python, datetime, csv
- **Repository:** GitHub - Batanounyy/TitaniumOS

4.2 System Architecture

Three-tier architecture: Presentation (Tkinter/TTKBootstrap GUI), Business Logic (DB class, TitaniumApp class), Data Layer (MySQL with 6 normalized tables).

Chapter 5

Implementation Details

5.1 Key Algorithms

5.1.1 Stock Management

Stock is automatically deducted on order placement and restored on cancellation by parsing item strings (e.g., "2x Burger, 1x Coke"). Stock validation prevents overselling by checking available inventory against cart quantities before order submission.

5.2 Financial Treasury (Wallet)

The wallet system prevents negative balances and acts as a double-entry ledger. Payment processing: wallet credits (total - discount), tax/service/tips added to respective pools, all transactions logged in wallet_logs. Pool payouts deduct from wallet and reset pools. Profit margin calculated from revenue minus cost. Receipts generated as text files (`receipt_[ID].txt`) with complete order details.

Chapter 6

Testing

6.1 Test Cases

Test Case	**Action**	**Expected Result**
Admin Access	Enter password "1234".	Dashboard loads.
Stock Validation	Order 10 items when stock is 5.	Error: "Not enough stock".
Order Cycle	Place Order → Ready → Pay.	Stock deducted, status updates, receipt generated, wallet credited.
Cancellation	Cancel order from Kitchen/Status.	Stock restored, table available.
Wallet Funds	Buy stock with insufficient funds.	Error: "Insufficient Funds".
Tax/Discount/Tip	Apply rates and discounts.	Correct calculations, pools updated.

Table 6.1: Key Test Cases

All test cases passed successfully, demonstrating robust error handling, accurate calculations, and reliable data persistence.

6.2 Output Screenshots

(Note: Include screenshots of the Menu Grid, Order Tab, and Admin Wallet Dashboard here in the final document.)

Chapter 7

Conclusion

7.1 Summary

Titanium OS successfully delivers a robust Restaurant Management System using Python-MySQL architecture (XAMPP). Key achievements: complete order lifecycle, financial integrity with audit trail, automatic inventory control, modern UI with multiple themes, and secure admin access. The system demonstrates modular code structure, efficient database queries, real-time updates, and comprehensive error handling.

Source Code Repository: https://github.com/Batanounyy/TitaniumOS

7.2 Future Enhancements

- Visual table map with drag-and-drop interface
- Multi-user system with role-based access
- Cloud sync for remote monitoring
- Advanced analytics (sales reports, best-sellers, peak hours)
- Print integration and barcode scanning
- Mobile app companion and email receipts