campusKubo - Boarding Bedspace Finder for Students

Project Report

Executive Summary

campusKubo is a comprehensive web application built with Flet (Python) that serves as a boarding house and bedspace finder specifically designed for students. The platform connects property managers (PMs) who list their boarding houses with students seeking affordable accommodation. The system features role-based access control with three user types: Administrators, Property Managers, and Tenants.

The application provides a complete solution for property listing, reservation management, payment processing, and administrative oversight. Key features include real-time availability checking, secure payment integration, notification systems, and comprehensive reporting tools.

Framework Chosen & Rationale

Primary Framework: Flet (Python) - **Rationale**: - Cross-platform compatibility (Web, Desktop, Mobile) - Pure Python development with no HTML/CSS/JS required - Built-in UI components and state management - Rapid prototyping and development - Strong community support and active development - Seamless integration with existing Python ecosystem

Supporting Technologies: - SQLite database for data persistence - Google Maps API integration for location services - Email services for notifications - File storage system for images and documents

Implemented Features

Baseline Features

- User registration and authentication (Admin, PM, Tenant roles)
- Property listing management with image uploads
- Search and browse functionality with filters
- Reservation system with availability checking
- Basic payment processing
- Notification system
- Responsive web interface

Enhancements Implemented

- Advanced admin dashboard with analytics

- Real-time activity monitoring
- Comprehensive reporting system
- PM verification workflow
- Tenant reservation history
- Advanced search filters
- Google Maps integration
- Email notifications
- File upload and management
- Session management and security

Architecture & Module Overview

The application follows a modular architecture with clear separation of concerns:

```
campusKubo/
  README.md
  requirements.txt
  app/
     __init__.py
     main.py
     __pycache__/
     assets/
        uploads/
     components/
        __init__.py
        admin_stats_card.py
        admin_user_table.py
        admin_utils.py
        advanced_filters.py
        chart_card.py
        dialog_helper.py
        footer.py
        listing_card.py
        login_form.py
        logo.py
        navbar.py
        notification_banner.py
        password_requirements.py
        profile_section.py
        reservation_form.py
        search_filter.py
        searchbar.py
        signup_banner.py
        signup_form.py
        table_card.py
        __pycache__/
```

```
config/
    __init__.py
    colors.py
    __pycache__/
models/
    __init__.py
    listing.py
    notification.py
    payment.py
    reservation.py
    settings.py
    user.py
    __pycache__/
services/
    __init__.py
    activity_service.py
    admin_service.py
    auth_service.py
    gmaps_service.py
    listing_service.py
    notification_service.py
    refresh_service.py
    report_service.py
    reservation_service.py
    settings_service.py
    user_service.py
    __pycache__/
state/
    __init__.py
    app_state.py
    profile_state.py
    session_state_backup.py
    session_state.py
    __pycache__/
storage/
    __init__.py
    db.py
    file_storage.py
    seed_data.py
    __pycache__/
    data/
    temp/
tests/
    __init__.py
    test_components.py
    test_integration.py
```

```
        test_models.py
        test_role_views.py
        test_services.py
        test_utils.py
    utils/
        __init__.py
        navigation.py
        __pycache__/
    views/
        __init__.py
        activity_logs_view.py
        admin_dashboard_view.py
        admin_listings_view.py
        admin_payments_view.py
        admin_pm_verification_view.py
        admin_profile_view.py
        admin_reports_view.py
        admin_reservations_view.py
        admin_settings_view.py
        admin_users_view.py
        browse_view.py
        forbidden_view.py
        home_view.py
        listing_detail_extended_view.py
        listing_detail_view.py
        login_view.py
        my_tenants_view.py
        pm_add_edit_view.py
        pm_dashboard_view.py
        pm_profile_view.py
        privacy_view.py
        profile_view.py
        property_detail_view.py
        reservation_view.py
        rooms_view.py
        signup_view.py
        tenant_dashboard_view.py
        tenant_messages_view.py
        tenant_reservations_view.py
        terms_view.py
        user_profile_view.py
        __pycache__/
docs
    CHANGELOG.md
    IAS Documentation
        Code_Documentation.md
```

```
            Project_Report.md
            Technical_Documentation.md
            Testing_Documentation.md
            User_Manual.docx
        README.md
        SRS.md
```

System Architecture Diagram

```
@startuml System Architecture
!theme plain
skinparam backgroundColor #FEFEFE
skinparam componentStyle uml2

package "Presentation Layer" as PL {
    [Web Interface] as Web
    [Admin Dashboard] as Admin
    [PM Dashboard] as PM
    [Tenant Dashboard] as Tenant
}

package "Application Layer" as AL {
    [Auth Service] as Auth
    [User Service] as UserSvc
    [Listing Service] as ListingSvc
    [Reservation Service] as ReservationSvc
    [Notification Service] as NotifSvc
}

package "Data Layer" as DL {
    [SQLite Database] as DB
    [File Storage] as Files
}

Web --> Auth
Web --> UserSvc
Web --> ListingSvc
Web --> ReservationSvc
Web --> NotifSvc

Auth --> DB
UserSvc --> DB
ListingSvc --> DB
ReservationSvc --> DB
NotifSvc --> DB
```

```
ListingSvc --> Files
UserSvc --> Files
@enduml
```

Threat Model & Security Controls

Threat Model   STRIDE Analysis:

1. **Spoofing**: Unauthorized access attempts
    - Controls: Password hashing, session validation, role-based access
2. **Tampering**: Data modification attacks
    - Controls: Input validation, parameterized queries, CSRF protection
3. **Repudiation**: Denial of actions
    - Controls: Audit logging, activity tracking
4. **Information Disclosure**: Sensitive data exposure
    - Controls: Encryption, access controls, secure file storage
5. **Denial of Service**: System unavailability
    - Controls: Rate limiting, resource monitoring
6. **Elevation of Privilege**: Unauthorized privilege escalation
    - Controls: Role validation, session management

Security Controls Implemented

- **Authentication**: Secure password hashing with bcrypt
- **Authorization**: Role-based access control (RBAC)
- **Session Management**: Secure session handling with expiration
- **Input Validation**: Comprehensive validation for all user inputs
- **SQL Injection Prevention**: Parameterized queries
- **XSS Protection**: Input sanitization and output encoding
- **CSRF Protection**: Token-based protection
- **File Upload Security**: Type validation and secure storage
- **Audit Logging**: Comprehensive activity logging

Design Decisions / Trade-offs

Framework Choice

- **Decision**: Flet over traditional web frameworks
- **Rationale**: Faster development, unified codebase, easier maintenance
- **Trade-off**: Less customization vs. rapid development

Database Choice

- **Decision**: SQLite over PostgreSQL/MySQL
- **Rationale**: Simplicity, no server setup, sufficient for application scale
- **Trade-off**: Concurrency limitations vs. ease of deployment

State Management

- **Decision**: Session-based state management
- **Rationale**: Simplicity and security
- **Trade-off**: Scalability limitations vs. development speed

File Storage

- **Decision**: Local file system over cloud storage
- **Rationale**: Simplicity and cost-effectiveness
- **Trade-off**: Scalability and backup complexity vs. ease of implementation

Limitations & Future Work

Current Limitations

1. **Scalability**: SQLite limitations for high concurrent users
2. **Real-time Features**: Limited real-time updates
3. **Mobile Optimization**: Basic responsive design
4. **Payment Integration**: Basic payment processing
5. **Search Capabilities**: Basic filtering only

Future Enhancements

1. **Database Migration**: PostgreSQL for better scalability
2. **Real-time Updates**: WebSocket integration for live updates
3. **Advanced Search**: Elasticsearch integration
4. **Mobile App**: Native mobile applications
5. **Payment Gateway**: Integration with payment processors
6. **Analytics**: Advanced reporting and business intelligence
7. **API Development**: RESTful API for third-party integrations
8. **Multi-language Support**: Internationalization
9. **Cloud Deployment**: Containerization and cloud hosting
10. **Advanced Security**: OAuth, 2FA, encryption at rest