MikroTik Captive Portal System

Complete Technical Documentation

MikroTik Captive Portal System

Technical Documentation & Implementation Guide

Version 1.0 | April 2025

Table of Contents

1. Core Architecture Files

The application is built on a Flask backend with SQLAlchemy for database interaction and follows an MVC-like structure.

main.py

Purpose: Application entry point and configuration.

Key Components:

- Database connection setup via SQLAlchemy
- Flask application initialization
- Environment variables configuration
- Server startup parameters (host="0.0.0.0", port=5000)

main.py

```
# Database initialization
db = SQLAlchemy(model_class=Base)
app = Flask(__name__)
app.secret_key = os.environ.get("SESSION_SECRET", "default_secret_key_for_
database_url = os.environ.get("DATABASE_URL")
app.config["SQLALCHEMY_DATABASE_URI"] = database_url
app.config["SQLALCHEMY_ENGINE_OPTIONS"] = {
    "pool_recycle": 300,
    "pool_pre_ping": True,
}
db.init_app(app)

# Create tables
with app.app_context():
    import models
    db.create_all()
```

models.py

Purpose: Defines database models using SQLAlchemy ORM.

Key Models:

1. **User:** Stores user information and credentials
    - Regular guests (Google Sheet validation)
    - Special users (staff, family, friends with custom credentials)
2. **LoginSession:** Tracks user login/logout activity
3. **BlockedDevice:** Manages blocked MAC addresses
4. **GoogleCredential:** Stores Google API credentials

Model Relationships:

- One-to-many relationship between `User` and `LoginSession`

> models.py
>
> ```
> class User(db.Model):
>     __tablename__ = 'users'
>
>     id = db.Column(db.Integer, primary_key=True)
>     mobile_number = db.Column(db.String(20), unique=True, nullable=False)
>     room_number = db.Column(db.String(20), nullable=True)  # For guests
>     password = db.Column(db.String(100), nullable=True)    # For special u
>     user_type = db.Column(db.String(20), default='guest')  # guest, staff,
>     is_active = db.Column(db.Boolean, default=True)
>     last_login = db.Column(db.DateTime, nullable=True)
>
>     # Relationships
>     login_sessions = db.relationship('LoginSession', backref='user', lazy=
> ```

app.py

Purpose: Contains all route handlers and application logic.

Key Routes:

- **User Authentication:**
  - `/`: Main index/login page
  - `/login`: Processes login form
  - `/logout`: Handles user logout
- **Admin Panel:**
  - `/admin/login`: Admin authentication
  - `/admin`: Dashboard with network stats
  - `/admin/users`: Registered users list
  - `/admin/sessions`: Login history
  - `/admin/blocked`: Blocked device management
  - `/admin/manage-users`: Special user management
- **API Endpoints:**
  - `/api/users`: Get active users (AJAX)
  - `/api/disconnect_user`: Remove user from network
  - `/api/refresh_sheet`: Update Google Sheet data

Authentication Logic:

- Guest authentication (via Google Sheets)
- Special user authentication (via database)
- Admin authentication (via environment variables)

Key Features:

- User session management with Flask sessions
- Automatic user creation for first-time logins
- MAC address tracking and blocking

2. External Integration Files

mikrotik.py

Purpose: Handles communication with MikroTik router via API.

Key Functions:

- `connect()`: Establishes connection to router
- `get_active_users()`: Retrieves connected clients
- `add_user()`: Authenticates users to the hotspot
- `remove_user()`: Disconnects users
- `_block_mac_address()`: Blocks MAC addresses

Error Handling:

- Connection timeouts
- Authentication failures
- Development mode for offline testing

> mikrotik.py (usage example)
>
> ```
> # Initialize MikroTik API
> mikrotik_api = MikroTikAPI(
>     host=os.environ.get("MIKROTIK_HOST", "192.168.88.1"),
>     username=os.environ.get("MIKROTIK_USERNAME", "admin"),
>     password=os.environ.get("MIKROTIK_PASSWORD", "")
> )
>
> # Get active users
> active_users = mikrotik_api.get_active_users()
>
> # Add a new user
> success = mikrotik_api.add_user(mobile_number, room_number)
>
> # Remove and block a user
> mikrotik_api.remove_user(user_id)
> ```

google_sheets.py

Purpose: Authenticates users against Google Sheets database.

Key Functions:

- `_get_credentials()`: Loads Google API credentials
- `get_credential_sheet()`: Fetches and caches spreadsheet data
- `normalize_room_number()`: Handles different room number formats
- `verify_credentials()`: Validates mobile/room combinations

Room Number Normalization:

Handles multiple formats like:

- R0/r0/r 0 (Room 0)
- F1/f1/f 1 (Floor 1)
- 1 Dorm/dormitory/1dorm (Dormitory 1)

> google_sheets.py
>
> ```
> def verify_credentials(mobile_number, room_number):
>     """Verify mobile and room number against Google Sheet"""
>     sheet_data = get_credential_sheet()
>     normalized_room = normalize_room_number(room_number)
>
>     # Search for matching mobile and room number
>     for row in sheet_data:
>         if (len(row) >= 2 and
>             str(row[0]).strip() == mobile_number and
>             normalize_room_number(str(row[1])) == normalized_room):
>             return True
>
>     return False
> ```

convert_to_pdf.py

Purpose: Generates PDF documentation from Markdown.

Features:

- Converts the MikroTik setup guide to PDF
- Preserves formatting and images
- Uses WeasyPrint for high-quality PDF rendering

3. User Authentication Flow

Guest Authentication

1. User enters mobile number and room number in login form
2. `app.py:login()` receives form data
3. System checks for blocked devices by MAC address
4. `google_sheets.py:verify_credentials()` validates against Google Sheet
5. If valid, user is created or updated in database
6. `mikrotik.py:add_user()` connects user to WiFi
7. Session is created to track login

> **Note:** Google Sheets validation ensures only hotel guests can connect, with mobile numbers matching room assignments.

Special User Authentication (Staff/Family/Friends)

1. User enters mobile number and password in same login form
2. `app.py:login()` recognizes user as special by `user_type` field
3. System validates password directly from database, not Google Sheets
4. Login session is created and tracked in database
5. User preferences and permissions applied based on user type

> **Feature Highlight:** This allows hotel staff, family members, and friends to have permanent access using custom credentials rather than temporary room assignments.

Admin Authentication

1. Admin navigates to /admin/login
2. Credentials validated against environment variables:
    - `ADMIN_USERNAME` (default: "admin")
    - `ADMIN_PASSWORD` (default: "admin123")
3. Upon successful login, admin session created
4. `@admin_required` decorator protects admin routes

> **Security Note:** For production, always change the default admin credentials by setting the environment variables.

4. Admin Panel Features

Dashboard

File: `templates/admin.html`

Purpose: Provides overview of network status

Key Components:

- Real-time active user count
- Data usage statistics
- Connected device list with disconnect options
- Login session history

JavaScript Integration:

- Chart.js for usage graphs
- AJAX polling for real-time updates

User Management

File: `templates/admin_users.html`

Purpose: View and manage regular guests

Features:

- List all users from Google Sheets
- View registration timestamps
- See session counts per user

Special User Management

File: `templates/admin_manage_users.html`

Purpose: Add and manage special users with custom credentials

Features:

- Create staff, family, and friend accounts
- Set custom passwords (not room numbers)
- Block/unblock users
- Delete user accounts
- Tabbed interface for filtering user types

> app.py (Special User Management)
>
> ```
> @app.route('/admin/add-user', methods=['POST'])
> @admin_required
> ```

```
def admin_add_user():
    """Add a new special user (staff, family, friend)"""
    mobile_number = request.form.get('mobile_number')
    password = request.form.get('password')
    user_type = request.form.get('user_type', 'guest')

    # Create user in database
    user = User(
        mobile_number=mobile_number,
        password=password,
        user_type=user_type,
        is_active=True
    )
    db.session.add(user)
    db.session.commit()

    return redirect(url_for('admin_manage_users'))
```

Session Management

File: `templates/admin_sessions.html`

Purpose: Track user login history

Features:

- View login timestamps
- Track session duration
- Monitor data usage
- See connection details (IP/MAC)

Device Blocking

File: `templates/admin_blocked.html`

Purpose: Manage blocked devices

Features:

- View blocked MAC addresses
- See blocking reason and timestamp
- Unblock devices
- Automatic blocking after admin disconnection

> app.py (Device Blocking)

```
@app.route('/admin/block-user/', methods=['POST'])
@admin_required
def admin_block_user(user_id):
    """Block a user"""
    user = User.query.get_or_404(user_id)
    user.is_active = False
    db.session.commit()

    # If user is currently active in MikroTik, disconnect them
    try:
        active_users = mikrotik_api.get_active_users()
        for active_user in active_users:
            if active_user.get('user') == user.mobile_number:
                mikrotik_api.remove_user(active_user.get('id'))

                # Add device to block list if MAC address is available
                mac_address = active_user.get('mac_address')
                if mac_address:
                    blocked_device = BlockedDevice(
                        mac_address=mac_address,
                        mobile_number=user.mobile_number,
                        reason="Blocked by administrator",
                        blocked_by=session.get('admin_username', 'admin')
                    )
                    db.session.add(blocked_device)
```

```
                    db.session.commit()
                break
    except Exception as e:
        logger.error(f"Error blocking user: {str(e)}")

    return redirect(url_for('admin_manage_users'))
```

5. Technical Implementation Details

Database Schema

Migration File: `migrations/update_user_table.sql`

Key Tables:

1. **users**:
   - Primary authentication table
   - Stores both guests and special users
   - Tracks user activity status
2. **login_sessions**:
   - Historical record of logins
   - Tracks session duration
   - Records data usage
3. **blocked_devices**:
   - MAC address blacklist
   - Blocking reason and admin information
   - Active/inactive status

Session Management

Implementation: Flask's `session` object

Key Session Variables:

- `user_mobile`: Mobile number
- `user_room`: Room number or password
- `authenticated`: Login status
- `login_time`: Timestamp
- `login_session_id`: Database record ID
- `mac`: Device MAC address
- `ip`: Device IP address

Error Handling

Strategy: Comprehensive logging and user feedback

Implementation:

- Flask flash messages for user feedback
- Detailed exception logging
- Development mode for offline testing
- Graceful degradation when external services unavailable

Security Considerations

Authentication:

- MAC address tracking and blocking
- Admin-only routes protection
- Session timeout handling

Environment Variables:

- `DATABASE_URL`: PostgreSQL connection
- `MIKROTIK_*`: Router credentials
- `GOOGLE_CREDENTIALS_*`: API authentication
- `ADMIN_*`: Admin panel access

6. Deployment Preparation

Required Environment Variables

Variable	Purpose	Example
`DATABASE_URL`	PostgreSQL connection string	`postgresql://user:pass@host:port/dbn`
`MIKROTIK_HOST`	Router IP address	`192.168.88.1`
`MIKROTIK_USERNAME`	Router admin username	`admin`
`MIKROTIK_PASSWORD`		`password123`

	Router admin password	
`GOOGLE_CREDENTIALS_JSON`	Service account credentials	`{"type": "service_account", ...}`
`SPREADSHEET_ID`	Google Sheet identifier	`1BxiMVs0XRA5nFMdKvBdBZjgmUUqptlbs74O`
`ADMIN_USERNAME`	Override default admin username	`administrator`
`ADMIN_PASSWORD`	Override default admin password	`secure_password_here`
`DEVELOPMENT_MODE`	Enable/ disable offline testing mode	`true` or `false`

Server Configuration

Flask Application:

- Served via Gunicorn on port 5000
- Binds to 0.0.0.0 for external access
- Uses Werkzeug ProxyFix for proper URL generation

Database:

- PostgreSQL with connection pooling
- Connection recycling every 300 seconds
- Pre-ping to verify connections

Static Files:

- Bootstrap CSS for styling
- Admin JavaScript files for dashboard interaction
- Custom CSS for UI enhancements

7. Additional Documentation

MikroTik Router Setup Guide

Files:

- `mikrotik_setup_guide.md`: Markdown source
- `mikrotik_setup_guide.pdf`: Generated PDF

Contents:

- Step-by-step router configuration
- Hotspot setup instructions
- Captive portal integration
- Security best practices
- Troubleshooting guide

This comprehensive system provides a complete solution for hotel/dormitory WiFi authentication with Google Sheets integration and advanced user management capabilities.