Temporary Password System Implementation Guide

Overview

This guide implements a comprehensive temporary password system for your POS application with the following features:

- **Automatic temporary password generation** when creating new users
- **Mandatory password change** on first login
- **Password strength validation** with real-time feedback
- **Password reset functionality** with secure tokens
- **Enhanced security** with password expiration options

1. Database Changes

Step 1: Run Migration Script

Execute the migration script to add new columns to your User table:

```
python

python migration_add_password_fields.py
```

New Database Fields Added:

- `is_temporary_password` (BOOLEAN) - Flags temporary passwords
- `password_reset_token` (VARCHAR(100)) - Secure reset tokens
- `password_reset_expires` (DATETIME) - Token expiration
- `must_change_password` (BOOLEAN) - Forces password change
- `last_password_change` (DATETIME) - Tracks password age

2. File Updates Required

Replace Your Current Files:

1. **models.py** - Updated User model with temporary password methods
2. **forms.py** - New forms for password management
3. **auth.py** - Enhanced authentication routes
4. **decorators.py** - Updated with password change enforcement

New Template Files to Add:

1. **templates/auth/temp_password.html** - Shows generated password to admin
2. **templates/auth/change_password.html** - Password change interface
3. **templates/auth/forgot_password.html** - Password reset request
4. **templates/auth/reset_token.html** - Reset token display (temp)

Update Existing Templates:

1. **templates/auth/login.html** - Add forgot password link
2. **templates/auth/register.html** - Show temporary password info

3. Key Features Implemented

3.1 User Creation Process

```python
# When admin creates a new user:
temp_password = User.generate_temporary_password()  # e.g., "Kj3mN9pR"
user.set_password(temp_password, is_temporary=True)
```

Flow:

1. Admin fills registration form
2. System generates secure temporary password
3. Admin receives credentials to share with user
4. User must change password on first login

3.2 Password Change Enforcement

```python
# Automatic redirection for temporary passwords
if current_user.needs_password_change():
    return redirect(url_for('auth.change_password'))
```

Features:

- Real-time password strength indicator
- Password requirement checklist

- Confirmation matching validation
- Prevention of reusing current password

3.3 Password Reset System

```python
# Generate secure reset token
token = user.generate_password_reset_token(expires_in=3600)  # 1 hour
```

Security Features:

- Time-limited tokens (1 hour expiry)
- Secure token generation using `secrets.token_urlsafe()`
- Token invalidation after use
- No user enumeration (same response for valid/invalid users)

4. Security Enhancements

Password Requirements:

- Minimum 8 characters
- At least one uppercase letter
- At least one lowercase letter
- At least one number
- At least one special character

Temporary Password Generation:

- 8 characters by default
- Excludes ambiguous characters (0, O, l, 1, I)
- Uses cryptographically secure random generation
- Unique per user

Token Security:

- 32-byte URL-safe tokens
- Automatic expiration
- Single-use tokens

- Cleared after password change

5. User Experience Flow

New User Onboarding:

1. **Admin creates user** → System generates temp password
2. **Admin shares credentials** → User receives username + temp password
3. **User logs in** → Automatic redirect to password change
4. **User sets new password** → Access granted to system

Password Reset:

1. **User requests reset** → Enters username
2. **System generates token** → Admin provides reset link
3. **User clicks link** → Enters new password
4. **Password updated** → User can log in normally

Existing Users:

- Current users continue normally
- Admin can reset any user's password to temporary
- Users can change their own passwords anytime

6. Admin Management Features

User Management Interface:

- View all users with password status indicators
- Reset user passwords (generates new temporary password)
- Toggle user active/inactive status
- Delete users (with safety checks)
- View user statistics (sales count, join date)

Enhanced User List Shows:

- Username and role
- Active/inactive status
- Temporary password indicators
- Sales count per user

- Join date

7. Installation Steps

Step-by-Step Implementation:

1. **Backup your database** before running migrations
2. **Run the migration script** to update database schema
3. **Replace your models.py** with the updated version
4. **Update your forms.py** with new form classes
5. **Replace your auth.py** routes with enhanced version
6. **Update decorators.py** with password change enforcement
7. **Add new HTML templates** to your templates/auth/ directory
8. **Update existing templates** (login.html, register.html)
9. **Test the system** with a new user creation

Testing Checklist:

- [ ] Create new user - receives temporary password
- [ ] New user login - forced to change password
- [ ] Password strength validation works
- [ ] Password reset flow functions
- [ ] Existing users can still log in
- [ ] Admin can reset user passwords
- [ ] All decorators enforce password changes

8. Configuration Options

Customizable Settings:

```python
# In your User model, you can adjust:
TEMP_PASSWORD_LENGTH = 8  # Default temporary password length
RESET_TOKEN_EXPIRY = 3600  # Reset token expiry in seconds (1 hour)
PASSWORD_MIN_LENGTH = 8   # Minimum password length
MAX_PASSWORD_AGE_DAYS = 90  # Optional password expiration
```

9. Email Integration (Future Enhancement)

The system is designed to work with email notifications:

```python
# Replace the temporary token display with actual email sending
def send_reset_email(user, token):
    reset_link = url_for('auth.reset_password', token=token, _external=True)
    # Send email with reset_link
```

10. Security Considerations

Best Practices Implemented:

- ✅ Secure password generation
- ✅ Time-limited reset tokens
- ✅ Password strength enforcement
- ✅ No password storage in plain text
- ✅ Protection against user enumeration
- ✅ Prevention of password reuse
- ✅ Automatic token cleanup

Additional Recommendations:

- Consider implementing rate limiting on login attempts
- Add logging for security events
- Regular password expiration (optional feature included)
- Two-factor authentication (future enhancement)

11. Troubleshooting

Common Issues:

- **Migration fails**: Check database permissions and existing column names
- **Templates not found**: Ensure templates are in correct directory structure
- **Import errors**: Verify all new form classes are imported properly
- **Redirect loops**: Check decorator order and password change logic

This implementation provides a robust, secure temporary password system that enhances your POS application's security while maintaining excellent user experience.