Watch Website Project Report

Executive Summary

This report provides an overview of the Watch Website project developed using Python Flask framework. The project aims to create a comprehensive e-commerce platform for watches, featuring user authentication, product catalog management, and shopping cart functionality.

1. Project Overview

1.1 Project Title
Watch Website - E-commerce Platform

1.2 Technology Stack
- **Backend Framework**: Python Flask
- **Database**: Python
- **Frontend**: HTML, CSS, JavaScript
- **Template Engine**: Jinja
- **Authentication**: Flask-Login
- **Forms**: Flask-WTF

1.3 Project Objectives
- Develop a user-friendly watch e-commerce website
- Implement secure user authentication and authorization
- Create an intuitive product browsing and search system
- Build a functional shopping cart and checkout process
- Ensure responsive design for mobile and desktop users

2. System Architecture

2.1 Application Structure
```
watch_website/
├── app.py (main application file)
```

├── models.py (database models)
├── forms.py (WTF forms)
├── templates/ (HTML templates)
├── static/ (CSS, JS, images)
├── requirements.txt

```

### 2.2 Database Design
**Key Tables:**
- Users (user_id, username, email, password_hash, created_at)
- Products (product_id, name, brand, price, description, image_url, stock_quantity)
- Categories (category_id, name, description)
- Orders (order_id, user_id, total_amount, order_date, status)
- Order_Items (item_id, order_id, product_id, quantity, price)

## 3. Key Features Implemented

### 3.1 User Management
- User registration and login system
- Password hashing for security
- Session management
- User profile management

### 3.2 Product Management
- Product catalog with detailed information
- Category-based product organization
- Search and filter functionality
- Product image display

### 3.3 Shopping Cart
- Add/remove items from cart
- Quantity adjustment
- Cart persistence across sessions
- Price calculation

### 3.4 Administrative Features
- Admin dashboard for product management
- Order management system
- User management interface
```

4. Technical Implementation

4.1 Flask Application Setup
```python
from flask import Flask, render_template, request, redirect,
url_for
from flask_login import LoginManager

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key'
```

4.2 Database Models
Key models include User, Product, Category, and Order models with appropriate relationships

4.3 Route Handlers
- Home page route
- Product listing and detail routes
- User authentication routes
- Shopping cart routes
- Admin routes

5. Security Measures

5.1 Authentication Security
- Session-based authentication
- CSRF protection with Flask-WTF
- Input validation and sanitization

6. Testing and Quality Assurance

6.1 Testing Strategy
- Unit testing for individual functions
- Integration testing for route handlers
- User acceptance testing for key workflows
- Performance testing for database queries

6.2 Code Quality
- Following PEP 8 Python style guidelines
- Code documentation and comments
- Error handling and logging

- Code review process

7. Challenges and Solutions

7.1 Session Management
Challenge: Maintaining shopping cart state across user sessions
Solution: Used Flask-Login for session management and database storage for cart persistence

7.2 Image Handling
Challenge: Efficient product image storage and display
Solution: Implemented file upload system with image validation and storage

8. Future Enhancements

8.1 Planned Features
- Payment gateway integration
- Email notifications for orders
- Product reviews and ratings
- Wishlist functionality
- Advanced search filters

8.2 Performance Optimizations
- Database query optimization
- Caching implementation
- Image compression and CDN integration
- API rate limiting

9. Deployment

9.1 Development Environment
- Local development using Flask development server
- Virtual environment for dependency management
- Git version control for team collaboration
-Also hosted in Python Anywhere website

10. Team Collaboration

10.1 Development Workflow
- Daily standups and progress tracking
- Code review process

- Documentation maintenance

10.2 Version Control
- Git branching strategy
- Commit message conventions
- Pull request workflow
- Continuous integration practices

11. Lessons Learned

11.1 Technical Skills
- Gained proficiency in Flask framework
- Improved understanding of web security practices
- Enhanced database design skill

11.2 Project Management
- Importance of clear requirements gathering
- Value of thorough testing procedures
- Benefits of modular code structure
- Significance of documentation

12. Conclusion

The Watch Website project successfully demonstrates the implementation of a full-stack web application using Flask. The project meets all specified requirements and provides a solid foundation for future enhancements. The development process has been valuable for learning modern web development practices and team collaboration skills.

Report Prepared By: [Swayamshree Mishra]
Internship Project: