Software Requirements Specification for Shelf Library Catalog

Toqa EZzatly/D-hub

December 12, 2024

Contents

0.1 Introduction 2
0.1.1 Purpose of this Document 2
0.1.2 Scope of the Project 2
0.1.3 Intended Audience 2
0.1.4 Document Overview 2
0.2 Overall Description 3
0.2.1 Product Perspective 3
0.2.2 Product Functions 3
0.2.3 User Classes and Characteristics 3
0.2.4 Operating Environment 3
0.2.5 Design and Implementation Constraints 3
0.2.6 User Documentation 4
0.2.7 Assumptions and Dependencies 4
0.3 Specific Requirements 5
0.3.1 Functional Requirements 5
0.3.2 Non-Functional Requirements 6
0.3.3 Evolution of the System 6
0.4 Appendix 8
0.4.1 Code Snippet 8
0.5 Diagrams Overview 10

0.1 Introduction

0.1.1 Purpose of this Document

This document specifies the software requirements for the "Shelf" library catalog system. It outlines the functionalities, design, and constraints of the system, ensuring clarity and alignment among the development team, stakeholders, and other involved parties.

0.1.2 Scope of the Project

The "Shelf" library catalog system will enable users to manage their library catalogs, track borrowed items, and keep a record of books they have read. The system will provide a web-based interface, user authentication, book management, and tracking of user reading history.

0.1.3 Intended Audience

This document is intended for all the parties involved in the development and use of the "Shelf" library catalog system, including:

- Developers of the system
- Testers of the system
- Stakeholders who are funding and overseeing the project
- End-users of the system

0.1.4 Document Overview

This document is organized as follows:

- **Introduction:** Provides an overview of the document and project scope.
- **Overall Description:** Gives general descriptions of the product and its operating environment.
- **Specific Requirements:** Outlines the detailed functional and non-functional requirements.
- **Appendix:** Contains any additional information such as code snippets or figures used.

0.2 Overall Description

0.2.1 Product Perspective

The "Shelf" library catalog is a web-based application. It will be built using the Flask framework in Python with an SQLite database for persistence. The system will operate in a client-server architecture, accessible through any web browser.

0.2.2 Product Functions

The core functionalities of the "Shelf" system include:

- **User Authentication:** Secure user registration and login functionality.
- **Book Management:** Add new books to the system, display all available books, and view detailed book information.
- **User Book Tracking:** Ability for users to track their read books.

0.2.3 User Classes and Characteristics

The primary user of this system will be:

- **Regular Users:** Users who need to manage and track their books, including the record of books that they have read.

0.2.4 Operating Environment

The "Shelf" system will operate in the following environment:

- **Web Browser:** Any modern web browser that supports HTML5, CSS3, and JavaScript
- **Server:** Flask server on a standard web server
- **Database:** SQLite database for data persistence.

0.2.5 Design and Implementation Constraints

The following constraints will affect the design and implementation:

- **Database:** The application uses SQLite, which may not be ideal for large-scale user and book data.
- **Security:** The app has basic user authentication and does not implement advanced security practices.
- **Scalability:** the app is not designed to support a very large number of users and books.

0.2.6 User Documentation

User documentation will be provided via:

- **Built-in tooltips** that will give context of the application.
- **README.md** a file detailing any specific steps for deploying the application.

0.2.7 Assumptions and Dependencies

The application assumes:

- **Python:** A python interpreter is installed in the running machine
- **Pip:** The python package manager is available to install dependencies
- **Web Browser:** User has access to the web browser in the machine running the app.

0.3 Specific Requirements

0.3.1 Functional Requirements

FR1: User Registration

Description: Users should be able to register using a unique username and password. **Input:** Username, password, and confirm password. **Processing:** Validate that the username is unique, both passwords match and save the user info to the database. **Output:** Success message or error message in case of any validation issues.

FR2: User Login

Description: Registered users should be able to log in using their username and password. **Input:** Username, password. **Processing:** Verify credentials against the database. **Output:** If valid user, redirects to the home page. Otherwise, error message on the login page.

FR3: Book Addition

Description: Users should be able to add new books to the system, including title and author. **Input:** Book title, author. **Processing:** Save the book info to the database **Output:** Success message, display the current book on the book listing.

FR4: Book Listing

Description: Users should be able to view all the available books in the system. **Input:** None **Processing:** Fetch all books from the database **Output:** Display of all books in the database

FR5: Book Details

Description: Users should be able to view the details of a book. **Input:** Unique ID of the book. **Processing:** Fetch the requested book from the database **Output:** Display of all book details.

FR6: User Book Tracking

Description: Users should be able to track the books they have read. **Input:** Unique ID of the book. **Processing:** Save the user and book association to the database. **Output:** Success message, and a redirect to the books page.

FR7: View User Books

Description: Users should be able to see the list of books they have recorded as read. **Input:** Unique ID of the logged-in user. **Processing:** Fetch the user and book association and output the list of books. **Output:** List of all read books.

FR8: User Logout

Description: Users should be able to log out from the system. **Input:** None **Processing:** Terminate the user session. **Output:** Redirect to the home page.

0.3.2 Non-Functional Requirements

NFR1: Usability

Description: The system must be intuitive and easy to use for all users. User interfaces shall be clear, concise and easily navigable.

NFR2: Performance

Description: The system should perform all actions with minimum response time. The loading times for page shall be less than 3 seconds.

NFR3: Security

Description: Users should be able to access the system with their login credentials. Passwords should not be stored in plain text.

NFR4: Maintainability

Description: The application should be designed in a modular fashion to ease future changes and debugging.

NFR5: Portability

Description: The web application should be easily deployed and run on different OS.

0.3.3 Evolution of the System

This section details all major changes and refactorings done to the application

Evolution 1: Initial Set-Up

Initial setup of the application with database, models, views, login, registration and tracking.

- Models created for Users, Books, and UserBooks.
- Login and Registration pages created with validation.
- Ability to Add new books and track read books.

Evolution 2: Styling and Aesthetic

Initial Styling of the application following a color palette and a vintage font

- Added CSS styles based on the color palette image.
- Added a vintage font to enhance the look and feel.
- The application was styled to have a solid background with each section colored based on the palette.

Evolution 3: Smoother Design

Smoother UI and gradients and animation

- Added smooth gradients and transition in the styling.
- The application was styled to have smooth transitions, gradient and better card looks.
- Refactor and made the UI smoother

Evolution 4: Text Size and Weight

Changed text weight and size

- All the text in the application was made larger and heavier.
- Added a fade-in animation to all the sections.

0.4 Appendix

0.4.1 Code Snippet

Python Code (app.py)

```
import os
from flask import Flask, render_template, redirect, url_for, flash, request
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, current_user, logout_user, login_required
from models import db, User, Book, UserBook
from forms import RegistrationForm, LoginForm, BookForm
from werkzeug.security import generate_password_hash

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///shelf.db'
db.init_app(app)

login_manager = LoginManager(app)
login_manager.login_view = 'login'


@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


@app.route('/')
def home():
    return render_template('home.html')


@app.route('/register', methods=['GET', 'POST'])
def register():
    form = RegistrationForm()
    if form.validate_on_submit():
        user = User(username=form.username.data)
        user.set_password(form.password.data)
        db.session.add(user)
        db.session.commit()
        flash('Registration successful! Please log in.')
        return redirect(url_for('login'))
    return render_template('register.html', form=form)


@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user and user.check_password(form.password.data):
            login_user(user)
            return redirect(url_for('home'))
        else:
```

```
            flash('Login failed. Please check your credentials.')
    return render_template('login.html', form=form)


@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('home'))


@app.route('/books', methods=['GET', 'POST'])
@login_required
def books():
    form = BookForm()
    if form.validate_on_submit():
        book = Book(title=form.title.data, author=form.author.data)
        db.session.add(book)
        db.session.commit()
        flash('Book Added successfully')
        return redirect(url_for('books'))

    all_books = Book.query.all()
    return render_template('books.html', form=form, books=all_books)


@app.route('/book/<int:book_id>')
@login_required
def book_detail(book_id):
    book = Book.query.get_or_404(book_id)
    return render_template('book_detail.html', book=book)


@app.route('/user-books')
@login_required
def user_books():
    user_id = current_user.id
    user_books = UserBook.query.filter_by(user_id=user_id).all()
    return render_template('user_books.html', user_books=user_books)


@app.route('/add-user-book/<int:book_id>')
@login_required
def add_user_book(book_id):
    book = Book.query.get_or_404(book_id)
    user = current_user
    if UserBook.query.filter_by(user_id=user.id, book_id=book.id).first
    ():
        flash('You have already recorded this book before!')
        return redirect(url_for('books'))

    user_book = UserBook(user_id=user.id, book_id=book.id)
    db.session.add(user_book)
    db.session.commit()
    flash(f'You have Recorded the book: {book.title}')

    return redirect(url_for('books'))

```

```
107
108 if __name__ == '__main__':
109       with app.app_context():
110          db.create_all()
111       app.run(debug=True)
```

Listing 1: Main app logic (app.py)

0.5 Diagrams Overview

This section provides an overview of the diagrams used in the system. You can navigate to each diagram by clicking the links below:

- Functional Diagrams
- Sequence Diagram
- Data Flow Diagram

Revision History

Version	Date	Author(s)	Description
1.0	2024-12-1	Toqa Ezzatly	A Very basic and proto-type
1.1	2024-12-1	Toqa Ezzatly	Proto-type with some edit option
1.2	2024-12-1	Toqa Ezzatly	UI design
2.0	2024-12-3	Toqa Ezzatly	First released version