Low Level Design (LLD)

# New and Old book Purchase System (Web Application)

Revision Number: 1.0

Last date of revision: 10/01/2024

# Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| **Date Issued** | **Version** | **Description** | **Author** |
| 10/01/2024 | 1.0 | Initial LLD – V1.0 | Rahul Suregaonkar |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**

[Document Version Control 2](#_heading=h.gjdgxs)

Abstract 4

1. [Introduction](#_heading=h.1fob9te) 5
   1. [Why this Low-Level Design Document? 5](#_heading=h.3znysh7)
   2. [Scope 6](#_heading=h.2et92p0)
   3. Constraints 6
   4. Risks 6
   5. Out of Scope 7
2. System Architecture 7
   1. System Architecture Workflow 7
   2. Django System Architecture 8
   3. Django ORM Architecture 8
   4. Data Base 9
   5. Web Server 12
   6. Application Server 15
   7. Logging and Monitoring 16
   8. User Interface Workflow 19
   9. Test Cases 20
3. Conclusion 21

# Abstract

Embark on a literary journey with our meticulously crafted web application, a synthesis of cutting-edge technology and user-centric design. Leveraging the Django framework, this platform seamlessly integrates a robust eCommerce module and an innovative book exchange marketplace. Users can not only browse, purchase, and discover their next favourite read but also contribute to sustainable reading habits by selling or exchanging their previously cherished books.

This comprehensive experience is underpinned by secure transaction handling, a captivating in-built blog system for literary exploration, and a streamlined data flow. The user interface, adorned with responsive design principles, offers an immersive journey through the realms of literature. Shared authentication ensures a unified experience, while the scalable architecture paves the way for future enhancements.

In the intersection of security, performance optimization, and user engagement, this web application emerges as a versatile and dynamic solution, beckoning bibliophiles to explore, engage, and contribute to a thriving online literary community.

# Introduction

## Why this Low-Level Design Document?

The purpose of this Low-Level Design (LLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding and can be used as a reference manual for how the modules interact at a high level.

## Main Objectives:

### User Registration and Authentication:

* Implement a secure user registration system that allows readers to create an account on the application.
* Enable authentication mechanisms to ensure secure access to user accounts.

### Book Catalog and Purchase:

* Develop a user-friendly interface for readers to search and browse through the book catalogue.
* Implement a seamless purchasing system that enables users to add books to their cart and complete the purchase process.

### Order Tracking:

* Provide a functionality for users to track the status of their ordered books.
* Display relevant information such as order confirmation, shipping details, and expected delivery dates.

### Payment Processing:

* Integrate a secure and reliable payment gateway to facilitate smooth transactions.
* Ensure support for various payment methods and implement secure handling of sensitive financial information.

### Book Selling:

* Allow users to upload details of books they wish to sell, including information such as title, author, condition, and price.
* Implement a streamlined process for users to add new books to the catalog for potential buyers.

### Historical Tracking:

* Develop a comprehensive system for users to track and view their historical activity on the platform.
* Include a log of purchased books, sold books, and any other relevant transactions.

### Feedback System:

* Enable users to provide feedback for each book they have purchased or sold.
* Implement a rating and review system to gather valuable insights for other potential buyers.

### Community Interaction:

* Create a platform for users to engage in discussions on various topics related to books.
* Implement features such as forums or chat rooms to foster a sense of community among readers.

## Scope

The LLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The Low-Level Design (LLD) document outlines the detailed specifications and architecture of the web application designed to sell both new and old books. This document serves as a comprehensive guide for developers and stakeholders involved in the implementation phase.

## Constraints

* + - Server resources, including processing power, memory, and storage, may impose constraints on the application's scalability and performance under high loads.
    - Reliance on external services and APIs, such as Stripe for payments, introduces dependencies that are subject to the stability and maintenance of these third-party providers.
    - Timely development and deployment are essential. Delays in the development process may impact the release schedule and user expectations.
    - Time Constraints.
    - The project must operate within specified budgetary constraints, influencing the selection of tools, technologies, and the scope of certain features.

## Risks

Document specific risks that have been identified or that should be considered.

## 1.5 Out of Scope

### 

### Network Configuration:

* Configuration of networking components, such as firewalls and routers.

### Third-Party Services Integration:

* Detailed implementation of third-party services not directly related to the outlined features.

### Mobile Application Development:

* Development considerations for a mobile application version.

### Browser Compatibility:

* Specifics on ensuring compatibility with various web browsers.

### Load Testing and Scalability:

* Detailed performance testing and scalability measures.

### Legal and Compliance Aspects:

* Legal requirements, licenses, or compliance with specific industry standards.

### Internationalization and Localization:

* Adaptation of the application for different languages or regions.

### User Training:

* Training materials or sessions for end-users.

# 2. System Architecture

# 2.1 System Architecture Workflow

A diagram of a computer

Description automatically generated

The system architecture of a Django web application typically follows the Model-View-Controller (MVC) architectural pattern, where Django introduces its own flavor known as Model-View-Template (MVT). Below is an overview of the system architecture of a Django framework:

## 2.2 Django System Architecture:

A diagram of a server

Description automatically generated

### 1. **Client-Side:**

### **Web Browser:**

* + Renders HTML, CSS, and JavaScript.
  + Handles user interactions and sends HTTP requests to the Django server.

### 2. **Server-Side:**

### **Web Server (e.g., Nginx or Apache):**

* Serves static files (CSS, JavaScript, images).
* Manages SSL/TLS termination for secure communication.
* Forwards dynamic requests to the Django application server.

### **Django Application Server (e.g., Gunicorn or uWSGI):**

* Executes the Django web application.
* Manages multiple worker processes to handle concurrent requests.

### 3. Django Framework Components (MVT):

### **Models:**

* Represents the data structure of the application.
* Defines database schema and relationships.
* Utilizes Django ORM (Object-Relational Mapping) for database interactions.

### **Views:**

* Handles the application's business logic.
* Receives requests from the client, processes data, and returns responses.
* Interacts with models to fetch or update data.

### **Templates:**

* Responsible for generating HTML dynamically.
* Embeds Python-like code to create dynamic content.
* Renders the final HTML displayed to the user.

### 4. **Database:**

### **Database Management System (DBMS):**

* Django supports various databases such as PostgreSQL, MySQL, SQLite, and Oracle.
* Interacts with the database using Django ORM.
* Manages data persistence and retrieval.

### 5. **Middleware:**

### **Django Middleware:**

* Components that process requests and responses globally before reaching views or after leaving views.
* Examples include authentication middleware, session middleware, and CSRF protection middleware.

### 6. URL Dispatcher:

### **URLconf (URL Configuration):**

* Maps URLs to view functions.
* Decides which view function to call based on the URL pattern.

### 7. **Settings:**

### **Django Settings:**

* Configures the behavior of the Django application.
* Includes database settings, middleware configurations, security settings, etc.

### 8. **Static Files and Media:**

### **Static Files:**

* CSS, JavaScript, images, and other files served directly by the web server.
* Managed by the **STATICFILES\_DIRS** setting.

### **Media Files:**

* User-uploaded files (images, documents).
* Managed by the **MEDIA\_ROOT** and **MEDIA\_URL** settings.

## Interaction Flow:

1. The client sends an HTTP request to the web server.
2. The web server forwards dynamic requests to the Django application server.
3. Django's URL dispatcher determines the appropriate view function based on the URL.
4. The view function processes the request, interacts with models if needed, and returns a response.
5. The template engine renders dynamic content, creating the final HTML.
6. The response is sent back to the client for display in the web browser.

# 2.4 Data Base

Django web application with PostgreSQL as the RDBMS managed by psycopg2, the database section outlines the structure and organization of data within the system.

### Entities and Attributes:

* + - Identify main entities (e.g., User, Post, Comment) and define their attributes.

### Relationships:

* + - Define relationships between entities, specifying cardinality and connections.

### Tables and Fields:

* + - Map entities to PostgreSQL tables and define fields (columns) for attributes.

### Primary and Foreign Keys:

* + - Specify primary keys for unique record identification.
    - Use foreign keys for establishing relationships between tables.

### Indexes:

* Identify columns for indexing to optimize query performance.

### Constraints:

* + - Define constraints to ensure data integrity (e.g., unique constraints, check constraints).

### Normalization and Denormalization:

* + - Apply normalization techniques to organize data efficiently.
    - Consider denormalization for specific performance scenarios.

### Psycopg2 Integration:

* Specify psycopg2 as the PostgreSQL adapter in Django settings.
* Configure database connection parameters.

### Considerations:

* Plan for transaction management to ensure ACID properties.
* Outline data migration strategies.
* Implement security measures for sensitive data.
* Establish backup and recovery procedures.
* Consider scalability requirements.

# 

### 

# 2.5 Web Server

A diagram of a software company

Description automatically generated

The web server section handles the communication between clients and the Django application. Nginx is used as the front-end server to serve static files, handle SSL termination, and forward dynamic requests to the Gunicorn application server.

### **Components:**

### **Nginx:**

* Serves as the front-end server.
* Manages SSL termination.
* Handles static file serving.
* Acts as a reverse proxy to forward requests to Gunicorn.

### **Gunicorn:**

* Acts as the application server for Django.
* Processes dynamic requests forwarded by Nginx.

### **Configuration Steps:**

### **Nginx Configuration:**

### **SSL Configuration:**

* + - Generate or obtain SSL certificates.
    - Configure Nginx to use SSL with the obtained certificates.
    - Set up SSL protocols and ciphers for security.

### **Static File Serving:**

* + - Define the root directory for serving static files.
    - Configure Nginx to handle requests for static content directly.

### **Reverse Proxy for Django:**

* + - Configure Nginx to act as a reverse proxy for Gunicorn.
    - Forward requests to the Gunicorn application server.

### **Load Balancing (Optional):**

* + - Implement load balancing if multiple Gunicorn instances are used.
    - Adjust Nginx configuration to distribute requests among Gunicorn instances.

## **Gunicorn Configuration:**

### **Bind Address and Port:**

* Configure Gunicorn to bind to a specific address and port.
* Ensure Nginx forwards requests to the correct address and port.

### **Number of Workers:**

* Set the number of worker processes based on server resources.
* Consideration for optimal performance and resource utilization.

### **Security Configuration:**

* Limit access to Gunicorn to trusted IP addresses if applicable.
* Enable Gunicorn's security features, such as secure headers.

## **Security Considerations:**

### **Nginx Security:**

### **Firewall (iptables):**

* Configure iptables rules to allow traffic on necessary ports (e.g., 80, 443).
* Restrict access to only trusted IP addresses.
* Implement rate limiting to mitigate potential DDoS attacks.

### **SSL Security:**

* Ensure SSL protocols and ciphers are configured securely.
* Regularly update SSL certificates.
* Implement HTTP Strict Transport Security (HSTS).

### **Error Handling:**

* Customize error pages to avoid exposing sensitive information.
* Log errors securely.

## **Gunicorn Security:**

### **Access Control:**

* Restrict access to Gunicorn to trusted IP addresses.
* Avoid exposing Gunicorn to the public network.

### **Worker Processes:**

* Set appropriate values for the number of worker processes.
* Monitor and adjust based on server load and performance.

### **Security Headers:**

* Enable security headers in Gunicorn to enhance security.
* Implement measures to prevent cross-site scripting (XSS) attacks.

## **Logging and Monitoring:**

### **Nginx Logging:**

* Configure Nginx to log access and error information.
* Store logs securely and regularly review them.

### **Gunicorn Logging:**

* Enable Gunicorn logging for access and error information.
* Integrate with a logging system for centralized monitoring.

# 2.6 Application Server

The application server section is responsible for executing the Django web application. Gunicorn (Green Unicorn) is chosen as the application server to handle dynamic requests from clients.

## **Components:**

### **Gunicorn (Green Unicorn):**

* Acts as the application server for Django.
* Manages multiple worker processes.
* Communicates with the Django web application.

## **Gunicorn Configuration:**

### **Bind Address and Port:**

* Configure Gunicorn to bind to a specific address and port.
* Ensure it matches the address and port configured in Nginx.

### **Number of Workers:**

* Set the number of worker processes based on server resources.
* Consideration for optimal performance and resource utilization.

### **Concurrency Model:**

* Choose an appropriate concurrency model (sync, async) based on application requirements.
* Adjust the number of worker processes accordingly.

### **Worker Class:**

* Select a worker class suitable for the application's needs (e.g., sync, gevent).
* Configure worker class parameters.

### **Request Timeout:**

* Set the maximum allowed time for a request to be processed.
* Prevents long-running requests from affecting server performance.

### **Security Configuration:**

* Limit access to Gunicorn to trusted IP addresses if applicable.
* Enable Gunicorn's security features, such as secure headers.

### **Logging Configuration:**

* Configure Gunicorn logging settings for access and error logs.
* Determine log file locations and log formats.

## **Security Considerations:**

### **Access Control:**

* Restrict access to Gunicorn to trusted IP addresses.
* Avoid exposing Gunicorn to the public network.

### **Worker Processes:**

* Set appropriate values for the number of worker processes.
* Monitor and adjust based on server load and performance.

### **Security Headers:**

* Enable security headers in Gunicorn to enhance security.
* Implement measures to prevent cross-site scripting (XSS) attacks.

## **Monitoring and Performance:**

### **Performance Tuning:**

* Adjust Gunicorn settings based on server hardware and application load.
* Monitor CPU and memory usage to optimize configuration.

### **Health Checks:**

* Implement health checks to monitor the status of Gunicorn workers.
* Ensure Gunicorn automatically restarts workers in case of failures.

# 2.7 Logging and Monitoring

## Access Logs:

### Configuration:

Define the access log format in Nginx configuration.

Example: log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /var/log/nginx/access.log main;

### Information Logged:

IP address, user identity, date, request, status, bytes sent, referrer, user agent, and forwarded-for header.

### Error Logs:

### Configuration:

Specify the error log file and logging level.

Example: error\_log /var/log/nginx/error.log error;

## Gunicorn Logging

### Access Logs:

### Configuration:

Specify the access log format and file.

Example: gunicorn myapp.wsgi:application --access-logfile /var/log/gunicorn/access.log --access-logformat '%(h)s %(l)s %(u)s %(t)s "%(r)s" %(s)s %(b)s "%(f)s" "%(a)s"'

### Information Logged:

IP address, user identity, date, request, status, bytes sent, referrer, user agent.

## Error Logs:

### Configuration:

Specify the error log file.

Example: gunicorn myapp.wsgi:application --error-logfile /var/log/gunicorn/error.log

## Gunicorn Monitoring:

### Gunicorn StatsD Integration:

### Configuration:

Install the **statsd** package. Configure Gunicorn to send metrics to a StatsD server.

Example: gunicorn myapp.wsgi:application --statsd-host=localhost:8125 --statsd-prefix=myapp

### Monitoring:

Set up a StatsD server and use monitoring tools like Grafana for visualization.

## Log Rotation:

## Nginx:

### Configuration:

Configure log rotation using tools like **logrotate**.

Example : /var/log/nginx/\*.log {

daily

missingok

rotate 14

compress

delaycompress

notifempty

create 0640 www-data adm

sharedscripts

postrotate

[ -f /var/run/nginx.pid ] && kill -USR1 `cat /var/run/nginx.pid`

endscript

}

## Gunicorn:

### Configuration:

Implement log rotation using **logrotate** or similar tools.

Example: /var/log/gunicorn/\*.log {

daily

missingok

rotate 14

compress

delaycompress

notifempty

create 0640 www-data adm

sharedscripts

postrotate

/bin/kill -s HUP `cat /var/run/gunicorn/myapp.pid 2>/dev/null` 2>/dev/null || true

endscript

}

# 2.8 User Interface Workflow

A diagram of a flowchart

Description automatically generated

## 2.9 Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Module/Model Tested | Description | Expected outcome |
| Customer Model Test | Customer | Test customer model functionality | pass |
| Product Model Test | Profile | Test profile model functionality | pass |
| Address Model test | Address | Test Address model  Functionality | pass |
| Customer Account Manager Model Test | Customer Account Manager | Test Customer Account manager test functionality | pass |
| User View Test | User View | Test User View Functionality | pass |
| Basket View Test | Basket View | Test Basket Functionalities | pass |
| Category Model Test | Product Category | Test Category Functionality | pass |
| Product Model Test | Product | Product Model testing | pass |
| Search view Test | Product Search | Product search test | pass |
| All Products View Test | Product view | Product view test | pass |
| Product Detail view Test | Product Detail | Product Detail test | Pass |
| Comments system Test | Comments | Comments System testing | Pass |

# Conclusion

In the Low-Level Design (LLD) for the BookVerse web application, a comprehensive and robust architecture has been crafted, encompassing features for buying and selling books within the same platform. The incorporation of a built-in blogging system enhances the user experience, providing a multifaceted platform for literature enthusiasts.

## Key Features:

## E-Commerce Capabilities:

The application facilitates seamless buying and selling of books through an intuitive and user-friendly interface.

## Integrated Blogging System:

An in-built blogging system enriches the user experience, allowing users to share insights, reviews, and literary content within the BookVerse community.

## Gunicorn Application Server:

Gunicorn is employed as the WSGI server, efficiently managing dynamic content processing and contributing to high concurrency and responsiveness.

## Nginx Load Balancing:

Nginx serves as a powerful load balancer, distributing incoming traffic across multiple Gunicorn instances, ensuring optimal resource utilization and high availability.

## Redis Integration:

Redis plays a pivotal role in performance optimization, serving as a caching layer, facilitating efficient session storage, and supporting real-time communication through Pub/Sub messaging.

## Ajax and jQuery Implementation:

The incorporation of Ajax and jQuery enhances the application's interactivity, providing a seamless and dynamic user experience. These technologies contribute to the real-time updating of content, improving responsiveness without the need for full-page reloads.

## User-Centric Approach:

The BookVerse web application is designed with a user-centric approach, offering a feature-rich environment that caters to the diverse needs of book enthusiasts. The platform seamlessly combines e-commerce functionalities, blogging capabilities, and real-time interactions to create a vibrant and engaging community.

## Future Enhancements:

The extensible architecture allows for future enhancements, such as the introduction of additional features, improved scalability, and continuous optimization to meet evolving user requirements.

In conclusion, BookVerse stands as a dynamic and versatile web application, bringing together the worlds of commerce, literature, and community. The thoughtful integration of various technologies and features creates a compelling platform that fosters a rich and interactive experience for users passionate about books and literary discussions.