

**THE UNIVERSITY OF ZAMBIA**

**SCHOOL OF NATURAL SCIENCES**

**DEPARTMENT OF COMPUTER SCIENCES**

**ASSIGNMENT 1: Design and Implement a Simple Database Management System for a University Library**

**NAME: MWALE Jonathan**

**COMPUTER NUMBER: 2021512967**

**COURSE NAME: Databases & Information Management Systems.**

**COURSE CODE: CSC 2702**

**DUE DATE: Sunday, 20 October 2024**

**ATTENTION: Zipani Tom Sinkala**

# Executive Summary

This report outlines my individual contributions to the University Library Management System project. My responsibilities included implementing the Django models, configuring the multiple database architecture, and customizing the admin interface with Django Unfold. The project was successfully delivered, offering a comprehensive library management system built on Django, PostgreSQL, and a user-friendly administrative interface.

# Introduction

As a core member of the development team, I focused on the following key areas:

- Implementing data models for the core entities of the system

- Configuring multiple databases within Django

- Customizing the Django admin interface using Django Unfold

- Creating documentation for system setup and usage

This report details my technical contributions, challenges, and solutions, as well as the impact and future recommendations for the project.

# System Implementation

## Data Models

I implemented Django models for core entities in the library system, ensuring the correct relationships and constraints for accurate data management.

Core Models:

- Book: Tracks metadata for each book, including ISBN, title, publisher, and publication year.

- Author: Manages author data and the relationship between authors and books.

- BookCopy: Tracks the individual copies of books and their availability status.

- Student: Manages student data and loan eligibility.

- Loan: Handles the loaning of books, including due dates and loan status.

- Fine: Calculates and manages fines for overdue books.

Key model example for `Book`:

```python

class Book(models.Model):

id = models.BigAutoField(primary\_key=True)

isbn = models.CharField(max\_length=13)

title = models.CharField(unique=True, max\_length=255)

publisher = models.ForeignKey("Publisher", models.CASCADE)

publication\_year = models.SmallIntegerField()

authors = models.ManyToManyField("Author", through="BookAuthor", blank=False)

@property

def is\_available(self):

copies = BookCopy.objects.filter(book=self.id, status="Available").count()

return copies > 0

@property

def available\_copies(self):

return BookCopy.objects.filter(book=self.id, status="Available").count()

```

The implementation of these models ensured robust relationships, including:

- Many-to-many relationships between `Books` and `Authors`

- One-to-many relationships for `Publisher` to `Books`

- Status tracking for each `BookCopy` to manage availability

## Admin Interface Customization

I customized the Django admin interface using the Django Unfold library, which provided a responsive, modern interface for library staff. The admin interface was tailored to support efficient library operations, with features such as inline editing, search capabilities, and custom validation for related models.

Example of the `BookAdminClass` customization:

```python

@admin.register(Book)

class BookAdminClass(ModelAdmin):

inlines = [BookAuthorInline]

list\_display = ["title", "isbn", "total\_copies", "available\_copies"]

search\_fields = ["title", "publication\_year", "publisher\_\_name"]

def save\_related(self, request, form, formsets, change):

super().save\_related(request, form, formsets, change)

if not form.instance.bookauthor\_set.exists():

raise ValidationError("At least one author is required.")

```

Key customizations included:

- Inline editing of related models (e.g., adding authors to books)

- Advanced search functionality across multiple fields for efficient record lookup

- Custom validation to ensure data integrity, such as ensuring each book has at least one author

## Forms and Business Logic

I also developed custom forms that integrated business rules to streamline operations like loan processing. For example, the `LoanForm` dynamically filtered available copies of books to ensure only those available for loan were presented.

```python

class LoanForm(ModelForm):

class Meta:

model = Loan

fields = "\_\_all\_\_"

def \_\_init\_\_(self, \*args, kwargs):

super().\_\_init\_\_(\*args, kwargs)

self.fields["book\_copy"].queryset = BookCopy.objects.filter(status="Available")

```

Key business logic features:

- Automatic filtering of available book copies for loans

- Automated fine calculation for overdue books

- Support for returning books and updating status in real-time

# Technical Achievements

## Database Integration

I successfully implemented multiple database support and complex model relationships:

- Many-to-many relationships between books and authors with custom through models

- Automated status tracking for book copies and loans

- Comprehensive data validation to maintain integrity across models

## Business Logic Implementation

The system includes:

- Book availability tracking: Dynamically updates book availability based on loans and returns.

- Loan processing: Automatically calculates due dates and tracks overdue items.

- Fine management: Automatically generates fines for overdue books.

# System Features

## Book Management

- ISBN and publication year tracking

- Multiple copy management with real-time availability status

## Loan Management

- Automated due date calculation

- Return processing and overdue tracking

- Fine calculation for overdue items

## Student Management

- Comprehensive student record management

- Loan history tracking and fine management

# Challenges and Solutions

## Complex Data Relationships

Challenge: Managing complex relationships between books, authors, and copies.

Solution: I implemented through models and properties to manage many-to-many relationships and status tracking efficiently.

## Status Management

Challenge: Maintaining accurate status updates for book copies and loans.

Solution: Automated status updates in the `Loan` model to ensure book availability is always current.

## Data Validation

Challenge: Ensuring data integrity across related models.

Solution: Developed custom validation logic in both forms and models, ensuring rules such as mandatory author associations for books.

# Results and Impact

The system significantly improved library operations by:

- Streamlining manual processes for staff

- Reducing errors in book tracking and loans

- Providing a user-friendly interface with clear workflows for loan processing and fine management

# Conclusion

The University Library Management System implementation was successful in delivering a comprehensive solution for managing core library operations. The use of Django's powerful ORM and admin interface customization provided a solid foundation for future enhancements. Through the project, I gained significant experience in Django model design, database integration, and UI/UX development. This report reflects my individual contributions and showcases how the system effectively meets the needs of the library while maintaining flexibility for future expansions.