**BUILDING DEPLOYING &OPERATING CONTAINERIZED WEB-APPLICATION USING AWS**



SUBMITTED BY:

DARSHIL SHAH (N01619259)

MAHIMA SHASTRI (N01588261)

VANSHITA VERMA (N01607294)



Introduction

The Django-Based Book Management System is a sophisticated web application designed to enhance book management operations using the Django framework. This report provides an in-depth analysis of the project, covering its objectives, technology stack, system architecture, detailed configurations, and discussions on various aspects of the development process. The primary focus is on the transition from SQLite to MySQL for backend data management, Docker containerization for efficient deployment, and AWS hosting for global accessibility.

Table of Contents

[1. Objectives 4](#_Toc152364036)

[2. Technology Stack 4](#_Toc152364037)

[3. System Architecture 4](#_Toc152364038)

[3.1 Django Farmework 4](#_Toc152364039)

[3.2 MySQL Database Integration 4](#_Toc152364040)

[3.3 Dockerization 4](#_Toc152364041)

[3.4 AWS Deployment 4](#_Toc152364042)

[4. Default Configurations 4](#_Toc152364043)

[4.1 CRUD Operations for Book Management 4](#_Toc152364044)

[4.2 Dockerfile 4](#_Toc152364045)

[4.3 Cloud Hosting on AWS 4](#_Toc152364046)

[4.4 MySQL Integration and Volume Usage 4](#_Toc152364047)

[4.5 Dependencies 4](#_Toc152364048)

[5. Demo 4](#_Toc152364049)

[5.1 Create 4](#_Toc152364050)

[5.2 Read 4](#_Toc152364051)

[5.3 Update 4](#_Toc152364052)

[5.4 Delete 4](#_Toc152364053)

[6. Challenges 4](#_Toc152364054)

[6.1 Adapting CRUD Operations for MySQL 4](#_Toc152364055)

[6.2 Database Configurations 4](#_Toc152364056)

[6.2.1 Limitations of SQLite in Django 4](#_Toc152364057)

[6.2.2 Transition to MySQL 4](#_Toc152364058)

[7. Discussion 4](#_Toc152364059)

[7.1 Technology Choices 4](#_Toc152364060)

[7.2 MySQL Transition Impact 4](#_Toc152364061)

[8. Conclusion 4](#_Toc152364062)

[8.1 Key Findings 4](#_Toc152364063)

[9. Recommendations 4](#_Toc152364064)

[9.1 Future Enhancements 4](#_Toc152364065)

[9.2 Continuous Learnings 4](#_Toc152364066)

[10. References 5](#_Toc152364067)

[11. Appendices 5](#_Toc152364068)

[12. GitHub Re[pository 5](#_Toc152364069)

# Objectives

The primary goal of our project is to streamline and optimize book management operations through a robust web application. Key objectives include the implementation of CRUD (Create, Read, Update, Delete) operations, seamless user interface design, and efficient backend data management.

# Technology Stack

Our technology stack encompasses Django, MySQL, and Docker. Django provides a high-level framework for web development, MySQL serves as the backend database for data persistence, and Docker ensures efficient containerization for deployment.

# System Architecture

## Django Farmework

The Django framework adopts the Model-Template-Controller (MTC) architecture, providing a modular and scalable structure for our web application.

## MySQL Database Integration

The decision to transition from SQLite to MySQL involves careful considerations. We discuss the advantages of MySQL, the changes made in the settings.py file to accommodate MySQL, and its impact on the scalability of our book management system.

## Dockerization

Docker containerization plays a pivotal role in ensuring consistency across different deployment environments. We delve into the Dockerfile, specifying dependencies, configurations, and the rationale behind choosing Docker for our project.

## AWS Deployment

Our choice to host the application on AWS EC2 is thoroughly discussed, highlighting the benefits of global accessibility, scalability, and the overall efficiency gained by leveraging AWS infrastructure.

# Default Configurations

## CRUD Operations for Book Management

In this section, we provide a detailed breakdown of how CRUD operations are implemented in our Django project. We explore changes made in the project structure, migrations, models, views, and templates to align with the MySQL database.

## Dockerfile

Our Dockerfile is examined in detail, emphasizing the significance of each line in the configuration. We discuss dependencies, settings, and best practices for containerizing our Django application.

## Cloud Hosting on AWS

The AWS deployment process is outlined, covering the steps taken to host our application on EC2. We discuss server specifications, optimizations, and the strategic use of AWS services for seamless hosting.

## MySQL Integration and Volume Usage

Innovative MySQL integration involves Docker volumes for persistent data storage. The benefits of persistent volumes are discussed, emphasizing data retention strategies and the use of cluster IPs for smooth communication.

## Dependencies

The project relies on specific Python packages and libraries to function seamlessly. The dependencies include:

* asgiref==3.7.2
* crispy-bootstrap4==2023.1
* Django==4.2.7
* django-crispy-forms==2.1
* mysql-connector-python==8.2.0
* protobuf==4.21.12
* PyMySQL==1.1.0
* sqlparse==0.4.4
* tzdata==2023.3
* mysqlclient

These dependencies are essential for maintaining compatibility, implementing specific functionalities, and ensuring the overall stability of the Django-Based Book Management System.

# Demo

## Create

A screenshot of a computer

Description automatically generated

## Read

A screenshot of a book

Description automatically generated

## Update

A screenshot of a computer

Description automatically generated

## Delete

A screenshot of a computer

Description automatically generated

# Challenges

## Adapting CRUD Operations for MySQL

One of the significant challenges faced during the project was adapting the existing CRUD operations to seamlessly work with MySQL. Since MySQL has a different syntax and approach compared to SQLite, meticulous adjustments were required in the project structure, migrations, models, and views. Solutions involved thorough testing, debugging, and refactoring to ensure a smooth transition.

## Database Configurations

Configuring MySQL to work optimally with our Django application posed another challenge. Changes in the settings.py file were made to accommodate MySQL-specific configurations, including setting up the database engine, defining connection parameters, and handling potential compatibility issues. Solutions involved extensive documentation referencing MySQL and Django documentation, ensuring a robust database configuration.

### Limitations of SQLite in Django

* Initial Choice:  
  SQLite chosen for simplicity and easy integration with Django for the Book Management System.
* Concurrency Limitations:  
  Efficient for low to moderate traffic but struggles with high concurrency, impacting scalability and performance.
* Limited Query Support:  
  Demonstrated limitations with intricate and resource-intensive queries, affecting advanced search functionalities and analytics.
* Scaling Challenges:  
  Serverless nature and lack of horizontal scaling capabilities posed challenges as user base and data volume expanded, prompting the need for a more scalable solution.

### Transition to MySQL

* Transition to MySQL:  
  Decision made to address limitations by transitioning from SQLite to MySQL.
* MySQL Advantages:  
  MySQL chosen for superior support in high concurrency, advanced querying, and scalability through clustering and replication.
* Careful Transition Execution:  
  Database schemas adjusted to MySQL conventions, and data migration scripts implemented for a smooth transfer of existing data.
* Configuration Update:  
  settings.py file updated to configure Django to use MySQL as the backend.
* Immediate Benefits:  
  Improved support for high concurrency and enhanced performance of complex queries resulted in more efficient simultaneous user interactions and increased overall system responsiveness.

# Discussion

## Technology Choices

In the discussion section, we delve into the rationale behind our technology choices. We evaluate the advantages and disadvantages of using Django, MySQL, and Docker, addressing considerations such as development speed, scalability, and ease of deployment. This section provides insights into the decision-making process and the impact of these choices on the overall project.

## MySQL Transition Impact

The transition from SQLite to MySQL is discussed in detail, emphasizing the impact on the project's scalability and data management capabilities. We evaluate the benefits of MySQL, such as improved performance with larger datasets and enhanced support for complex queries. Additionally, potential challenges and lessons learned during the transition are candidly addressed.

# Conclusion

## Key Findings

The conclusion section provides a comprehensive summary of the key findings throughout the project. It emphasizes the successful transition from SQLite to MySQL, the efficiency gained through Docker containerization, and the benefits of AWS hosting. Key metrics and performance improvements are highlighted, showcasing the project's success in achieving its objectives.

# Recommendations

## Future Enhancements

Future recommendations explore potential enhancements, additional features, security measures, and optimization strategies. Consideration is given to alternative database solutions based on specific project requirements. As the project evolves, continuous monitoring and assessment of the technology stack's relevance are recommended.

* Implementing Docker orchestration through tools such as Kubernetes.
* Implementing CI/CD pipeline
* Hosting on EKS or ECS for better scalability

## Continuous Learnings

The project presented numerous learning opportunities for the team. Recommendations include ongoing education and exploration of emerging technologies, frameworks, and best practices in web development and cloud computing.

# References

All external resources, documentation, and references are meticulously listed, ensuring transparency and proper accreditation. This includes references to Django, MySQL, Docker, AWS documentation, and other relevant sources that contributed to the project's success.

* <https://youtu.be/Mezody4yiXw?si=QbeFGsZMNmwziBa3>
* <https://youtu.be/N6jzspc2kds?si=EGFnQhy709mWa8w8>
* <https://docs.djangoproject.com/en/4.2/>
* <https://dev.to/foadlind/dockerizing-a-django-mysql-project-g4m>

# Appendices

Project files, directory structures, and key code snippets are provided in the appendices for reference and clarity. This includes the Dockerfile, relevant sections of the project file. These appendices serve as a valuable resource for developers and stakeholders seeking a deeper understanding of the project's technical aspects.

**DockerFile :**

A screenshot of a computer

Description automatically generated**Docker-Compose File:**

A screenshot of a computer program

Description automatically generated

# GitHub Repository

- [GitHub Repository](<https://github.com/vvanshita01/project2>)