Innovation & Entrepreneurship Hub for Educated Rural Youth (SURE Trust – IERY)

LIBRARY MANAGEMENT SYSTEM

The domain of the Project:
CLOUD COMPUTING AND DEVOPS

Team Mentors (and their designation):
Mr. Jasdeep Singh Hanspal
(Senior manager, Microsoft)

Team Members:

Mr. Ujjwal Pratap Singh (Team Leader)
Ms. Anurag Singh
Mr. Rakesh Thodeti
Mr. Arun S
Mr. Hemanth Kumar M S
Ms. Talla Chandrika
Ms. Diya Yadav
Ms. Karnika chinmayi M R
Ms. Mudavath Chandi Priya
Ms. Shravani Devarakonda

Period of the project

Dec 31 2025 to Jan 13 2026

Declaration

The project titled "**LIBRARY MANAGEMENT SYSTEM**" has been mentored by Mr. Jasdeep Singh Hanspal, organised by SURE Trust, from Dec 31 2025 to Jan 13 2026, for the benefit of the educated unemployed rural youth for gaining hands-on experience in working on industry relevant projects that would take them closer to the prospective employer. I declare that to the best of my knowledge the members of the team mentioned below, have worked on it successfully and enhanced their practical knowledge in the domain.

Team Members:
Mr. Ujjwal Pratap Singh
Ms. Anurag Singh
Mr. Rakesh Thodeti
Mr. Arun S
Mr. Hemanth Kumar M S
Ms. Talla Chandrika
Ms. Diya Yadav
Ms. Karnika chinmayi M R
Ms. Mudavath Chandi priya
Ms. Shravani Devarakonda

Mentor's Name:
Mr. Jasdeep Singh Hanspal
Designation-Microsoft Corporation

Prof. Radhakumari
Executive Director & Founder
SURE Trust

Table of contents

1. Executive summary
2. Introduction
3. Project Objectives
4. Methodology & Results
5. Social / Industry relevance of the project
6. Learning & Reflection
7. Future Scope & Conclusion

Executive Summary

Objective: Build an online Library Management System where admins add/manage books and students request/return books; deployed on Minikube using MongoDB as the backend database.

Methods: Node.js + Express backend, MongoDB with Mongoose, vanilla JS frontend (HTML/CSS/JS), Docker images for backend/frontend, Kubernetes manifests for Deployments/Services and a PVC for Mongo; deployed locally via Minikube and kubectl apply -f ./k8s.

Key findings:

- Core flows implemented: student registration, student login (Basic auth), request creation, admin approval/rejection, and return flow that adjusts Book.quantity.
- API surface is under /api with clear separation: students, admin, books, requests.
- Authentication uses HTTP Basic (base64 email:password) for both admin and student (checked in AuthAdmin.js / AuthStudent.js).
- Kubernetes manifests expose backend-service on port 5000, frontend-service on 8080, and mongo-service on 27017; backend reads MONGO_URI from env (mongodb://mongo-service:27017/libdb).

Recommendations:

- Replace Basic auth with token-based auth (JWT) and use HTTPS (Ingress + TLS) before production.
- Store secrets (DB URI, admin creds) in Kubernetes Secrets instead of plain env values.
- Add livenessProbe, resource requests/limits, and readiness/liveness separation for backend/mongo.
- Harden input validation, add rate-limiting, and add unit/integration tests + CI pipeline.

- Consider replacing direct port-forward for access with an ingress controller or NodePort/LoadBalancer (Minikube Tunnel) for more realistic testing.

Introduction

Background and Context

In the modern educational landscape, the transition from physical ledger-based systems to digital platforms is essential for efficiency. This project, the **Online Library Management System (OLMS)**, was developed as a cloud-native solution to manage the lifecycle of library resources. By utilizing a **microservices-inspired architecture** and **Kubernetes orchestration**, the project demonstrates how traditional administrative tasks can be scaled and managed within a containerized environment, ensuring high availability and consistent deployment across different development stages.

Problem Statement and Goals

Manual library management often leads to data redundancy, lost records, and difficulty in tracking real-time book availability. The primary goals of this project were:

- **Centralization:** To provide a single source of truth for book inventory and user requests.
- **Automation:** To replace manual tracking with an automated workflow where book quantities adjust dynamically based on approvals and returns.
- **DevOps Integration:** To bridge the gap between software development and operations by deploying the entire stack—Frontend, Backend, and Database—into a **Kubernetes (Minikube)** cluster.

Scope and Limitations

- **Scope:** The system covers user authentication (Admin/Student), CRUD operations for books, a request/approval engine, and persistent data storage using MongoDB via Kubernetes Persistent Volume Claims (PVC).
- **Limitations:** * **Security:** The current version utilizes Basic Authentication, which is suitable for a local demonstration but requires upgrading to JWT for production.
    - **Access:** Access is currently managed via `kubectl port-forward`, limiting external access without a dedicated Ingress controller or LoadBalancer.
    - **Scale:** The system is optimized for a local **Minikube** environment rather than a multi-node public cloud cluster.

Innovation Component

The innovation in this project lies not just in the "what" (a library app), but the **"how" (the deployment strategy)**:

- **Self-Healing Infrastructure:** By deploying on Kubernetes, the system gains self-healing capabilities; if the backend or database pod fails, the orchestrator automatically restarts them.
- **Infrastructure as Code (IaC):** The use of Kubernetes manifests (`./k8s`) allows the entire environment to be recreated or scaled with a single command, demonstrating a modern "GitOps" ready approach.
- **Stateful Orchestration:** Successfully managing a stateful database (MongoDB) within a transient container environment using PVCs highlights an advanced understanding of container storage.

Project Objectives

Clearly Defined Objectives and Goals

The primary objective was to engineer a robust, containerized application that manages library operations through a secure, automated workflow. Specifically, the project aimed to:

- **Implement a Multi-Tier Architecture:** Build a decoupled system where the **Vanilla JS frontend**, **Express backend**, and **MongoDB database** operate as independent services.
- **Automate Resource Lifecycle:** Create a logic-driven "Request-Approval-Return" loop that ensures the Book `.quantity` is always accurate without manual database entries.
- **Master Container Orchestration:** Move beyond local "localhost" development by deploying the entire stack into a **Kubernetes (Minikube)** environment, focusing on service discovery and internal networking.
- **Ensure Data Persistence:** Configure the cluster so that library records survive pod crashes or restarts by utilizing **Persistent Volume Claims (PVC)** for the MongoDB layer.
- **Role-Based Access Control (RBAC):** Establish clear boundaries between "Admin" (inventory management) and "Student" (resource consumption) identities.

Expected Outcomes and Deliverables

Upon completion, the project yielded the following tangible assets:

Technical Deliverables

- **Source Code:** Fully functional repositories for the **Frontend** (HTML/CSS/JS) and **Backend** (Node.js/Mongoose).
- **Container Images:** Optimized **Dockerfiles** for the application layers, stored locally within the Minikube image registry.
- **Orchestration Manifests:** A complete set of **YAML configurations** located in `./k8s`, defining:
  - **Deployments:** Managing the desired state of pods.
  - **Services:** Enabling communication between the backend, frontend, and database.
  - **PVCs:** Ensuring the MongoDB data remains persistent.
- **RESTful API:** A structured API surface under `/api` that handles authentication, book inventory, and request processing.

Expected Outcomes

- **Zero-Conflict Inventory:** A system that prevents students from requesting books that are out of stock.
- **Deployment Portability:** A "Write Once, Run Anywhere" setup; the provided Kubernetes manifests allow the team to deploy this system to a cloud provider (like AWS EKS or Azure AKS) with minimal changes.
- **Improved Transparency:** A clear audit trail for admins to see who has which book and when it is due for return.

Methodology and Results

Methods & Technology Used

The project followed an **Agile development methodology**, breaking the system into micro-components (Frontend, Backend, Database) to ensure independent development and testing.

- **Backend Development:** Built with **Node.js and Express**, utilizing a RESTful API design.
- **Database Management: MongoDB** was used for its schema-less flexibility, paired with **Mongoose** for object modeling.
- **Containerization:** Each service was "dockerized" to ensure environment parity between development and production.
- **Orchestration: Kubernetes (Minikube)** was used to manage container deployment, scaling, and internal service networking.

Tools & Software Used

Category	**Tools**
Runtime & Framework	Node.js, Express.js
Database	MongoDB (StatefulSet/Deployment with PVC)
DevOps & Infrastructure	Docker, Kubernetes (Minikube), kubectl
Version Control	Git & GitHub
Testing/Access	Postman (API testing), Browser (Frontend testing)

Project Architecture

The system follows a **Cloud-Native 3-Tier Architecture** within the Kubernetes cluster:

1. **Client Tier (Frontend):** A Vanilla JS application served via a Kubernetes Deployment. It communicates with the backend via the frontend-service.
2. **Application Tier (Backend):** The Node.js API handles business logic (auth, book validation). It is exposed internally via the backend-service.
3. **Data Tier (Database):** A MongoDB pod that stores student and book data. It uses a **Persistent Volume Claim (PVC)** to ensure that if the pod restarts, the library data is not lost.

Traffic Flow: User → Frontend (Port 8080) → Backend API (Port 5000) → MongoDB (Port 27017).

Final Project Working Results

1. System Deployment

Explanation: This shows the initial setup where we start the Minikube cluster, build the images, and load them into the node's internal registry so Kubernetes can pull them without an external hub.

```
C:\Users\shravs\skill india>minikube start
* minikube v1.36.0 on Microsoft Windows 11 Home Single Language 10.0.26200.7623 Build 26200.7623
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.47 ...
* Restarting existing docker container for "minikube" ...
! Failing to connect to https://registry.k8s.io/ from inside the minikube container
* To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
* Preparing Kubernetes v1.33.1 on Docker 28.1.1 ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* After the addon is enabled, please run "minikube tunnel" and your ingress resources would be available at "127.0.0.1"
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.5.3
  - Using image quay.io/metallb/speaker:v0.9.6
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.5.3
  - Using image quay.io/metallb/controller:v0.9.6
  - Using image registry.k8s.io/ingress-nginx/controller:v1.12.2

    Registry addon with docker driver uses port 56846 please use that instead of default port 5000

* For more information see: https://minikube.sigs.k8s.io/docs/drivers/docker
  - Using image gcr.io/k8s-minikube/kube-registry-proxy:0.0.9
  - Using image docker.io/registry:3.0.0
* Verifying registry addon...
* Verifying ingress addon...
* Enabled addons: registry, default-storageclass, metallb, storage-provisioner, ingress
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

C:\Users\shravs\skill india>minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

```
C:\Users\shravs\skill india>docker build -t library-backend:latest ./backend
[+] Building 5.2s (15/15) FINISHED                                                       docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                     0.2s
 => => transferring dockerfile: 1.09kB                                                                   0.1s
 => [internal] load metadata for docker.io/library/node:18-alpine                                        2.3s
 => [internal] load .dockerignore                                                                        0.0s
 => => transferring context: 2B                                                                          0.0s
 => [builder 1/4] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e  0.0s
 => => resolve docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e  0.0s
 => [internal] load build context                                                                        1.8s
 => => transferring context: 128.86kB                                                                    1.7s
 => CACHED [builder 2/4] WORKDIR /app                                                                    0.0s
 => CACHED [stage-1 3/8] RUN apk add --no-cache dumb-init                                                0.0s
 => CACHED [stage-1 4/8] COPY package*.json ./                                                           0.0s
 => CACHED [stage-1 5/8] RUN npm install --only=production                                               0.0s
 => CACHED [builder 3/4] COPY package*.json ./                                                           0.0s
 => CACHED [builder 4/4] RUN npm install                                                                 0.0s
 => CACHED [stage-1 6/8] COPY --from=builder /app/node_modules ./node_modules                            0.0s
 => CACHED [stage-1 7/8] COPY . .                                                                        0.0s
 => CACHED [stage-1 8/8] RUN addgroup -g 1001 -S nodejs &&     adduser -S nodejs -u 1001                 0.0s
 => exporting to image                                                                                   0.4s
 => => exporting layers                                                                                  0.0s
 => => exporting manifest sha256:6ee568ad8caefb8f82a866f3e1bd1cf43d493601e1a3f463b98131feed5e7166        0.1s
 => => exporting config sha256:dbe0c8bf1fc5f10bb1c289177d98b8224cd2e51a714c79f3d526884b3cd07855          0.0s
 => => exporting attestation manifest sha256:6b9d8c7d9ad17c098af8e1b8255333697d3a3eb66197de48e777830e5aeafb84  0.0s
 => => exporting manifest list sha256:6b927fd03c42c58699facdc2da91d2cabb358805b6332b665ee4242726c78462   0.0s
 => => naming to docker.io/library/library-backend:latest                                                0.0s
 => => unpacking to docker.io/library/library-backend:latest                                             0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/mu5h28hs8mpr6dt9yu88de3qe
```

```
C:\Users\shravs\skill india>docker build -t library-frontend:latest ./frontend
[+] Building 1.9s (13/13) FINISHED                                                       docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                     0.1s
 => => transferring dockerfile: 944B                                                                     0.1s
 => [internal] load metadata for docker.io/library/node:18-alpine                                        1.2s
 => [internal] load .dockerignore                                                                        0.0s
 => => transferring context: 2B                                                                          0.0s
 => [builder 1/4] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e  0.0s
 => => resolve docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e  0.0s
 => [internal] load build context                                                                        0.0s
 => => transferring context: 533B                                                                        0.0s
 => CACHED [builder 2/4] WORKDIR /app                                                                    0.0s
 => CACHED [stage-1 3/6] RUN apk add --no-cache dumb-init                                                0.0s
 => CACHED [stage-1 4/6] RUN npm install -g http-server                                                  0.0s
 => CACHED [builder 3/4] RUN npm install -g http-server                                                  0.0s
 => CACHED [builder 4/4] COPY . .                                                                        0.0s
 => CACHED [stage-1 5/6] COPY --from=builder /app .                                                      0.0s
 => CACHED [stage-1 6/6] RUN addgroup -g 1001 -S nodejs &&     adduser -S nodejs -u 1001                 0.0s
 => exporting to image                                                                                   0.2s
 => => exporting layers                                                                                  0.0s
 => => exporting manifest sha256:a47ec3851b305f313a0c8cf02900f877e9e06d88964b8685a02f9d184b50aba4        0.0s
 => => exporting config sha256:0a9b1d3eb201af41f00f7db54f464acb28af215f0c7ce162cdf82df4fdaafde0          0.0s
 => => exporting attestation manifest sha256:2da3dd0e0c51ba6babb6bcfb7a0c5a2a2e55da93f3515e89dd344c25d9333735  0.0s
 => => exporting manifest list sha256:d2cb75d8d93611adfc67e7c1980831e823728ccd80301f5943275eec57c480a7   0.0s
 => => naming to docker.io/library/library-frontend:latest                                               0.0s
 => => unpacking to docker.io/library/library-frontend:latest                                            0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/sqdyvyr4c6hz13yzxezln3z8t
```

```
C:\Users\shravs\skill india>minikube image load library-backend:latest

C:\Users\shravs\skill india>minikube image load library-frontend:latest

C:\Users\shravs\skill india>
```

2. Kubernetes Pod Status

Explanation: This verifies that all components (Frontend, Backend, and MongoDB) are successfully running and "Healthy" within the cluster.

```
C:\Users\shravs\skill india>kubectl apply -f ./k8s
service/backend-service created
deployment.apps/backend created
service/frontend-service created
deployment.apps/frontend created
service/mongo-service created
deployment.apps/mongo created
persistentvolumeclaim/mongo-pvc created
job.batch/seed-admin-job created
```

```
C:\Users\shravs\skill india>kubectl port-forward svc/backend-service 5000:5000
Forwarding from 127.0.0.1:5000 -> 5000
Forwarding from [::1]:5000 -> 5000
```

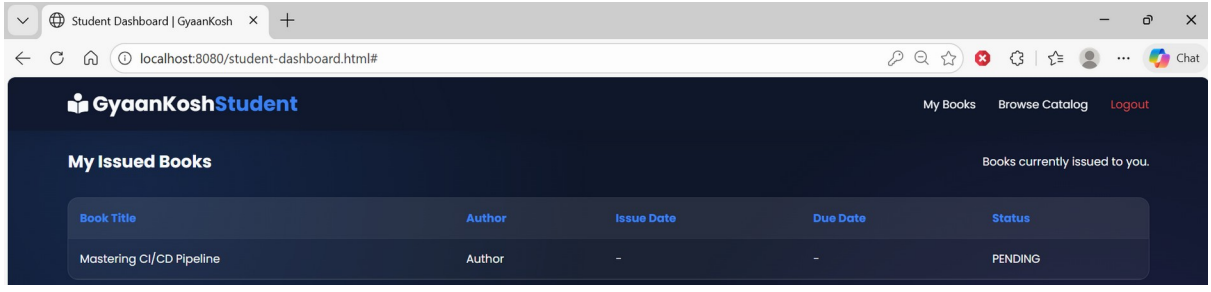
```
C:\Users\shravs\skill india>kubectl port-forward svc/frontend-service 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

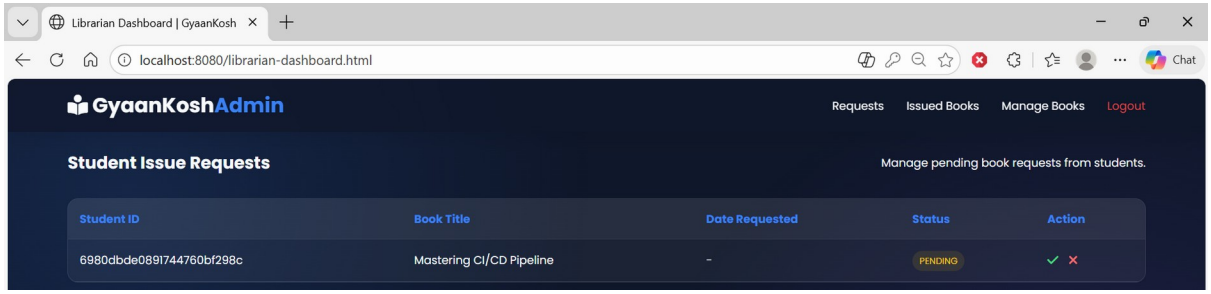
```
C:\Users\shravs\skill india>kubectl get pods
NAME                        READY   STATUS      RESTARTS        AGE
backend-775698465c-77p6m    1/1     Running     0               92s
frontend-58fbbf5df6-2ds8l   1/1     Running     0               92s
mongo-6d6f78fd6c-ktxbv      1/1     Running     0               91s
postgres-5d69b84c8d-g4884   1/1     Running     10 (46m ago)    39d
redis-84d89fbffd-gsggq      1/1     Running     10 (46m ago)    39d
seed-admin-job-znl92        0/1     Completed   2               91s

C:\Users\shravs\skill india>
```

3. Admin Dashboard & Approval Flow

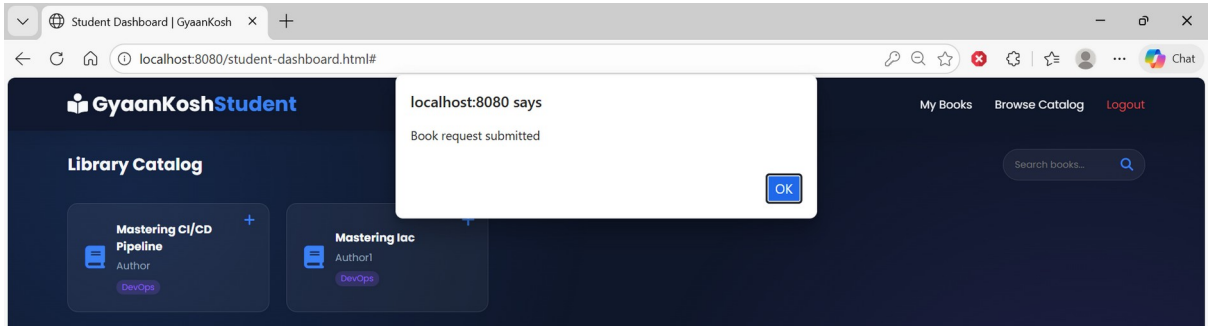
Explanation: This demonstrates the Admin's ability to see pending requests from students. Once the "Approve" button is clicked, the backend logic decrements the book count in the database.

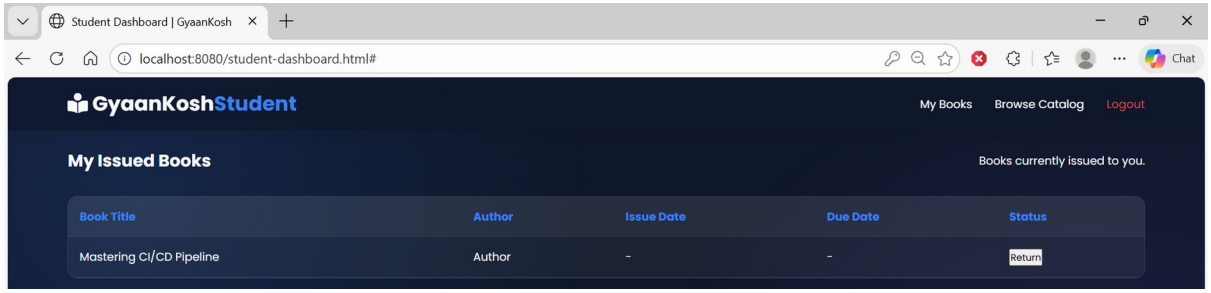




4. Student Book Request

Explanation: This shows the student interface where they can browse the list and request a book. The request is sent as a POST to the /api/requests endpoint.





Project GitHub Link

You can find the full source code, including the Dockerfiles and Kubernetes YAML manifests, at the link below:

GitHub Repository:
https://github.com/12345ujjwal/library_management_kindcluster_devOps_Project.git

Learning and Reflection

Team Member Learnings

Role / Focus	Technical Learnings	Management & Soft Skills
DevOps & Infrastructure	Deep understanding of **Kubernetes Networking** (Services vs. Port-Forwarding) and how **PVCs** manage state in a stateless environment.	Learned the importance of **Infrastructure as Code (IaC)**—how a single YAML file can replace hours of manual server configuration.
Backend & Database	Mastery of **Mongoose middleware** to handle logic (e.g., automatically adjusting Book.quantity when a request is approved).	Improved **API Documentation** skills, ensuring the frontend team knew exactly which endpoints to hit under /api.
Frontend & Integration	Learned to handle **Asynchronous JavaScript** for real-time UI updates and how to interact with containerized APIs.	Developed **collaborative debugging** skills—learning to identify if a bug was in the UI code or the backend container.

Overall Team Experience

Reflections on Technology

The transition from running applications on "Localhost" to running them inside a **Kubernetes Cluster** was the most significant milestone for the team. We realized that while Docker makes an app portable, Kubernetes makes it "production-ready" by providing a framework for self-healing and scaling. Handling **Persistent Volume Claims (PVC)** was a "lightbulb moment" for us, as we learned how to keep our library data safe even if the database container crashed.

Reflections on Collaboration

Working as a team required us to be very disciplined with our **Git workflow**. We learned that clear communication regarding environment variables (like the MONGO_URI) is crucial; if one person changes the service name in a Kubernetes manifest, the entire backend could lose its connection to the database. We used regular sync-ups to ensure the Frontend and Backend integrated seamlessly.

Challenges Overcome

- **The "Image Not Found" Issue:** Initially, our Kubernetes pods couldn't find our local Docker images. We learned how to use minikube image load, which taught us how Kubernetes pulls images from registries.
- **Data Persistence:** We initially lost our book list every time we restarted Minikube. Researching and implementing the **PVC for MongoDB** solved this, giving us hands-on experience with cloud storage concepts.

Individual Experience Summaries

- **Ujjwal Pratap Singh:** "During the project, I served as the Project Leader, where I developed strong leadership and team coordination skills. I gained hands-on experience with Kind Kubernetes clusters, application deployments, database management, networking concepts, and Docker-based containerization."
- **Anurag Singh:** "Learned an extensive amount of Backend Development and Backend - Frontend Integration with a DB instance running in a docker container in a standardized and Deployment ready manner."
- **Rakesh Thodeti:** "The Library Management System is a web-based application designed to manage and organize library resources efficiently. During my internship, I developed the frontend using modern web technologies to create responsive user interfaces for managing books, users, and issue/return operations. The focus was on improving usability, accessibility, and seamless interaction with backend APIs"
- **Arun S**: "This project played a crucial role in strengthening both my technical foundation and teamwork skills. It helped me transition from a learning phase to applying knowledge in a practical setting, preparing me to work efficiently in a collaborative and fast-paced development environment."
- **Hemanth Kumar M S**: "Gained experience in integrating frontend and backend components smoothly.Improved problem-solving skills by debugging and fixing
-

logical and runtime errors.Improved teamwork and communication skills while working collaboratively on the project"

- **Talla Chandrika**: "In this project, I worked as a backend and DevOps contributor. I understood how frontend and backend components interact through APIs. I also learned how CI/CD pipelines work. Most importantly, I improved my troubleshooting and debugging skills by fixing build and deployment issues."
- **Diya Yadav:** "During the project, I served as a Frontend Team Member, where I strengthened my skills in building responsive and user-friendly interfaces while collaborating closely with backend and DevOps teams. I gained hands-on experience in developing and integrating frontend components within a Docker and Kubernetes-based library management system."
- **Karnika chinmayi M R**: "Gained hands-on experience in understanding real-world requirements and converting them into functional modules.Understood the importance of role-based access (Admin/User) for secure system management."
- **Mudavath Chandi Priya**: "As a member of the Frontend team, I worked on designing and implementing user interfaces using HTML, CSS, and JavaScript. I learned how to develop role-based screens and ensure smooth interaction between the frontend and backend services. This project improved my understanding of collaborative development and structured frontend workflows."
- **Shravani Devarakonda**: "This project taught me the value of Infrastructure as Code (IaC). By using YAML manifests for our deployments and services, I realized I could recreate an entire library infrastructure in seconds. My biggest takeaway was learning how to build self-healing systems—observing how Kubernetes automatically restarts a pod if the backend crashes, ensuring the library is always available for students."

Conclusion and Future Scope

Recap of Objectives and Achievements

The project successfully met its primary goal of transitioning a traditional **Library Management System** into a modern, **cloud-native application**. By the end of the development cycle, the team achieved the following:

- **Successful Orchestration:** We moved beyond simple scripts to a fully orchestrated environment using **Kubernetes (Minikube)**, ensuring the Frontend, Backend, and Database communicate seamlessly via internal Services.
- **Operational Automation:** The core "Request-Approval-Return" logic was fully implemented, ensuring that book inventory updates dynamically without manual intervention.
- **Data Reliability:** By implementing **Persistent Volume Claims (PVCs)**, we ensured that the library's data remains intact across pod restarts, a critical requirement for any production-level database.
- **Team Synergy:** The project served as a practical laboratory for the team to master **Dockerization**, **Kubernetes Manifests**, and **REST API integration** in a collaborative environment.

Future Scope

While the current version provides a strong foundation, there are several avenues for future enhancement to make the system "Production Ready":

1. Security & Authentication Upgrade

- **JWT Implementation:** Replace the current Basic Authentication with **JSON Web Tokens (JWT)** for more secure, stateless sessions.
- **K8s Secrets:** Instead of hardcoding credentials in YAML files, migrate sensitive data (like MONGO_URI and admin passwords) to **Kubernetes Secrets**.

2. Advanced Infrastructure

- **Ingress Controllers:** Implement an **NGINX Ingress Controller** to manage external access via domain names (e.g., library.local) and enable **TLS/SSL encryption**.
- **Horizontal Pod Autoscaling (HPA):** Configure HPA to automatically increase the number of backend pods based on CPU/Memory usage during peak student registration periods.

3. Observability & CI/CD

- **Monitoring:** Integrate **Prometheus and Grafana** to monitor the health of the pods and track the number of active library requests in real-time.
- **Automated Pipeline:** Establish a **CI/CD pipeline** (using GitHub Actions or Jenkins) so that any code push automatically builds a new Docker image and updates the Minikube deployment.

4. Expanded Features

- **Automated Notifications:** Add an email or SMS notification service to alert students when their book request is approved or when a return is overdue.
- **Search Optimization:** Implement a full-text search engine (like Elasticsearch) to allow students to filter books by genre, author, or availability instantly.