Readers Garden Library

Library Management System

With DevOps Integration

Submitted By: Ayan Yadav

Reg.No: 12222524

Roll.No: 22 (KO129)

Submitted to: Arshiya Mam

Course Code: INT332

Table of Contents

2. Introduction

Readers Garden Library is a web application designed to manage library operations efficiently.
It features two primary interfaces:

- **Admin Panel:** Allows library administrators to manage books, view and manage user profiles, handle borrow/return requests, and apply late return charges.

- **Client Interface:** Lets users browse the available book collection, request books, and update their dashboard or profile.

The project focuses on delivering a seamless and scalable library management system using modern web development and DevOps tools.

3. Objectives

- Build a robust system for managing books in a library.
- Enable CRUD operations on books and user profiles.
- Allow admins to confirm borrow and return requests.
- Implement late return fee tracking.
- Containerize the full application using Docker.
- Automate CI/CD workflows using GitHub Actions.
- Provide a user-friendly and efficient interface for library users.

The following technologies are utilized in the development of Readers Garden Library:

- **Frontend:** React.js, Tailwind CSS,
- **Backend:** Node.js, Express.js, MongoDB

- **Authentication:** JWT (JSON Web Tokens) for secure user login

4. Implementation

4.1 Architecture Overview

- **Admin Panel:** Allows management of books, users, borrow requests, return confirmations, and overdue charges.
- **Client Interface:** Enables users to browse books, request them, and manage their profiles.
- **Backend Services:** APIs built using Express.js handle all application logic.
- **Database:** MongoDB stores book data, user profiles, and transaction history.

4.2 Docker Containerization

Each part of the application is containerized:

- **Frontend (React.js):** Served as a separate container.
- **Backend (Express.js):** Runs in its own container, exposing API endpoints.
- **Database (MongoDB):** Containerized using official MongoDB image.
- **Docker Compose:** Used for orchestrating the services together.

Steps:

1. Write Dockerfiles for frontend and backend.
2. Define services in docker-compose.yml.
3. Build and run the stack using:

   " **DOCKER COMPOSE UP"**

4.3 GitHub Actions CI/CD Pipeline

CI/CD is handled with GitHub Actions to automate testing and deployment:

- Workflows are defined in .github/workflows.
- On push to main or pull requests, the pipeline runs:
  - Linting and testing stages.
  - Docker build and push to registry (optional).
  - Deployment scripts to the target server or service.
- Logs and deployment status can be viewed on GitHub's Actions tab.

Each microservice is containerized using Docker. Steps involved:

1. Dockerize Backend Services:

Readers-Garden-Library / backend / **Dockerfile**

AyanYadav24 Initial commit with project files — cf57070 · 2 weeks ago — History

Code | Blame | 13 lines (7 loc) · 118 Bytes | Code 55% faster with GitHub Copilot | Raw

```
FROM node:18-alpine

WORKDIR /app

COPY package.json .

RUN npm i

COPY . .

EXPOSE 5000

CMD [ "npm", "run", "dev" ]
```

2. Dockerize React Frontend:

Readers-Garden-Library / frontend / Dockerfile

AyanYadav24 Initial commit with project files cf57070 · 2 weeks ago History

Code Blame 13 lines (7 loc) · 117 Bytes Code 55% faster with GitHub Copilot Raw

```
FROM node:18-alpine

WORKDIR /app

COPY package.json .

RUN npm i

COPY . .

EXPOSE 5173

CMD [ "npm", "run", "dev" ]
```

3. MongoDB Docker Setup:

- Use official MongoDB image in docker-compose.yml

4. Docker Compose for Orchestration:

Readers-Garden-Library / docker-compose.yml

AyanYadav24 Update docker-compose.yml 7407b00 · 2 weeks ago History

Code Blame 26 lines (23 loc) · 380 Bytes Code 55% faster with GitHub Copilot Raw

```
version: "3"
services:
  backend:
    build: ./backend
    ports:
      - "5000:5000"
    volumes:
      - logs:/app/logs
      - ./backend:/app
      - /app/node_modules

  frontend:
    build: ./frontend
    ports:
      - "5173:5173"
    volumes:
      - ./frontend/src:/app/src
    stdin_open: true
    tty: true
    depends_on:
      - backend

volumes:
  data:
  logs:
```

5. Run All Services: docker-compose up --build

4.3 Github Actions CI/CD Pipeline

Steps to automate testing and deployment using Jenkins:

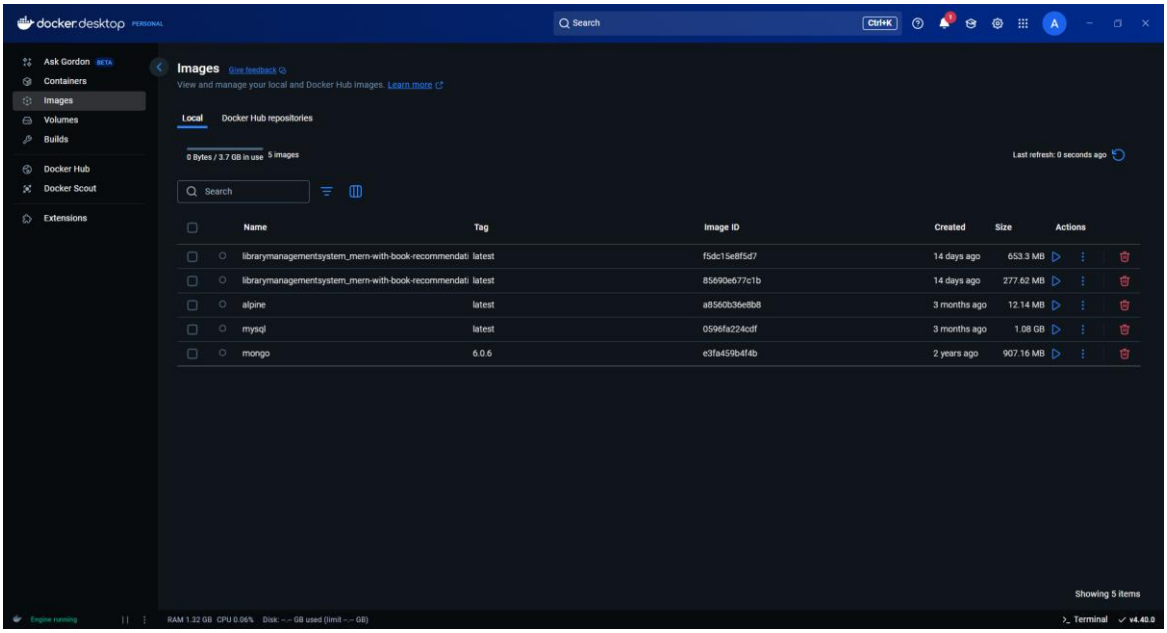
- Go to Actions tab in the repository to setup the CI/CD pipeline.
- Setup the required secrets and variables required to run the app like docker username and password under the secrets and variables section in settings.
- Sample File .github/workflows / ci-cd.yml:

Readers-Garden-Library / .github / workflows / ci-cd.yml

Code | Blame | 61 lines (49 loc) · 1.54 KB | Code 55% faster with GitHub Copilot | Raw

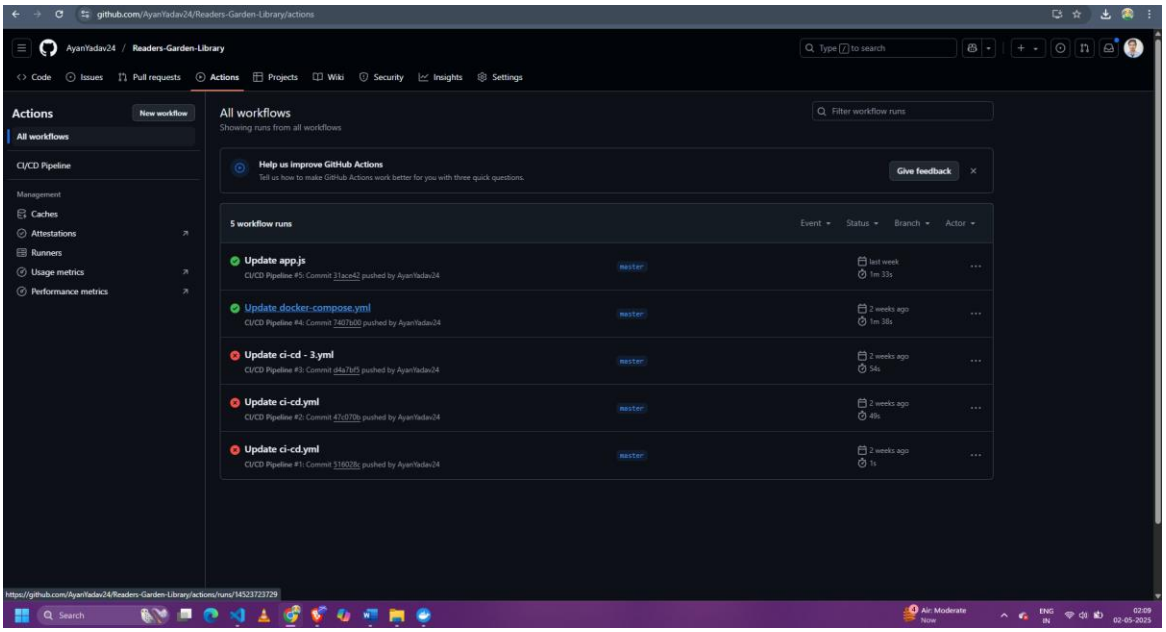
```yaml
  push:
    branches:
      - master
  pull_request:
    branches:
      - master

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    services:
      mongo:
        image: mongo:6.0.6
        ports:
          - 27017:27017

    env:
      CONNECTION_URL: mongodb://localhost:27017/library%mtsystem
      CONNECTION_PORT: 5000
      JWT_SECRET: ${{ secrets.JWT_SECRET }}
      JWT_REFRESH_SECRET: ${{ secrets.JWT_REFRESH_SECRET }}
      JWT_LIFE: 30m
      JWT_REFRESH_LIFE: 365d
      EMAIL_USERNAME: ${{ secrets.EMAIL_USERNAME }}
      EMAIL_PASSWORD: ${{ secrets.EMAIL_PASSWORD }}

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Log in to Docker Hub (Optional)
        if: env.DOCKER_USERNAME != '' && env.DOCKER_PASSWORD != ''
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}
        env:
          DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
          DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build Backend Image
        run: docker build -t my-backend ./backend

      - name: Build Frontend Image
        run: docker build -t my-frontend ./frontend

      - name: Start Full Stack with Docker Compose
        run: docker compose up -d --build

      - name: Docker Compose Logs
        run: docker compose logs

      - name: Shut Down Docker Compose
        run: docker compose down
```

Created Repositories for Integrateing CI/CD:



1. Integrate with GitHub/GitLab Webhooks for auto-trigger on push.

2. Monitor Logs and Deployment from Github Actions.

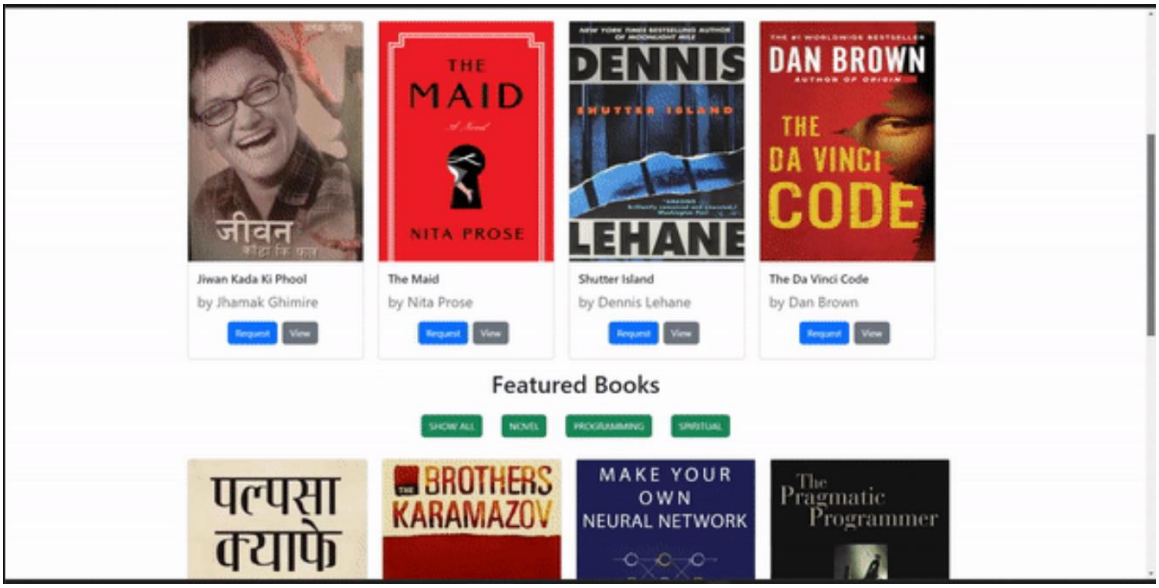


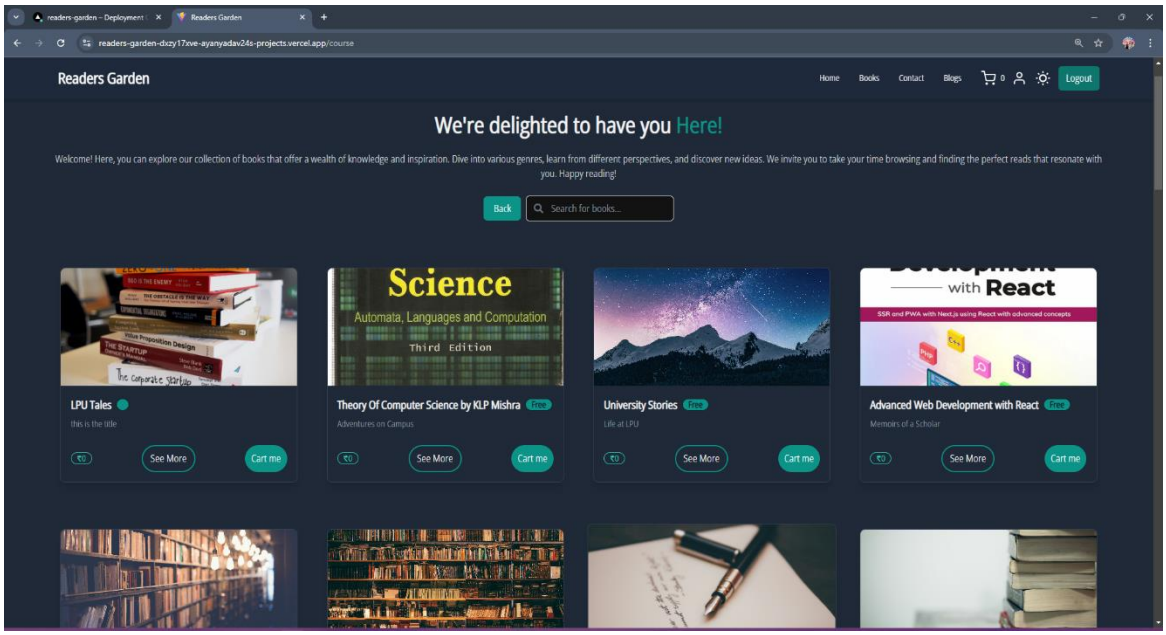
5. Outcomes

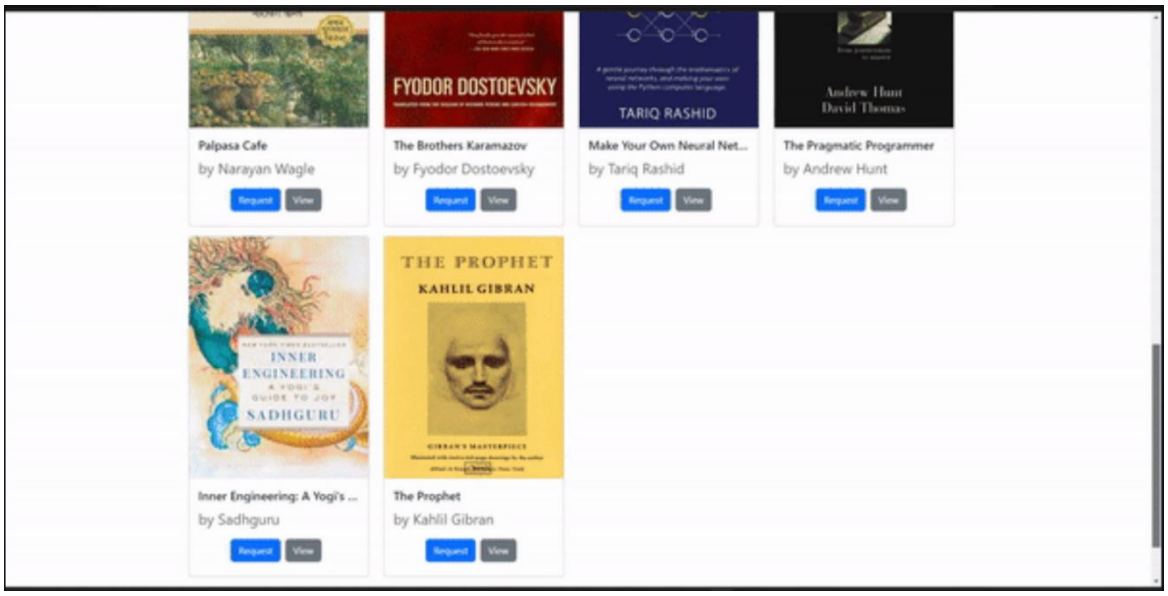
- A fully functional, scalable, and modular library management system.

- Separation of concerns via frontend/backend/containerization.
- Docker ensures consistent deployments across environments.
- GitHub Actions allows for fast and reliable integration and delivery.
- Enhances the experience for both admins and clients through automation.

6. Project Visuals:







7. Conclusion

Readers Garden Library showcases how modern web development and DevOps practices can be applied to a real-

world domain like library management.
By leveraging **React.js**, **Express.js**, **MongoDB**, **Docker**, and **GitHub Actions**, the project ensures:

- Clean modular design,
- Scalable deployment,
- Streamlined development lifecycle,
- And an intuitive user experience.

The project reflects strong integration of backend logic, frontend UI, automation, and containerized deployment workflows.

GitHub: https://github.com/AyanYadav24/Readers-Garden-Library.git