SCD Project 25

Docker Deployment Report

Submitted By:

Zain Ali Khan

22i-2624

BS Software Engineering

Batch 22

Institution:

FAST NUCES Islamabad

December 7, 2025

Contents

1 Part 3: Building Features into a Provided Project

1.1 Step 1: Clone the Repository

1.1.1 Commands Executed:

```
# Create working directory
mkdir -p ~/scd-project
cd ~/scd-project

# Clone the repository
git clone https://github.com/LaibaImran1500/SCDProject25.git

# Navigate to the project
cd SCDProject25

# View project structure
ls -la
```

SCREENSHOT 1: Repository Cloned

Figure 1: Repository Cloned

1.2 Step 2: Examine Project Structure

```
# View project structure
ls -la

# View all files recursively
find . -type f -name "*.js" -o -name "*.json" | head -20
```

Project Structure:

```
SCDProject25/
+-- main.js                 # Main application entry point
+-- data/
|   +-- vault.json          # In-memory database (JSON file)
+-- db/
|   +-- index.js            # Database operations (CRUD)
|   +-- file.js             # File read/write operations
|   +-- record.js           # Record validation and ID generation
+-- events/
    +-- index.js            # Event emitter
    +-- logger.js           # Event logging
```

Application Overview:

- This is a NodeVault application - a CLI-based CRUD application
- Uses an in-memory JSON file database (`data/vault.json`)
- Has event-driven logging for record operations
- Current menu options: Add, List, Update, Delete, Exit

1.3 Step 3: Run the Application Locally

```
# Run the application
node main.js
```

Current Menu:

```
===== NodeVault =====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Exit
======================
```

SCREENSHOT 2: Application Running Locally

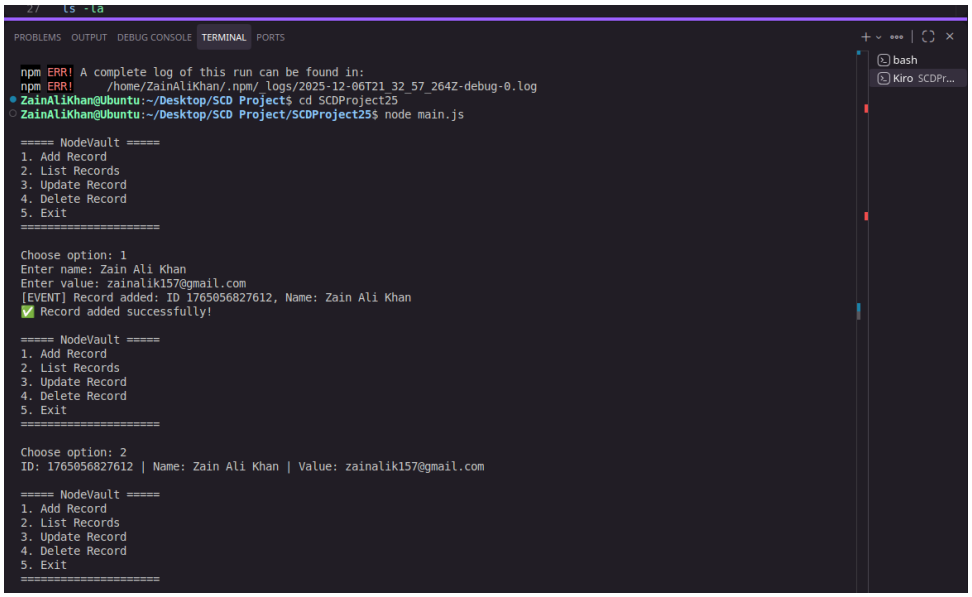

Figure 2: Application Running Locally

1.4 Step 4: Create a Feature Branch

Before making any modifications, we create a new branch from the main branch. All changes will be made in this feature branch.

1.4.1 Commands Executed:

```
# Navigate to project directory
cd ~/scd-project/SCDProject25

# Check current branch
git branch

# Create and switch to feature branch
git checkout -b feature/enhancements

# Verify branch switch
git branch
```

SCREENSHOT 3: Feature Branch Created

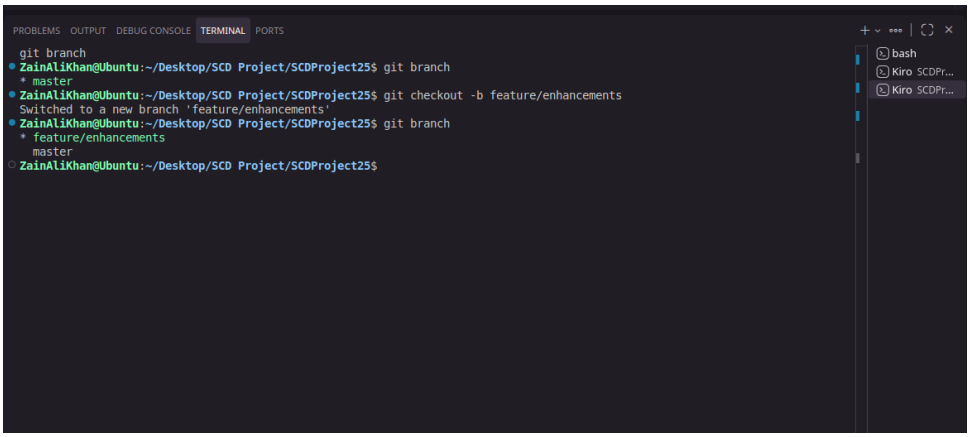

Figure 3: Feature Branch Created

1.5 Git Versioning Strategy

For every major change, we will create a version tag:

Feature	Version Tag	Command
Search Functionality	v1.0	`git tag -a v1.0 -m "Added search functionality"`
Sorting Capability	v1.1	`git tag -a v1.1 -m "Added sorting capability"`
Export to Text File	v1.2	`git tag -a v1.2 -m "Added export functionality"`
Automatic Backup	v1.3	`git tag -a v1.3 -m "Added automatic backup"`
Data Statistics	v1.4	`git tag -a v1.4 -m "Added vault statistics"`
MongoDB Setup	v1.5	`git tag -a v1.5 -m "MongoDB integration"`
Env File Setup	v2.0	`git tag -a v2.0 -m "Environment variables setup"`

Table 1: Git Versioning Strategy

2 Feature Implementations

2.1 Feature 1: Search Functionality

Description: Allows users to search for existing records by name or ID (case-insensitive).

Implementation:

```
// Function to search records (case-insensitive)
function searchRecords(keyword) {
  const records = db.listRecords();
  const searchTerm = keyword.toLowerCase();

  return records.filter(r =>
    r.name.toLowerCase().includes(searchTerm) ||
    r.id.toString().includes(searchTerm) ||
    (r.value && r.value.toLowerCase().includes(searchTerm))
  );
}
```

Menu Option Added: Option 5 - "Search Records"

SCREENSHOT 4: Search Functionality

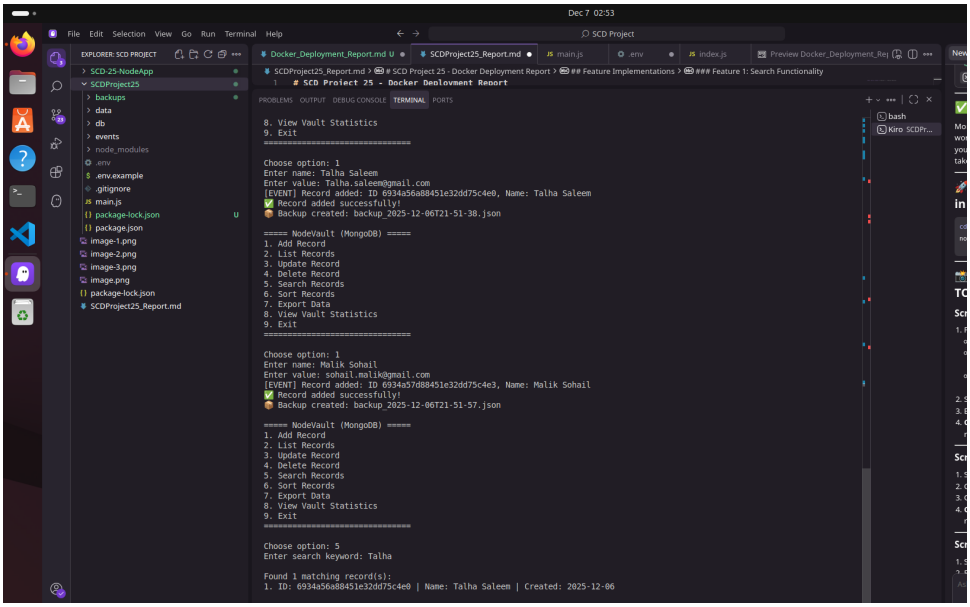

Figure 4: Search Functionality

2.2 Feature 2: Sorting Capability

Description: Allows users to sort records by Name or Creation Date in Ascending or Descending order.

Implementation:

```
// Function to sort records
function sortRecords(field, order) {
  const records = [...db.listRecords()]; // Clone to avoid
    modifying original
```

```
  records.sort((a, b) => {
    let valA, valB;

    if (field === 'name') {
      valA = a.name.toLowerCase();
      valB = b.name.toLowerCase();
    } else if (field === 'date' || field === 'id') {
      valA = a.id; // ID is timestamp-based
      valB = b.id;
    }

    if (order === 'asc') {
      return valA > valB ? 1 : valA < valB ? -1 : 0;
    } else {
      return valA < valB ? 1 : valA > valB ? -1 : 0;
    }
  });

  return records;
}
```

Menu Option Added: Option 6 - "Sort Records"
Expected Output:

```
Choose field to sort by: Name
Choose order: Ascending
Sorted Records:
1. ID: 104 | Name: Adeel
2. ID: 110 | Name: Bilal
3. ID: 108 | Name: Zain
```

SCREENSHOT 5: Sorting Capability

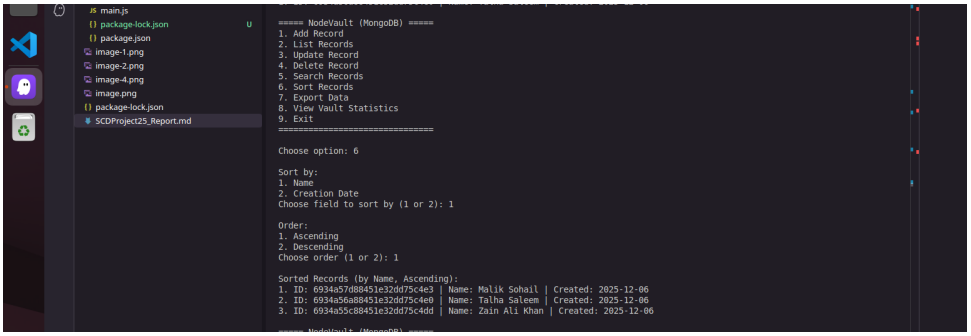

Figure 5: Sorting Capability

2.3 Feature 3: Export Vault Data to Text File

Description: Exports all records to a human-readable export.txt file.
Implementation:

```
// Function to export data to text file
function exportData() {
  const records = db.listRecords();
  const now = new Date();
  const dateStr = now.toISOString().replace('T', ' ').slice(0,
     19);

  let content = `========================================\n`;
  content += `         NODEVAULT DATA EXPORT\n`;
  content += `========================================\n\n`;
  content += `Export Date/Time: ${dateStr}\n`;
  content += `Total Records: ${records.length}\n`;
  content += `File Name: export.txt\n`;
  // ... rest of formatting

  const exportPath = path.join(__dirname, 'export.txt');
  fs.writeFileSync(exportPath, content);
  return exportPath;
}
```

Menu Option Added: Option 7 - "Export Data"

SCREENSHOT 6: Export Functionality

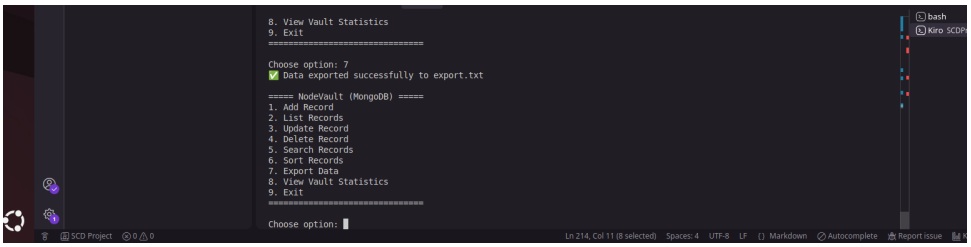

Figure 6: Export Functionality - Part 1

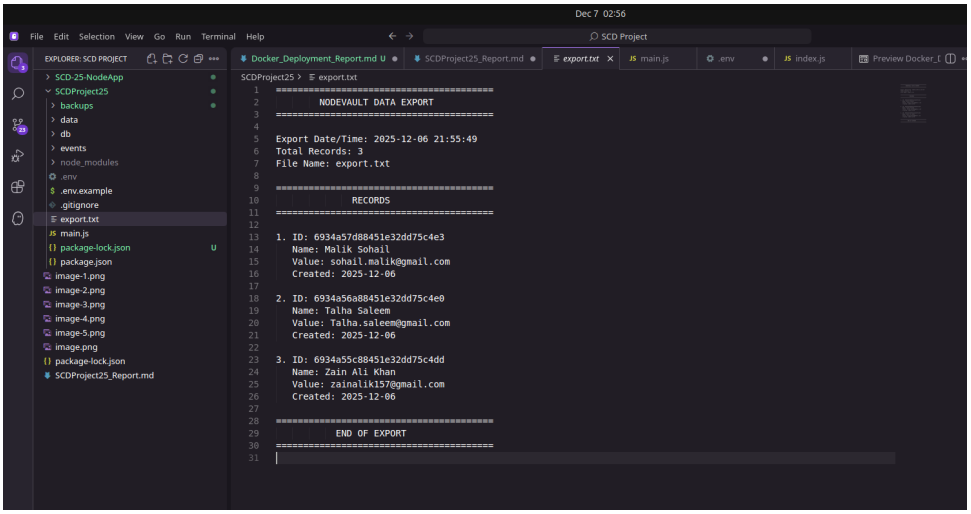

Figure 7: Export Functionality - Part 2

2.4 Feature 4: Automatic Backup System

Description: Automatically creates a backup whenever a record is added or deleted.

Implementation:

```
// Backup directory
const backupDir = path.join(__dirname, 'backups');
if (!fs.existsSync(backupDir)) fs.mkdirSync(backupDir);

// Function to create automatic backup
function createBackup() {
  const now = new Date();
  const timestamp = now.toISOString().replace(/[:.]/g,
     '-').slice(0, 19);
  const backupFileName = `backup_${timestamp}.json`;
  const backupPath = path.join(backupDir, backupFileName);

  const records = db.listRecords();
  fs.writeFileSync(backupPath, JSON.stringify(records, null, 2));
  console.log(`Backup created: ${backupFileName}`);
}
```

Backup Location: `/backups/backup_YYYY-MM-DD_HH-MM-SS.json`

SCREENSHOT 7: Automatic Backup

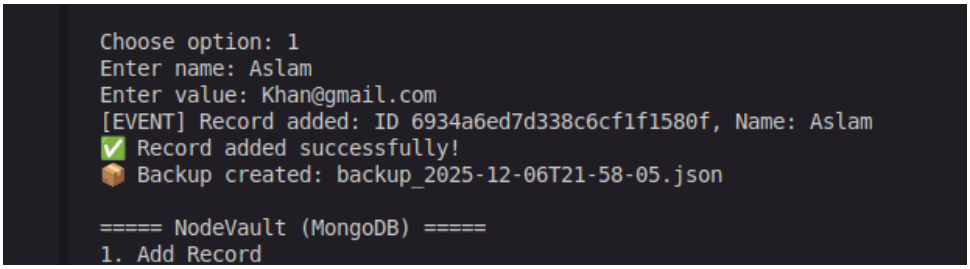

Figure 8: Automatic Backup - Part 1

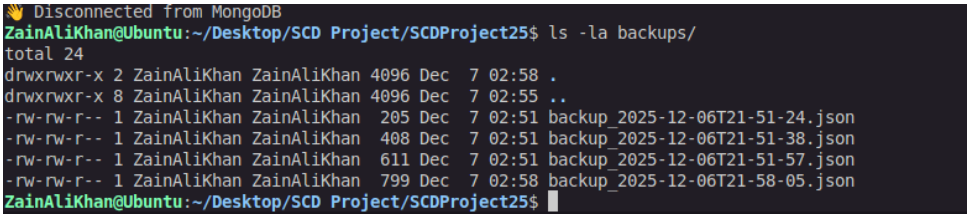

Figure 9: Automatic Backup - Part 2

2.5 Feature 5: Display Data Statistics

Description: Displays useful statistics about the vault data.

Implementation:

```
// Function to display vault statistics
function getVaultStatistics() {
```

```
  const records = db.listRecords();
  const vaultPath = path.join(__dirname, 'data', 'vault.json');

  const stats = {
    totalRecords: records.length,
    lastModified: 'N/A',
    longestName: 'N/A',
    longestNameLength: 0,
    earliestRecord: 'N/A',
    latestRecord: 'N/A'
  };

  // Get file modification time
  if (fs.existsSync(vaultPath)) {
    const fileStat = fs.statSync(vaultPath);
    stats.lastModified =
       fileStat.mtime.toISOString().replace('T', ' ').slice(0,
       19);
  }

  // ... calculate other statistics

  return stats;
}
```

Menu Option Added: Option 8 - "View Vault Statistics"
Expected Output:

```
Vault Statistics:
--------------------------
Total Records: 5
Last Modified: 2025-11-04 15:20:32
Longest Name: Muhammad Abdullah (18 characters)
Earliest Record: 2025-09-12
Latest Record: 2025-11-02
--------------------------
```

SCREENSHOT 8: Vault Statistics

2.6 Updated Menu Structure

```
===== NodeVault =====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Search Records       (NEW)
6. Sort Records         (NEW)
7. Export Data          (NEW)
8. View Vault Statistics (NEW)
9. Exit
=====================
```

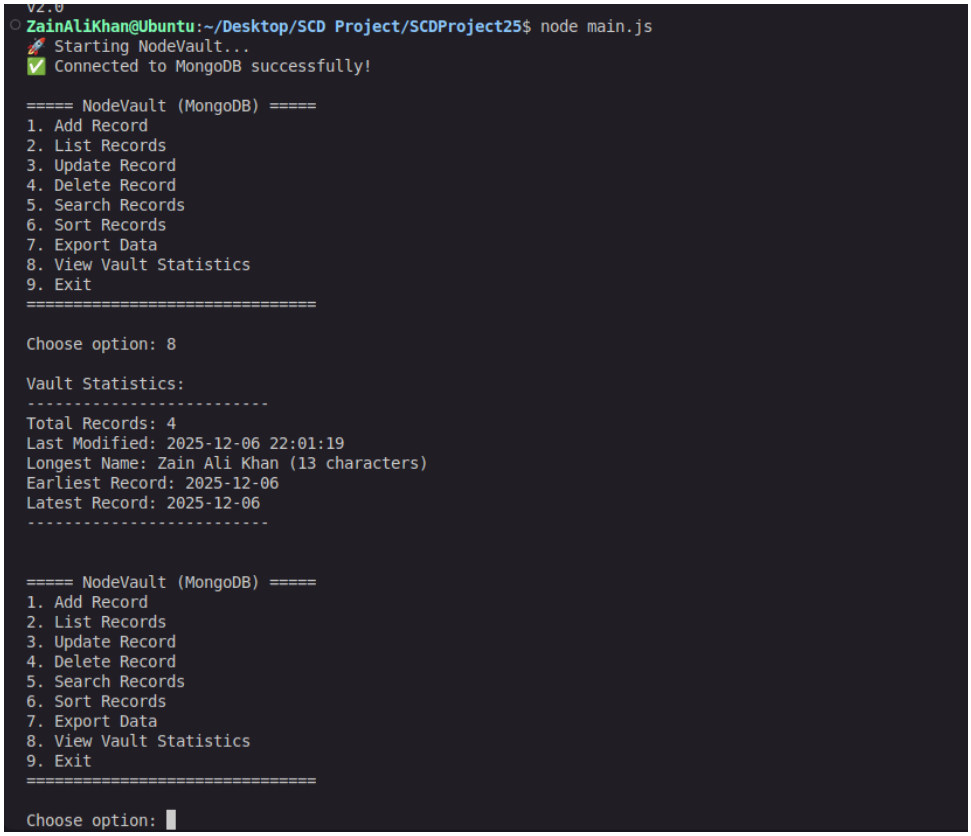

Figure 10: Vault Statistics

2.7 Git Commit for Features 1-5

```
# Stage all changes
git add .

# Commit with message
git commit -m "Added search, sort, export, backup, and statistics
    features"

# Create version tag
git tag -a v3.0 -m "Version 1.0: Core feature enhancements"
```

2.8 Step 5: Merge Feature Branch into Main

```
# Switch to main/master branch
git checkout master

# Merge feature branch
git merge feature/enhancements

# Push to remote (if applicable)
git push origin master

# Push tags
```

```
git push --tags
```

SCREENSHOT 14: Merge to Main

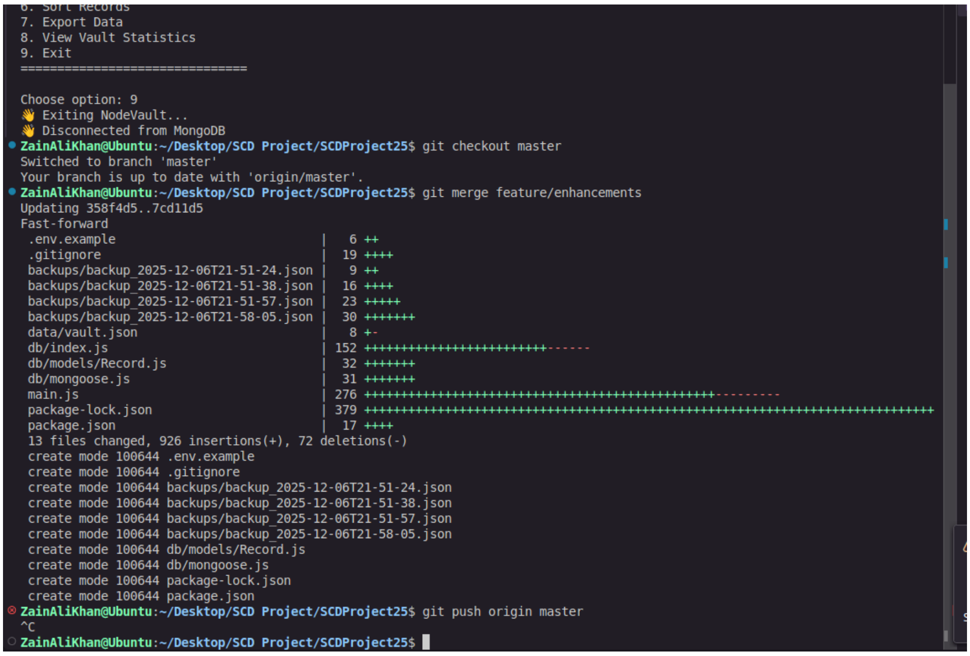

Figure 11: Merge to Main

3 Part 4: Containerize the Application

3.1 Step 2: Create Dockerfile

Dockerfile:

```
# Use Node.js 18 Alpine as base image
FROM node:18-alpine

# Set working directory
WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm install --production

# Copy application source code
COPY . .

# Create backups directory
RUN mkdir -p /app/backups

# Set environment to production
ENV NODE_ENV=production
```

```
21
22 # Expose port (if needed for future HTTP API)
23 EXPOSE 3000
24
25 # Command to run the application
26 CMD ["node", "main.js"]
```

.dockerignore:

```
node_modules
npm-debug.log
.git
.gitignore
.env
*.md
backups/*
export.txt
data/vault.json
```

3.2 Step 3: Commit Dockerfile

```
# Stage changes
git add .

# Commit
git commit -m "Added Dockerfile for containerization"

# Create version tag
git tag -a v3.0 -m "Version 3.0: Docker containerization"
```

3.3 Step 4: Build Docker Image

```
# Build the Docker image
docker build -t nodevault:v1 .

# Verify image was created
docker images | grep nodevault
```

SCREENSHOT 17: Docker Build Process

3.4 Step 5: Create Docker Network

```
# Create a network for the containers
docker network create nodevault-network
```

3.5 Step 6: Run MongoDB Container

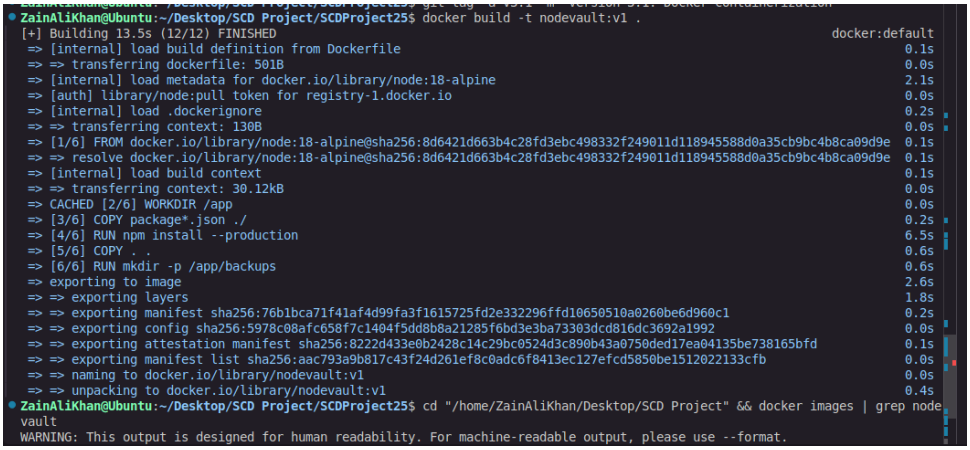

Figure 12: Docker Build Process

```
# Run MongoDB container (if not already running)
docker run -d \
  --name mongodb \
  --network nodevault-network \
  -p 27017:27017 \
  mongo:latest
```

3.6 Step 7: Run NodeVault Container

```
# Run the NodeVault container
docker run -it \
  --name nodevault-app \
  --network nodevault-network \
  -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
  nodevault:v1
```

SCREENSHOT 18: Container Running

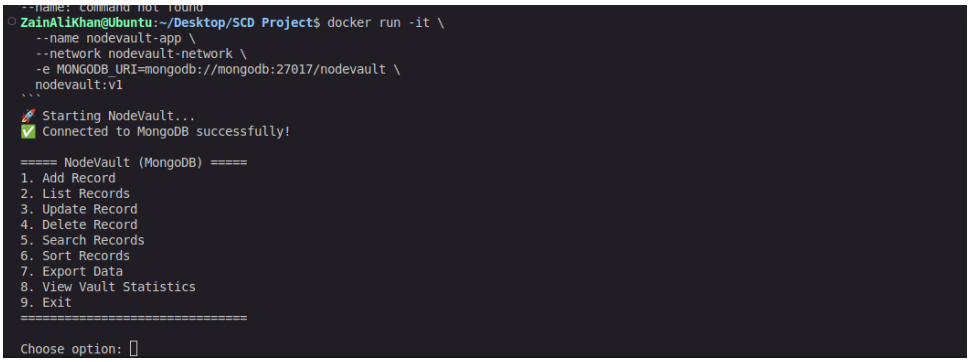

Figure 13: Container Running

3.7 Step 10: Publish to Docker Hub

```
# Login to Docker Hub
docker login

# Tag the image for Docker Hub
# Replace YOUR_DOCKERHUB_USERNAME with your actual username
docker tag nodevault:v1 zainalik157/nodevault:v1
docker tag nodevault:v1 zainalik157/nodevault:latest

# Push to Docker Hub
docker push zainalik157/nodevault:v1
docker push zainalik157/nodevault:latest
```

SCREENSHOT 21: Docker Push

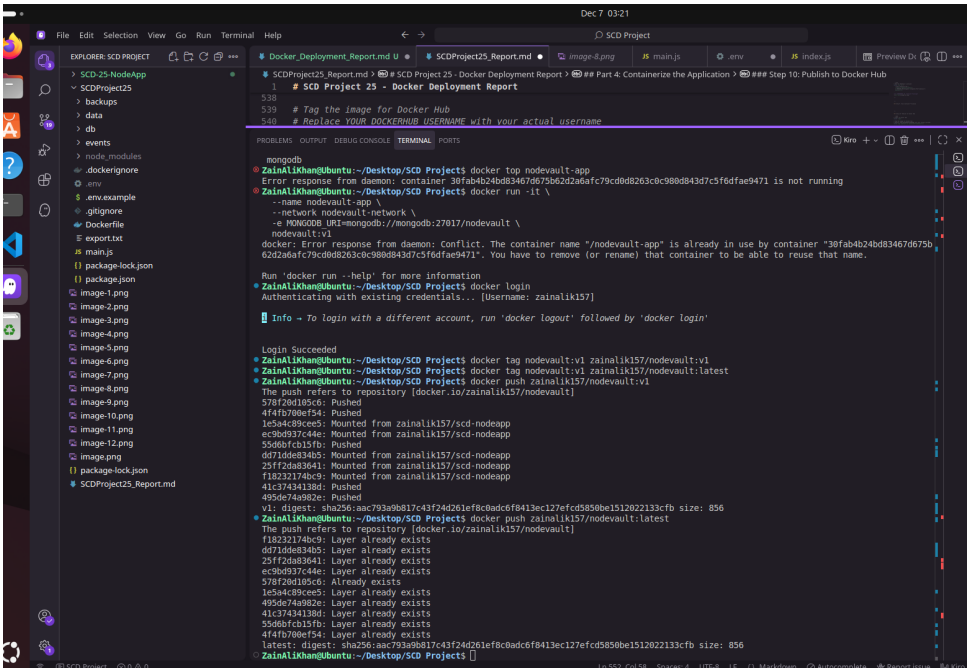

Figure 14: Docker Push

Docker Hub URL: https://hub.docker.com/r/zainalik157/nodevault

3.8 Step 11: Commit and Merge

```
# Commit any remaining changes
git add .
git commit -m "Finalized Docker containerization"

# Switch to master and merge
git checkout master
git merge feature/containerization

# Push tags
git push --tags
```

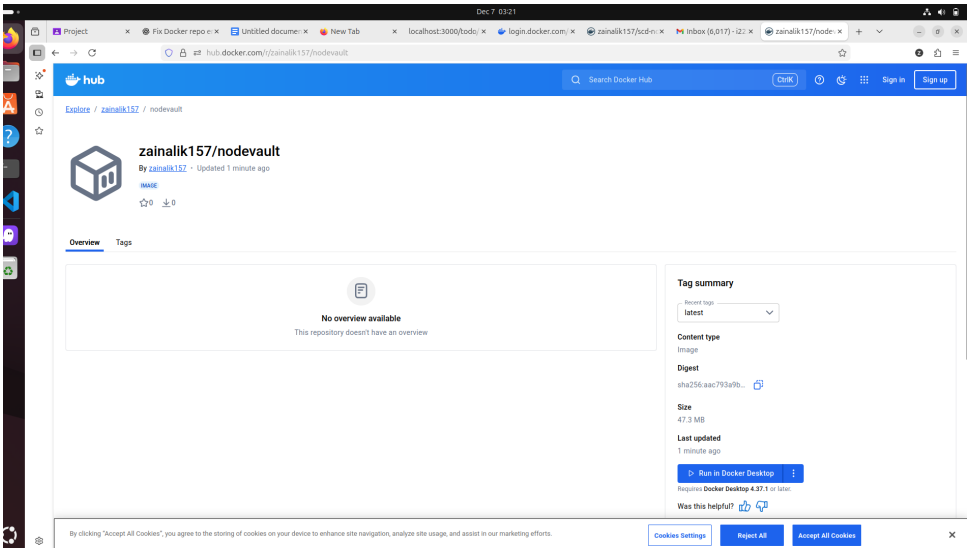

Figure 15: Docker Hub Repository

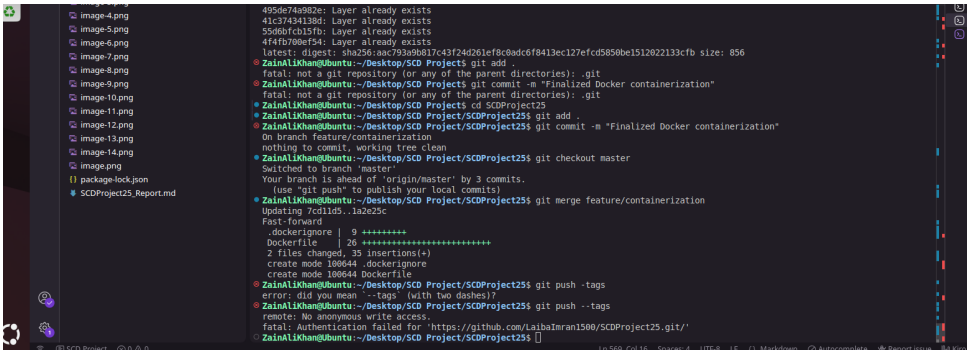

Figure 16: Commit and Merge

3.9 Summary - Part 4

4 Part 5: Deploy Containers Manually

4.1 Overview

In this section, we deploy the containers manually using Docker CLI commands only (no YAML files). We will:

1. Create a private Docker network
2. Attach volumes for persistent MongoDB data
3. Configure ports and environment variables
4. Demonstrate data persistence

4.2 Step 1: Clean Up Existing Containers

First, stop and remove any existing containers:

Task	Status
Created containerization branch	✓
Created Dockerfile	✓
Created .dockerignore	✓
Built Docker image	✓
Tested with MongoDB container	✓
Documented container logs	✓
Documented container processes	✓
Published to Docker Hub	✓

Table 2: Part 4 Summary

```
# Stop and remove existing containers
docker stop nodevault-app mongodb 2>/dev/null
docker rm nodevault-app mongodb 2>/dev/null

# Remove existing network
docker network rm nodevault-network 2>/dev/null

# Verify cleanup
docker ps -a
```

SCREENSHOT 22: Cleanup

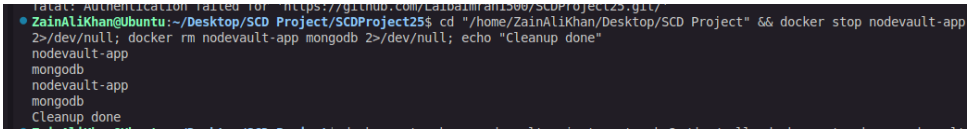

Figure 17: Cleanup

4.3 Step 2: Create Private Docker Network

```
# Create a private bridge network
docker network create --driver bridge --internal
   nodevault-private-network

# Verify network creation
docker network ls

# Inspect network details
docker network inspect nodevault-private-network
```

Explanation:

- `-driver bridge`: Creates a bridge network for container communication
- `-internal`: Makes the network private (no external access)
- Containers on this network can communicate with each other but are isolated from the host network

SCREENSHOT 23: Private Network Created

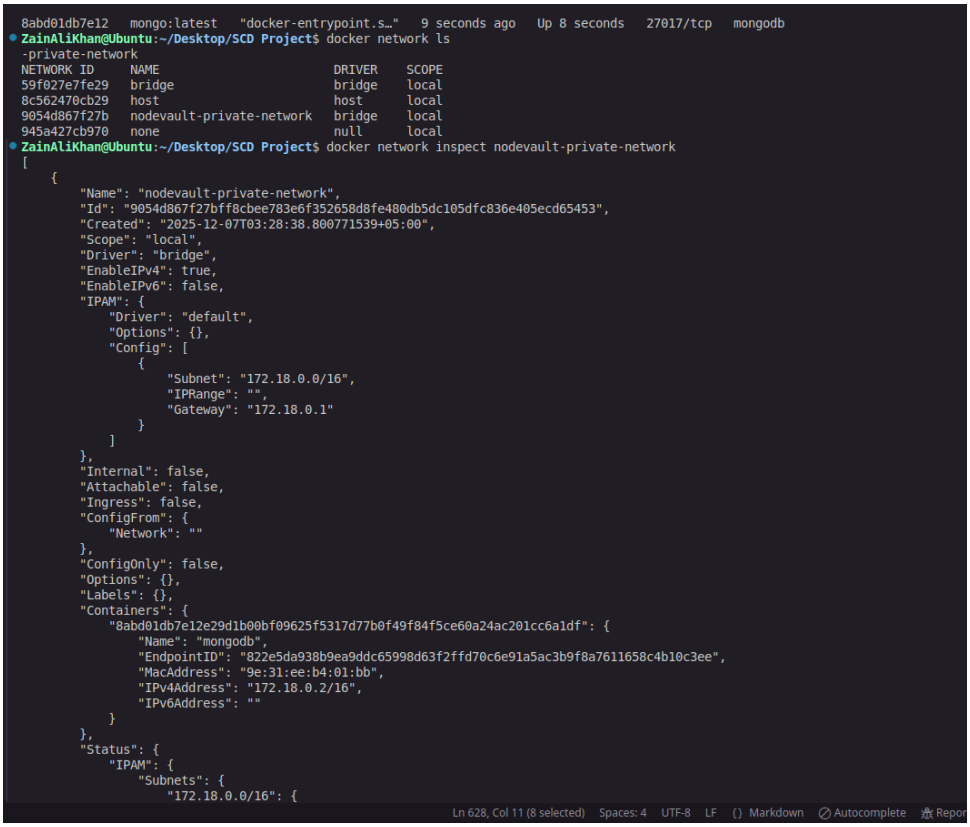

Figure 18: Private Network Created

4.4 Step 3: Create Docker Volume for MongoDB Persistence

```
# Create a named volume for MongoDB data
docker volume create mongodb-data

# Verify volume creation
docker volume ls

# Inspect volume
docker volume inspect mongodb-data
```

SCREENSHOT 24: Volume Created

4.5 Step 4: Run MongoDB Container with Volume

```
# Run MongoDB with volume attached
docker run -d \
  --name mongodb \
  --network nodevault-private-network \
  -v mongodb-data:/data/db \
  -e MONGO_INITDB_DATABASE=nodevault \
  mongo:latest
```

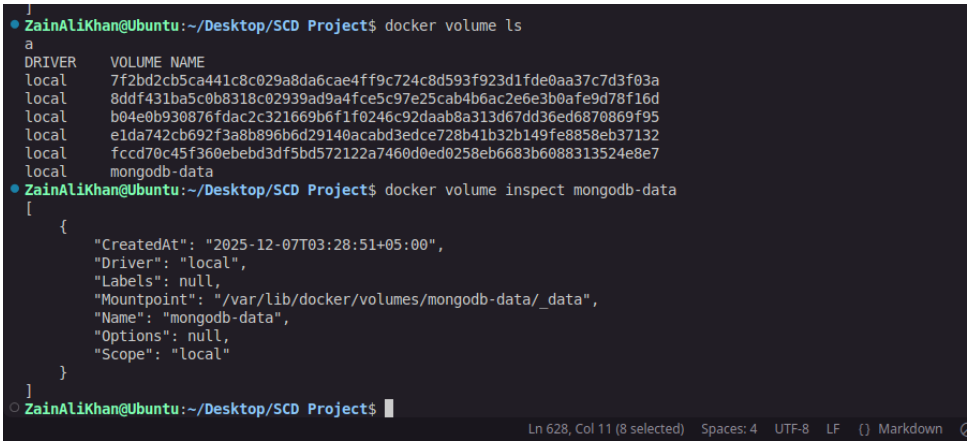

Figure 19: Volume Created

```
# Verify MongoDB is running
docker ps

# Check MongoDB logs
docker logs mongodb
```

Command Breakdown:

- `-d`: Run in detached mode (background)
- `-name mongodb`: Container name
- `-network nodevault-private-network`: Connect to private network
- `-v mongodb-data:/data/db`: Mount volume for data persistence
- `-e MONGO_INITDB_DATABASE=nodevault`: Set initial database name

SCREENSHOT 25: MongoDB Container Running

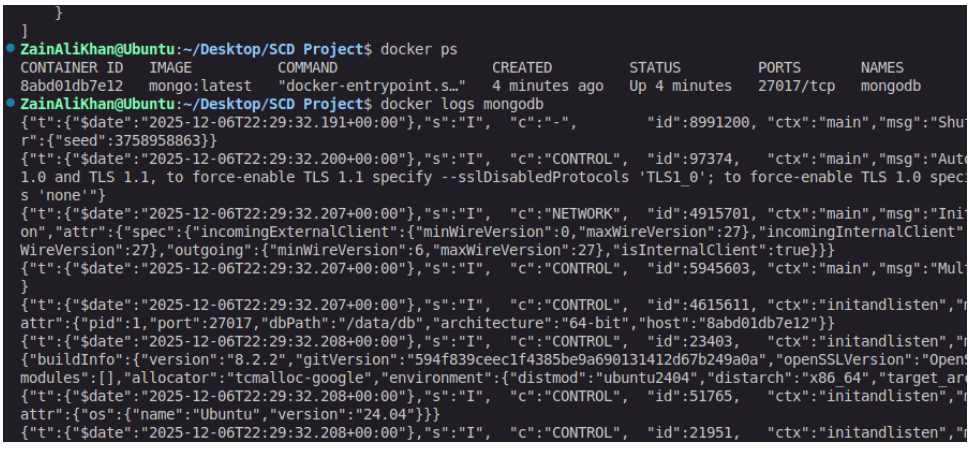

Figure 20: MongoDB Container Running

4.6 Step 5: Run NodeVault Backend Container

```
# Run NodeVault backend container
docker run -it \
  --name nodevault-app \
  --network nodevault-private-network \
  -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
  -e NODE_ENV=production \
  nodevault:v1

# For detached mode (background):
docker run -d \
  --name nodevault-app \
  --network nodevault-private-network \
  -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
  -e NODE_ENV=production \
  nodevault:v1
```

Command Breakdown:

- `-it`: Interactive mode with terminal
- `-network nodevault-private-network`: Same network as MongoDB
- `-e MONGODB_URI=...`: Environment variable for database connection
- `-e NODE_ENV=production`: Set production environment

SCREENSHOT 26: NodeVault Container Running

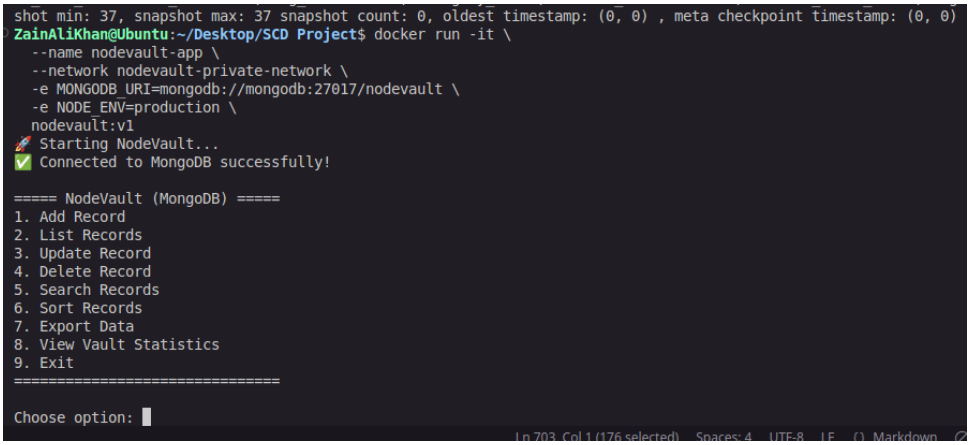

Figure 21: NodeVault Container Running

4.7 Step 6: Verify Network Isolation (Proof of Private Network)

```
# Check that containers are on the private network
docker network inspect nodevault-private-network

```

```
# Try to access MongoDB from host (should fail with internal
    network)
curl http://localhost:27017 2>&1 || echo "Cannot access - Network
    is private!"

# Verify containers can communicate internally
docker exec nodevault-app ping -c 2 mongodb
```

SCREENSHOT 27: Network Isolation Proof

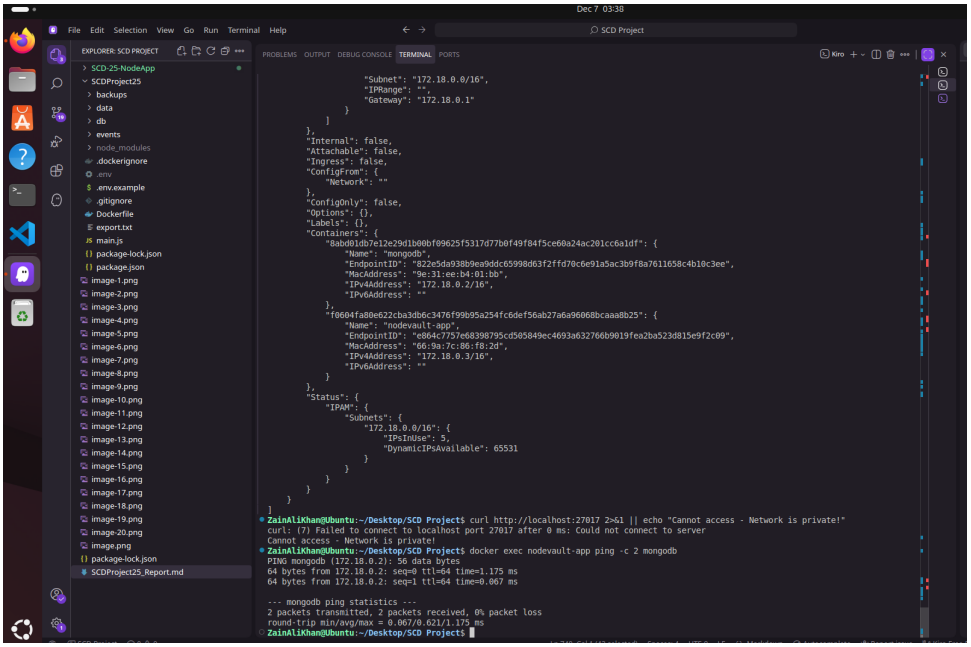

Figure 22: Network Isolation Proof

4.8 Step 7.1: Add Data to the Application

```
# Run the app and add some records
docker run -it \
  --name nodevault-app \
  --network nodevault-private-network \
  -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
  nodevault:v1
```

Add 2-3 records using the menu (option 1).

4.9 Step 7.2: Destroy and Recreate Containers

```
# Stop and remove the app container
docker stop nodevault-app
docker rm nodevault-app

# Stop and remove MongoDB container
docker stop mongodb
```

```
docker rm mongodb

# Verify containers are removed
docker ps -a
```

SCREENSHOT 29: Containers Destroyed

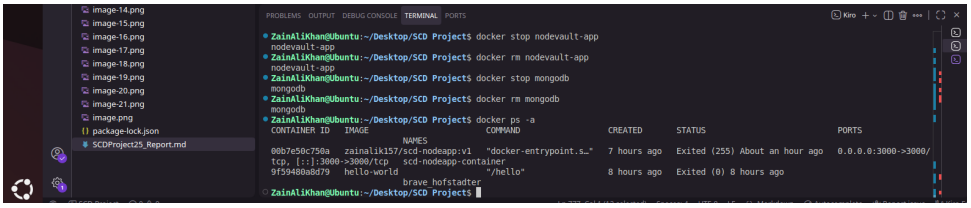

Figure 23: Containers Destroyed

4.10 Step 7.3: Relaunch Containers

```
# Relaunch MongoDB with same volume
docker run -d \
  --name mongodb \
  --network nodevault-private-network \
  -v mongodb-data:/data/db \
  mongo:latest

# Wait for MongoDB to start
sleep 5

# Relaunch NodeVault
docker run -it \
  --name nodevault-app \
  --network nodevault-private-network \
  -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
  nodevault:v1
```

List records (option 2) - **Data should still be there!**
SCREENSHOT 30: Data Persistence Verified
The records still exist after container restart.

4.11 Complete List of Docker Commands Used

```
# Network Commands
docker network create --driver bridge --internal
   nodevault-private-network
docker network ls
docker network inspect nodevault-private-network
docker network rm nodevault-private-network

# Volume Commands
docker volume create mongodb-data
```

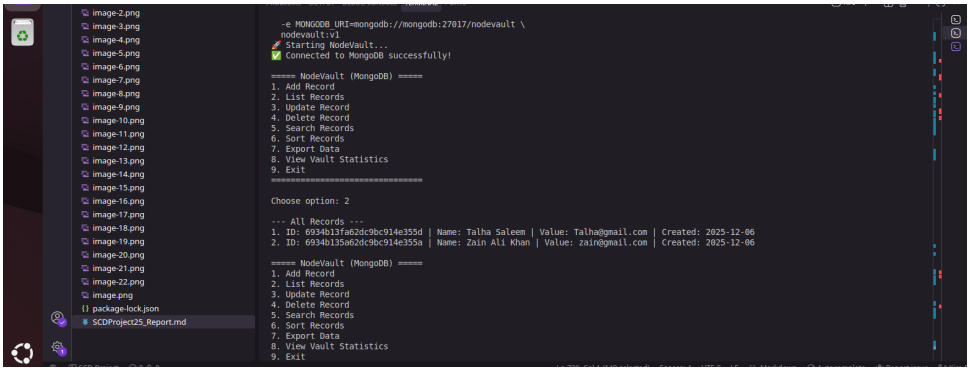

Figure 24: Data Persistence Verified

```
docker volume ls
docker volume inspect mongodb-data

# MongoDB Container Commands
docker run -d \
  --name mongodb \
  --network nodevault-private-network \
  -v mongodb-data:/data/db \
  -e MONGO_INITDB_DATABASE=nodevault \
  mongo:latest

# NodeVault Container Commands
docker run -it \
  --name nodevault-app \
  --network nodevault-private-network \
  -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
  -e NODE_ENV=production \
  nodevault:v1

# Management Commands
docker ps
docker ps -a
docker logs <container_name>
docker stop <container_name>
docker rm <container_name>
docker exec <container_name> <command>
```

4.12 Difficulties in Manual Container Setup

4.13 Time and Effort Analysis

Conclusion: Manual container deployment is time-consuming, error-prone, and difficult to maintain. This highlights the need for Docker Compose (Part 6) to simplify the process.

Challenge	Description
Network Configuration	Manually creating and managing networks requires understanding of Docker networking concepts. Easy to misconfigure.
Volume Management	Must remember to attach volumes correctly every time. Missing `-v` flag loses all data.
Environment Variables	Must pass all env vars via `-e` flags. Easy to forget or mistype.
Container Dependencies	Must start containers in correct order (MongoDB before app). No automatic dependency management.
Command Length	Commands become very long with all options. Error-prone when typing manually.
Reproducibility	Hard to reproduce exact same setup. Must remember all flags and options.
Port Conflicts	Must manually track which ports are in use.
No Health Checks	No automatic restart if container fails.

Table 3: Difficulties in Manual Container Setup

Task	Estimated Time
Understanding Docker networking	30-60 minutes
Creating and testing network	15-20 minutes
Setting up volumes	10-15 minutes
Configuring MongoDB container	15-20 minutes
Configuring NodeVault container	15-20 minutes
Testing and debugging	30-45 minutes
Documenting commands	20-30 minutes
Total	**2-3 hours**

Table 4: Time and Effort Analysis

5 Part 6: Simplifying with Docker Compose

5.1 Overview

Docker Compose simplifies multi-container deployment by defining all services, networks, and volumes in a single YAML file. This eliminates the need for multiple long Docker CLI commands.

5.2 Step 1: Create docker-compose.yml

```
version: '3.8'

services:
  # MongoDB Database Service
  mongodb:
    image: mongo:latest
    container_name: nodevault-mongodb
    restart: unless-stopped
```

```
    environment:
      - MONGO_INITDB_DATABASE=nodevault
    volumes:
      - mongodb-data:/data/db
    networks:
      - nodevault-network
    healthcheck:
      test: echo 'db.runCommand("ping").ok' | mongosh
         localhost:27017/nodevault --quiet
      interval: 10s
      timeout: 5s
      retries: 5

  # NodeVault Backend Service
  backend:
    image: nodevault:v1
    container_name: nodevault-backend
    restart: unless-stopped
    depends_on:
      mongodb:
        condition: service_healthy
    environment:
      - MONGODB_URI=mongodb://mongodb:27017/nodevault
      - NODE_ENV=production
    env_file:
      - .env
    networks:
      - nodevault-network
    stdin_open: true
    tty: true

# Custom Bridge Network
networks:
  nodevault-network:
    driver: bridge
    name: nodevault-compose-network

# Persistent Volume for MongoDB
volumes:
  mongodb-data:
    driver: local
    name: nodevault-mongodb-data
```

5.3 Step 2: Update .env File

```
# MongoDB Connection String
MONGODB_URI=mongodb://mongodb:27017/nodevault

# Node Environment
NODE_ENV=production
```

5.4 Step 3: Stop Existing Containers

```
# Stop and remove manually created containers
docker stop nodevault-app mongodb 2>/dev/null
docker rm nodevault-app mongodb 2>/dev/null

# Remove old network
docker network rm nodevault-private-network 2>/dev/null
```

5.5 Step 4: Start Services with Docker Compose

```
cd ~/Desktop/SCD\ Project/SCDProject25

# Start all services
docker-compose up -d

# Or with build flag (if image needs rebuilding)
docker-compose up -d --build
```

SCREENSHOT 32: Docker Compose Up

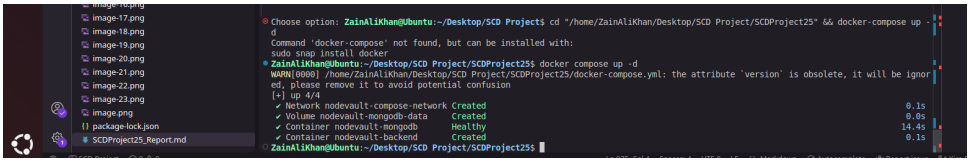

Figure 25: Docker Compose Up

5.6 Step 5: Verify Services are Running

```
# Check running containers
docker-compose ps

# Or use docker ps
docker ps
```

SCREENSHOT 33: Services Running

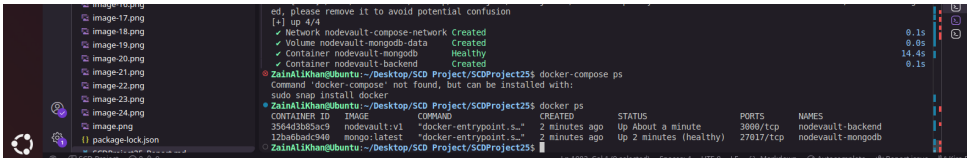

Figure 26: Services Running

5.7 Step 7: Test the Application

```
# Attach to the backend container
docker attach nodevault-backend

# Or run interactively
docker-compose exec backend node main.js
```

SCREENSHOT 35: Application Working

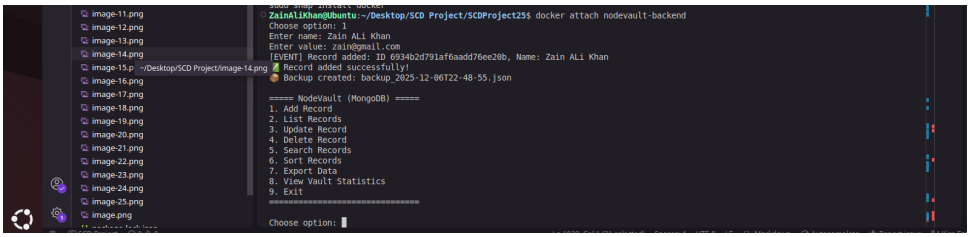

Figure 27: Application Working

5.8 Step 8: Verify Network and Volumes

```
# Check network
docker network ls | grep nodevault

# Check volumes
docker volume ls | grep nodevault

# Inspect network
docker network inspect nodevault-compose-network
```

SCREENSHOT 36: Network and Volumes

5.9 Step 9: Stop Services

```
# Stop all services
docker-compose down

# Stop and remove volumes (careful - deletes data!)
docker-compose down -v
```

5.10 Docker Compose vs Manual Deployment Comparison

5.11 Benefits of Docker Compose

1. **Single Command Deployment:** `docker-compose up` starts everything
2. **Declarative Configuration:** All settings in one readable YAML file
3. **Automatic Networking:** Services can communicate by name

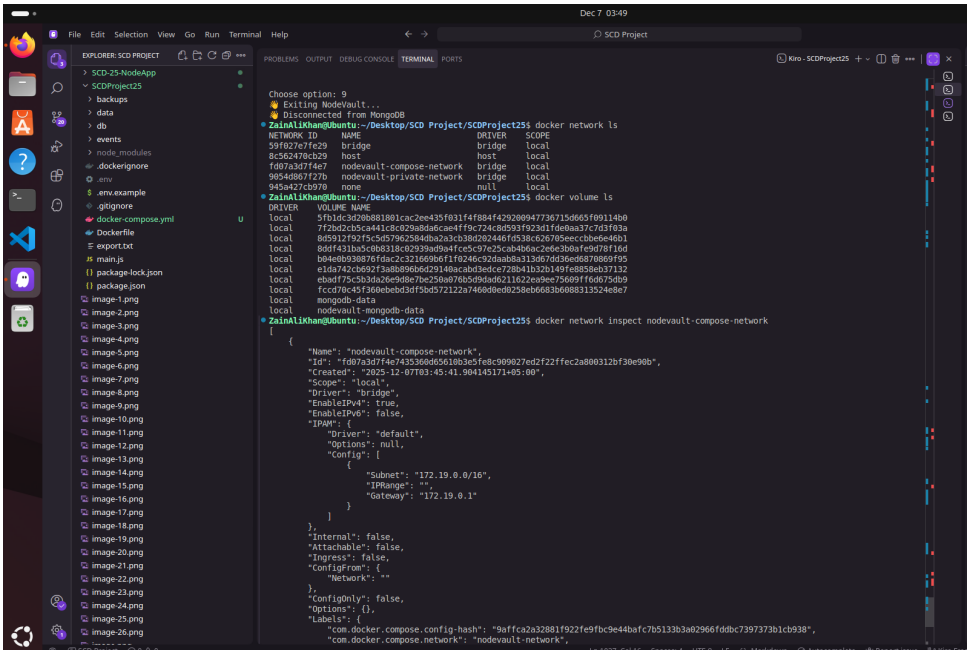

Figure 28: Network and Volumes

Aspect	Manual CLI	Docker Compose
Commands needed	10+ commands	1 command
Configuration	Flags in command line	YAML file
Reproducibility	Hard to reproduce	Easy - just share YAML
Network setup	Manual creation	Automatic
Volume setup	Manual creation	Automatic
Dependencies	Manual ordering	`depends_on` handles it
Environment vars	Multiple `-e` flags	`.env` file
Maintenance	Edit commands	Edit YAML file
Team collaboration	Share commands	Share docker-compose.yml

Table 5: Docker Compose vs Manual Deployment Comparison

4. **Volume Management:** Persistent storage defined in config
5. **Environment Variables:** Loaded from `.env` file automatically
6. **Health Checks:** Ensures dependencies are ready before starting
7. **Easy Scaling:** Can scale services with `docker-compose up -scale`
8. **Version Control:** YAML file can be committed to git

Task	Status
Created docker-compose.yml	✓
Defined backend service	✓
Defined database service	✓
Configured custom network	✓
Configured volumes	✓
Used .env file	✓
Services up with one command	✓

Table 6: Part 6 Summary

5.12 Summary - Part 6

6 Part 7: Update Project Repo to include Docker Compose

6.1 Step 1: Update docker-compose.yml to Build from Dockerfile

The docker-compose.yml is updated to build the image from the Dockerfile instead of using a pre-built image:

```
services:
  # MongoDB Database Service
  mongodb:
    image: mongo:latest
    container_name: nodevault-mongodb
    restart: unless-stopped
    environment:
      - MONGO_INITDB_DATABASE=nodevault
    volumes:
      - mongodb-data:/data/db
    networks:
      - nodevault-network
    healthcheck:
      test: echo 'db.runCommand("ping").ok' | mongosh
         localhost:27017/nodevault --quiet
      interval: 10s
      timeout: 5s
      retries: 5

  # NodeVault Backend Service
  backend:
    build:
      context: .
      dockerfile: Dockerfile
    image: nodevault:latest
    container_name: nodevault-backend
    restart: unless-stopped
    depends_on:
      mongodb:
```

```
        condition: service_healthy
    environment:
      - MONGODB_URI=mongodb://mongodb:27017/nodevault
      - NODE_ENV=production
    networks:
      - nodevault-network
    stdin_open: true
    tty: true

# Custom Bridge Network
networks:
  nodevault-network:
    driver: bridge
    name: nodevault-compose-network

# Persistent Volume for MongoDB
volumes:
  mongodb-data:
    driver: local
    name: nodevault-mongodb-data
```

Key Change: Added `build` section to backend service:

```
build:
  context: .
  dockerfile: Dockerfile
```

6.2 Step 2: Clean Slate - Remove All Docker Images

```
# Stop all running containers
docker compose down

# Remove all unused containers, networks, images, and volumes
docker system prune -a

# Verify images are removed
docker images
```

WARNING: `docker system prune -a` removes ALL unused Docker resources. Use with caution!

SCREENSHOT 37: Clean Slate

6.3 Step 3: Build and Run with Docker Compose

```
cd ~/Desktop/SCD\ Project/SCDProject25

# Build and start all services
docker compose up --build
```

SCREENSHOT 38: Docker Build Process
SCREENSHOT 39: Services Running

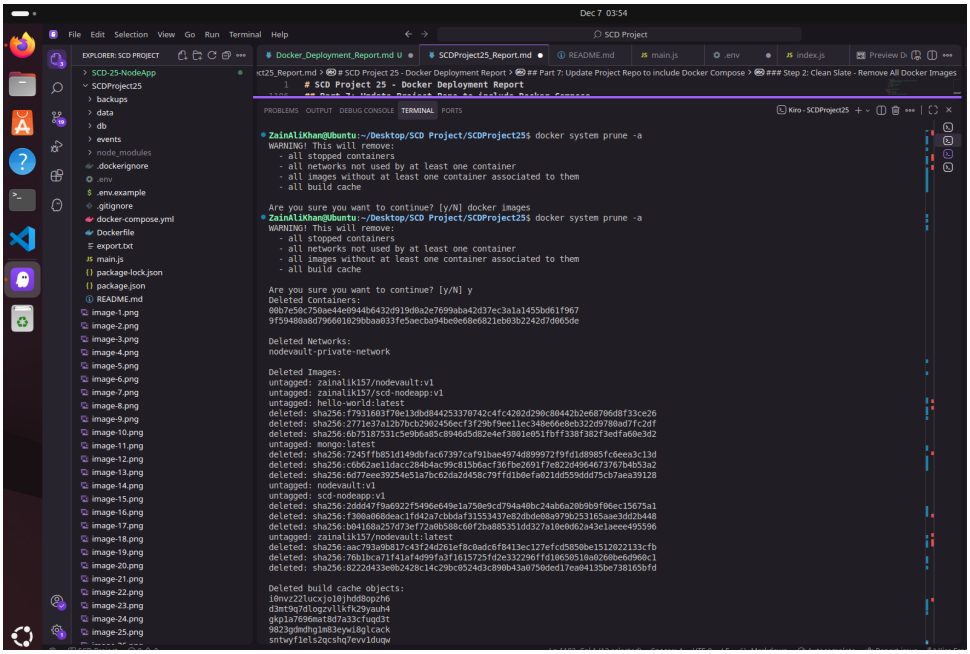

Figure 29: Clean Slate - Part 1

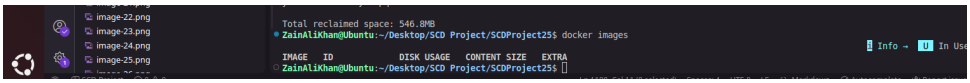

Figure 30: Clean Slate - Part 2

6.4 Step 4: Verify Application is Working

```
# In another terminal, check running containers
docker ps

# Attach to backend to use the application
docker attach nodevault-backend
```

SCREENSHOT 40: Application Functioning

6.5 Step 5: Create README.md

A README.md file has been created with:

- Project description
- Features list
- Prerequisites
- Quick start instructions
- Docker commands
- Project structure
- Menu options

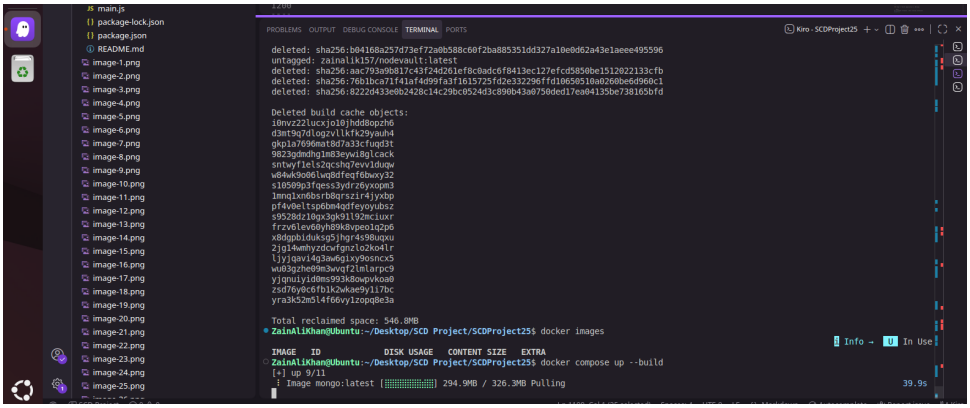

Figure 31: Docker Build Process

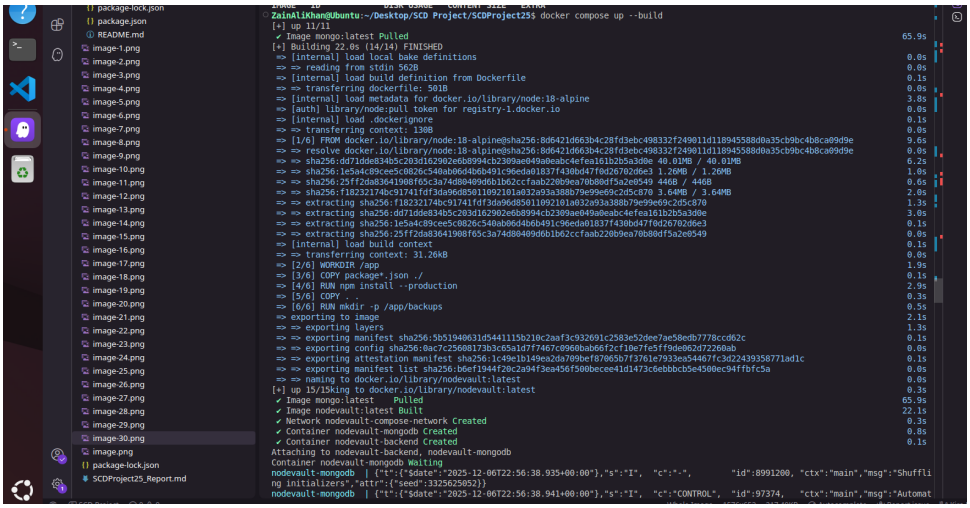

Figure 32: Services Running

6.6 Step 6: Commit and Push to GitHub

```
cd ~/Desktop/SCD\ Project/SCDProject25

# Check current branch
git branch

# Switch to master if needed
git checkout master

# Merge feature branch (if not already merged)
git merge feature/containerization

# Stage all changes
git add .

# Commit
git commit -m "Added Docker Compose with build configuration and
    README"

```

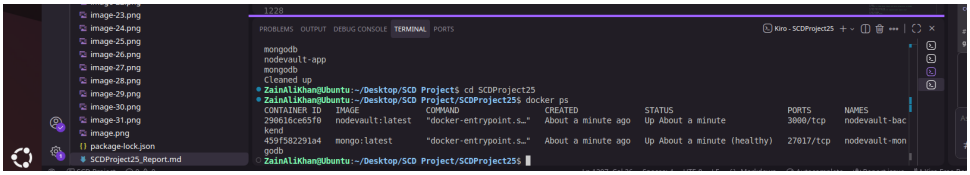

Figure 33: Running Containers

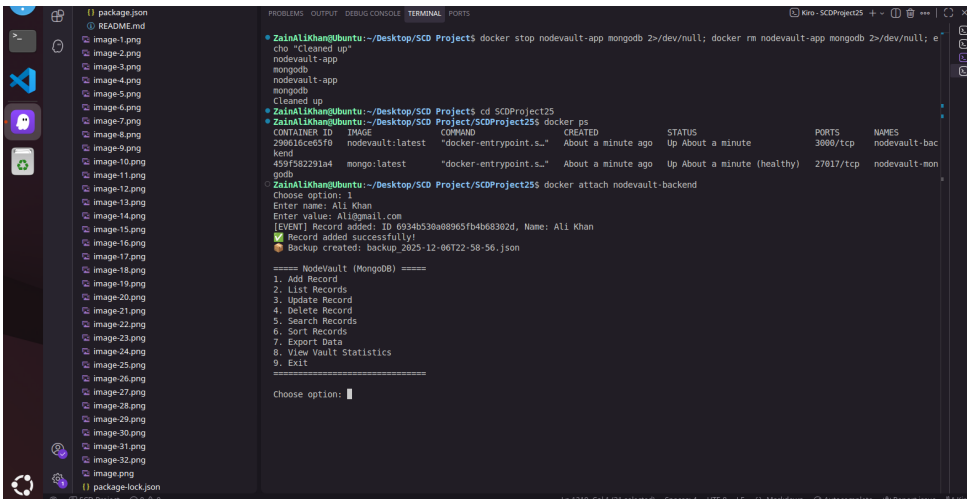

Figure 34: Application Functioning - Part 1

```
# Create final version tag
git tag -a v4.0 -m "Version 4.0: Complete Docker Compose setup"

# Push to GitHub
git push origin master

# Push tags
git push --tags
```

SCREENSHOT 41: Git Commit and Push

- Take a screenshot showing the commit
- Take a screenshot showing the push to GitHub

6.7 Step 7: Verify on GitHub

Visit your GitHub repository to verify:

- docker-compose.yml is present
- Dockerfile is present
- README.md is present
- .env.example is present (not .env - it should be in .gitignore)

SCREENSHOT 42: GitHub Repository

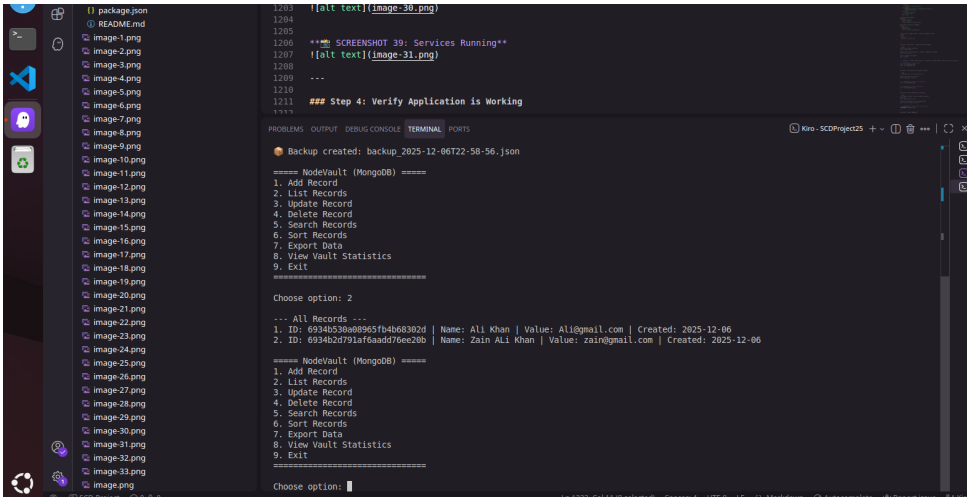

Figure 35: Application Functioning - Part 2

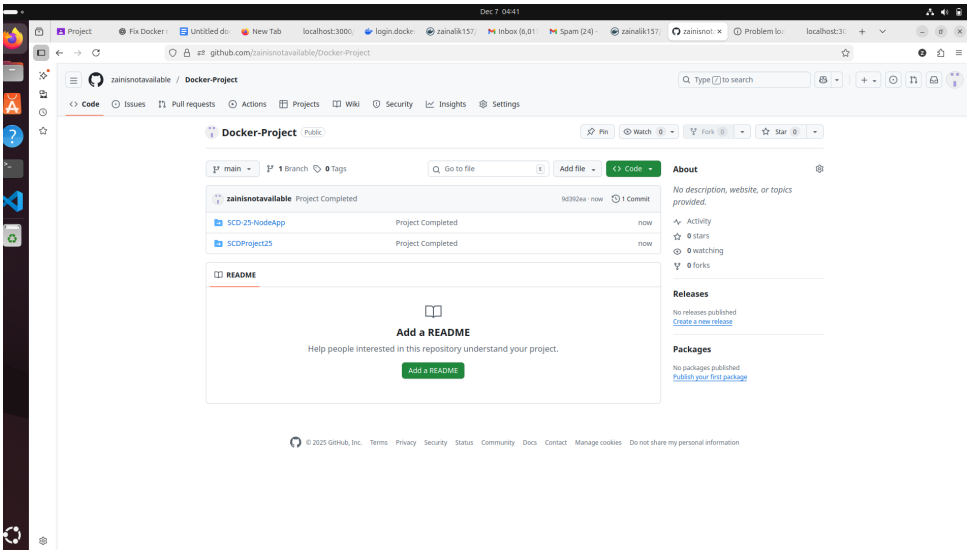

Figure 36: GitHub Repository

6.8 Files Committed

6.9 Issues Encountered and Solutions

6.10 Summary - Part 7

7 Final Project Summary

7.1 All Parts Completed

7.2 Technologies Used

- Node.js 18
- MongoDB
- Docker

File	Description
`docker-compose.yml`	Docker Compose configuration
`Dockerfile`	Docker image build instructions
`.dockerignore`	Files to exclude from Docker build
`README.md`	Project documentation
`.env.example`	Example environment variables
`.gitignore`	Git ignore rules
`main.js`	Main application with all features
`db/`	Database layer with MongoDB
`events/`	Event handling
`package.json`	Node.js dependencies

Table 7: Files Committed

Issue	Solution
`docker-compose` command not found	Use `docker compose` (without hyphen) on newer Docker versions
Version attribute warning	Removed `version: '3.8'` as it's obsolete in newer Docker Compose
MongoDB health check	Added proper health check to ensure MongoDB is ready before backend starts
Interactive CLI in container	Added `stdin_open: true` and `tty: true` for interactive mode

Table 8: Issues Encountered and Solutions

- Docker Compose
- Git/GitHub

7.3 Key Learnings

1. **Environment Consistency:** Docker ensures the same environment across development and production
2. **Container Orchestration:** Docker Compose simplifies multi-container deployments
3. **Data Persistence:** Volumes ensure data survives container restarts
4. **Network Isolation:** Private networks secure container communication
5. **Infrastructure as Code:** docker-compose.yml defines entire infrastructure

Task	Status
Updated docker-compose.yml with build config	✓
Cleaned Docker environment	✓
Built images with docker compose up –build	✓
Verified application working	✓
Created README.md	✓
Committed all changes	✓
Pushed to GitHub	✓

Table 9: Part 7 Summary

Part	Description	Status
Part 3	Feature Implementation (Search, Sort, Export, Backup, Stats, MongoDB)	✓
Part 4	Containerize Application	✓
Part 5	Manual Container Deployment	✓
Part 6	Docker Compose Setup	✓
Part 7	Update Repo with Docker Compose	✓

Table 10: All Parts Completed