Data Backup and Restore Feature Documentation

Overview

This documentation outlines the implementation of a data backup and restore feature for a full-stack book management application. This feature ensures data integrity and recovery in case of data loss or corruption by creating backups of the database and restoring them when necessary.

Backup Strategy

1. **Frequency**: Backups are created every 24 hours at 2a.m.
2. **Storage Location**: Backups are stored in the '`./backups`' directory.
3. **Retention Policy**: Backups older than 1 day are automatically deleted.
4. **Backup Type**: Full database dump.

Backup Process

1. **Automated Backup**: Configured using a task scheduler to run daily at 2 AM.
2. **Manual Backup**: Can be initiated via a backend API endpoint.

API Endpoint

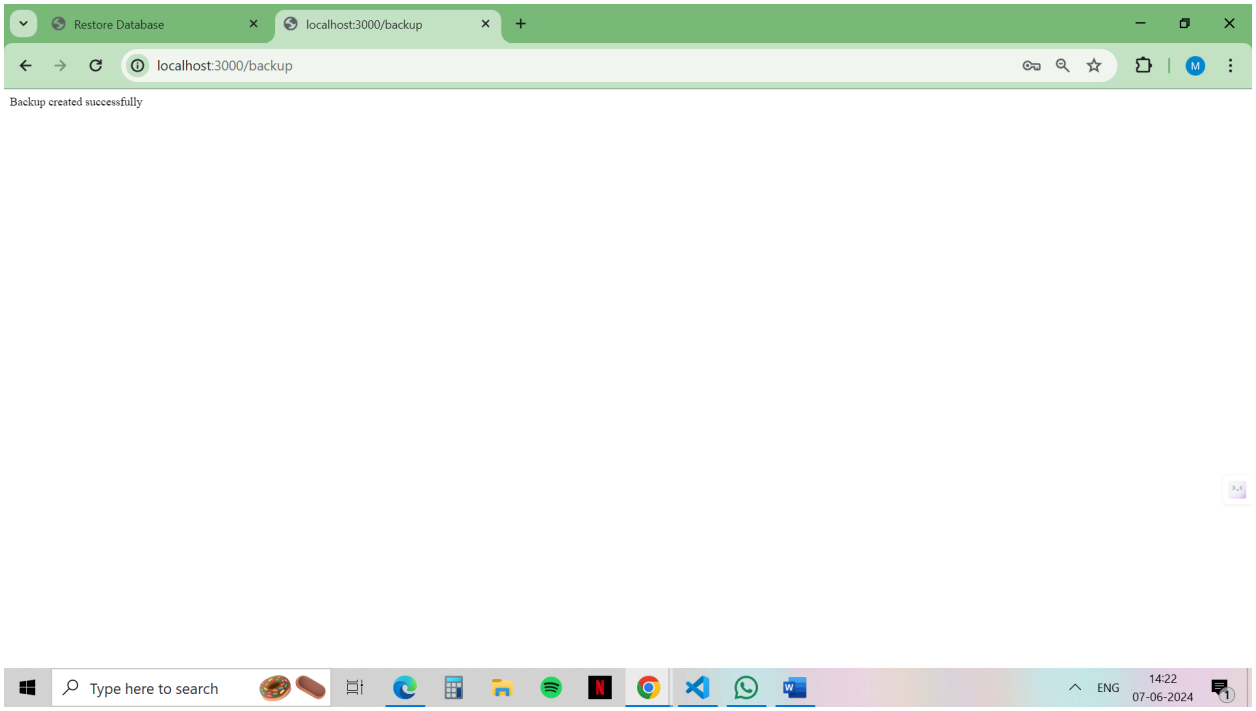
- **Link : http://localhost:3000/backup**
- **Endpoint**: '`/backup`'
- **Method**: POST
- **Parameters**:
    - `Password` : The password for authentication.
- **Response**: Confirmation message on success or error message on failure.

Implementation

```
app.post('/backup', (req, res) => {
    const { password } = req.body;
    if (password !== PASSWORD) {
        return res.status(401).send('Unauthorized: Incorrect password');
    }
```

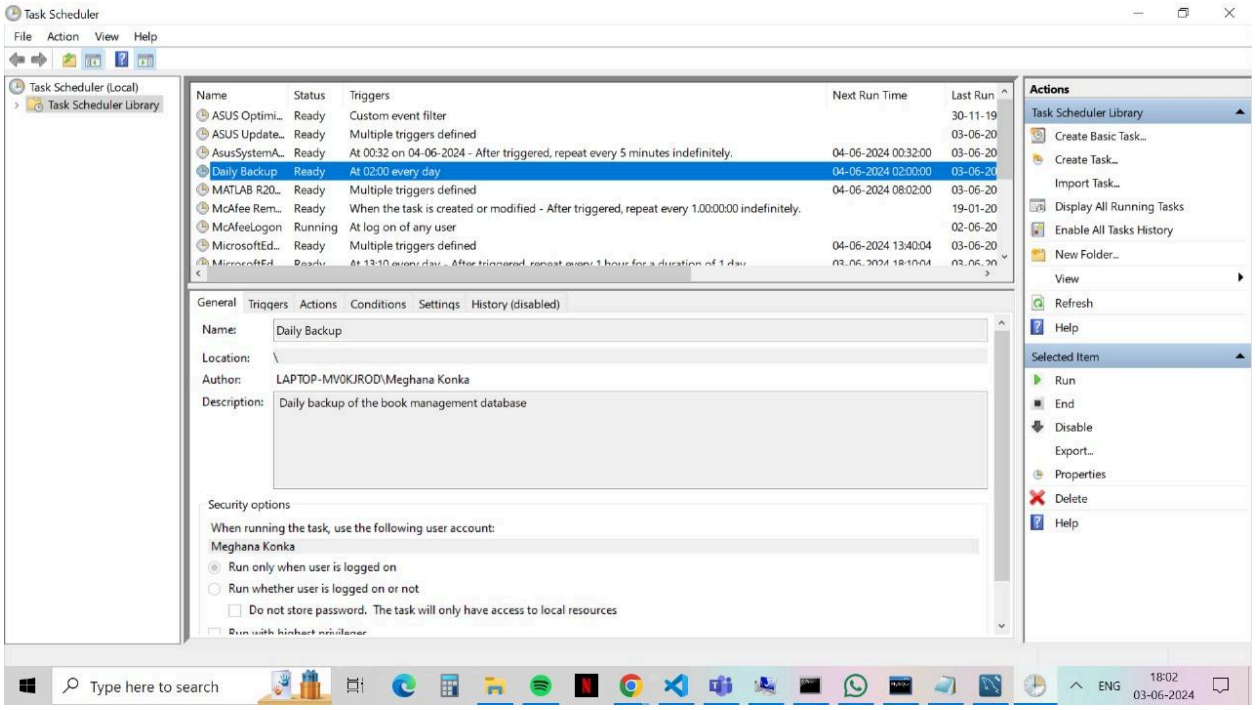
```
    const backupFile = path.join(__dirname, 'backups',
`${Date.now()}.sql`);
    const command = `mysqldump -u${process.env.DB_USER}
-p${process.env.DB_PASSWORD} ${process.env.DB_NAME} > ${backupFile}`;

    exec(command, (error, stdout, stderr) => {
        if (error) {
            console.error(`Error creating backup: ${stderr}`);
            return res.status(500).send('Error creating backup');
        }
        res.send('Backup created successfully');
    });
});
```

BACKUP Webpage:



Automated Backup:

Schedule Backup
Use Task Scheduler on Windows to automate the backup process every 24 hours.
Create a task that runs a script to hit the /backup endpoint. this is done via task scheduling



Restore Strategy

When you restore data using the endpoint http://localhost:3000/restore, the data from the backup file is loaded into your MySQL database specified in your configuration. The restored data is not saved to a new location but rather it overwrites the existing data in the configured database.

Restore Process

1. **Manual Restore**: Can be initiated via a backend API endpoint or through a web form.
2. **Safeguards**: Requires password authentication to prevent accidental data overwrites.

API Endpoint

- **Link : http://localhost:3000/restore**
- **Endpoint**: '`/restore`'
- **Method**: POST
- **Parameters**:
    - `password`: The password for authentication.
    - `backupFile`: The backup file to be restored.
- **Response**: Confirmation message on success or error message on failure.

Implementation

```
app.post('/restore', upload.single('backupFile'), (req, res) => {
    const { password } = req.body;
    if (password !== PASSWORD) {
        return res.status(401).send('Unauthorized: Incorrect password');
    }

    const backupFile = req.file.path;
    const command = `mysql -u${process.env.DB_USER}
-p${process.env.DB_PASSWORD} ${process.env.DB_NAME} < ${backupFile}`;

    exec(command, (error, stdout, stderr) => {
        if (error) {
            console.error(`Error restoring backup: ${stderr}`);
            return res.status(500).send('Error restoring backup');
        }
        res.send('Backup restored successfully');
    });
});
```

Restore Webpage:

Backup Retention and Cleanup

- **Script**: ‘`cleanup_backup.js`’
- **Frequency**: Runs daily at 2 AM using a task scheduler.
- **Function**: Deletes backup files older than 1 day.

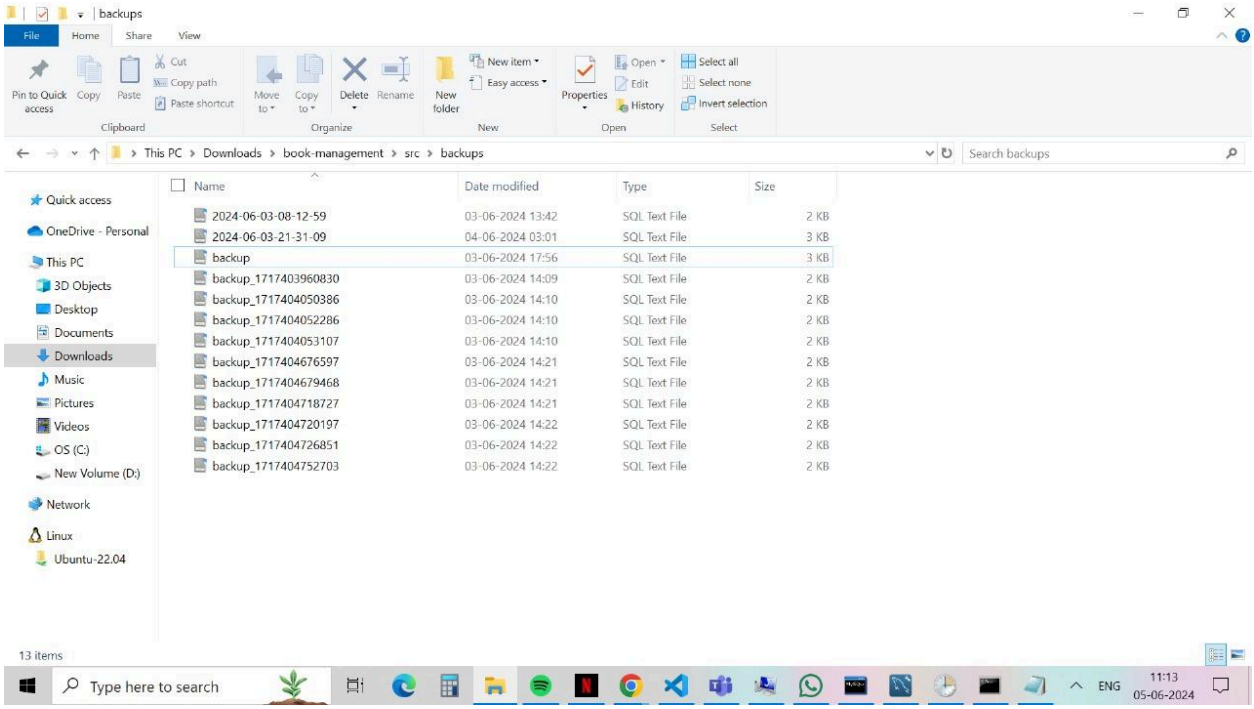
Implementation

```
const fs = require('fs');
const path = require('path');

// Update this path to your actual backup directory
const backupDir = path.join(__dirname, 'backups'); // Change 'backups' to
your backup directory name if different

// Set the retention period in days
const retentionDays = 1 // Change 2 to your desired retention period

const now = Date.now();

fs.readdir(backupDir, (err, files) => {
  if (err) throw err;

  files.forEach(file => {
    const filePath = path.join(backupDir, file);
    fs.stat(filePath, (err, stat) => {
      if (err) throw err;

      const age = (now - stat.mtime.getTime()) / (1000 * 60 * 60 * 24); //
Age in days
      if (age > retentionDays) {
        fs.unlink(filePath, err => {
          if (err) throw err;
          console.log(`Deleted old backup: ${file}`);
        });
      }
    });
  });
});
```

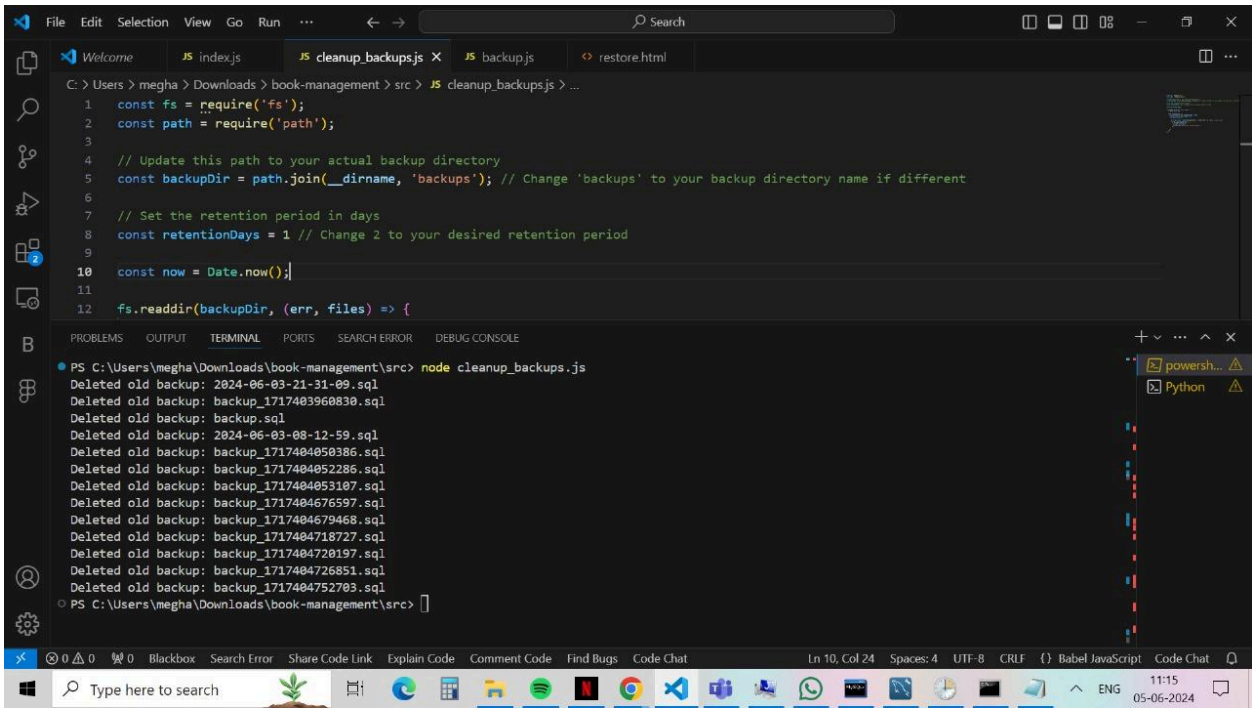
Cleanup_backup:

```
PS C:\Users\megha\Downloads\book-management\src> node cleanup_backups.js
Deleted old backup: backup-1717349899268.sql
PS C:\Users\megha\Downloads\book-management\src>
```

Before cleanup:

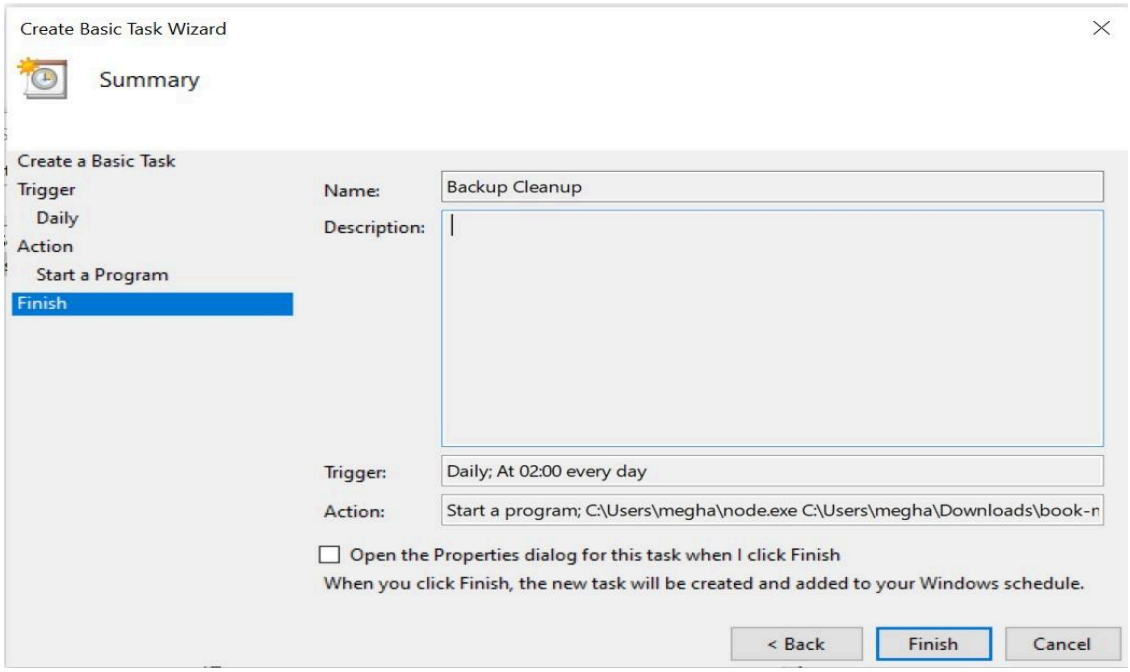


After cleanup



Automated Cleanup:

This is done using task scheduler

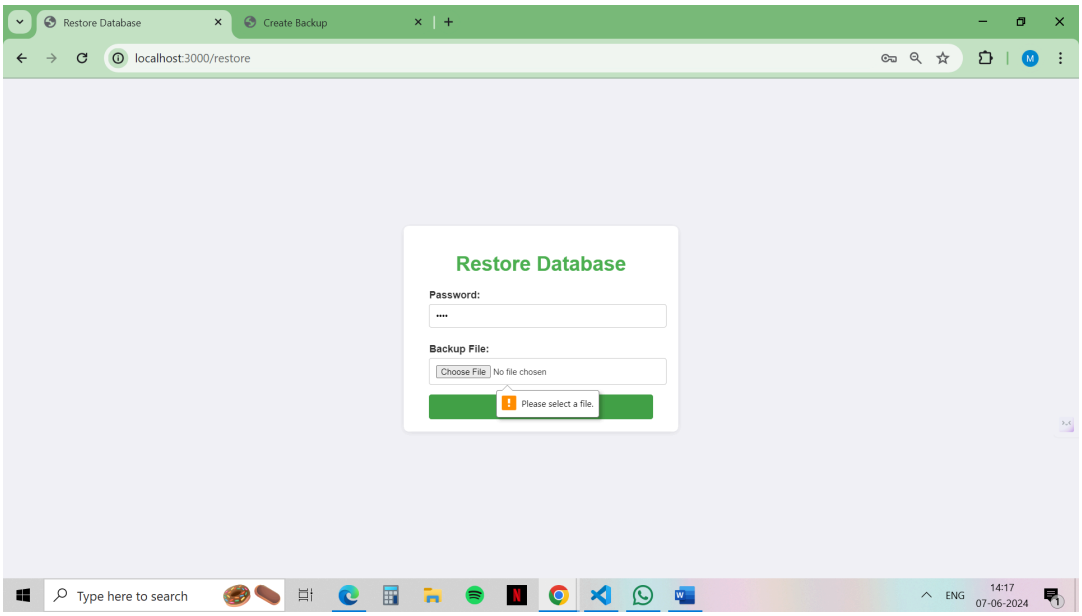


Error Handling and Logging

- Errors during backup and restore processes are logged for troubleshooting.
- Responses include appropriate error messages for failed operations.

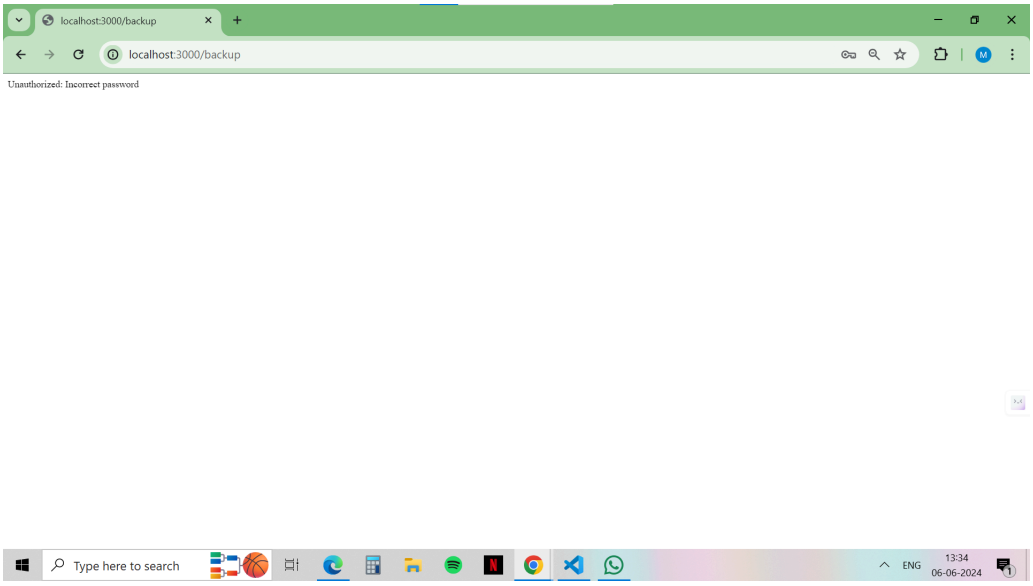
Step 1: Missing Backup File

- I attempted to restore without selecting a backup file.
- Verified that the application responds with an appropriate error message.

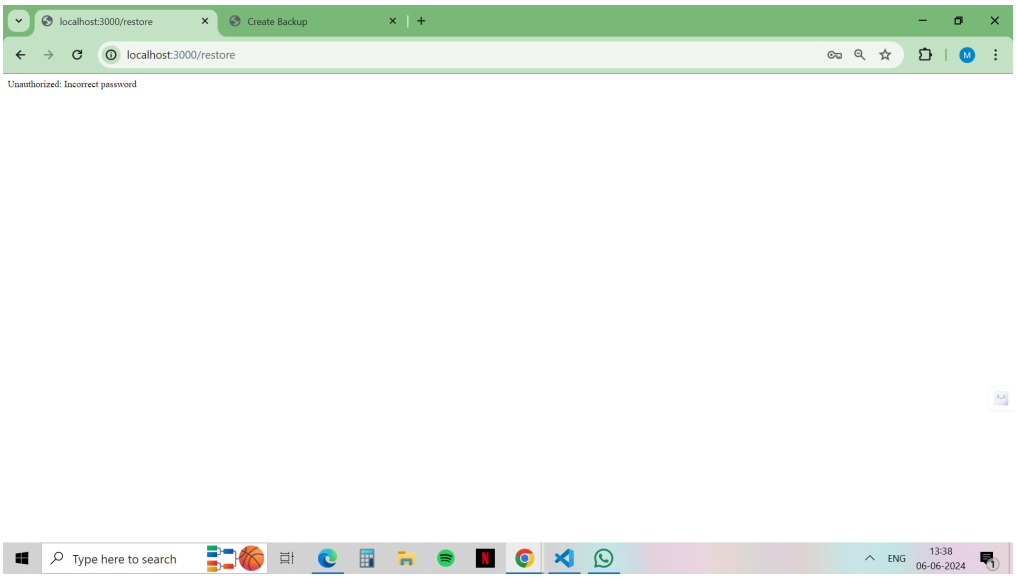


Step 2: Passing invalid password

- Giving incorrect password for backing up

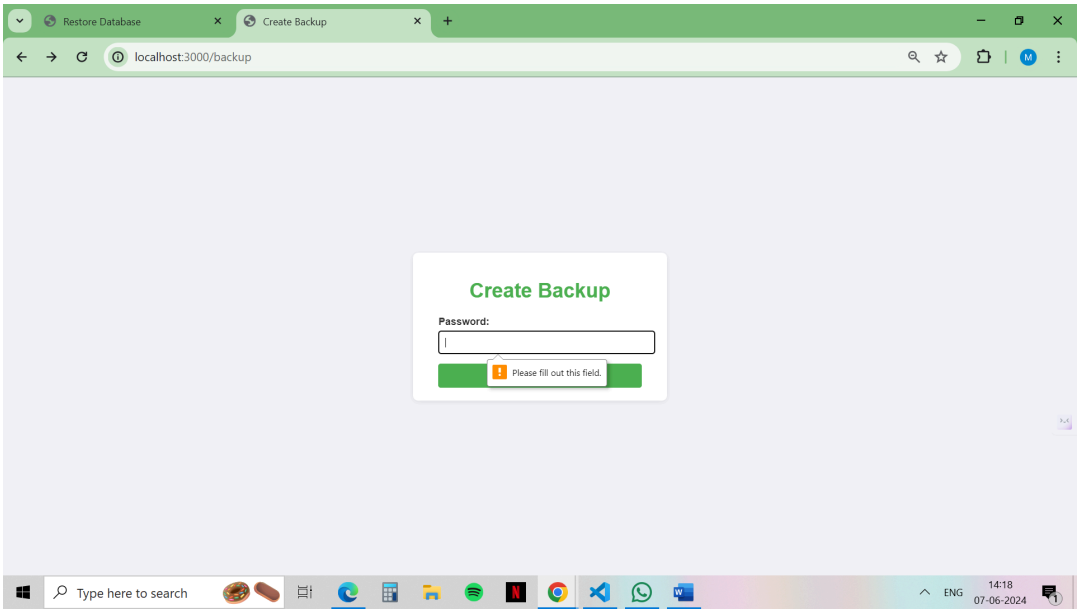


- Giving incorrect password for restoring up

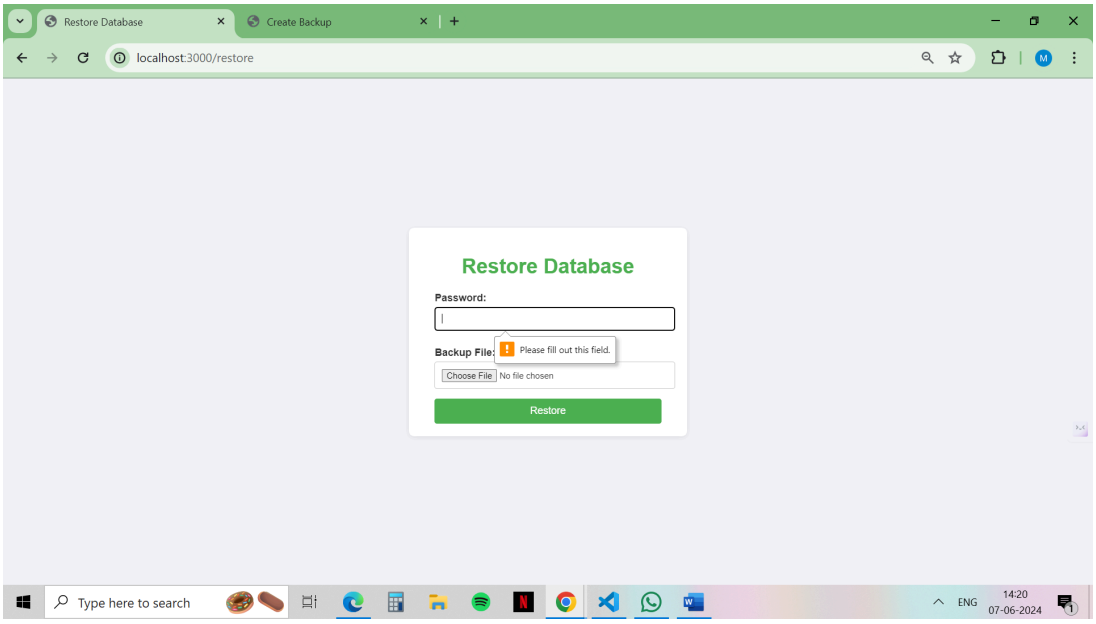


Step 3:Trying to backup or restore without passwords:

- **Backup:**

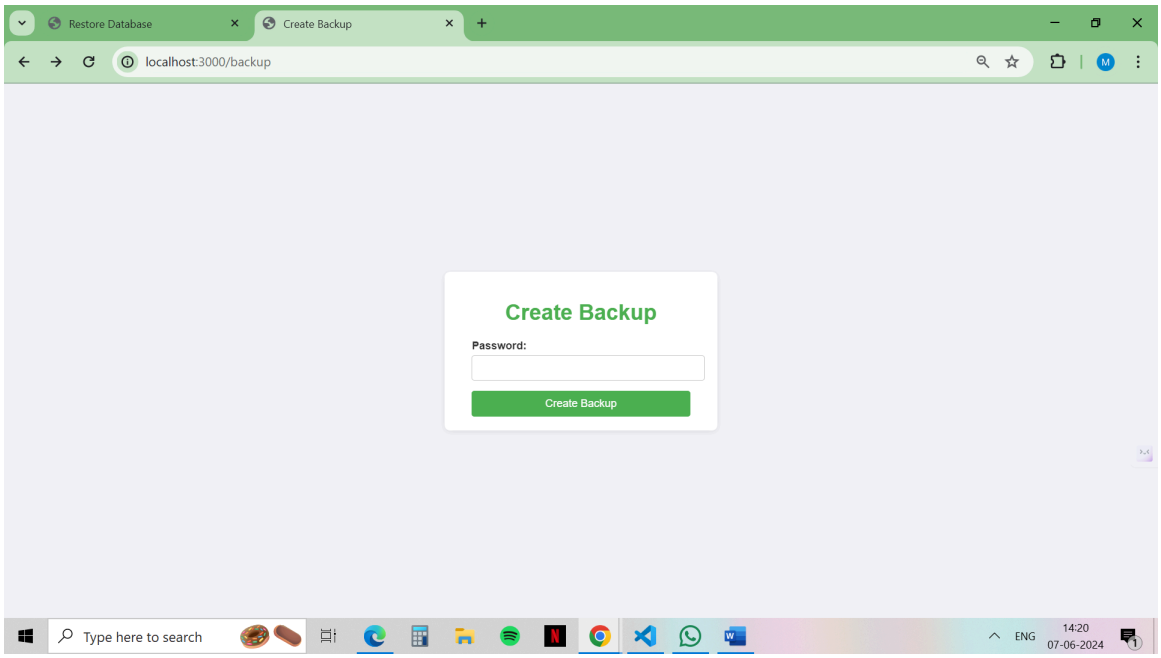


- **Restore:**

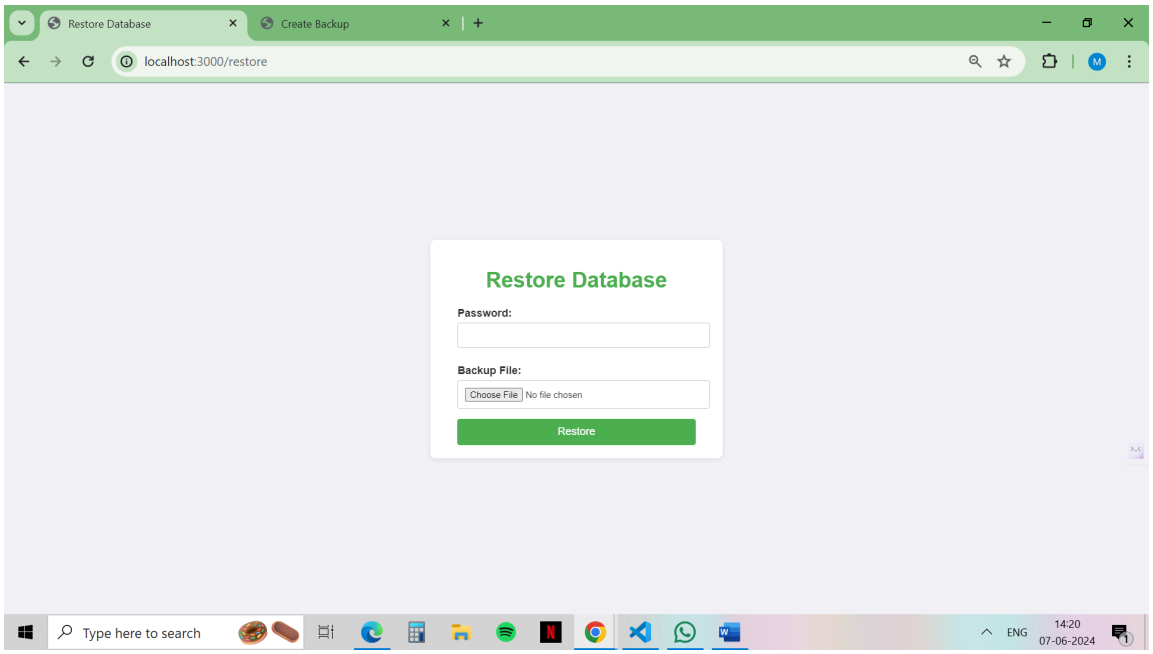


Security Considerations

- Backup and restore operations require password authentication.
- Ensure the '`.env`' file containing database credentials is secure and not exposed.
- For Back up:

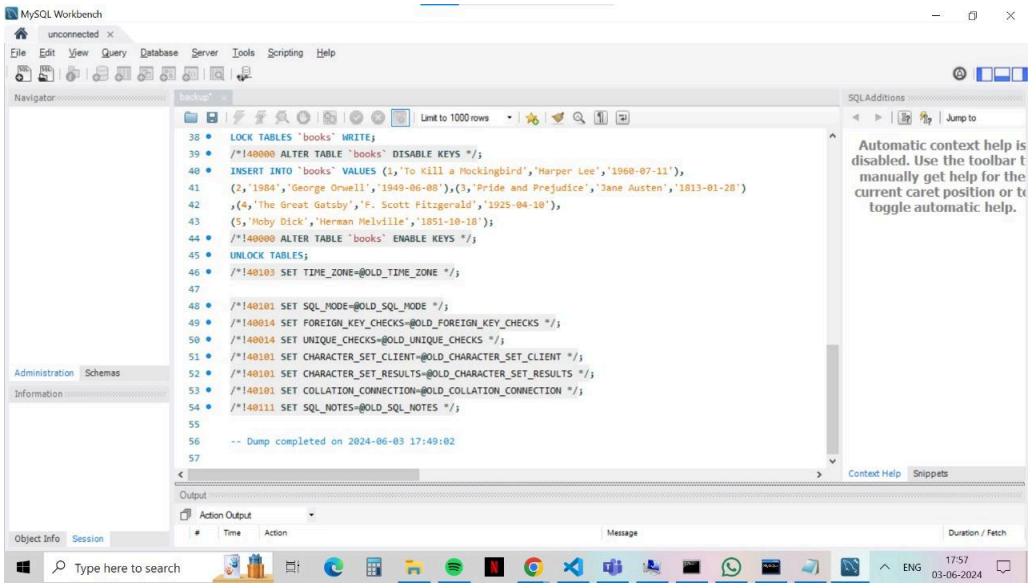


- For Restoring:

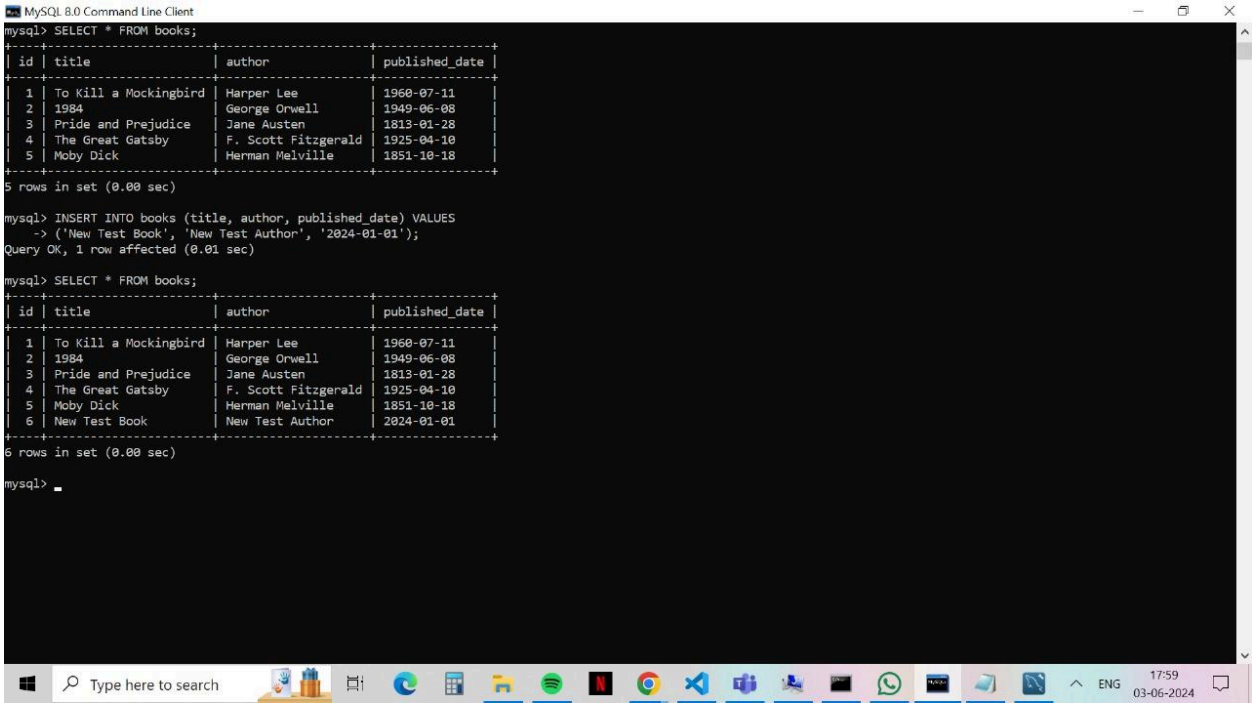


Testing

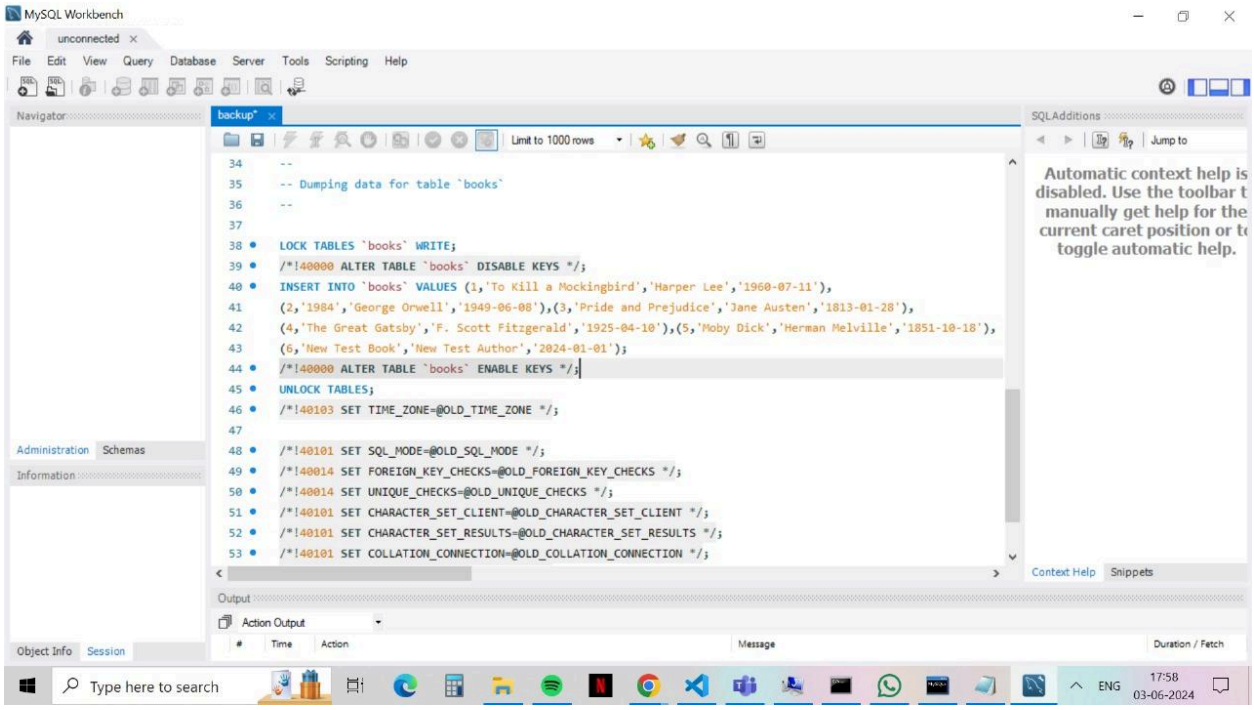
- Created and restored backups in various scenarios to ensure data integrity.
- Verify restored data matches the original dataset.
- Before adding into the database



- After adding into the database



- After Adding a new term:



Deployment and Post-Deployment Checks

- Deployment tasks and post-deployment monitoring are to be planned and executed in the production environment.
- Monitor backup and restore processes to ensure functionality.

Usage Instructions

Backup

1. **Automated Backup**: Configured via task scheduler, no manual intervention is needed.
2. **Manual Backup**: Send a POST request to '`/backup`' with the `password` parameter.

Restore

1. **Web Form**: Access '`/restore`' and upload the backup file with the correct password.
2. **Manual Restore**: Send a POST request to '`/restore`' with the `password` parameter and '`backupFile`' uploaded.

Configuration

- **Environment Variables**:
  - `DB_HOST`: Database host.
  - `DB_USER`: Database user.
  - `DB_PASSWORD`: Database password.
  - `DB_NAME`: Database name.
  - `BACKUP_DIR`: Directory to store backup files.
  - `SESSION_SECRET`: Session secret for authentication.

Example Database Content

```
mysql> SELECT * FROM books;
+----+-----------------------+--------------------+----------------+
| id | title                 | author             | published_date |
+----+-----------------------+--------------------+----------------+
|  1 | To Kill a Mockingbird | Harper Lee         | 1960-07-11     |
|  2 | 1984                  | George Orwell      | 1949-06-08     |
|  3 | Pride and Prejudice   | Jane Austen        | 1813-01-28     |
|  4 | The Great Gatsby      | F. Scott Fitzgerald | 1925-04-10     |
|  5 | Moby Dick             | Herman Melville    | 1851-10-18     |
|  6 | New Test Book         | New Test Author    | 2024-01-01     |
+----+-----------------------+--------------------+----------------+
```

This documentation provides a comprehensive guide to the backup and restore feature, ensuring administrators can maintain data integrity and recover from data loss effectively.