

ACKNOWLEDGEMENT

I would also like to extend my appreciation to all the faculty members of the Computer Science department who have played a vital role in shaping my academic career. Their insightful guidance, valuable feedback, and constructive criticism have been instrumental in enabling me to develop a better understanding of the subject matter and to enhance my problem-solving skills.

I would like to express my gratitude to my classmates who have been a constant source of motivation and support throughout the project. Their valuable inputs and suggestions have helped me to refine my ideas and to improve the quality of my work.

I would like to thank my friends and colleagues for their encouragement and support during the project. Their unwavering belief in my abilities and their willingness to lend a helping hand whenever needed have been crucial in keeping me motivated and focused.

I would like to acknowledge the invaluable support and assistance provided by the staff of the Royal College of Arts, Commerce & Science. Their cooperation and prompt response to my requests have made the project a smooth and hassle-free experience.

I would also like to extend my heartfelt thanks to Prof. **Ritika Lala**, who has been a constant source of inspiration, guidance, and support throughout the project. Her unwavering faith in my abilities and her constructive feedback have been instrumental in shaping my ideas and refining my work. Her invaluable inputs and suggestions have helped me to develop a better understanding of the subject matter and to enhance my skills.

I would also like to express my gratitude to the entire Computer Science department for their support and encouragement during the project. Their expertise and experience have been invaluable in helping me to overcome challenges and to achieve my goals. Their willingness to share their knowledge and to provide me with the resources I needed has been crucial in enabling me to complete this project successfully.

I would like to thank my family for their unwavering support, motivation, and encouragement throughout the project. Their belief in me and their constant encouragement have been my guiding light, and I could not have completed this project without their love and support. Their sacrifices and unwavering faith in me have been the cornerstone of my success, and I am truly grateful to them for everything they have done for me.

Finally, I would like to thank the almighty for blessing me with the strength, perseverance, and determination to complete this project successfully. Without His grace and guidance, this accomplishment would not have been possible.

Salman Sayyed

DECLARATION

I, Mr.Salman Sayyed hereby confirm that the project titled "Design and Implementation of ChemBreak" is the result of my independent work and that I have not used any unauthorized assistance or material in the completion of this project. The project has been undertaken as part of my course curriculum for the Bachelor's Degree in Computer Science at Mumbai University.

Throughout the project, I have personally designed the system architecture, developed the programming logic, and performed the required testing and validation. I have made sure to incorporate all the necessary features and functionalities required to meet the project objectives and to ensure a seamless user experience.

Moreover, I acknowledge that the project may require modifications in the future as per the user's requirements or due to changes in the technological landscape. In this regard, I have incorporated flexibility in the system design to enable any necessary modifications or updates. I am confident that I can make the required changes by modifying the file design or the program code, if necessary, to ensure the system's continued smooth operation.

I would also like to state that I have taken all necessary precautions to ensure that the system design and implementation are secure and comply with industry standards. I have followed best practices in data protection, authentication to ensure that user data remains secure and confidential.

I have developed a deep understanding of the system's underlying technology and functionality throughout the project's development, and I am confident that I can handle any modifications or updates that may be required in the future. I have created the system with a modular and scalable design, enabling me to incorporate changes or enhancements without disrupting the system's overall functioning.

Finally, I affirm that this project report represents my own work, and all sources used have been duly cited and referenced. Any resemblance to other works is purely coincidental.

Index

INDEX

Sr no	Topic	Page no
1.	Introduction	
2.	Proposed System and Advantages	
3.	System Requirements	
4.	Phase Title	
5.	Gantt Chart	
6.	Class Diagram	
7.	Event Table	
8.	Use Case Diagram	
9.	Sequence Diagram	
10.	System Coding	
11.	Snapshots	
12.	Future Enhancements	
13.	Reference and Bibliography	

Preliminary Investigation

Introduction

Chembreak is a modern and innovative app designed to streamline the process of recording and managing damages that occur in chemistry laboratories. The primary objective of Chembreak is to provide a comprehensive and efficient solution to manage breakage reports, record incidents of broken apparatus and charge the respective students in an automated way. The app is an essential tool that eliminates the need for manual documentation of incidents, reduces the workload on lab staff, and ensures a smooth and efficient workflow in the lab.

The need for Chembreak arises from the increasing frequency of damages that occur in chemistry labs and the cumbersome process of managing the records. Chembreak simplifies the task of documentation by providing a user-friendly interface that enables users to create, manage, and share breakage reports in real-time. With Chembreak, lab staff can easily track and monitor breakages, identify the culprits responsible for the damages and charge them accordingly. Moreover, the app generates comprehensive reports that help in analyzing trends and identifying areas of improvement. In summary, Chembreak is an essential tool for any chemistry lab that seeks to enhance its efficiency and productivity by automating the process of managing breakages.

Proposed System and Advantages.

Chembreak is a digital solution that simplifies the process of recording and managing damages that occur in chemistry labs. The app is designed to replace the traditional pen-and-paper method of documenting breakages and incidents. Chembreak allows users to record incidents of broken apparatus and generate breakage reports with just a few clicks. The app also automates the process of charging students for damages, eliminating the need for manual calculations and record-keeping. With Chembreak, lab staff can easily monitor and manage breakages in real-time, ensuring a smooth workflow and efficient operation of the lab.

Advantages of Chembreak:

- 1) Improved Efficiency: Chembreak streamlines the process of recording and managing breakages, reducing the workload on lab staff and enhancing the overall efficiency of the lab.
- 2) Real-time Monitoring: With Chembreak, lab staff can monitor breakages in real-time, ensuring that damages are quickly identified and addressed.
- 3) Automated Charging: Chembreak automates the process of charging students for damages, eliminating the need for manual calculations and record-keeping.
- 4) Comprehensive Reporting: The app generates comprehensive reports that help in analyzing trends and identifying areas of improvement.
- 5) User-friendly Interface: Chembreak has a user-friendly interface that enables users to create, manage, and share breakage reports with ease.
- 6) Cost-effective: Chembreak is a cost-effective solution that eliminates the need for manual documentation and record-keeping, reducing the overall operational costs of the lab.

In summary, Chembreak is a modern and innovative solution that enhances the efficiency and productivity of chemistry labs by automating the process of managing breakages. With its real-time monitoring, automated charging, and comprehensive reporting, Chembreak is a valuable tool that simplifies the task of documentation and enhances the overall performance of the lab.

System Requirements.

Hardware:

- Processor: 1.5 GHz or higher
- RAM: 2 GB or higher
- Storage: 50 MB or more free disk space

Software:

- Operating System: Any platform that supports Python3, such as Windows, Linux, or macOS
- Python: Version 3.6 or higher
- Flask: Version 2.0.2 or higher
- SQLAlchemy: Version 1.4.0 or higher
- SQLite: Version 3.36.0 or higher

Other Required Libraries:

- Jinja2: Version 3.0.1 or higher
- WTForms: Version 3.0.0 or higher
- Flask-WTF: Version 1.0.0 or higher
- Werkzeug: Version 2.0.2 or higher

Cloud Hosting:

- Any cloud hosting platform that supports Python applications and SQLite databases, such as Amazon Web Services, Google Cloud Platform, or Microsoft Azure.

TYBSC Computer Science Semester 6
2022 -2023

Phase Title	Expected Date of Completion	Actual Time of Completion with Guide's Signature	Remarks
I. Preliminary Investigation	}		
(i) Organizational Overview			
(ii) Present System and its advantages		10/12/2022	
(iii) System Requirements			
(iv) Feasibility Study	}		
(v) Fact Finding Methods			
(vi) Phase Title		20/12/2022	
(vii) Gantt Chart			
II. System Analysis	}		
(i) Event Table			
(ii) Use Case Diagram		09/01/2023	
(iii) ERD			
(iv) Class Diagram	}		
III. System Design			
(i) Sequence Diagram	}		
IV. System Coding			
(i) System Coding		06/02/2023	
(ii) Form Layouts			
(iii) Report Layouts	}		
V. Future Enhancements			
VI. Reference and Bibliography	}	20/03/2023	

System Analysis

Event Table

Event Id	Event Name	Description
1	User Login	A user has logged into the application.
2	Apparatus Created	A new apparatus has been added to the system
3	Apparatus Edited	An existing apparatus has been edited in the system
4	Apparatus Deleted	An existing apparatus has been deleted from the system
5	Breakage Added	A new breakage has been recorded for a specific apparatus.
6	Report Printed	A specific report has been printed by a user of the system

System Design

System Coding

Wsgi.py

```
from app import create_app
from config import Config

app = create_app(config=Config)

if __name__ == '__main__':
    app.run()
```

app/__init__.py

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager

db = SQLAlchemy()

def create_app(config=None):
    app = Flask(__name__)
    if config:
        app.config.from_object(config)

    db.init_app(app)
    login_manager = LoginManager()
    login_manager.login_view = 'auth.login'
    login_manager.init_app(app)
```

```

    with app.app_context():
        from app.models import User, Breakage, Student, Apparatus,
Record, Bank
        db.create_all()

from app.models import User

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

# register blueprints
from .auth import auth as auth_blueprint
app.register_blueprint(auth_blueprint)

from .routes import main as main_blueprint
app.register_blueprint(main_blueprint)

return app

```

app/models.py

```

from app import db
from flask_login import UserMixin
from datetime import datetime

```



```

class User(db.Model, UserMixin):
    """
    User Model for authentication
    parms:
        id: user id
        username: user name
        password: user password
    """
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(15), unique=True, nullable=False)
    password = db.Column(db.String(80), nullable=False)

    def __repr__(self):
        return f"User('{self.username}')"

    def check_password(self, password):
        return self.password == password


class Breakage(db.Model):
    """
    Breakage Model
    parms:
        id: breakage id
        date: date of breakage
        item_id: item that broke (apparatus id)
        quantity: quantity of item that broke

```

```

        student_unique_id: student unique id
    """

    __tablename__ = "breakage"

    id = db.Column(db.Integer, primary_key=True)
    date = db.Column(db.DateTime, nullable=False,
default=datetime.utcnow)
    item_id = db.Column(db.Integer, db.ForeignKey(
        "apparatus.id"), nullable=False)
    quantity = db.Column(db.Integer, nullable=False)
    student_unique_id = db.Column(db.Integer, db.ForeignKey(
        "student.unique_id"), nullable=False)
    total_ammount = db.Column(db.Integer, nullable=False, default=0)
    student = db.relationship(
        'Student', backref='breakage', lazy=True)
    apparatus = db.relationship(
        'Apparatus', backref='breakage', lazy=True)

    def __repr__(self) -> str:
        return f"Breakage('{self.date}', '{self.item_id}',
'{self.quantity}', '{self.student_unique_id}',
'{self.total_ammount}')"

    def __init__(self, item_id, quantity, student_unique_id,
total_ammount, date):
        self.item_id = item_id
        self.quantity = quantity
        self.student_unique_id = student_unique_id

```

```

        self.total_ammount = total_ammount
        self.date = date

def get_dd_mm_yyyy(self):
    utc_datetime = datetime.datetime.strptime(
        self.date, "%Y-%m-%d %H:%M:%S.%f")
    date_str = utc_datetime.strftime("%d-%m-%Y")
    return date_str

class Student(db.Model):
    """
    Student Model
    parms:
        id: student id
        unique_id: student unique id
        roll_no: student roll number
        class: student class (fy, sy, ty)
        section: department section (Chemistry)
    """

    __tablename__ = "student"

    id = db.Column(db.Integer, primary_key=True)
    unique_id = db.Column(db.String(100), unique=True, nullable=False)
    roll_no = db.Column(db.String(10), nullable=False)
    class_ = db.Column(db.String(10), nullable=False)

```

```

        section = db.Column(db.String(10), nullable=False,
default="Chemistry")

        total_amount = db.relationship('Bank', backref='student',
lazy=True)

    def __repr__(self) -> str:
        return f"Student('{self.unique_id}', '{self.roll_no}',
'{self.class_}', '{self.section}')"

    def __init__(self, unique_id, roll_no, class_, section):
        self.unique_id = unique_id
        self.roll_no = roll_no
        self.class_ = class_
        self.section = section

class Apparatus(db.Model):
    """
    Apparatus Model
    parms:
        id: apparatus id
        name: apparatus name
        size: apparatus size
        price: apparatus price
    """

    __tablename__ = "apparatus"

    id = db.Column(db.Integer, primary_key=True)

```

```

name = db.Column(db.String(100), nullable=False)
size = db.Column(db.String(100), nullable=False)
price = db.Column(db.Integer, nullable=False)
apparatus = db.relationship(
    'Breakage', backref='breakage', lazy=True, cascade='all,
delete-orphan')

def __repr__(self) -> str:
    return f"Apparatus('{self.name}', '{self.size}',
'{self.price}')"

def __init__(self, name, size, price):
    self.name = name
    self.size = size
    self.price = price

class Record(db.Model):
    """
    Record Model
    parms:
        id: record id
        date: date of record
        message: message of record
        student_unique_id: student unique id
    """

    __tablename__ = "record"

```

```

id = db.Column(db.Integer, primary_key=True)
date = db.Column(db.DateTime, default=datetime.utcnow)
message = db.Column(db.String(100), nullable=False)
student_unique_id = db.Column(db.String(100), db.ForeignKey(
    "student.unique_id"), nullable=False)

def __repr__(self) -> str:
    return f"Record('{self.date}', '{self.message}',
'{{self.student_unique_id}}')"

def __init__(self, message, student_unique_id):
    self.message = message
    self.student_unique_id = student_unique_id

class Bank(db.Model):
    """
    Bank Model
    parms:
        id: bank id
        amount: amount of money
        student_unique_id: student id
    """

    __tablename__ = "bank"

    id = db.Column(db.Integer, primary_key=True)
    amount = db.Column(db.Integer, nullable=False, default=0)

```

```

unique_student_id = db.Column(db.String(100), db.ForeignKey(
    "student.unique_id"), nullable=False)

def __repr__(self) -> str:
    return f"Bank('{self.amount}', '{self.unique_student_id}')"

def __init__(self, amount, unique_student_id):
    self.amount = amount
    self.unique_student_id = unique_student_id

```

app/routes.py

```

from flask import Blueprint, send_file, send_from_directory
from flask import render_template, redirect, url_for, request, flash,
abort
from flask_login import login_required
from app.models import Apparatus, Breakage, Bank, Student, Record,
User
from app.view_classes import ViewRecord, CollectMoney
from app import db
import datetime
import pytz
from sqlalchemy import and_

main = Blueprint('main', __name__)

@main.route('/')
def index():
    return redirect(url_for('auth.login'))

```

```

@main.route('/admin/create', methods=['GET', 'POST'])
def create_admin():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User(username=username, password=password)
        db.session.add(user)
        db.session.commit()
        return redirect(url_for('auth.login'))
    return render_template('create_account.html')

```

```

@main.route('/home')
@login_required
def home():
    """
    List all modules.
    """
    return render_template('home.html')

```

```

@main.route('/home/breakage')
@login_required
def breakage():
    """
    Add records

```



```

"""

apparatus_list = Apparatus.query.all()
display_name_list = [apparatus.name + " " + apparatus.size
                      for apparatus in apparatus_list]

return render_template('breakage.html',
id_dname=zip(apparatus_list, display_name_list))

@main.route('/home/breakage', methods=['POST'])
@login_required
def post_breakage():
    """
    Add Breakage records
    """
    if request.method == 'POST':
        item = request.form['apparatus_id']
        quantity = request.form['quantity']
        roll_no = request.form['roll_no']
        s_class = request.form['class']
        section = request.form['section']
        date_ = request.form['date']
        total_ammount = int(quantity) *
int(Apparatus.query.get(item).price)

        print(date_)
        # convert date to datetime
        date_obj = datetime.datetime.strptime(
            date_, "%Y-%m-%d").astimezone(pytz.utc)
        print(date_obj)

```

```

print(datetime.datetime.utcnow())

# check if student exists with roll_no and class
student = Student.query.filter_by(roll_no=roll_no,
                                   class_=s_class).first()

# if student does not exist, create a new student
if student is None:
    student = create_student(roll_no, s_class, section)

breakage = Breakage(item_id=item,
                    quantity=quantity,
student_unique_id=student.unique_id, total_ammount=total_ammount,
date=date_obj)

record_message = student.unique_id + " " +
str(breakage.quantity) + " " + Apparatus.query.get(
    breakage.item_id).name + " " +
Apparatus.query.get(breakage.item_id).size

create_record(record_message, student.unique_id)

create_bank(total_ammount, student.id, student.unique_id)

db.session.add(breakage)
db.session.commit()
return redirect(url_for('main.breakage'))

```

```

def create_student(rollno, s_class, section):
    """
    Create a new student.
    """
    year = str(datetime.datetime.now().year)
    unique_id = str(s_class) + str(rollno) + "Y" + str(year[2:])

    student = Student(unique_id=unique_id, roll_no=rollno,
                      class_=s_class, section=section)

    db.session.add(student)
    db.session.commit()
    return student


def create_record(message, student_id):
    """
    Create a new record.
    """
    record = Record(
        message=message, student_unique_id=student_id)
    db.session.add(record)
    db.session.commit()
    return record


def create_bank(amount, student_id, unique_id):
    """

```

Create a new bank record.

```
"""
```

```
# check if unique_id exists
```

```
bank = Bank.query.filter_by(unique_student_id=unique_id).first()
```

```
if bank is not None:
```

```
    bank.amount = int(bank.amount) + int(amount)
```

```
    db.session.commit()
```

```
    return bank
```

```
bank = Bank(amount=amount,
```

```
            unique_student_id=unique_id)
```

```
db.session.add(bank)
```

```
db.session.commit()
```

```
return bank
```

```
@main.route('/home/report')
```

```
@login_required
```

```
def report():
```

```
    """
```

```
Print report from a selected range of dates from a calendar.
```

```
Possilbe inputs :
```

```
    class (fy, sy, ty)
```

```
        date (by default generate a months data.)
```

```
    """
```

```
return render_template('print_report.html')
```

```

@main.route("/home/records")
@login_required
def records():
    """
    List all records.
    Three subsections :
        1. Fy
        2. Sy
        3. Ty
    """
    return render_template('records.html')

```

```

@main.route('/home/help')
@login_required
def help():
    """
    Help page.
    """
    return render_template('help.html')

```

```

@main.route('/home/apparatus')
@login_required
def apparatus():
    """
    Add / Update / Delete apparatus.
    """

```

```
        return render_template('apparatus.html',
                                apparatuses=Apparatus.query.all())
```

```
@main.route("/home/apparatus", methods=['POST'])
```

```
@login_required
```

```
def new_apparatus():
```

```
    """
```

```
    Add new apparatus.
```

```
    """
```

```
    if request.method == 'POST':
```

```
        name = request.form['name']
```

```
        size = request.form['size']
```

```
        price = request.form['price']
```

```
        apparatus = Apparatus(name=name, size=size, price=price)
```

```
        db.session.add(apparatus)
```

```
        db.session.commit()
```

```
    return redirect(url_for('main.apparatus'))
```

```
@main.route("/home/apparatus/<int:id>", methods=['GET', 'POST'])
```

```
@login_required
```

```
def update_apparatus(id):
```

```
    """
```

```
    Update apparatus.
```

```
    """
```

```
    if request.method == 'GET':
```

```
        return render_template('update_apparatus.html',
                                apparatus=Apparatus.query.get(id))
```

```
    if request.method == 'POST':
        name = request.form['name']
        size = request.form['size']
        price = request.form['price']
        apparatus = Apparatus.query.get(id)
        apparatus.name = name
        apparatus.size = size
        apparatus.price = price
        db.session.commit()
    return redirect(url_for('main.apparatus'))
```

```
@main.route("/home/apparatus/<int:id>/delete", methods=['POST'])
```

```
@login_required
```

```
def delete_apparatus(id):
    """
    Delete apparatus.
    """
    if request.method == 'POST':
        apparatus = Apparatus.query.get(id)
        db.session.delete(apparatus)
        db.session.commit()
    return redirect(url_for('main.apparatus'))
```

```

@main.route("/home/records/<string:class_name>")
@login_required
def class_records(class_name):
    """
    List all records from a specific class.
    Three subsections :
        1. Fy
        2. Sy
        3. Ty
    """
    valid_classes = ['fy', 'sy', 'ty']
    if class_name.lower() not in valid_classes:
        abort(404)

    class_students =
Student.query.filter_by(class_=class_name.lower()).all()
    class_records = []
    for student in class_students:
        utc_date_str = str(Breakage.query.filter_by(
            student_unique_id=student.unique_id).first().date)
        utc_datetime = datetime.datetime.strptime(
            utc_date_str, "%Y-%m-%d %H:%M:%S.%f")
        date_str = utc_datetime.strftime("%d-%m-%Y")

        # only getting a single student record
        item_id = Breakage.query.filter_by(
            student_unique_id=student.unique_id).first().item_id

```



```

        new_record = ViewRecord(
            date=date_str,
            roll_no=student.roll_no,
            class_=student.class_,
            section=student.section,
            apparatus=Apparatus.query.get(item_id).name,
            quantity=Breakage.query.filter_by(
                student_unique_id=student.unique_id).first().quantity,
            price=Apparatus.query.get(item_id).price,
            total_ammount=Breakage.query.filter_by(
student_unique_id=student.unique_id).first().total_ammount,
            )

        class_records.append(new_record)

    sorted_records = sorted(class_records, key=lambda x: x.roll_no)
    return render_template('class_records.html',
records=sorted_records, class_name=class_name.upper())

@main.route("/test/home/records/<string:class_name>")
@login_required
def test_class_records(class_name):
    valid_classes = ['fy', 'sy', 'ty']
    if class_name.lower() not in valid_classes:
        abort(404)

```

```

class_students =
Student.query.filter_by(class_=class_name.lower()).all()

view_records = []

for student in class_students:
    breakages = Breakage.query.join(Apparatus).filter(
        and_(Breakage.student_unique_id ==
student.unique_id)).all()

    # loop through breakages and create view records
    for breakage in breakages:
        view_record = ViewRecord(
            date=breakage.date.strftime('%d/%m/%Y'),
            roll_no=student.roll_no,
            class_=student.class_,
            section=student.section,
            apparatus=breakage.apparatus.name + " " +
breakage.apparatus.size,
            quantity=breakage.quantity,
            price=breakage.apparatus.price,
            total_ammount=breakage.quantity *
breakage.apparatus.price
        )

        view_records.append(view_record)

sorted_records = sorted(view_records, key=lambda x: x.roll_no)
return render_template('class_records.html',
records=sorted_records, class_name=class_name.upper())

```

```

@main.route("/home/records/getMoney/<string:class_name>")
@login_required
def getMoney(class_name):
    valid_classes = ['fy', 'sy', 'ty']
    if class_name.lower() not in valid_classes:
        abort(404)

    class_students =
Student.query.filter_by(class_=class_name.lower()).all()
    class_records = []

    for student in class_students:
        bank = Bank.query.filter_by(
            unique_student_id=student.unique_id).first().amount
        collect_money = CollectMoney(
            rollno=student.roll_no,
            total_cash=bank,
        )
        class_records.append(collect_money)

    sorted_records = sorted(class_records, key=lambda x: x.rollno)
    return render_template('collect_money.html',
collect_money_list=sorted_records, class_name=class_name.upper())

@main.route("/home/reset_and_bakup")
@login_required
def reset_and_bakup():

```

```

        return render_template('reset_and_backup.html')

@main.route('/download_backup', methods=['POST'])
@login_required
def download_backup():
    return send_file('../app.db', as_attachment=True)

@main.route('/empty_user_table', methods=['POST'])
@login_required
def empty_user_table():
    db.session.query(User).delete()
    db.session.commit()

    return "User table has been emptied"

@main.route('/empty_breakage_table', methods=['POST'])
@login_required
def empty_breakage_table():
    db.session.query(Breakage).delete()
    db.session.commit()
    flash("Breakage table has been emptied", "success")
    return render_template('reset_and_backup.html')

@main.route('/empty_student_table', methods=['POST'])

```

```

@login_required
def empty_student_table():
    db.session.query(Student).delete()
    db.session.commit()
    flash("Student table has been emptied", "success")
    return render_template('reset_and_backup.html')


@main.route('/empty_apparatus_table', methods=['POST'])
@login_required
def empty_apparatus_table():
    db.session.query(Apparatus).delete()
    db.session.commit()
    flash("Apparatus table has been emptied", "success")
    return render_template('reset_and_backup.html')


@main.route('/empty_records_table', methods=['POST'])
@login_required
def empty_records_table():
    db.session.query(Record).delete()
    db.session.commit()
    flash("Records table has been emptied", "success")
    return render_template('reset_and_backup.html')


@main.route('/empty_bank_table', methods=['POST'])
@login_required

```

```
def empty_bank_table():
    db.session.query(Bank).delete()
    db.session.commit()
    flash("Bank table has been emptied", "success")
    return render_template('reset_and_backup.html')

@main.route('/complete_reset', methods=['POST'])
@login_required
def complete_reset():
    db.session.query(Breakage).delete()
    db.session.query(Student).delete()
    db.session.query(Record).delete()
    db.session.query(Bank).delete()
    db.session.commit()
    flash("All tables have been emptied", "success")
    return render_template('reset_and_backup.html')
```

Test Cases

Test Case	Steps	Expected Result	Actual Result
User Login	1. Enter valid username and password. 2. Click on "Login" button	User is successfully logged in and redirected to home page	User is successfully logged in and redirected to home page
Create Apparatus	1. Navigate to "Apparatus" page 2. Enter valid apparatus details 3. Click on "Submit" button	Apparatus is successfully created and added to the system	Apparatus is successfully created and added to the system
Edit Apparatus	1. Navigate to "Edit Apparatus" page. 2. Select an existing apparatus. 3. Modify the apparatus details. 4. Click on "Submit" button	Apparatus details are successfully updated in the system	Apparatus details are successfully updated in the system
Delete Apparatus	1. Navigate to "Apparatus" page 2. Select an existing apparatus 3. Confirm the deletion	Apparatus is successfully removed from the system	Apparatus is successfully removed from the system
Add Breakage	1. Navigate to "Add Breakage" page 2. Select an existing apparatus 3. Enter valid breakage details 4. Click on "Submit" button	Breakage is successfully recorded for the selected apparatus	Breakage is successfully recorded for the selected apparatus
Generate Report	1. Navigate to " Report" page. 2. Select a date range for the report. 3. Click on "Submit" button.	Report is generated and displays all breakages recorded within the selected date range	Report is generated and displays all breakages recorded within the selected date range

ChemBreak

127.0.0.1:5000/chem/home/records/fy


CHEM-BREAK Breakage Print Report Apparatus Reset & Backup Log Out Help

Royal College Chemistry Dept

Breakage Charges

FY

Click to print



Date	Roll No	Class	Apparatus	Quantity	Price	Total
13/03/2023	1	fy	Test Tube 12 ml	1	25 ₹	25 ₹
13/03/2023	1	fy	Flask 120ml	1	60 ₹	60 ₹
13/03/2023	10	fy	Flask 120ml	1	60 ₹	60 ₹
13/03/2023	15	fy	Beaker 50ml	1	30 ₹	30 ₹

ChemBreak

127.0.0.1:5000/chem/home/records/fy

CHEM-BREAK Breakage Print Report Apparatus Reset & Backup Log Out Help


Royal College Chemistry Dept

Breakage Charges

Total Amount

FY

Click to Print



Roll No	Total Cash
1	85 ₹
10	60 ₹
15	30 ₹

Future Enhancement

- 1) Integration with Payment Gateway: In the current version, the application records the breakage details and calculates the corresponding damage charges for the selected apparatus. However, the application can be enhanced to integrate with a payment gateway so that the damage charges can be collected directly from the students.
- 2) Notification System: The application can be enhanced to include a notification system that alerts the lab in-charge or the faculty whenever a breakage occurs. This would help ensure that the breakage is attended to promptly, and also enable the lab staff to take preventive measures to reduce the incidence of breakages.
- 3) Barcode Scanning: To improve the accuracy of the inventory management system, the application can be enhanced to include barcode scanning functionality. Lab staff can simply scan the barcode on the apparatus to update its status (e.g. available, in-use, damaged), which would reduce the need for manual data entry and minimize errors.
- 4) Data Analytics and Visualization: The application can be enhanced to include data analytics and visualization features, which would allow lab staff and faculty to gain insights into the patterns and trends of breakages. For example, they can analyze the frequency of breakages for different apparatus and identify any apparatus that are more prone to breakage than others. This information can be used to optimize the lab setup and minimize breakages.
- 5) Mobile Application: A mobile application can be developed as an extension of the current web-based application, which would enable lab staff and faculty to access the application on-the-go. They can use their mobile devices to scan barcodes, record breakages, and generate reports, which would enhance the overall efficiency and convenience of the application.

Reference and Bibliography

Reference:

- Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
[<https://www.oreilly.com/library/view/flask-web-development/9781491991725/>]
- Bouchenak, S., & Defude, B. (2018). Web development using Flask, a Python microframework. Journal of Computing Sciences in Colleges, 33(3), 31-37.
[<https://dl.acm.org/doi/abs/10.5555/3276687.3276694>]
- Shah, S., & Gupta, A. (2017). Rapid Web Application Development using Flask. International Journal of Computer Applications, 162(2), 1-6.
[https://www.researchgate.net/publication/318030561_Rapid_Web_Application_Development_using_Flask]

Bibliography:

- [1] Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
- [2] Bouchenak, S., & Defude, B. (2018). Web development using Flask, a Python microframework. Journal of Computing Sciences in Colleges, 33(3), 31-37.
- [3] Shah, S., & Gupta, A. (2017). Rapid Web Application Development using Flask. International Journal of Computer Applications, 162(2), 1-6.