Project Report

On

**Genesis**

Blockchain Application

Using React Js

Designed and Developed

By

**Mr. Naqi Ansari**

TYBSc CS 26

2022-2023

And Guided By

Prof Ritika Lala

Submitted in partial fulfillment of academic project

[Bachelor of Computer Science]

University of Mumbai



**DEPARTMENT OF COMPUTER SCIENCE**

Class: TYBSc                                                                                                        Roll. No.  26

                                                                                                                               Seat No.\_\_\_\_\_\_\_

**Certificate**

Certified that **Naqi Ansari** of T.Y.BSc Semester-VI has

successfully completed the project as prescribed by the University of

Mumbai on **Genesis** as partial fulfillment of requirement for completing

Bachelor’s Degree in Computer Science during the academic year 2022-2023.

Signature of Project Guide

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature of Examiner

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

                   H.O.D

Dept. of Computer Science

**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. **Anushka** , 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.

**Naqi Ansari**

**DECLARATION**

I, Mr.Naqi Ansari hereby confirm that the project titled "Design and Implementation of Genesis" 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**

The Genesis Crowd Funding Application is a decentralized crowdfunding platform built on blockchain technology that provides individuals and organizations with a transparent and secure way to raise funds for their projects, campaigns, or businesses without the need for intermediaries such as banks, payment gateways, or traditional crowdfunding platforms. The application uses Solidity, React JS, Hardhat, and Metamask on the Goerli network to ensure the highest level of security and transparency in the fundraising process.

Objectives:

The main objective of the Genesis Crowd Funding Application is to provide a decentralized crowdfunding platform that empowers individuals and organizations to raise funds for their projects, campaigns, or businesses without the need for intermediaries. The application aims to eliminate the risks of fraud and manipulation by third parties, increase transparency and accountability, provide a higher level of security for the funds raised, and reduce fees significantly. The application also aims to provide global access to fundraising campaigns and simplify the process of starting and contributing to a crowdfunding campaign.

Need for the App:

The need for the Genesis Crowd Funding Application arises from the limitations of traditional crowdfunding platforms that rely on intermediaries such as banks and payment gateways, which can be costly, slow, and prone to fraud and manipulation. Additionally, traditional crowdfunding platforms often have limited access to global participation, making it challenging for individuals and organizations to reach a broader audience. The Genesis Crowd Funding Application addresses these limitations by providing a decentralized, transparent, and secure platform that is accessible to anyone with an internet connection, enabling global participation in fundraising campaigns.

**Proposed System and Advantages**.

The proposed system is a decentralized crowdfunding platform that uses blockchain technology to enable individuals and organizations to raise funds for their projects, campaigns, or businesses without the need for intermediaries such as banks and payment gateways. The platform will allow users to create and manage crowdfunding campaigns using Ether (ETH) on the blockchain network. The system will use smart contracts to enforce the rules of the campaign and release the funds raised to the campaign owner when the fundraising goal is reached before the deadline. The system will provide transparency, security, and lower fees than traditional crowdfunding platforms.

Advantages over Traditional Crowdfunding with Centralized Banking and Intermediary Fees:

1) Decentralization: The proposed system is decentralized, which means it is not controlled by any central authority or intermediary. This eliminates the risk of fraud or manipulation by third parties.

2) Transparency: The transactions and activities on the blockchain are transparent and can be viewed by anyone, providing greater accountability and trust in the crowdfunding process.

3) Security: The blockchain technology is secure and tamper-proof, providing a higher level of security for the funds raised.

4) Lower Fees: Traditional crowdfunding platforms charge high fees for their services. With this blockchain-based crowdfunding platform, there are no intermediaries, so fees are significantly lower.

5) Global Access: The proposed system is accessible to anyone with an internet connection, enabling global participation in fundraising campaigns.

6) Faster Transaction Processing: The proposed system uses blockchain technology, which can process transactions faster than traditional crowdfunding platforms, reducing the time it takes to receive funds.

7) More Control for Campaign Owners: The proposed system gives more control to campaign owners by enabling them to manage their campaigns directly without the need for intermediaries. This allows for greater customization and flexibility in the fundraising process.

**System Requirements**.

Hardware Requirements:

* A computer or laptop with a minimum of 4 GB RAM and a dual-core processor.
* An internet connection with a minimum speed of 5 Mbps.

Software Requirements:

* Operating System: Windows, macOS, or Linux.
* Node.js: version 12 or later.
* NPM: version 6 or later.
* Metamask wallet: version 10 or later.
* Hardhat: version 2.0 or later.
* Alchemy API key.

Note: The above software requirements are necessary for both the development and deployment of the Genesis Crowdfunding Application.

Deployment Requirements:

* Goerli network: The Genesis Crowdfunding Application is deployed on the Goerli network, so you will need to connect to this network using your Metamask wallet.
* Ethereum (ETH): You will need some Ethereum (ETH) on the Goerli network to interact with the smart contract.

TYBSC Computer Science Semester 6

2022 -2023

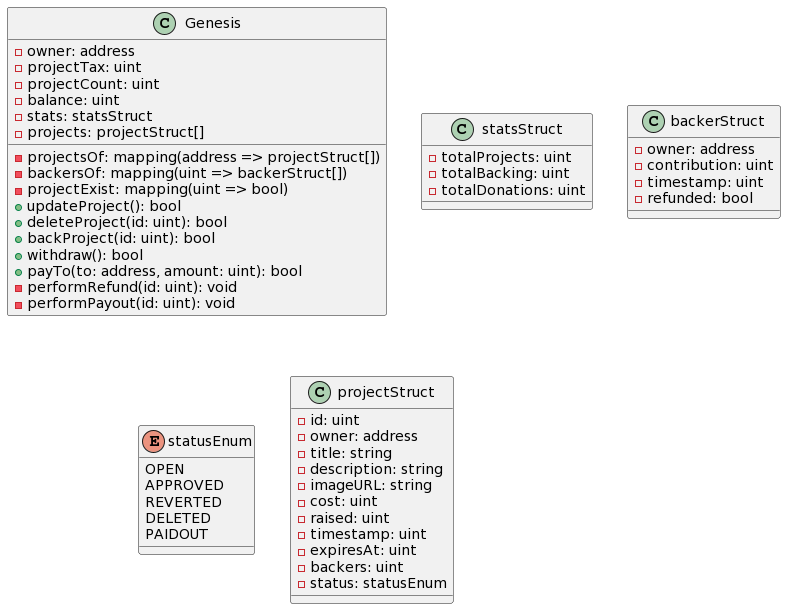
|  |  |  |  |
| --- | --- | --- | --- |
| **Phase Title** | **Expected Date of Completion** | **Actual Time of Completion with Guide’s Signature** | **Remarks** |
| **I. Preliminary Investigation** |  |  |  |
| (i) Organizational Overview |  |  |  |
| (ii) Present System and its advantages | 10/12/2022 |  |  |
| (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) 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 |  |  |

**Gantt Chart**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TYBSc Computer Science Semester 6 Project Gantt Chart** | | | | **Time Requirement** | | Year 2019 - 20 | | | | | | | | | | | | |
| Weeks | | | | | | | | | | | | |
| December | | January | | | | February | | | | March | | |
| W3 | W4 | W1 | W2 | W3 | W4 | W1 | W2 | W3 | W4 | W1 | W2 | W3 |
| **I** | **Preliminary Investigation** | | | Estimated | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actual | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **II** | **Requirement Gathering** | | | Estimated | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actual | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **III** | **System Analysis** | | | Estimated | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actual | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **IV** | **System Design** | | | Estimated | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actual | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **V** | **System Coding** | | | Estimated | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actual | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **VI** | **Testing** | | | Estimated | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actual | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **VII** | **Implementation** | | | Estimated | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actual | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **VIII** | **Deployment** | | | Estimated | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actual | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Estimated |  | |  | |  |  |  |  |
|  | Actual |  | |  | |  |  |  |  |

**System Analysis**

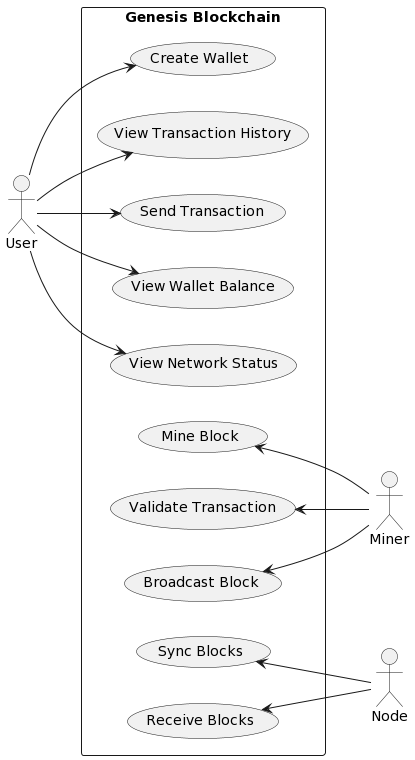
**Class Diagram**



**Event Table**

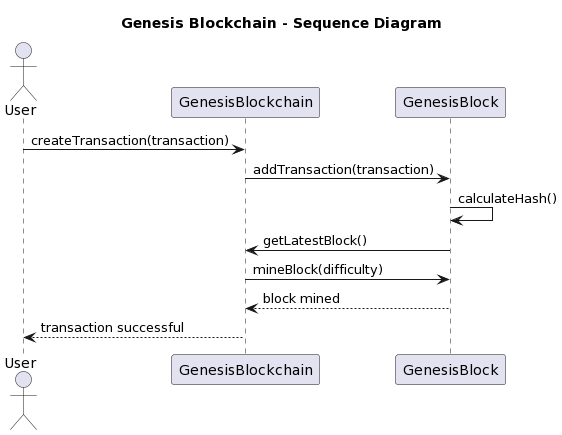
|  |  |  |  |
| --- | --- | --- | --- |
| Event | Trigger | Precondition | Post condition |
| User creates campaign | User specifies project details, fundraising goal, and deadline | User has a Metamask wallet on the Goerli network | Smart contract is created on the blockchain |
| User contributes to campaign | User sends Ether (ETH) through Metamask wallet | Campaign smart contract exists on the blockchain | Funds are added to the campaign balance |
| Campaign reaches fundraising goal | Campaign balance reaches or exceeds fundraising goal | Campaign smart contract exists on the blockchain | Campaign owner can withdraw funds |
| Campaign deadline passes | Campaign deadline is reached | Campaign smart contract exists on the blockchain | If fundraising goal is not met, funds are returned to contributors |
| Campaign owner withdraws funds | Campaign owner initiates withdrawal | Campaign balance is greater than 0 and fundraising goal has been met | Funds are transferred from the campaign smart contract to the campaign owner's wallet |

**Use Case**



**System Design**

**Sequence Diagram**



**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()

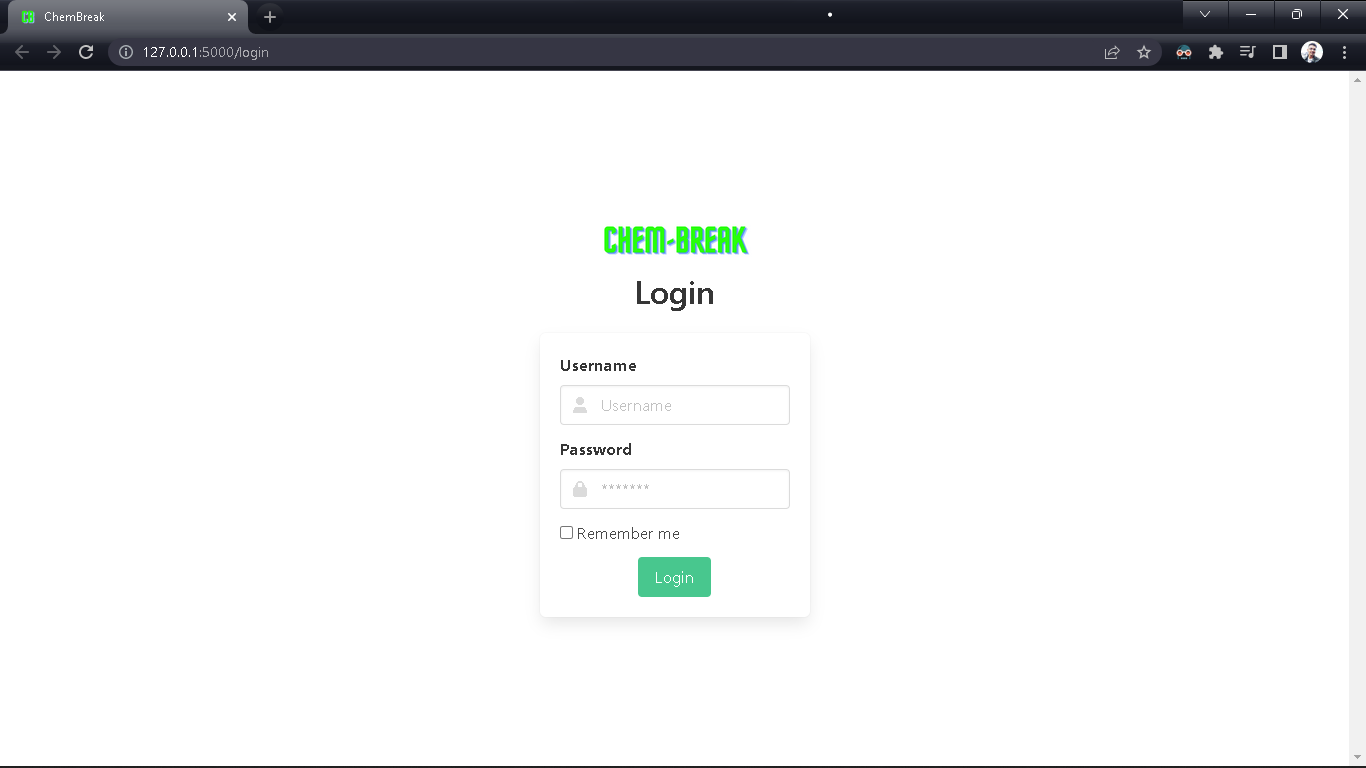
flash("All tables have been emptied", "success")

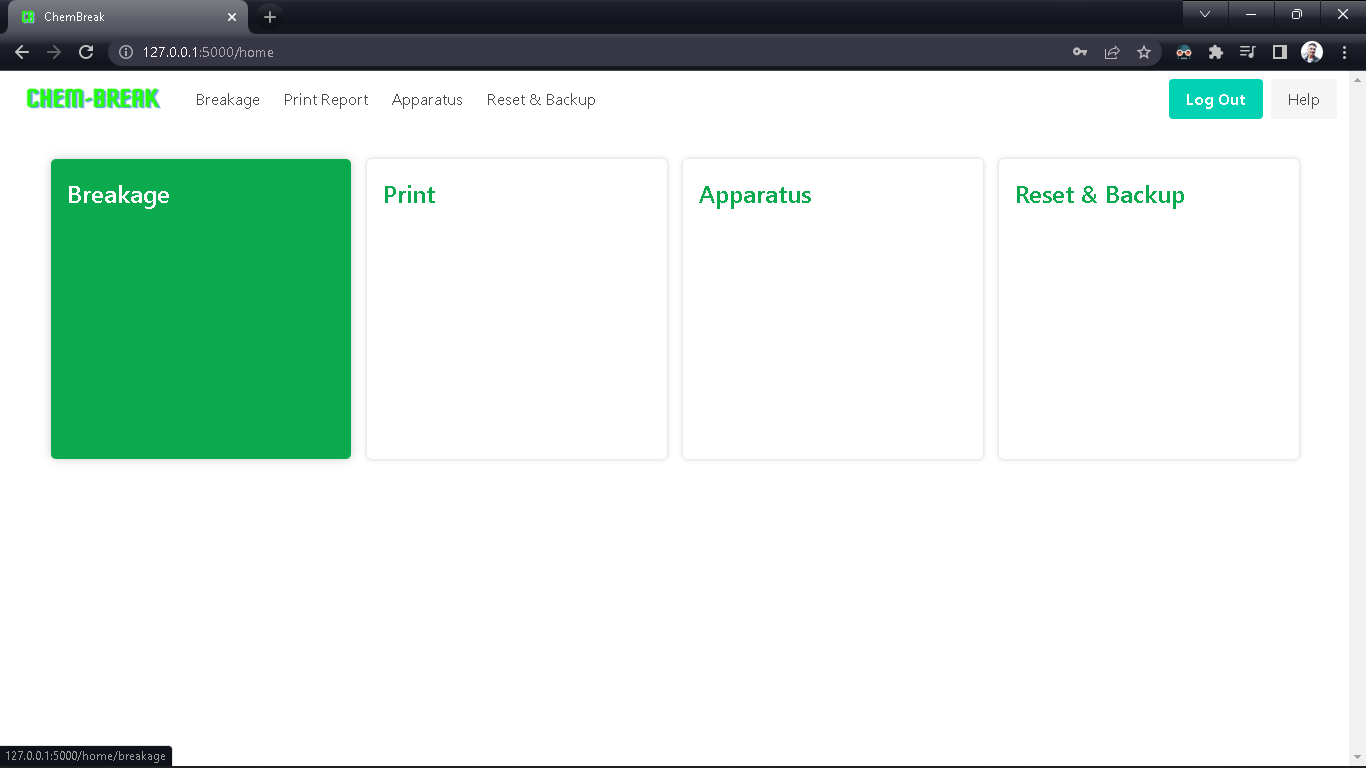
return render\_template('reset\_and\_backup.html')

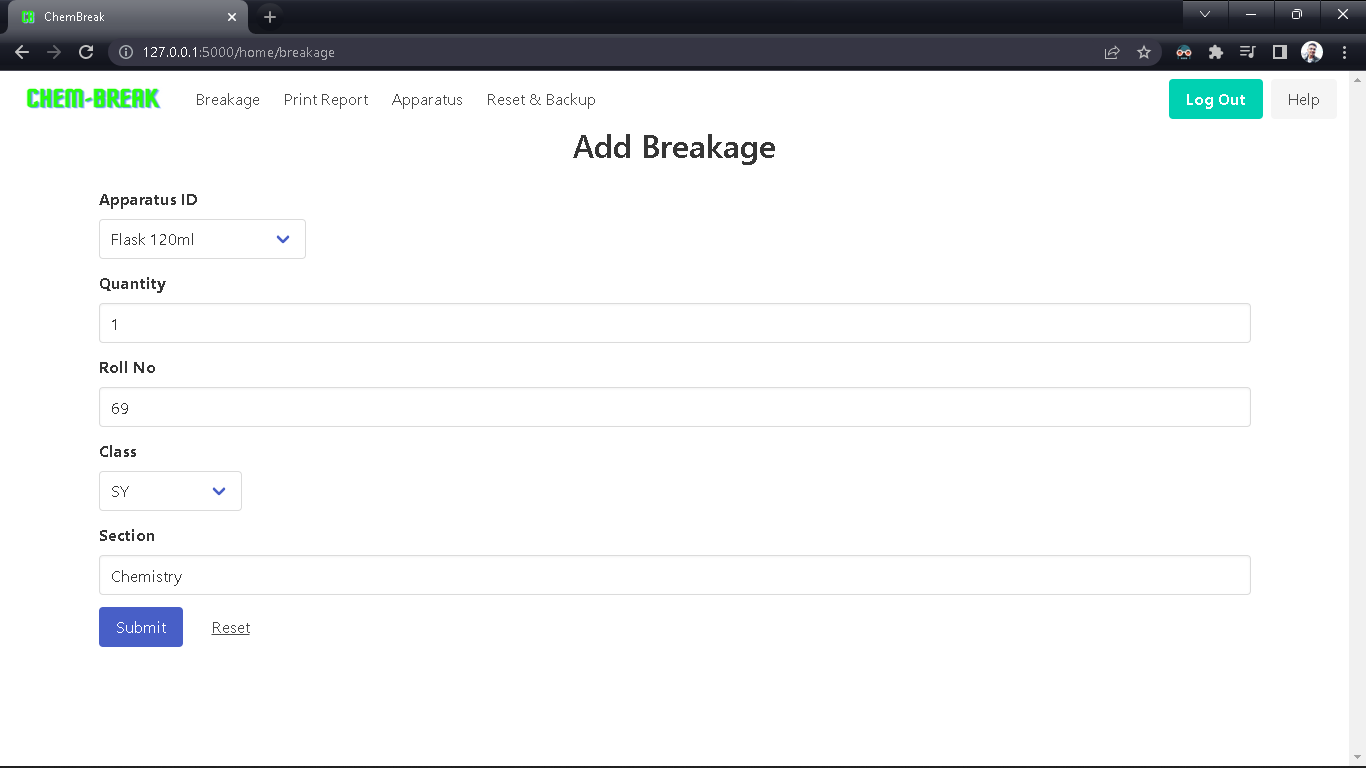
**Test Cases**

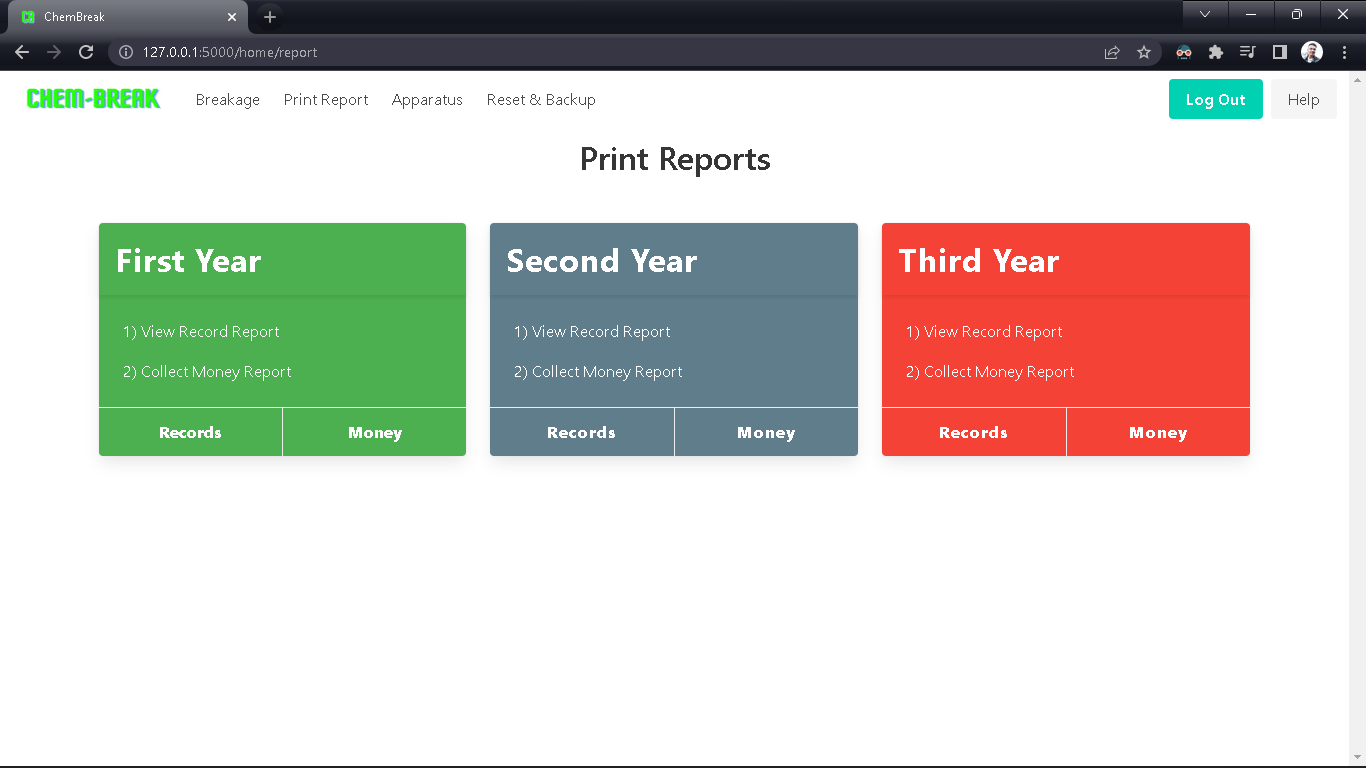
|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Steps** | **Expected Result** | **Actual Result** |
| User Login | 1. Enter valid username and password.  2. Click on "Login" button | User is successfully logged in and redirected to home page | User is successfully logged in and redirected 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 |

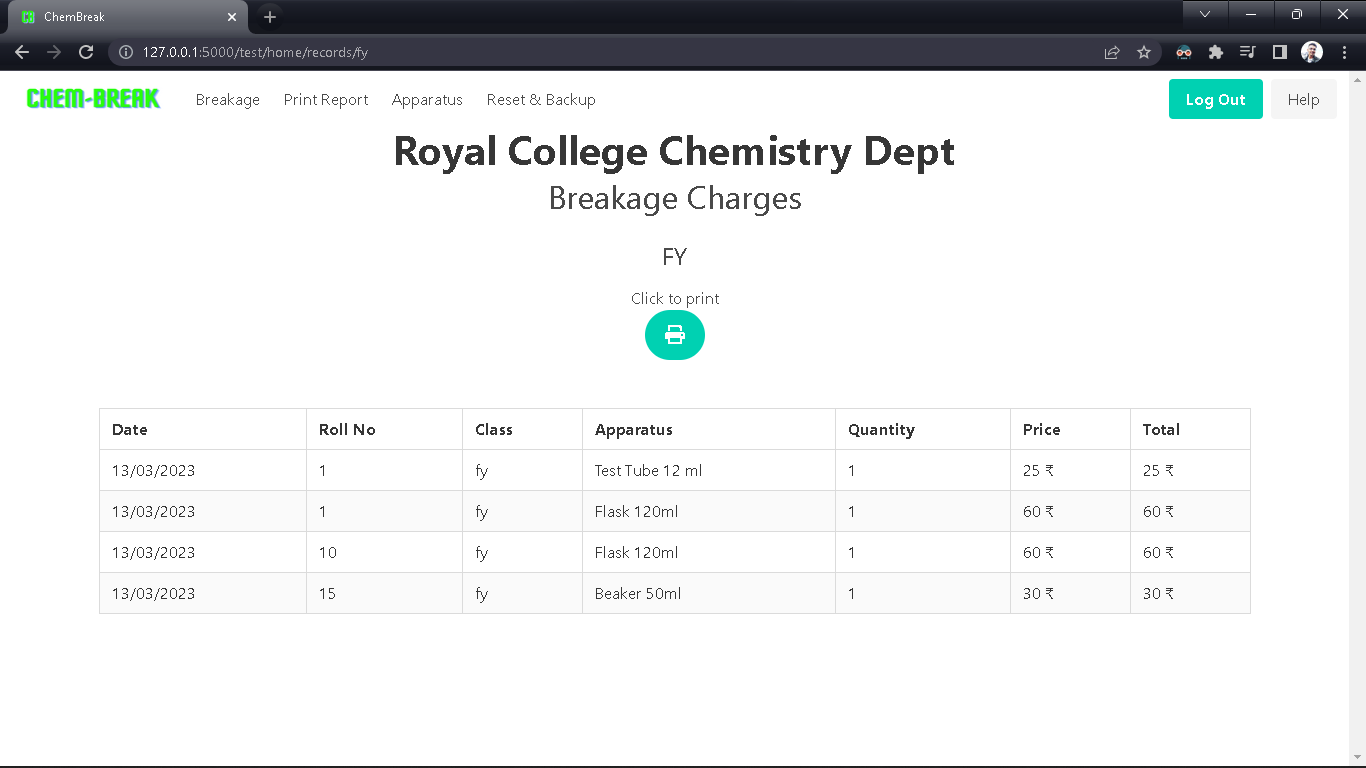
**Snapshot**

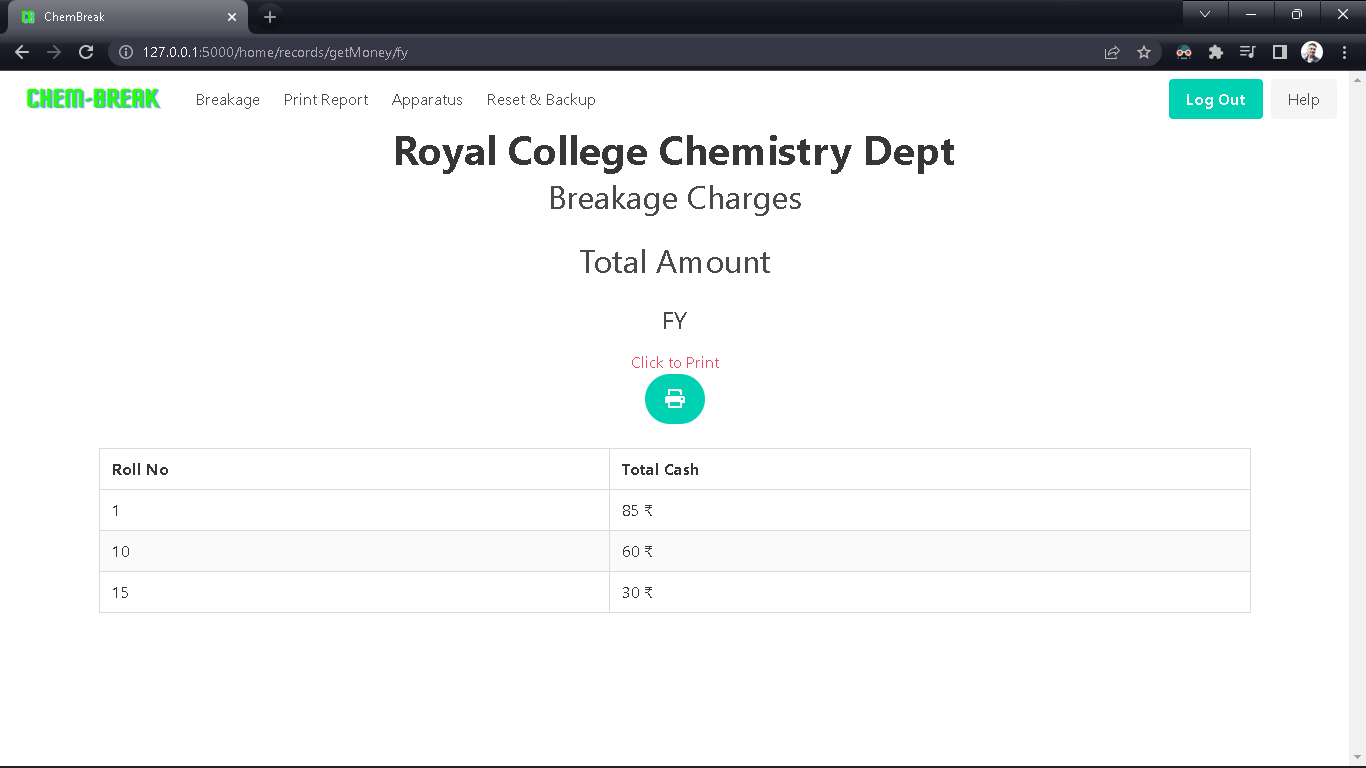












**Future Enhancement**

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.