NEWS TRACKER APPPLICATION

USING CLOUD

Submitted by

Team ID: PNT2022TMID03031

Vasunthra AS

Yoganithi T

Retheeka D

Revathi V

Team ID	**PNT2022TMID03031**
Project Name	**NEWS TRACKER APPPLICATION**
College Name	**SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY**

ABSTRACT

Everyone has the right to freedom of speech. However, this right is being misused to differentiate and attack others, physically or verbally, in the name of free speech. This discrimination is known as hate speech. Hate speech can be well-defined as language used to express hate towards a person or a group of people based on characteristics such as race, religion, ethnicity, gender, nationality, disability and sexual orientation. The increasing usage of social sites and information sharing has specified major benefits to humanity. However, this has also assumed rise to a variety of challenges including the spreading and sharing of hate speech messages. Thus, to solve this emerging issue in social media sites, recent studies employed a variety of machine learning and deep learning algorithms with text mining algorithm to automatically detect the hate speech messages on real time datasets. Hence, the aim of this Project is to analyses the comments on social networks using Natural Language processing technique (NLP) and Deep learning algorithm named as Back propagation neural network algorithm. Using NLP technique, can extract the keywords from user generated content and implement Back Propagation neural network to classify the text whether it is positive or negative. If it is negative means, automatically block the comments as per user wish and also block the friends based on pre-defined threshold values. Experimental results shows that the proposed framework implemented in real time social network site with improved notification .

TABLE OF FIGURES

News Tracker Application

1. INTRODUCTION

Mobile app ecosystems are transforming patterns of news consumption. Until quite recently, reading the news was a niche use for smartphones [12], mostly for when users were ‘on the go’; now however, two in every three users of mobile devices in the US regularly access news and as many as one in five read in-depth news articles daily [2]; a similar picture is found in the UK [1]. This growth in mobile news access continues the migration of news consumers to the Internet.

1.1 Project overview

As news is increasingly accessed on smartphones and tablets, the need for personalising news app interactions is apparent. We report a series of three studies addressing key issues in the development of adaptive news app interfaces. We first surveyed users’ newsreading preferences and behaviors’; analysis revealed three primary types of reader. Wethen implemented and deployed an Android news app that logs users’ interactions with the app. We used the logs to train a classifier and showed that it is able to reliably recognise a user according to their reader type. Finally we evaluated alternative, adaptiveuser interfaces for each reader type. The evaluation demonstrates the differential benefitof the adaptation for different users of the news app and the feasibility of adaptive interfaces for news apps.

1.2 Purpose

As our lives are very busy these days, we often feel we need more than 24 hrs. a day to cope up with everything we have in our schedule. Well, that's not possible but reducing the time by changing the conventional method of reading news can help. Just tell us what market news you're interested in and get a quick peek for the day. Only read what you feel is relevant and save your time. This app helps you to query for all information about Indices, Commodities, Currencies, Future Rates, Bonds, etc.… as on official websites.

2. LITERATURE SURVEY

2.1 Existing problem

We often feel we need more than 24 hrs. a day to cope up with everything we have in our schedule. Well, that's not possible but reducing the time by changing the conventional method of reading news can help.

2.2 References

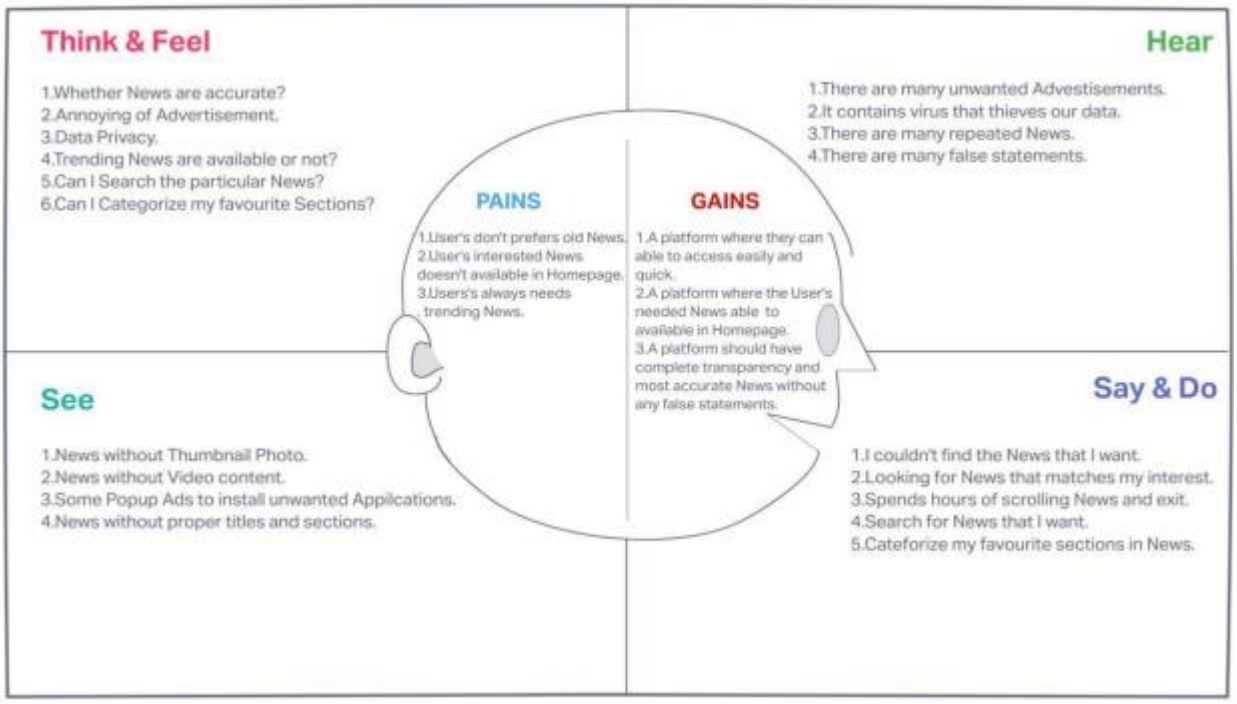
Like articles, websites, blogs.

2.3 Problem Statement Definition

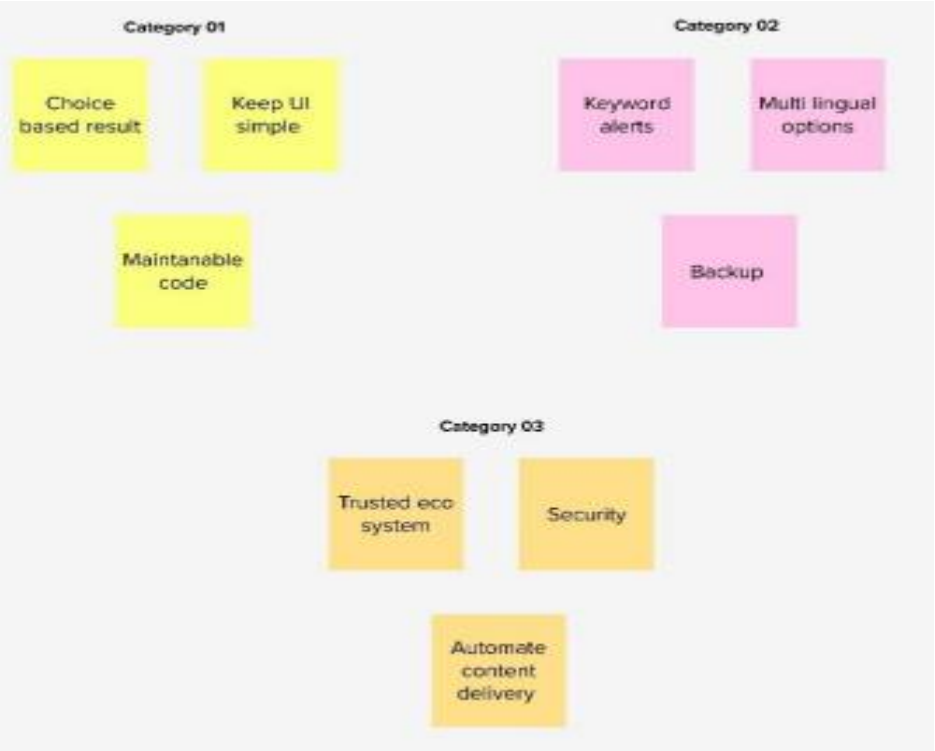
1. The user needs a way to organize the news on daily basis so that he can cope up with daily events in his tight schedule.
2. The user wants to read news only about particular topics so that he can be informed about his interest.
3. The user wants to get informed from only certified news outlets .
4. The user needs a way to search about the news on topics he wants to.
5. The user wants to know the news about his surrounding using GPS location.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.1.2 Ideation & Brainstorming

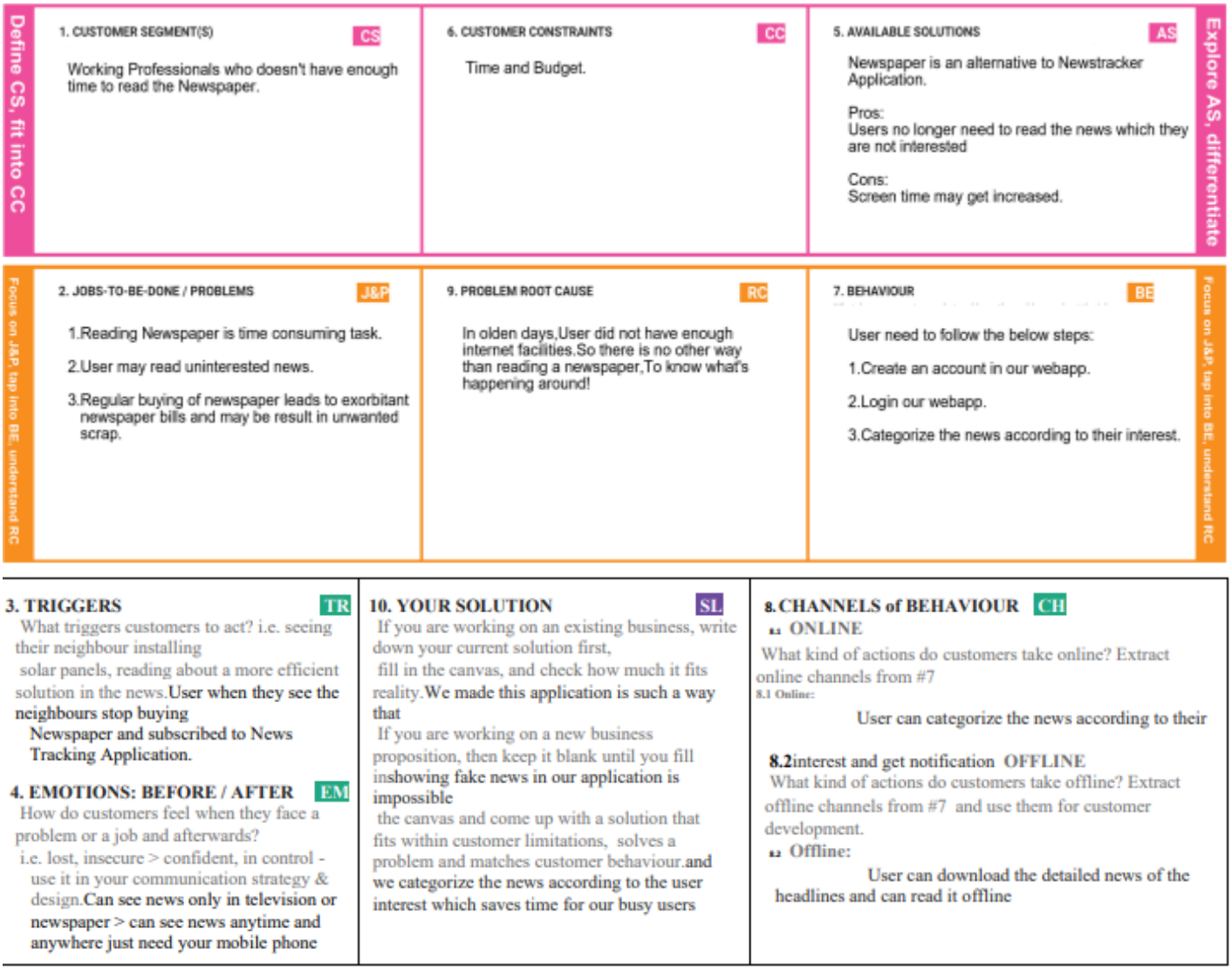


3.2 Proposed Solution

Robert is a businessman who's busy 24/7, so he has practically no time to consume news through the form of a physical newspaper and it's difficult for a person like him to carry a physical newspaper, because he travels a lot to attend his business meetings. And it's difficult for him to select the kind of news which interests him and also useful for him, so if he goes for a physical newspaper it's going be a time consuming task. So with the help of a application it's easier for him to read the kind of news he wants to consume and the news which actually keeps him well informed in his desired field despite his busy schedule and travel..

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	A single user may utilise several different newssharing applications, which are frequently inundated with notifications. Additionally, a lot of phoney news is disseminated. A newssharing app aims to make it simple for users to obtain daily relevant and vital news, as well as make it clear that the content is authentic and comes from reliable sources.
2.	Idea / Solution description	Since I'm presuming that consumers can't tell the difference between true and fake news, the app should list all of the reliable sources from across the world and then indicate in each article which source has verified this news.
3.	Novelty / Uniqueness	Creating a user experience that is appealing to clients and delivering news headlines with more details
4.	Social Impact / Customer Satisfaction	Making people aware of how to combat negative thinking and disseminating news that is socially responsible. To ensure consumer satisfaction, we track all news. In today's world, knowledge will be more useful.
5.	Business Model (Revenue Model)	In addition to promoting advertisements for quality items, one of our first and most important tasks is to provide the public with quality information or news.
6.	Scalability of the Solution	The high data storage capacity, processing power, and networking capabilities of this application can all be scaled using cloud computing infrastructure. Scaling is quick and simple to do.

3.3 Problem Solution fit



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form
FR-2	User Confirmation	Confirmation via Email
FR-3	User Login	Login Via Email ID & Password
FR-4	Home Page	Shows the NEWS Headlines.
FR-5	Tracker Page	Shows the NEWS elaboratively when the NEWS Headline of the Home page is clicked.

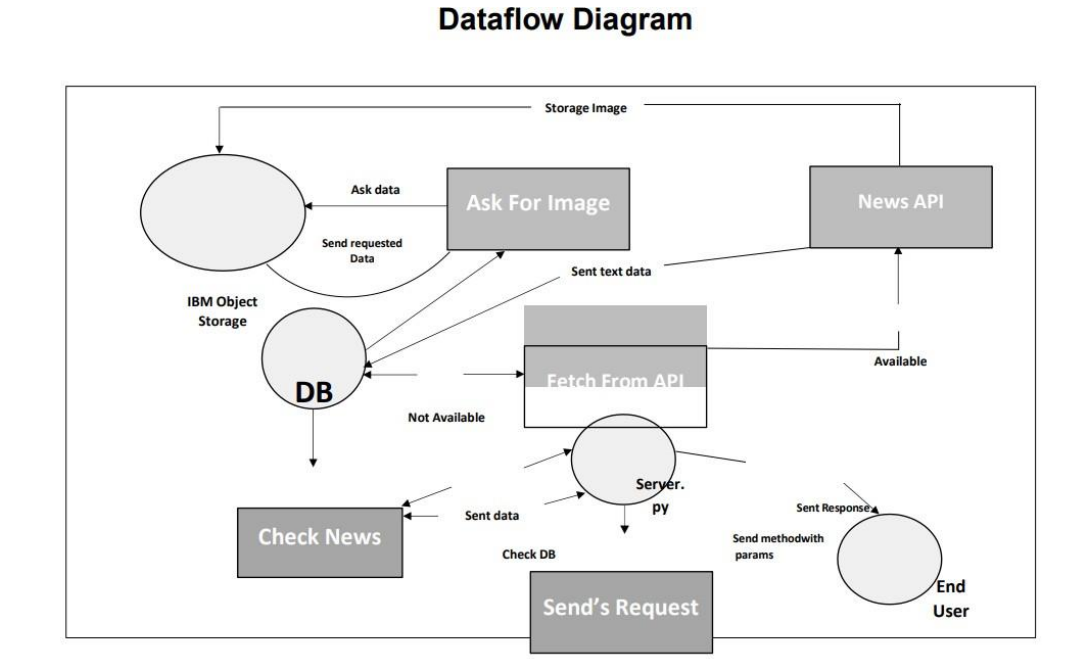
4.2 Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

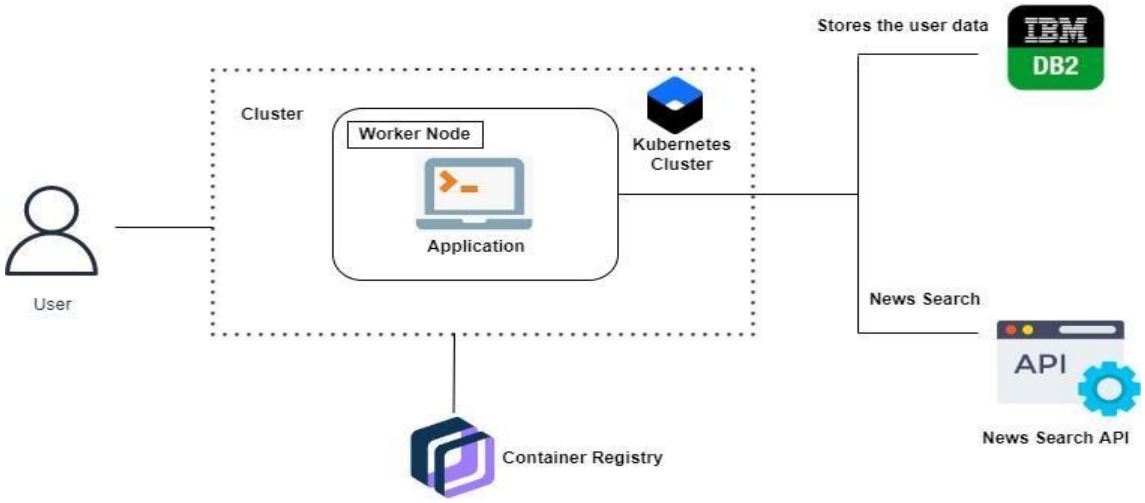
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Our Website is used to Read News and gain Knowledge according to user's interest.
NFR-2	**Security**	User can only login with their own Email ID and password without their Email and password no one can use the App.
NFR-3	**Reliability**	Our Website delivers the reliable content only from NEWS API.
NFR-4	**Performance**	It is a high performable App with friendly User Interface and shares lot of information without any Issues.
NFR-5	**Availability**	Our Website is supported with all search Engine so it will Avail for User at any time.
NFR-6	**Scalability**	This Website is built with high data storage capacity,processing power and networking can all be scaled using cloud computing

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
	Login	USN-3	As a user, I can log into the application by entering email & password	I can access my dashboard	High	Sprint-2
	Home Page	USN-4	As a user I can view the headlines of the news that interest me	I can read the news elaborately by clicking on the headlines	High	Sprint-2
		USN-5	As a user I can view the elaborate content of the headlines	I can download that to read it when iam free even in offine mode	Medium	Sprint-3
		USN-6	As a user I can search a news I want	I can read the news elaborately	High	Sprint-4
Customer Care Executive	Chatbot	USN-6	As a user I can solve my doubts about the application with the help of the chatbot.If the doubt is not resolved the chatbot guides us on how to contact the customer care executive	I can contact the customer care executive if needed	Medium	Sprint-3
Administrator	About us	USN-7	As a user I can view about the application and the developers of the application	I can be able to see their other applications	low	Sprint-4

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Account Creation	USN-1	As a user, I can create an account on the application.	5	High	Vasunthra A S, Yoganithi T
Sprint-1	Login	USN-2	I can successfully login to the application using provided login credentials.	5	High	Retheeka, Revathi V
Sprint-1	Storage	USN-3	As a user, my data will be stored on cloud in IBM Database.	5	High	Vasunthra A S, Yoganithi T
Sprint-2	News API integration	USN-4	I need to know the latest news in the app which can be pulled from the News API.	3	Low	Retheeka, Revathi V
Sprint-2	Add routes to display news with respect to user preferences	USN-5	As a user, I want only the news that I prefer.	10	High	Vasunthra A S, Yoganithi T
Sprint-3	Front End	USN-6	Create front end for all above listed services and connect them to back end and database.	5	Medium	Retheeka, Revathi V
Sprint-3	Front End	USN-7	Create front end for all above listed services and connect them to back end and database.	5	Medium	Vasunthra A S, Yoganithi T
Sprint-4	Deploy the application	USN-8	Deploy the application to Kubernetes and host the application using IBM services	5	Medium	Retheeka, Revathi V
Sprint-4	Additional Features	USN-9	Implement all additional features of the application	5	Low	Vasunthra A S, Yoganithi T

6.2 Sprint Delivery Schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Account Creation	USN-1	As a user, I can create an account on the application.	5	IIigh	Vasunthra A S, Yoganithi T
Sprint-1	Login	USN-2	I can successfully login to the application using provided login credentials.	5	IIigh	Retheeka, Revathi V
Sprint-1	Storage	USN-3	As a user, my data will be stored on cloud in IBM Database.	5	IIigh	Vasunthra A S, Yoganithi T
Sprint-2	News API integration	USN-4	I need to know the latest news in the app which can be pulled from the News API.	3	Low	Retheeka, Revathi V
Sprint-2	Add routes to display news with respect to user preferences	USN-5	As a user, I want only the news that I prefer.	10	High	Vasunthra A S, Yoganithi T
Sprint-3	Front End	USN-6	Create front end for all above listed services and connect them to back end and database.	5	Medium	Retheeka, Revathi V
Sprint-3	Front End	USN-7	Create front end for all above listed services and connect them to back end and database.	5	Medium	Vasunthra A S, Yoganithi T
Sprint-4	Deploy the application	USN-8	Deploy the application to Kubernetes and host the application using IBM services	5	Medium	Retheeka, Revathi V
Sprint-4	Additional Features	USN-9	Implement all additional features of the application	5	Low	Vasunthra A S, Yoganithi T
Sprint-4	Testing	Testing	Testing all the features of the application.	15	IIigh	Retheeka, Revathi V

7. CODING & SOLUTIONING

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_mysqldb import MySQL
import MySQLdb.cursors
import re

app = Flask(__name__)

app.secret_key = 'your secret key'

app.config['MYSQL_HOST'] = 'localhost'
```

```
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'your password'
app.config['MYSQL_DB'] = 'geeklogin'

mysql = MySQL(app)



@app.route('/')
@app.route('/login', methods=['GET', 'POST'])
def login():
    msg = ''
    if request.method == 'POST' and 'username' in request.form and 'password' in
request.form:
        username = request.form['username']
        password = request.form['password']
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM accounts WHERE username = % s AND
password = % s', (username, password,))
        account = cursor.fetchone()
        if account:
            session['loggedin'] = True
            session['id'] = account['id']
            session['username'] = account['username']
            msg = 'Logged in successfully !'
            return render_template('index.html', msg=msg)
        else:
            msg = 'Incorrect username / password !'
    return render_template('login.html', msg=msg)



@app.route('/logout')
```

```
def logout():
    session.pop('loggedin', None)
    session.pop('id', None)
    session.pop('username', None)
    return redirect(url_for('login'))


@app.route('/register', methods=['GET', 'POST'])
def register():
    msg = ''
    if request.method == 'POST' and 'username' in request.form and 'password' in
request.form and 'email' in request.form:
        username = request.form['username']
        password = request.form['password']
        email = request.form['email']
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM accounts WHERE username = % s', (username,))
        account = cursor.fetchone()
        if account:
            msg = 'Account already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email address !'
        elif not re.match(r'[A-Za-z0-9]+', username):
            msg = 'Username must contain only characters and numbers !'
        elif not username or not password or not email:
            msg = 'Please fill out the form !'
        else:
            cursor.execute('INSERT INTO accounts VALUES (NULL, % s, % s, % s)',
(username, password, email,))
            mysql.connection.commit()
            msg = 'You have successfully registered !'
```

```
elif request.method == 'POST':
    msg = 'Please fill out the form !'
return render_template('register.html', msg=msg)
```

8. TESTING

8.1 Test Cases

Test case

Test case	feature	component	Test scenario	Expected result	Actual result	status	comments	bug
Sign in	**Functional**	**Login page**	**Verify user can see the sign in option**	**can visible**	**Yes visible**	**pass**	**successful**	**-**
Sign up	Functional	Login page	Verify user has the option to sign up	Can visible	Yes visible	pass	Successful	-
Forgot password	Functional	Login page	Verify user has the option to forgot password	Yes the option is available	Option is available	pass	Successful	-
Fetch news	**Functional**	**Home page**	**Verify user can get the news**	**News will be eed to the app**	**404 error**	**Fail**	**unsuccessful**	**App integeration problem**
Types of news available in the fetch news page	Functional	Fetch news	Types of news available	Weather, Sport, Economy.	Hover buttons will be Shown.	Yes	successful	-

Result:

News tracker application using cloud is developed and executed at the level of completed progress.

10. ADVANTAGES & DISADVANTAGES

Advantages:-

- In this app news is already categorization.
- Easily accessible and portable.
- Better user experience.
- Apps help you convert visitors to loyal readers.
- Minute by minute updates of news.
- This app helps you to get local news updates instantly.
- To explore and discover trending news and topics.
- News feeds for you based on your interest.

Disadvantages:-

- Some apps demand premium subscription from user.
- Occurance of Advertisment disturb the user.
- Sometimes the news gives brief information.
- Prevailance of fake and uncertain news can confuse the user lead to misconception.
- Fake news may mislead the readers.

11. CONCLUSION

We explored the feasibility of recognising patterns of news reading interactions and evaluated three adaptive interface designs for different news reader types. We show that from their interaction log, a specific user can be recognised as one of three kinds. The reader types emerging from the online survey are well defined and distinct. The evaluation of the three variant interfaces suggests that different news reader types need different user interfaces. We have demonstrated a method for monitoring users' news reading behaviour and inferring news reader type from it. In the future we will further explore the design of adaptive interfaces, in order to be in a position to demonstrate a complete adaptive mobile news framework providing automatic personalisation

of news apps.

12. FUTURE SCOPE

In the future we will further explore the design of adaptive interfaces, in order to be in a position to demonstrate a complete adaptive mobile news framework providing automatic personalisation of news apps.

13. APPENDIX

Source Code:

Backend_

App.py

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_mysqldb import MySQL
import MySQLdb.cursors
import re

app = Flask(__name__)

app.secret_key = 'your secret key'

app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'your password'
app.config['MYSQL_DB'] = 'geeklogin'

mysql = MySQL(app)



@app.route('/')
@app.route('/login', methods=['GET', 'POST'])
def login():
    msg = ''
    if request.method == 'POST' and 'username' in request.form and 'password' in
request.form:
        username = request.form['username']
        password = request.form['password']
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM accounts WHERE username = % s AND
```

```
password = % s', (username, password,))
        account = cursor.fetchone()
        if account:
            session['loggedin'] = True
            session['id'] = account['id']
            session['username'] = account['username']
            msg = 'Logged in successfully !'
            return render_template('index.html', msg=msg)
        else:
            msg = 'Incorrect username / password !'
    return render_template('login.html', msg=msg)


@app.route('/logout')
def logout():
    session.pop('loggedin', None)
    session.pop('id', None)
    session.pop('username', None)
    return redirect(url_for('login'))


@app.route('/register', methods=['GET', 'POST'])
def register():
    msg = ''
    if request.method == 'POST' and 'username' in request.form and 'password' in
request.form and 'email' in request.form:
        username = request.form['username']
        password = request.form['password']
        email = request.form['email']
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM accounts WHERE username = % s', (username,))
```

```
        account = cursor.fetchone()
        if account:
            msg = 'Account already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email address !'
        elif not re.match(r'[A-Za-z0-9]+', username):
            msg = 'Username must contain only characters and numbers !'
        elif not username or not password or not email:
            msg = 'Please fill out the form !'
        else:
            cursor.execute('INSERT INTO accounts VALUES (NULL, % s, % s, % s)',
(username, password, email,))
            mysql.connection.commit()
            msg = 'You have successfully registered !'
    elif request.method == 'POST':
        msg = 'Please fill out the form !'
    return render_template('register.html', msg=msg)
```

Routes_

CheckEmail.py

```
from flask import request
from flask_restful import Resource
from utils.dbQuery import selectQuery

class CheckEmail(Resource):
    def post(self):
        data=request.json
        email=str(data["email"])
        res=selectQuery("SELECT email FROM USER WHERE EMAIL=?",(email,))
        print(res)
        if(not res):
            return {"status":True},200
    return {"status":False},400
```

getNews.py

```
from random import random
from flask_restful import Resource
from utils.cookieChecker import token_required
from utils.dbQuery import selectQuery
from utils.apiFetch import apiData
import random
from utils.testData import retArray


class Personal(Resource):
    @token_required
    def get(email, self):
        topicsArr = ["sport", "tech", "world", "finance", "politics", "business",
```

```
                "economics", "entertainment", "beauty", "travel", "music", "food",
"science", "cricket"]
        fav = selectQuery("SELECT favourites from user where email=?", (email,))[
            'FAVOURITES']
        fav = fav.split(',')
        for x in fav:
            topicsArr.remove(x)
        retArr = []
        favList = []
        # favList = retArray
        nonFavList = []
        for x in fav:
            try:
                data = apiData(x)
            except:
                data=None
            if (data is not None):
                for y in data:
                    favList.append(y)
        try:
            random.shuffle(favList)
            favList = sorted(favList, key=lambda k: k['date'], reverse=True)
        except:
            favList=[]
        for x in topicsArr:
            try:
                data = apiData(x)
            except:
                data=None
            if(data is not None):
                for y in data:
```

```
                nonFavList.append(y)
        try:
            random.shuffle(nonFavList)
            nonFavList = sorted(nonFavList, key=lambda k: k['date'], reverse=True)
        except:
            nonFavList=[]
        retArr = favList+nonFavList
        return {"data": retArr}, 200

class News(Resource):
    @token_required
    def get(email,self,topic):
        topicsArr = ["headline","sport", "tech", "world", "finance", "politics", "business",
                     "economics", "entertainment", "beauty", "travel", "music", "food",
"science", "cricket"]
        if(topic not in topicsArr):
            return {"status":"Not a valid topic"},404
        try:
            retArr=apiData(topic)
            random.shuffle(retArr)
            retArr=sorted(retArr,key=lambda k:k['date'],reverse=True)
        except:
            retArr=[]
    return {"data":retArr},200
```

Islogin.py

```
from flask import request, after_this_request
from utils.password import checkPassword
```

```
from flask_restful import Resource
from utils.cookieChecker import token_required


class IsLogin(Resource):
    @token_required
    def get(email, self):
    return {"status": "Logged in"}, 200
```

login.py

```
from datetime import datetime
from flask_restful import Resource
from flask import request,after_this_request
from utils.password import checkPassword
from utils.dbQuery import *
from dateparser import parse
from utils.tokener import generate_confirmation_token
from utils.emailSender import emailSender
import app
import jwt

class Login(Resource):
    def post(self):
        data=request.json
        ip=request.headers.get("ip")
        email=data["email"]
        password=data["password"]
        queryRes=selectQuery("SELECT * from user where email=?",(email,))
        res=checkPassword(password,queryRes["PASSWORD"])
        if(not res):
```

```
            return {"status":"Wrong credentials"},400
        if(not queryRes["VERIFIED"]):
            lastTime=parse(queryRes["RESEND_TIME"])
            currTime=datetime.now()
            diff=currTime-lastTime
            diff=diff.total_seconds()
            if(diff>3600):
                token=generate_confirmation_token(email)
                emailSender(email,token)
                insertQuery("UPDATE user set RESEND_TIME=? where
email=?",(datetime.now(),email))
            return {"status":"Not verified"},400

        @after_this_request
        def cookieSender(response):
            access_token=jwt.encode(
                {"email":email,"ip":ip},app.app.config['SECRET_KEY']
            )
            #
response.set_cookie("access_token",str(access_token),httponly=True,samesite=None,p
ath="/")
            #
response.set_cookie("email",str(email),httponly=True,samesite=None,path="/")
            response.headers.add('Set-Cookie',f'access_token={str(access_token)};
SameSite=None; Secure; HttpOnly; Path=/')
            response.headers.add('Set-Cookie',f'email={str(email)}; SameSite=None;
Secure; HttpOnly; Path=/')
            return response

       return{"status":"Successfully Logged in"},200
```

Register.py

```
from datetime import datetime
from flask import request,after_this_request
from flask_restful import Resource
from utils.dbQuery import insertQuery
from utils.emailSender import newEmailSender
from utils.password import genHash

class Register(Resource):
    def post(self):
        req=request.json
        name=req['name']
        email=req['email']
        password=req['password']
        favourite=req['favourite']
        fav=','.join(favourite)
        if(name==" or email==" or password==" or fav=="):
            return {"status":"Missing data"},404
        password=genHash(password)
        t=(name,email,password,fav,datetime.now())
        res=insertQuery('INSERT INTO user
(name,email,password,favourites,resend_time) values (?,?,?,?,?)',t)
        if(not res):
            return {"status":"Error while registering"},400

        @after_this_request
        def emailer(response):
            newEmailSender(email)
```

```
            return response


    return {"status":"Successfully registered"},200
```

Utils_

apiFetch.py

```
from datetime import datetime
from time import sleep
import warnings
from dotenv import dotenv_values
from threading import *
import requests
from dateparser import parse



class Api:
    warnings.simplefilter('ignore')
    __key = dotenv_values(".env")
    __key = __key["key"]
    __apiMap = {}
    __mainApiMap = {}
    __url = "https://newscatcher.p.rapidapi.com/v1/latest_headlines"
    __headers = {
        "X-RapidAPI-Key": str(__key),
        "X-RapidAPI-Host": "newscatcher.p.rapidapi.com"
    }

    def __newCatcherRunner(self, title):
        querystring = {"topic": title, "lang": "en",
                       "media": "True", "country": "IN"}
        respone = requests.request(
            "GET", url=self.__url, headers=self.__headers, params=querystring)
        respone = respone.json()
        retArr = []
        for x in respone["articles"]:
            newJson = {}
```

```
            newJson["url"] = x["link"]
            newJson["title"] = x["title"]
            newJson["img"] = x["media"]
            newJson["topic"] = x["topic"]
            currTime = parse(x["published_date"])
            newJson["date"] = currTime.strftime("%d/%m/%Y")
            retArr.append(newJson)
        return retArr

    def __topHeadlinesFetcher(self):
        querystring = {"topic":"news","lang": "en","media": "True", "country": "IN"}
        respone = requests.request("GET", url=self.__url, headers=self.__headers, params=querystring)
        respone = respone.json()
        retArr = []
        for x in respone["articles"]:
            newJson = {}
            newJson["url"] = x["link"]
            newJson["title"] = x["title"]
            newJson["img"] = x["media"]
            newJson["topic"] = x["topic"]
            currTime = parse(x["published_date"])
            newJson["date"] = currTime.strftime("%d/%m/%Y")
            retArr.append(newJson)
        self.__apiMap["headline"]=retArr
        print("headline fetched at "+str(datetime.now()))

    def __newsCatcherApiFetcher(self):
        arr = ["sport", "tech", "world", "finance", "politics", "business",
               "economics", "entertainment", "beauty", "travel", "music", "food", "science"]
        for x in arr:
            self.__apiMap[x] = self.__newCatcherRunner(x)
        print("NewsCatcher fetched at "+str(datetime.now()))

    def __cricketFetcher(self):
        url = "https://cricbuzz-cricket.p.rapidapi.com/news/v1/index"
        headers = {
            "X-RapidAPI-Key": self.__key,
```

```
        "X-RapidAPI-Host": "cricbuzz-cricket.p.rapidapi.com"
    }
    response = requests.request("GET", url, headers=headers)
    response = response.json()
    response = response["storyList"]
    retArr = []
    for x in response:
        try:
            x = x["story"]
            newJson = {}
            newJson["url"] = f'https://www.cricbuzz.com/cricket-news/{x["id"]}/newsTrakcer'
            newJson["title"] = x["hline"]
            newJson["image"] = f'https://www.cricbuzz.com/a/img/v1/500x500/i1/c{x["id"]}/abc.jpg'
            currTime = datetime.fromtimestamp(int(x["pubTime"])/1e3)
            newJson["date"] = currTime.strftime("%d/%m/%Y")
            newJson["topic"] = "cricket"
            retArr.append(newJson)
        except:
            pass
    self.__apiMap["cricket"] = retArr
    print("Cricbuzz fetched at "+str(datetime.now()))

def newsCatcherThreader(self):
    while True:
        print("NewsCatcher fetching.... at "+str(datetime.now()))
        try:
            self.__newsCatcherApiFetcher()
            self.__mainApiMap = self.__apiMap
        except:
            print("Error NewsCatcher fetching.... at "+str(datetime.now()))
            pass
        sleep(30*60)

def topHeadlinesThreader(self):
    while True:
        print("Headline fetching.... at "+str(datetime.now()))
        try:
```

```
            self.__topHeadlinesFetcher()
            self.__mainApiMap = self.__apiMap
        except:
            print("Error headline fetching.... at "+str(datetime.now()))
            pass
        sleep(30*60)

    def cricbuzzThreader(self):
        while True:
            print("Cricbuzz fetching.... at "+str(datetime.now()))
            try:
                self.__cricketFetcher()
                self.__mainApiMap = self.__apiMap
            except:
                print("Error Cricbuzz fetching.... at "+str(datetime.now()))
                pass
            sleep(15*60)

    def dataGetter(self, topic):
        return self.__mainApiMap[str(topic)]



a = Api()

def apiRunner():
    t1 = Thread(target=a.topHeadlinesThreader)
    t2 = Thread(target=a.newsCatcherThreader)
    t3 = Thread(target=a.cricbuzzThreader)
    t1.daemon=True
    t2.daemon=True
    t3.daemon=True
    t1.start()
    t2.start()
    t3.start()

def apiData(topic):
return a.dataGetter(topic)
```

CookieChecker.py

```
import app
from functools import wraps
import jwt
from flask import request,after_this_request

def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.cookies.get("access_token")
        try:
            data = jwt.decode(token, app.app.config['SECRET_KEY'],algorithms=['HS256'])
            ip=request.headers.get("ip")
            cookieIp=data['ip']
            if(ip!=cookieIp):
                resp={"status":"not logged in"}
                @after_this_request
                def deleter(response):
                    response.delete_cookie("access_token",path="/")
                    response.delete_cookie("email",path="/")
                    return response
                return resp,401
        except:
            resp = {"status":"not logged in"}
            @after_this_request
            def deleter(response):
                response.delete_cookie("access_token",path="/")
                response.delete_cookie("email",path="/")
                return response
            return resp, 401
        return f(data['email'],*args, **kwargs)
    return decorated
```

dbConfig.py

```
from dotenv import dotenv_values
def getDbCred():
    data=dotenv_values(".env")
    dsn_hostname=data["ibm_host_name"]
    dsn_uid=data["ibm_user_id"]
    dsn_pwd=data["ibm_password"]
    dsn_driver=data["ibm_driver"]
    dsn_database=data["ibm_db_name"]
    dsn_port=data["ibm_port"]
    dsn_protocol=data["ibm_protocol"]
    dsn=(
        "DATABASE={1};"
        "HOSTNAME={2};"
        "PORT={3};"
        "PROTOCOL={4};"
        "UID={5};"
        "PWD={6};"

"SECURITY=SSL").format(dsn_driver,dsn_database,dsn_hostname,dsn_port,dsn_protocol,dsn_uid,d
sn_pwd)
return dsn
```

dbQuery.py

```
import ibm_db
from utils.dbConfig import getDbCred

conn=ibm_db.connect(getDbCred(),"","")

def selectQuery(query,params=None):
    try:
        stmt=ibm_db.prepare(conn,query)
        if(params==None):
            ibm_db.execute(stmt)
            data=ibm_db.fetch_assoc(stmt)
```

```
            return data
        ibm_db.execute(stmt,params)
        data=ibm_db.fetch_assoc(stmt)
        return data
    except:
        return False

def insertQuery(query,params):
    try:
        stmt=ibm_db.prepare(conn,query)
        ibm_db.execute(stmt,params)
        return True
    except:
    return False
```

emailSender.py

```
from __future__ import print_function
from utils.tokener import generate_confirmation_token
import sib_api_v3_sdk
import app
from sib_api_v3_sdk.rest import ApiException
from datetime import datetime


def emailSender(email, token):
    configuration = sib_api_v3_sdk.Configuration()
    configuration.api_key['api-key'] = app.data['mail_api_key']
    api_instance = sib_api_v3_sdk.TransactionalEmailsApi(
        sib_api_v3_sdk.ApiClient(configuration))
    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
```

```
    msg = {}
    msg['Subject'] = "Verfiy your NewsTracker Account"
    msg['From'] = {"name": "News Tracker Dev Team",
                   "email": "verify@newstracker.com"}
    msg['To'] = [{"email": email}]
    msg['Text']=f'Please click this <a
href="http://127.0.0.1:5500/frontend/pages/verify.html?token={token}">link</a> to
verify your account'
    html = f"""\
    <html>
      <head></head>
      <body>
      <p>நன்றி, for joining NewsTracker 🙏 </p>
      <br>
      <p>Hurray 🥳, you just registerd at NewsTracker<br><br>
      Please click the following link to verify your account:<br>
      <a href="http://127.0.0.1:5500/frontend/pages/verify.html?token={token}">Click
Here to Verify 😎</a>
      </p>
      <br>
      <p> ⚠️ Note: This link expires within one hour from the time sent</p>
      <br><br>
      <p>Regrads,<br></p>
      <p><a href="https://localhost:5000">NewsTracker Dev Team</a></p>
      <br><br>
      <p>Email sent at {dt_string}</p>
      </body>
    </html>
    """
    send_smtp_email = sib_api_v3_sdk.SendSmtpEmail(
      to=msg['To'], html_content=html, sender=msg['From'],
```

```
subject=msg['Subject'],text_content=msg['Text'])
    try:
        api_response = api_instance.send_transac_email(send_smtp_email)
        print(api_response)
    except ApiException as e:
        print("Exception when calling SMTPApi->send_transac_email: %s\n" % e)


def newEmailSender(email):
    token = generate_confirmation_token(email)
emailSender(email, token)
```

password.py

```
import bcrypt

def genHash(password):
    salt=bcrypt.gensalt()
    bytes=password.encode('utf-8')
    hash=bcrypt.hashpw(bytes,salt)
    print(hash)
    return hash

def checkPassword(password,hash):
    hash=hash.encode('utf-8')
    bytes=password.encode('utf-8')
    res=bcrypt.checkpw(bytes,hash)
return res
```

tokener.py

```
from itsdangerous import URLSafeTimedSerializer
import app
```

```
def generate_confirmation_token(email):
    serializer=URLSafeTimedSerializer(app.app.config['SECRET_KEY'])
    return
serializer.dumps(email,salt=app.app.config['SECURITY_PASSWORD_SALT'])

def confirm_token(token,expiration=3600):
    serializer=URLSafeTimedSerializer(app.app.config['SECRET_KEY'])
    try:
        email=serializer.loads(
            token,
            salt=app.app.config['SECURITY_PASSWORD_SALT'],
            max_age=expiration
        )
    except:
        return False
return email
```

JAVASCRIPT :

Login.js

```
import { poster } from "./modules/server.js";
const signupbutton = document.getElementById("signup");
const signinbutton = document.getElementById("signin");
const container = document.getElementById("container");

let signupVal = false;

signupbutton.addEventListener("click", () => {
  container.classList.add("right-panel-active");
```

```
});
signinbutton.addEventListener("click", () => {
  container.classList.remove("right-panel-active");
});

const signupForm = document.querySelector("#signup-form");
signupForm.querySelector("button").addEventListener("click", async (e) => {
  const name = signupForm.elements[0].value;
  const email = signupForm.elements[1].value;
  const password = signupForm.elements[2].value;
  const repassword = signupForm.elements[3].value;
  if (
    name === "" ||
    email === "" ||
    password === "" ||
    repassword === "" ||
    !signupVal
  ) {
    return;
  }
  if (password !== repassword) {
    signupForm.querySelector("h2").innerText = " ⚠️ Password doesn't match";
    return;
  } else {
    signupForm.querySelector("h2").innerText = "";
  }
  const checkboxes = document.querySelectorAll(
    ".container-checkbox input[type='checkbox']"
  );
  let favArr = [];
  checkboxes.forEach((t) => {
```

```
    if (t.checked) {
      favArr.push(t.value);
    }
  });
  signupForm.querySelector("h2").innerText = "⏳ Saving data in Server...";
  let postData = {
    name: name,
    email: email,
    password: password,
    favourite: favArr,
  };
  let response = await poster("register", postData);
  if (response.status === "Successfully registered") {
    signupForm.querySelector("h2").innerText =
      "User registered successfully ✅...";
  }
  setTimeout(() => {
    signupForm.querySelector("h2").innerText =
      "⚠️ Please verify your email before log in";
  }, 1000);
  setTimeout(() => {
    location.reload();
  }, 3000);
});

signupForm.elements[2].addEventListener("change", (e) => {
  let passwordRegex =
    /(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[^A-Za-z0-9])(?=.{8,})/;
  if (!e.target.value.match(passwordRegex)) {
    e.target.setCustomValidity(
      "Your password should be of 8 character with atleast one uppercase, one lowercase,
```

```
one number and one special digit"
    );
    e.target.value = "";
  } else {
    e.target.setCustomValidity("");
  }
});

signupForm.elements[1].addEventListener("change", async (e) => {
  let emailRegex =
    /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;
  if (e.target.value.match(emailRegex)) {
    let data = await poster("register/check", {
      email: e.target.value,
    });
    signupVal = false;
    if (data["status"] === false) {
      signupForm.querySelector("h2").innerText = " ⚠️ Email already in use";
      signupForm.elements[1].value = "";
    } else {
      signupForm.querySelector("h2").innerText = "";
      signupVal = true;
    }
  } else {
    e.target.value = "";
    signupVal = false;
  }
});

const signinForm = document.querySelector(".sign-in-container form");
signinForm.addEventListener("submit", async (e) => {
```

```
  e.preventDefault();
  const email = signinForm.elements[0].value;
  const password = signinForm.elements[1].value;
  const data = {
    email: email,
    password: password,
  };
  let retdata;
  let emailCheckData = await poster("register/check", {
    email: email,
  });
  if (emailCheckData["status"] === false) {
    retdata = await poster("login", data);
    if(retdata["status"]==="Not verified"){
      signinForm.querySelector("h2").innerText=" ⚠️ Please verify your mail by
clicking the link sent to your mail ID"
    }
    else if(retdata["status"]==="Wrong credentials"){
      signinForm.querySelector("h2").innerText=" ⚠️ Wrong credentials"
    }
    else if(retdata["status"]==="Successfully Logged in"){
      signinForm.querySelector("h2").innerText="Logged In ✅...Redirecting ⌛ "
    }
    else{
      signinForm.querySelector("h2").innerText=""
    }
  }
  else{
    signinForm.querySelector("h2").innerText=" ⚠️ Email not registered"
  }
```

});

Verify.js :

```
import { poster } from "./modules/server.js";

const queryString = window.location.search;
const urlParams = new URLSearchParams(queryString);

const token = urlParams.get("token");
async function verifyer() {
  let data = await poster("register/verify", {
    token: token,
  });
  if (data["status"] === "Couldn't verify") {
    document.querySelector(
      ".status-cont"
    ).innerHTML = `<div class="welcome-text">Uh oh... 😞 </div>
        <div class="sub-text">Couldn't verify you at this moment...</div>`;
    setTimeout(() => {
      location.href = "login.html";
    }, 8000);
  }
  if (data["status"] === "verified successfully") {
    document.querySelector(
      ".status-cont"
    ).innerHTML = `<div class="welcome-text">வணக்கம் 🙏 </div>
        <div class="sub-text">Let's know what's happening around us...</div>`;
    setTimeout(() => {
      location.href = "login.html";
    }, 8000);
```

```
  }
 }
verifyer();
```

Frontend_

Index.html

```
<html>
  <head>
    <meta charset="UTF-8">
    <title> Index </title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body></br></br></br></br></br>
    <div align="center">
     <div align="center" class="border">
       <div class="header">
         <h1 class="word">Index</h1>
       </div></br></br></br>
         <h1 class="bottom">
             Hi {{session.username}}!!</br></br> Welcome to the index
page...
         </h1></br></br></br>
         <a href="{{ url_for('logout') }}" class="btn">Logout</a>
     </div>
    </div>
  </body>
</html>
```

Login.html

```
<html>
   <head>
      <meta charset="UTF-8">
      <title> Login </title>
      <link rel="stylesheet" href="style.css">
   </head>
   <body></br></br></br></br></br>
      <div align="center">
       <div align="center" class="border">
         <div class="header">
           <h1 class="word">Login</h1>
         </div></br></br></br>
         <h2 class="word">
           <form action="{{ url_for('login') }}" method="post">

              <input id="username" name="username" type="text"
placeholder="Enter Your Username" class="textbox"/></br></br>
              <input id="password" name="password" type="password"
placeholder="Enter Your Password" class="textbox"/></br></br></br>
              <input type="submit" class="btn" value="Sign In"></br></br>
           </form>
         </h2>
         <p class="bottom">Don't have an account?  <a class="bottom"
href="{{url_for('register')}}"> Sign Up here</a></p>
        </div>
      </div>
   </body>
</html>
```

Register.html

```
<html>
   <head>
```

```
        <meta charset="UTF-8">
        <title> Register </title>
        <link rel="stylesheet" href="style.css">
    </head>
    <body></br></br></br></br></br>
        <div align="center">
          <div align="center" class="border">
            <div class="header">
              <h1 class="word">Register</h1>
            </div></br></br></br>
            <h2 class="word">
              <form action="{{ url_for('register') }}" method="post">
                <input id="username" name="username" type="text"
placeholder="Enter Your Username" class="textbox"/></br></br>
                <input id="password" name="password" type="password"
placeholder="Enter Your Password" class="textbox"/></br></br>
                <input id="email" name="email" type="text"
placeholder="Enter Your Email ID" class="textbox"/></br></br>
                <input type="submit" class="btn" value="Sign Up"></br>
              </form>
            </h2>
            <p class="bottom">Already have an account?  <a class="bottom"
href="{{url_for('login')}}"> Sign In here</a></p>
          </div>
        </div>
    </body>
</html>
```

Style.css

```
.header{
```

```
    padding: 5px 120px;
    width: 150px;
    height: 70px;
    background-color: #236B8E;
}

.border{
    padding: 80px 50px;
    width: 400px;
    height: 450px;
    border: 1px solid #236B8E;
    border-radius: 0px;
    background-color: #9AC0CD;
}

.btn {
    padding: 10px 40px;
    background-color: #236B8E;
    color: #FFFFFF;
    font-style: oblique;
    font-weight: bold;
    border-radius: 10px;
}

.textbox{
    padding: 10px 40px;
    background-color: #236B8E;
    text-color: #FFFFFF;
    border-radius: 10px;
}
```

```
::placeholder {
    color: #FFFFFF;
    opacity: 1;
    font-style: oblique;
    font-weight: bold;
}

.word{
    color: #FFFFFF;
    font-style: oblique;
    font-weight: bold;
}

.bottom{
    color: #236B8E;
    font-style: oblique;
    font-weight: bold;
}
```

Footer

© 2022 GitHub, Inc.

Footer navigation

Terms

Privacy

Security

GitHub

https://github.com/IBM-EPBL/IBM-Project-25253-1659956120