ONLINE VOTING SYSTEM

MINI PROJECT (REVIEW2)

Submitted by

MURALI KRISHNA S 212223230129

in partial fulfilment for the award of

the degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SAVEETHA ENGINEERING COLLEGE, THANDALAM

An Autonomous Institution Affiliated to

ANNA UNIVERSITY - CHENNAI 600 025

NOVEMBER 2025

SAVEETHA ENGINEERING COLLEGE

AUTONOMOUS

NAAC A GRADE

Affiliated to Anna University | Approved by AICTE

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Project report **" ONLINE VOTING SYSTEM "** is the bonafide work of **MURALI KRISHNA S (212223230129),** who carried out this project work under my supervision.

SIGNATURE

Associate Professor

SUPERVISOR

Dept of AI&DS

Saveetha Engineering College,
Thandalam, Chennai 602105

SIGNATURE

Dr. Karthi Govindharaju, M.E., Ph.D.,

Professor

HEAD OF THE DEPARTMENT

Dept of Artificial Intelligence
and Machine Learning,

Saveetha Engineering College,
Chennai 602105.

DATE OF THE VIVA VOCE EXAMINATION: ……

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to our esteemed Founder President **Dr. N. M. Veeraiyan**, our President **Dr. Saveetha Rajesh**, our Director **Dr. S. Rajesh**, and the entire management team for providing the essential infrastructure.

I extend my sincere appreciation to our principal, **Dr. V. Vijaya Chamundeeswari, M.Tech., Ph.D.,** for creating a supportive learning environment for this project.

I am very thankful to our Dean of ICT, **Mr. Obed Otto,** M.E., for facilitating a conducive atmosphere that allowed me to complete my project successfully.

My thanks go to **Dr. Karthi Govindharaju, M.E., Ph.D.,**, Professor and Head of the Department of Artificial Intelligence and Data Science at Saveetha Engineering College, for his generous support and for providing the necessary resources for my project work.

I would also like to express my profound gratitude to my Supervisor, **<<Supervisor Name with Designation>>**, and my Project Coordinator **Dr. N.S. Gowri Ganesh**, Associate Professor at Saveetha Engineering College, for their invaluable guidance, suggestions, and constant encouragement, which were instrumental in the successful completion of this project. Their timely support and insights during the review process were greatly appreciated.

I am grateful to all my college faculty, staff, and technicians for their cooperation throughout the project. Finally, I wish to acknowledge my loving parents, friends, and well-wishers for their encouragement in helping me achieve this milestone.

ABSTRACT

Our country, India is the largest democratic country in the world. So it is essential to make sure that the governing body is elected through a fair election. The project is mainly aimed at providing a secured and user friendly Online Voting System. The problem of voting is still critical in terms of safety and security. Which is inefficient and subpar because it needs a lot of labourers and takes a long time to process and broadcast the results. The system needs to alter in order to address these drawbacks and become effective. The additional feature of the model is that the online voter can confirm if his\ her vote has gone to correct candidate id. In this model a person can also vote from outside or inside her\his zone of preferred location. Nowadays with the rise in population the need for checking the validity of the voters has become a problem. As the modern communications and Internet, today are almost accessible electronically, the computer technology users, brings the increasing need for electronic services and their security.

Usages of new technology in the online voting process improve the elections in natural. After the industrialisation more number of people leave their native places and come to the cities for the job sake. But many of them still have their voter ids in the address of their native places. In online system mode, you can put a vote in any of the following places: native, job sake, and some other places. These drawbacks can overcome by Online Voting System. This is a voting system by which any voter can use his/her voting rights from anywhere in the country. Voter can cast their votes from anywhere in the country without visiting to voting booths, in highly secured way. That makes voting a fearless of violence and that increases the percentage of online voting. It provides enough security in the online voting system that it can reduce the number of dummy votes.

TABLE OF CONTENTS

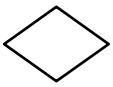
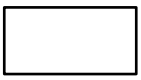
LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

OVS	**ONLINE VOTING SYSTEM**
EVM	**ELECTRONIC VOTING MACHINE**
OTP	**ONE TIME PASSWORD**
GUI	**GRAPHICAL USER INTERFACE**
HTML	**HYPERTEXT MARKUP LANGUAGE**

LIST OF SYMBOLS

S.NO.	SYMBOL NAME	SYMBOL
1.	Usecase	
2.	Actor	
3.	Process	
4.	Start	
5.	Decision	
6.	Unidirectional	
7.	Entity set	
8.	Stop	

Chapter 1

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

The **Online Voting System** is a web-based application developed to modernize and simplify the traditional voting process. It enables voters to cast their votes electronically in a secure, convenient, and efficient manner from any location with internet access. This system eliminates the need for manual paper-based voting, reduces human errors, and ensures faster vote counting and result generation.

The project aims to provide a transparent and reliable platform where voter authentication, vote casting, and vote counting are handled digitally. Each voter is assigned a unique login credential, ensuring that only authorized individuals can access the system and cast their votes. The system uses database management to store and validate voter information securely, preventing duplicate voting and unauthorized access.

Furthermore, the Online Voting System enhances the democratic process by making voting more accessible to people who may not be able to visit polling stations physically. It also minimizes administrative costs, time consumption, and the risk of tampering compared to traditional methods. Overall, this project demonstrates how technology can be effectively used to improve fairness, security, and efficiency in elections.

1.2 PROBLEM DEFINITION

In traditional voting systems, the process of conducting elections involves significant manual effort, time, and cost. Voters are required to visit designated polling stations to cast their votes, which often leads to long queues, human errors during vote counting, and challenges in verifying voter identities. Additionally, paper-based systems are prone to tampering, mismanagement, and fraudulent practices, which can affect the integrity of election results.

The main problem addressed by the **Online Voting System** is to design a secure, transparent, and user-friendly platform that ensures voter authentication, prevents duplicate voting, and automates the counting process to produce accurate results in real-time.

Chapter 2

LITERATURE SURVEY

2.1 INTRODUCTION

The **Literature Survey** provides an overview of previous research, methodologies, and systems developed in the field of electronic and online voting. It helps in understanding the existing approaches, their limitations, and the technological advancements that have contributed to the evolution of secure digital voting platforms.

Over the years, various researchers and developers have proposed online voting systems using technologies such as database management systems, encryption algorithms, and web-based authentication mechanisms to ensure voter security and data integrity. Early systems focused mainly on digitizing the voting process, while recent works have introduced advanced security features such as biometric verification, blockchain integration, and end-to-end encryption to enhance trust and transparency.

The review of existing literature highlights the common challenges faced by online voting systems, including data privacy, authentication reliability, and prevention of multiple voting. By analyzing these studies, the current project identifies gaps and proposes an improved model that ensures secure, efficient, and accessible online voting.

2.2 LITERATURE SURVEY

2.2.1 Helios: Web-based Open-Audit Voting.

Author Name : Ben Adida.

Year of Publish : 2008

Helios is one of the earliest open-audit online voting systems designed for web-based elections such as university or organizational voting. The system uses advanced cryptographic techniques to ensure ballot privacy and verifiability. Voters can verify that their votes were correctly recorded and included in the final tally without revealing their choices. Helios demonstrated that secure electronic voting could be achieved using standard web technologies, laying the foundation for

future online voting systems.

2.2.2 Scantegrity II: End-to-End Verifiability for Optical-Scan Election Systems

Author Name : David Chaum et al.

Year of Publish : 2008

Scantegrity II extends the traditional optical-scan voting method with invisible ink confirmation codes. When voters mark their ballots, unique codes are revealed, allowing them to verify online that their votes were included in the final count. This system ensures both voter privacy and verifiability, combining paper-based integrity with digital security. It was a major step toward creating hybridsystems that bridge manual and electronic verification.

2.2.3 Punchscan.

Author Name : Carback, Chaum, Clark, Essex, Popoveniuc et al.

Year of Publish : 2006-2007

Punchscan introduced a receipt-based end-to-end verifiable voting system that allows voters to check whether their votes were counted correctly, without exposing how they voted. It employs a two-layer paper ballot with cryptographic shuffling to maintain anonymity and prevent tampering. The project highlighted the importance of software independence — meaning the correctness of election outcomes should not depend solely on trusted software.

2.2.4 Prêt à Voter: A Systems Perspective

Author Name : P.Y.A. Ryan & T. Peacock

Year of Publish : 2005

Prêt à Voter proposed a novel cryptographic approach to electronic voting using randomized candidate lists. Each voter receives a ballot with candidates in a random order, and after casting their vote, a part of the ballot containing the order is destroyed to preserve secrecy. The remaining portion acts as a verifiable receipt. This system became one of the most influential frameworks for end-to-end verifiable electronic voting

2.2.5 Cryptographic Voting Protocols: A Systems Perspective

Author Name : Chris Karlof, N. Sastry & D. Wagner

Year of Publish : 2005

This study analyzed the design and implementation challenges of cryptographic voting protocols from a system-level perspective. It pointed out that while cryptographic security is crucial, practical deployment requires considerations like usability, voter authentication, and system scalability. The paper emphasized the balance between theoretical security and real-world practicality, influencing how later systems were designed.

2.2.6 An Overview of End-to-End Verifiable Voting Systems

Author Name : Various Authors

Year of Publish : 2013-2016

This survey paper reviewed the evolution of end-to-end verifiable voting systems and discussed the privacy, security, and usability features of major voting protocols. It compared systems like Helios, Scantegrity, and Prêt à Voter, highlighting their advantages and limitations. The study concluded that while significant progress had been made, scalability and voter accessibility remained key challenges in implementing nationwide electronic elections.

2.2.7 Attacking Paper-Based End-to-End Voting Systems

Author Name : J. Kelsey et al.

Year of Publish : 2015

This paper explored vulnerabilities in hybrid voting systems that combine paper and electronic verification. It demonstrated practical attacks that could manipulate ballots or compromise anonymity. The research stressed the importance of rigorous testing, cryptographic proofs, and secure audit mechanisms. It also motivated the development of more resilient protocols and voter-verifiable systems.

2.2.8 A Secure End-to-End Verifiable E-Voting System Using Blockchain

Author Name : Various Authors

Year of Publish : 2019-2021

Recent studies introduced blockchain technology to enhance the transparency and immutability of online voting systems. Blockchain ensures that once votes are recorded, they cannot be altered or deleted. Each transaction (vote) is cryptographically signed and stored on a distributed ledger, providing public verifiability and resistance to tampering. These systems address previous challenges like central database vulnerability and trust issues, marking a new phase in e-voting innovation.

2.1. LITERATURE SURVEY SUMMARY

S.No	Research	Technique	Features Used	Domain	Disadvantage / Advantage	Future Direction
1.	Dr.Kriti Patidar (2019)	Blockchain Network (e.g., Ethereum, Hyperledger)	Cryptographic Hashing, Digital Signatures, Smart Contracts	E-Governance	Immutability, Decentralization, Transparency,and Security against data tampering and fraud.	Integration of Biometrics for enhanced voter verification.
2.	Dr.Sushil kumar S.Salve (2025)	Embedded Systems Engineering, Secure Communication Protocols, and Real-Time Data Processing Mechanisms.	Cryptographic Routines (e.g., lightweight AES) for data integrity/ vote encryption	Electronic Voting Systems (E-VS)	Ensures end-to-end system reliability and enhanced tamper resistance	Further integration and leveraging of Blockchain Technology to lock up vote data

3.	D. L. Xu (2021)	Asymmetric Encryption	Decentralized and Non-Tampering	Electronic Voting Systems (E-VS)	Better suited for elections with more than two candidates.	Improving the efficiency and scalability of the encryption and anonymity- preserving process.

Chapter 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing voting process in many regions still relies on traditional methods such as paper- based ballots or electronic voting machines (EVMs). While these methods have been used for decades, they suffer from numerous limitations related to accessibility, efficiency, transparency, and security. In the conventional paper ballot system, voters must visit their designated polling stations in person to cast their votes. This process can be time-consuming, particularly in densely populated areas, and often discourages participation due to long queues, transportation difficulties, and limited voting hours.

People living far from their home constituencies—such as students, working professionals, and citizens residing abroad—are frequently unable to vote, leading to reduced voter turnout. Electronic Voting Machines (EVMs), though introduced to reduce manual counting errors, still require physical polling locations and trained staff for setup, supervision, and maintenance. EVMs are also prone to technical malfunctions, tampering, or unauthorized access if not properly secured.

3.2 DISADVANTAGES OF EXISTING SYSTEM

- The manual voting process involves setting up polling booths, verifying voter identities, and physically counting votes, which takes a significant amount of time.
- Traditional elections require large expenditures on manpower, printing ballots, transportation, and security arrangements.
- Manual counting and voter verification can lead to mistakes, miscounts, or misplacement of ballots, affecting accuracy.
- Voters must be physically present at polling stations, making it difficult for people living abroad, those with disabilities, or those in remote areas to participate.

3.3 PROPOSED SYSTEM

The proposed **Online Voting System** is designed to overcome the challenges and inefficiencies of the traditional voting process by leveraging modern web technologies, secure authentication mechanisms, and real-time data processing. This system enables authorized voters to cast their votes remotely from any location using a computer or mobile device with an internet connection. Each voter is assigned a unique login ID and password generated at the time of registration, ensuring that only verified users can access the voting portal.

The system uses secure encryption techniques to protect voter credentials and voting data, maintaining privacy and preventing unauthorized access or tampering. The voting interface is designed to be simple and user-friendly, allowing users to easily select their preferred candidate and submit their vote with a single click. Once a vote is cast, it is encrypted and stored securely in the database, making it impossible to alter or duplicate. The system includes an administrative panel that allows the election authority or administrator to manage voter registrations, monitor voting activities, generate election IDs, and publish results once the voting period ends.

3.4 ADVANTAGES OF PROPOSED SYSTEM

- The online voting system significantly reduces the time required for voting and counting, enabling faster result generation and declaration.
- The need for physical polling stations, printed ballots, and manual labor is minimized, which reduces the overall cost of conducting elections.
- The proposed system uses secure authentication methods such as OTP verification or biometric validation to ensure that only eligible voters can cast their votes.
- Automated vote counting eliminates human error, enduring accuarte and tamper-proof results

3.5 FEASIBILITY STUDY

A feasibility study is conducted to assess whether the proposed Online Voting System can be developed and implemented successfully within the available resources, time, and technical constraints. The study analyzes various factors such as technical, economic, and operational aspects to ensure the system's practicality and sustainability.

3.6 HARDWARE ENVIRONMENT

- Processor : Intel i5 and above
- Hard disk : 1 TB Minimum
- RAM : 4 GB or Higher Version
- Keyboard : 110 keys enhanced

3.7 SOFTWARE ENVIRONMENT

- Operating system : Windows 10 and 11
- Language : Python,HTML,CSS

3.8 TECHNOLOGIES USED

- IDE - Visual Studio
- Framework – Python flask
- My Sql

3.8.1 Python

Python is a high-level, interpreted programming language that is widely used in various domains such as web development, data science, artificial intelligence, scientific computing, and more. It was first released in 1991 and has since become one of the most popular programming languages in the world. Some key features of Python include:

- Easy to Learn: Python has a simple and easy-to-learn syntax, which makes it an ideal language for beginners.
- Interpreted Language: Python is an interpreted language, which means that the code is executed line by line, making it easier to test and debug.
- Cross-Platform: Python can be run on various platforms, including Windows, macOS, and Linux.
- Large Standard Library: Python has a large standard library that provides a wide range of built-in modules for various tasks, such as file I/O, regular expressions, networking, and more.
- Open Source: Python is open-source software, which means that the source code is freely available to anyone and can be modified and redistributed.
- Object-Oriented: Python is an object-oriented language, which means that it supports object-oriented programming concepts such as encapsulation, inheritance, and polymorphism.

3.8.2 My Sql

MySQL is an open-source **Relational Database Management System (RDBMS)** that uses **Structured Query Language (SQL)** to store, organize, and manage data efficiently. It is one of the most widely used database systems due to its speed, reliability, and ease of integration with web applications.

In the **Online Voting System**, MySQL plays a critical role in managing and maintaining all the essential data related to the voting process. It stores various entities such as **voter details, user authentication credentials, candidate information, and voting results** in well-structured tables. Each table is linked through relational keys, which helps maintain **data integrity and consistency**.

MySQL provides **high security** features, including user authentication, role-based access control, and encrypted connections, ensuring that sensitive data such as voter information and votes remain confidential. The database supports **ACID (Atomicity,**

Consistency, Isolation, Durability) properties, ensuring that all transactions are executed accurately and completely without any data loss or corruption.

Chapter 4

SYSTEM DESIGN

4.1 ENTITY-RELATIONSHIP DIAGRAM

‘The relationships between database entities can be seen using an entity- relationship diagram (

ERD). The entities and relationships depicted in an ERD can have further detail added to them via data object descriptions. In software engineering, conceptual and abstract data descriptions are represented via entity- relationship models (ERMs). Entity-relationship diagrams (ERDs), entity-relationship diagrams (ER), or simply entity diagrams are the terms used to describe the resulting visual representations of data structures that contain relationships between entities. As such, a data flow diagram can serve dual purposes. To demonstrate how data is transformed across the system. To provide an example of the procedures that affect the data flow.

Fig 4.1 Entity Relationship Diagram

4.2 DATA FLOW DIAGRAM (DFD)

The whole system is shown as a single process in a level DFD. Each step in the system's assembly process, including all intermediate steps, are recorded here. The "basic system model" consists of this and 2-level data flow diagrams.They are often elements of a formal methodology such as Structured Systems Analysis and Design Method (SSADM). Superficially, DFDs can resemble flow charts or Unified Modeling Language (UML), but they are not meant to represent details of software logic. DFDs make it easy to depict the business requirements of applications by representing the sequence of process steps and flow of information using a graphical representation or visual representation rather than a textual description.

Fig 4.2.1 Level 0 of Data Flow Diagram

Fig 4.2.2 Level 1 of Data Flow Diagram

4.3 UML DIAGRAMS

4.3.1 Use Case Diagram

A use case diagram is a type of Unified Modeling Language (UML) diagram that represents the interactions between a system and its actors, and the various use cases that the system supports. It is a visual representation of the functional requirements of the system and the actors that interact with it. Use case diagrams typically include the following elements:

- Actors: Actors are external entities that interact with the system. They can be human users, other systems, or devices.
- Use Cases: Use cases are the specific functions or tasks that the system can perform. Each use case represents a specific interaction between an actor and the system.
- Relationships: Relationships are used to indicate how the actors and use cases are related to each other. The two main relationships in a use case diagram are "uses" and "extends". "Uses" relationship indicates that an actor uses a specific use case, while "extends" relationship indicates that a use case extends or adds functionality to another use case.
- System Boundary: The system boundary is a box that contains all the actors and use cases in the system. It represents the physical or logical boundary of the system being modeled.

4.3.2 Class Diagram

In essence, this is a "context diagram," another name for a contextual diagram. It simply stands for the very highest point, the 0 Level, of the procedure. As a whole, the system is shown as a single process, and the connection to externalities is shown in an abstract manner.

- A + indicates a publicly accessible characteristic or action.
- A - a privately accessible one.
- A # a protected one.
- A - denotes private attributes or operations.

4.3.3 Sequence Diagram

These are another type of interaction-based diagram used to display the workings of the system. They record the conditions under which objects and processes cooperate. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.

Chapter 5

SYSTEM ARCHITECTURE

5.1 ARCHITECTURE DIAGRAM

This graphic provides a concise and understandable description of all the entities currently integrated into the system. The diagram shows how the many actions and choices are linked together. You might say that the whole process and how it was carried out is a picture. The figure below shows the functional connections between various entities.

Fig 5.1 Architecture Diagram

The system architecture of the fig 5.1 clearly shows that the input is given as video then using the YOLO v5 model the accidents are detected and classified based on the probability .After the detection of the accident the alert message is sent to the defined user through SMS.

5.2 ALGORITHMS

5.2.1 User Authentication Algorithm

The **Online Voting System** operates using a series of algorithms designed to ensure security, accuracy, and reliability throughout the voting process. The first and most crucial algorithm is the **User Authentication Algorithm**, which verifies voter identities before granting access. During login, the system prompts the user to enter their voter ID and password. These credentials are then checked against securely stored, hashed passwords in the database using algorithms such as **SHA-256** or **bcrypt**. If the information matches, access is granted; otherwise, the system denies entry. This step ensures that only legitimate and registered users can participate in the election process.

Once the user is authenticated, the **Vote Casting Algorithm** takes effect. This algorithm ensures that each voter can vote only once by checking the `has_voted` flag in the database. If the voter has not yet voted, the system presents the list of candidates, and upon selection, the chosen vote is **encrypted using RSA (Asymmetric Encryption)** for confidentiality before being stored. The system then updates the voter's status to indicate that they have voted, preventing duplicate votes.

Step 1: Start

Step 2: User Authentication

- Input: Voter ID and Password.
- If valid → Proceed.
- Else → Show "Invalid login" and terminate.

Step 3: Check Voting Eligibility

- Verify if the voter has already voted (has_voted = False).
- If not voted → Continue.
- If already voted → Deny access.

Step 4: Display Election & Candidate List

- Fetch from the election and candidate database.

Step 5: Cast Vote

- Voter selects a candidate.
- System encrypts the vote before storing it.

Step 6: Store Vote

- Save the encrypted vote in the vote database.
- Update has_voted = True for the voter.

Step 7: Count Votes

- After voting ends, decrypt votes securely.
- Count votes for each candidate.

Step 8: Display Results

- Publish results to admin and optionally to voters.

Step 9: End

Chapter 6

SYSTEM IMPLEMENTATION

6.1 MODULE 1: USER AUTHENTICATION AND REGISTRATION

This module handles the registration and authentication of voters and administrators. New voters can register by providing valid identification details, which are verified against the voter database before approval. Once registered, the voter receives a secure login credential. The authentication process uses encrypted passwords to ensure data privacy and prevent unauthorized access. The system validates login credentials before allowing access to the voting portal or admin dashboard.

6.2 MODULE 2: CANDIDATE MANAGEMENT

This module is managed by the administrator, who adds and updates candidate details for the election. Each candidate's information, including name, party, and description, is stored securely in the system database. The admin also defines election parameters such as election name, start date, and end date. The candidate data is retrieved dynamically during the voting process for display to eligible voters.

6.3 MODULE 3: VOTING PROCESS

This module is the core of the system. Once authenticated, voters can view the list of active elections and their respective candidates. The voter selects one candidate per election, and the vote is then encrypted using a secure cryptographic algorithm (such as RSA) before being stored in the database. The system ensures that each voter can vote only once by updating the has_voted status after a successful vote submission.

6.4 MODULE 4: DATABASE MANAGEMENT

The database is the backbone of the Online Voting System. It maintains all critical data such as voter information, candidate details, election records, and encrypted votes. The system uses MySQL as its relational database, designed with normalized tables to avoid redundancy and maintain data integrity.

Database access is role-based — administrators have privileges to manage elections and candidates, while voters can only view and vote. Foreign key relationships are defined to maintain data consistency across tables.

6.5 MODULE 5: RESULT GENERATION AND DISPLAY

Once the election period concludes, the administrator triggers the result generation process. The system retrieves all encrypted votes from the database and decrypts them using the RSA private key. Each decrypted vote is then matched to its candidate ID, and a counter is incremented accordingly.

The results are calculated automatically and displayed in a structured tabular format, showing the total number of votes received by each candidate. The system also highlights the winning candidate. Graphical visualizations (such as bar or pie charts) may be generated for better readability.

6.7 MODULE 6: SECURITY IMPLEMENTATION

Security is a fundamental requirement in an online voting environment. This module ensures end-to-end protection of user data, votes, and system integrity. All communications between the client and server are secured using **SSL/TLS protocols** to prevent interception. Votes are encrypted before transmission and remain confidential until decrypted for counting.

The system implements **two-factor authentication (2FA)** for administrators to prevent unauthorized access. Additionally, all activities (logins, votes, and modifications) are logged in an **audit trail** to ensure traceability and accountability. CAPTCHA verification is used during registration and login to prevent automated bots from interacting with the system.

Chapter 7

SYSTEM TESTING

7.1 BLACK BOX TESTING

During this kind of testing, the user does not have access to or knowledge of the internal structure or specifics of the data item being tested. In this method, test cases are generated or designed only based on the input and output values, and prior knowledge of either the design or the code is not necessary. The testers are just conscious of knowing about what is thought to be able to do, but they do not know how it is able to do it.

Fig 7.1 Black Box Testing

For example, without having any knowledge of the inner workings of the website, we test the web pages by using a browser, then we authorize the input, and last, we test and validate the outputs against the intended result.

7.2 WHITE BOX TESTING

During this kind of testing, the user is aware of the internal structure and details of the data item, or they have access to such information. In this process, test cases are constructed by referring to the code. Programming is extremely knowledgeable of the manner in which the application of knowledge is significant. White Box Testing is so called because, as we all know, in the tester's eyes it appears to be a white box, and on the inside, everyone can see clearly.This is how the testing got its name.

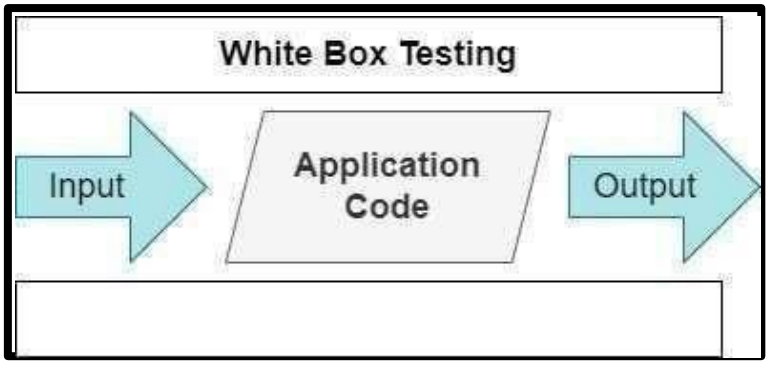


Fig 7.2 White Box Testing

As an instance, a tester and a developer examine the code that is implemented in each field of a website, determine which inputs are acceptable and which are not, and then check the output to ensure it produces the desired result. In addition, the decision is reached by analyzing the code that is really used.

7.3 TEST CASES

TEST REPORT: 01

PRODUCT: ONLINE VOTING SYSTEM

USE CASE: VOTER REGISTERATION

TEST CASE ID	TEST CASE/ ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/FAIL
01	Enter valid voter details and submit the registration form	Registration successful message displayed	As Executed	PASS

02	Try to register with an existing Voter ID	Error message “Voter already registered” displayed	As Executed	PASS
03	Leave mandatory fields empty and submit	Validation message “All fields are required” displayed	As Executed	PASS

Table-7.3.1 Test Case For Voter Registration

TEST REPORT: 02

PRODUCT: ONLINE VOTING SYSTEM

USE CASE: LOGIN

TEST CASE ID	TEST CASE/ ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/FAIL
01	Enter valid username and password	Redirected to voter dashboard	As Executed	PASS
02	Enter invalid password	Error message “Invalid credentials” displayed	As Executed	PASS
03	Leave username or password blank	Error message “Fields cannot be empty” displayed	As Executed	PASS

Table-7.3.2 Test Case For Login

Chapter 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 CONCLUSION

The development and analysis of the Online Voting System demonstrate its profound potential to modernize and streamline democratic processes, achieving core objectives of **increased accessibility, efficiency, and election integrity**. By allowing eligible voters to cast encrypted ballots from any internet-enabled location, the system promises to significantly boost voter participation while drastically reducing the time and immense cost of manual tabulation, enabling near real-time results. However, the system's viability ultimately rests on its ability to overcome the critical challenge of public trust, which requires more than just technical security; it demands **absolute transparency and verifiability**. Moving forward, the OVS must evolve through the integration of technologies like **Blockchain or Distributed Ledger Technology** to decentralize data and ensure immutable records, coupled with rigorous, independent auditing protocols. The OVS is a necessary and transformative tool for civic engagement, provided its future design continues to prioritize and prove its resilience, anonymity, and end-to-end auditability.

8.2 FUTURE ENCHANCEMENT

The most significant enhancement involves integrating the OVS with a Permissioned Blockchain or Distributed Ledger Technology (DLT). Each encrypted vote would be recorded as a transaction block, leveraging the blockchain's core attributes of immutability and transparency. This eliminates reliance on a single central server, drastically reducing the risk of internal administrative manipulation or a single point of failure. The public could be allowed to view the encrypted, auditable log of all transactions, providing universal verifiability that every vote cast was received and counted without alteration, all while maintaining the voter's anonymity.

To completely counter identity fraud and double-voting, the authentication layer should evolve to include Biometric Verification (e.g., facial or fingerprint recognition) during the login or ballot issuance phase. Crucially, this must be paired with Zero-Knowledge Proofs (ZKPs). ZKPs

are a cryptographic technique that allows the system to verify the voter's identity and eligibility (e.g., "Yes, this voter is a registered citizen over 18, and has not yet voted") without revealing the underlying personal data to the voting server itself. This maintains a robust check on eligibility while maximally protecting voter privacy.

Chapter 9

APPENDIX 1 – SAMPLE CODING

homePage.py:

```
import subprocess as sb_p
import tkinter as tk
from tkinter import *
from Admin import AdmLogin
from voter import voterLogin
from PIL import Image, ImageTk
import pygame
import threading
import os

# Initialize pygame mixer for audio
def init_audio():
    try:
        pygame.mixer.init()
        return True
    except:
        print("Audio initialization failed")
        return False

# Function to play background music in a separate thread
def play_background_music():
    try:
        # Replace with your actual music file path
        music_file = "audio/song.mp3"
        if os.path.exists(music_file):
            pygame.mixer.music.load(music_file)
            pygame.mixer.music.play(-1)  # -1 means loop indefinitely
            print("Background music started")
        else:
            print(f"Music file not found: {music_file}")
    except Exception as e:
        print(f"Error playing music: {e}")

# Function to stop background music
def stop_background_music():
    try:
        pygame.mixer.music.stop()
        print("Background music stopped")
```

```
    except:
        pass

def Home(root, frame1, frame2):
    for frame in root.winfo_children():
        for widget in frame.winfo_children():
            widget.destroy()

    # Set background image using PIL for PNG support
    try:
        bg_image = Image.open("img/bg.png")
        bg_image = bg_image.resize((root.winfo_screenwidth(), root.winfo_screenheight()),
Image.Resampling.LANCZOS)
        bg_photo = ImageTk.PhotoImage(bg_image)
        bg_label = Label(frame1, image=bg_photo)
        bg_label.image = bg_photo
        bg_label.place(x=0, y=0, relwidth=1, relheight=1)
    except Exception as e:
        print(f"Background image error: {e}")
        frame1.config(bg='lightgray')

    # Remove frame2 (navigation bar) and place Home button directly on background
    frame2.pack_forget()

    # Home button directly on background at top left
    home_btn = Button(frame1, text="· Home", font=('Helvetica', 14, 'bold'),
                      command=lambda: Home(root, frame1, frame2),
                      bg='lightblue', fg='black', relief='raised', bd=3,
                      width=12, height=1, cursor='hand2')
    home_btn.place(x=20, y=20)

    # Music control button
    music_btn = Button(frame1, text="· Mute", font=('Helvetica', 12, 'bold'),
                       command=stop_background_music,
                       bg='red', fg='white', relief='raised', bd=2,
                       width=8, height=1, cursor='hand2')
    music_btn.place(x=20, y=70)

    root.title("Home")

    # Title directly on background
    title_label = Label(frame1, text="Online Voting System", font=('Helvetica', 35, 'bold'),
                        bg='black', fg='white', bd=0)
    title_label.place(relx=0.5, rely=0.1, anchor='center')

    # Buttons directly on background with transparent effect
    admin = Button(frame1, text="Admin Login", width=20, height=3, font=('Helvetica', 18, 'bold'),
```

```
            command=lambda: AdmLogin(root, frame1), bg='#4CAF50', fg='white',
            relief='raised', bd=4, cursor='hand2')
    admin.place(relx=0.5, rely=0.4, anchor='center')

    voter = Button(frame1, text="Voter Login", width=20, height=3, font=('Helvetica', 18, 'bold'),
            command=lambda: voterLogin(root, frame1), bg='#2196F3', fg='white',
            relief='raised', bd=4, cursor='hand2')
    voter.place(relx=0.5, rely=0.55, anchor='center')

    newTab = Button(frame1, text="New Window", width=20, height=3, font=('Helvetica', 18, 'bold'),
             command=lambda: sb_p.call('start python homePage.py', shell=True),
             bg='#FF9800', fg='white', relief='raised', bd=4, cursor='hand2')
    newTab.place(relx=0.5, rely=0.7, anchor='center')

    frame1.pack(fill=BOTH, expand=True)
    root.mainloop()

def new_home():
    root = Tk()
    root.attributes('-fullscreen', True)
    root.bind('<Escape>', lambda e: root.attributes('-fullscreen', False))

    # Initialize audio and play background music in a separate thread
    if init_audio():
        music_thread = threading.Thread(target=play_background_music, daemon=True)
        music_thread.start()

    frame1 = Frame(root)
    frame2 = Frame(root)  # Keep frame2 but we'll hide it
    Home(root, frame1, frame2)

if __name__ == "__main__":
    new_home()
```

Admin.py

```
import subprocess as sb_p
import tkinter as tk
import registerVoter as regV
import admFunc as adFunc
from tkinter import *
from registerVoter import *
from admFunc import *

def AdminHome(root,frame1,frame3):
    root.title("Admin")
```

```
    for widget in frame1.winfo_children():
        widget.destroy()

    # Admin button at top left corner with larger size (matching home page)
    admin_btn = Button(frame3, text="Admin", font=('Helvetica', 14, 'bold'),
                       command = lambda: AdminHome(root, frame1, frame3),
                       bg='lightblue', relief='raised', bd=3,
                       width=12, height=2)
    admin_btn.grid(row=1, column=0, sticky='w', padx=20, pady=10)

    # Add some spacing
    Label(frame3, text="", width=100).grid(row=1, column=1)

    frame3.pack(side=TOP, fill=X)

    Label(frame1, text="Admin Dashboard", font=('Helvetica', 30, 'bold')).grid(row=0, column=1,
rowspan=1, pady=20)
    Label(frame1, text="").grid(row=1, column=0)

    # Run Server - Larger button matching home page style
    runServer = Button(frame1, text="Run Server", width=20, height=3, font=('Helvetica', 16, 'bold'),
                       command=lambda: sb_p.call('start python Server.py', shell=True),
                       bg='#4CAF50', fg='white', relief='raised', bd=4)

    # Register Voter - Larger button matching home page style
    registerVoter = Button(frame1, text="Register Voter", width=20, height=3, font=('Helvetica', 16,
'bold'),
                           command=lambda: regV.Register(root, frame1),
                           bg='#2196F3', fg='white', relief='raised', bd=4)

    # Show Votes - Larger button matching home page style
    showVotes = Button(frame1, text="Show Votes", width=20, height=3, font=('Helvetica', 16, 'bold'),
                       command=lambda: adFunc.showVotes(root, frame1),
                       bg='#FF9800', fg='white', relief='raised', bd=4)

    # Reset All - Larger button matching home page style
    reset = Button(frame1, text="Reset All", width=20, height=3, font=('Helvetica', 16, 'bold'),
                   command=lambda: adFunc.resetAll(root, frame1),
                   bg='#f44336', fg='white', relief='raised', bd=4)

    # Adjusted spacing to better center the buttons vertically
    Label(frame1, text="", height=1).grid(row=2, column=0)
    runServer.grid(row=3, column=1, columnspan=2, pady=15)

    Label(frame1, text="", height=1).grid(row=4, column=0)
    registerVoter.grid(row=5, column=1, columnspan=2, pady=15)

    Label(frame1, text="", height=1).grid(row=6, column=0)
```

```
    showVotes.grid(row=7, column=1, columnspan=2, pady=15)

    Label(frame1, text="", height=1).grid(row=8, column=0)
    reset.grid(row=9, column=1, columnspan=2, pady=15)

    # Add bottom padding to center the buttons
    Label(frame1, text="", height=2).grid(row=10, column=0)

    frame1.pack(expand=True)
    root.mainloop()

def log_admin(root,frame1,admin_ID,password):

    if(admin_ID=="Admin" and password=="admin"):
        frame3 = root.winfo_children()[1]
        AdminHome(root, frame1, frame3)
    else:
        msg = Message(frame1, text="Either ID or Password is Incorrect", width=500)
        msg.grid(row=6, column=0, columnspan=5)

def AdmLogin(root,frame1):

    root.title("Admin Login")
    for widget in frame1.winfo_children():
        widget.destroy()

    # Create a main container for better centering
    main_container = Frame(frame1)
    main_container.pack(expand=True)

    # Centered title - using columnspan to center properly
    Label(main_container, text="Admin Login", font=('Helvetica', 25, 'bold')).grid(row=0, column=0,
columnspan=2, pady=40)

    # Labels
    Label(main_container, text="Admin ID:", font=('Helvetica', 16, 'bold')).grid(row=1, column=0,
padx=20, pady=20, sticky='e')
    Label(main_container, text="Password:", font=('Helvetica', 16, 'bold')).grid(row=2, column=0,
padx=20, pady=20, sticky='e')

    admin_ID = tk.StringVar()
    password = tk.StringVar()

    # Entry fields
    e1 = Entry(main_container, textvariable=admin_ID, font=('Helvetica', 14), width=20)
    e1.grid(row=1, column=1, padx=20, pady=20)

    e2 = Entry(main_container, textvariable=password, show='*', font=('Helvetica', 14), width=20)
```

```
    e2.grid(row=2, column=1, padx=20, pady=20)

    # Function to handle login
    def perform_login():
        log_admin(root, frame1, admin_ID.get(), password.get())

    # Centered login button - using columnspan to center across both columns
    sub = Button(main_container, text="Login", width=15, height=2, font=('Helvetica', 16, 'bold'),
                 command=perform_login,
                 bg='#4CAF50', fg='white', relief='raised', bd=4)
    sub.grid(row=3, column=0, columnspan=2, pady=30)

    # Bind Enter key to login function
    root.bind('<Return>', lambda event: perform_login())
    e1.bind('<Return>', lambda event: perform_login())
    e2.bind('<Return>', lambda event: perform_login())

    root.mainloop()

# if __name__ == "__main__":
#         root = Tk()
#         root.geometry('500x500')
#         frame1 = Frame(root)
#         frame3 = Frame(root)
#         AdminHome(root,frame1,frame3)
```

admnFunc.py:

```
import tkinter as tk
import dframe as df
from tkinter import *
from dframe import *
from PIL import ImageTk,Image
import Admin  # Import the Admin module to access AdminHome function

def resetAll(root,frame1):
    #df.count_reset()
    #df.reset_voter_list()
    #df.reset_cand_list()
    Label(frame1, text="").grid(row = 10,column = 0)
    msg = Message(frame1, text="Reset Complete", width=500)
    msg.grid(row = 11, column = 0, columnspan = 5)

def showVotes(root,frame1):

    result = df.show_result()
    root.title("Votes")
```

```
    for widget in frame1.winfo_children():
        widget.destroy()

    Label(frame1, text="Vote Count", font=('Helvetica', 25, 'bold')).grid(row = 0, column = 1,
rowspan=1, pady=30)
    Label(frame1, text="").grid(row = 1,column = 0)

    vote = StringVar(frame1,"-1")

    # BJP - Larger images and better spacing
    bjpLogo =
ImageTk.PhotoImage((Image.open("img/bjp.png")).resize((45,45),Image.Resampling.LANCZOS))
    bjpImg = Label(frame1, image=bjpLogo).grid(row = 2,column = 0, padx=20, pady=10)

    # Congress - Larger images and better spacing
    congLogo =
ImageTk.PhotoImage((Image.open("img/cong.jpg")).resize((35,48),Image.Resampling.LANCZOS))
    congImg = Label(frame1, image=congLogo).grid(row = 3,column = 0, padx=20, pady=10)

    # DMK - Larger images and better spacing
    dmkLogo =
ImageTk.PhotoImage((Image.open("img/dmk.jpg")).resize((45,35),Image.Resampling.LANCZOS))
    dmkImg = Label(frame1, image=dmkLogo).grid(row = 4,column = 0, padx=20, pady=10)

    # ADMK - Larger images and better spacing
    admkLogo =
ImageTk.PhotoImage((Image.open("img/admk.jpg")).resize((45,40),Image.Resampling.LANCZOS))
    admkImg = Label(frame1, image=admkLogo).grid(row = 5,column = 0, padx=20, pady=10)

    # TVK - Larger images and better spacing
    tvkLogo =
ImageTk.PhotoImage((Image.open("img/tvk.jpg")).resize((45,40),Image.Resampling.LANCZOS))
    tvkImg = Label(frame1, image=tvkLogo).grid(row = 6,column = 0, padx=20, pady=10)

    # NOTA - Larger images and better spacing
    notaLogo =
ImageTk.PhotoImage((Image.open("img/nota.jpg")).resize((40,30),Image.Resampling.LANCZOS))
    notaImg = Label(frame1, image=notaLogo).grid(row = 7,column = 0, padx=20, pady=10)

    # Party labels with larger font and better spacing
    Label(frame1, text=" BJP          :      ", font=('Helvetica', 16, 'bold')).grid(row = 2, column = 1,
pady=10)
    Label(frame1, text=result['bjp'], font=('Helvetica', 16, 'bold')).grid(row = 2, column = 2, pady=10)

    Label(frame1, text=" Congress      :      ", font=('Helvetica', 16, 'bold')).grid(row = 3, column = 1,
pady=10)
    Label(frame1, text=result['cong'], font=('Helvetica', 16, 'bold')).grid(row = 3, column = 2, pady=10)
```

```
    Label(frame1, text=" DMK              :       ", font=('Helvetica', 16, 'bold')).grid(row = 4, column = 1,
pady=10)
    Label(frame1, text=result['dmk'], font=('Helvetica', 16, 'bold')).grid(row = 4, column = 2, pady=10)

    Label(frame1, text=" ADMK           :       ", font=('Helvetica', 16, 'bold')).grid(row = 5, column = 1,
pady=10)
    Label(frame1, text=result['admk'], font=('Helvetica', 16, 'bold')).grid(row = 5, column = 2,
pady=10)

    Label(frame1, text=" TVK              :       ", font=('Helvetica', 16, 'bold')).grid(row = 6, column = 1,
pady=10)
    Label(frame1, text=result['tvk'], font=('Helvetica', 16, 'bold')).grid(row = 6, column = 2, pady=10)

    Label(frame1, text=" NOTA           :       ", font=('Helvetica', 16, 'bold')).grid(row = 7, column = 1,
pady=10)
    Label(frame1, text=result['nota'], font=('Helvetica', 16, 'bold')).grid(row = 7, column = 2, pady=10)

    # Fixed Back button - goes back to Admin home
    def go_back():
        # Clear current frame
        for widget in frame1.winfo_children():
            widget.destroy()
        # Call AdminHome function
        from Admin import AdminHome
        frame3 = root.winfo_children()[1]  # Get the frame3 from root
        AdminHome(root, frame1, frame3)

    back_btn = Button(frame1, text="Back", width=15, height=2, font=('Helvetica', 14, 'bold'),
                      command=go_back,
                      bg='#f44336', fg='white', relief='raised', bd=3)
    back_btn.grid(row=8, column=1, columnspan=2, pady=30)

    frame1.pack()
    root.mainloop()

# if __name__ == "__main__":
#         root = Tk()
#         root.geometry('500x500')
#         frame1 = Frame(root)
#         showVotes(root,frame1)
```

dframe.py:

```
import pandas as pd
from pathlib import Path
import os
```

```
# path = Path("C:/Users/Desktop/Sem-5/CS301 CN/Project/Voting/database")
path = Path("database")

# Ensure database directory exists
path.mkdir(exist_ok=True)

def initialize_candidate_list():
    """Initialize candidate list with new parties if it doesn't exist"""
    cand_file = path / 'cand_list.csv'
    if not cand_file.exists():
        reset_cand_list()
        print("Candidate list initialized with new parties")

def count_reset():
    df=pd.read_csv(path/'voterList.csv')
    df=df[['voter_id','Name','Gender','Zone','City','Passw','hasVoted']]
    for index, row in df.iterrows():
        df['hasVoted'].iloc[index]=0
    df.to_csv(path/'voterList.csv')

    df=pd.read_csv(path/'cand_list.csv')
    df=df[['Sign','Name','Vote Count']]
    for index, row in df.iterrows():
        df['Vote Count'].iloc[index]=0
    df.to_csv (path/'cand_list.csv')

def reset_voter_list():
    df = pd.DataFrame(columns=['voter_id','Name','Gender','Zone','City','Passw','hasVoted'])
    df=df[['voter_id','Name','Gender','Zone','City','Passw','hasVoted']]
    df.to_csv(path/'voterList.csv')

def reset_cand_list():
    # Updated with new parties: DMK, ADMK, TVK instead of AAP and Shiv Sena
    df = pd.DataFrame({
        'Sign': ['bjp', 'cong', 'dmk', 'admk', 'tvk', 'nota'],
        'Name': ['BJP', 'Congress', 'DMK', 'ADMK', 'TVK', 'NOTA'],
        'Vote Count': [0, 0, 0, 0, 0, 0]
    })
    df=df[['Sign','Name','Vote Count']]
    df.to_csv(path/'cand_list.csv')
    print("Candidate list reset with new parties")

def verify(vid,passw):
    # Initialize candidate list on first run
    initialize_candidate_list()

    df=pd.read_csv(path/'voterList.csv')
    df=df[['voter_id','Passw','hasVoted']]
```

```
    for index, row in df.iterrows():
        if df['voter_id'].iloc[index]==vid and df['Passw'].iloc[index]==passw:
            return True
    return False

def isEligible(vid):
    df=pd.read_csv(path/'voterList.csv')
    df=df[['voter_id','Name','Gender','Zone','City','Passw','hasVoted']]
    for index, row in df.iterrows():
        if df['voter_id'].iloc[index]==vid and df['hasVoted'].iloc[index]==0:
            return True
    return False

def vote_update(st,vid):
    if isEligible(vid):
        df=pd.read_csv (path/'cand_list.csv')
        df=df[['Sign','Name','Vote Count']]
        for index, row in df.iterrows():
            if df['Sign'].iloc[index]==st:
                df['Vote Count'].iloc[index]+=1

        df.to_csv (path/'cand_list.csv')

        df=pd.read_csv(path/'voterList.csv')
        df=df[['voter_id','Name','Gender','Zone','City','Passw','hasVoted']]
        for index, row in df.iterrows():
            if df['voter_id'].iloc[index]==vid:
                df['hasVoted'].iloc[index]=1

        df.to_csv(path/'voterList.csv')

        return True
    return False

def show_result():
    # Initialize candidate list on first run
    initialize_candidate_list()

    df=pd.read_csv (path/'cand_list.csv')
    df=df[['Sign','Name','Vote Count']]
    v_cnt = {}
    for index, row in df.iterrows():
        v_cnt[df['Sign'].iloc[index]] = df['Vote Count'].iloc[index]
    return v_cnt

def taking_data_voter(name,gender,zone,city,passw):
    # Initialize candidate list on first run
    initialize_candidate_list()
```

```
    df=pd.read_csv(path/'voterList.csv')
    df=df[['voter_id','Name','Gender','Zone','City','Passw','hasVoted']]
    row,col=df.shape
    if row==0:
        vid = 10001
        df = pd.DataFrame({"voter_id":[vid],
                    "Name":[name],
                    "Gender":[gender],
                    "Zone":[zone],
                    "City":[city],
                    "Passw":[passw],
                    "hasVoted":[0]},)
    else:
        vid=df['voter_id'].iloc[-1]+1
        df1 = pd.DataFrame({"voter_id":[vid],
                    "Name":[name],
                    "Gender":[gender],
                    "Zone":[zone],
                    "City":[city],
                    "Passw":[passw],
                    "hasVoted":[0]},)

        df = pd.concat([df, df1],ignore_index=True)

    df.to_csv(path/'voterList.csv')

    return vid

# Initialize on import
initialize_candidate_list()
```

registerVoter.py:

```
import tkinter as tk
import dframe as df
import Admin as adm
from tkinter import ttk
from Admin import *
from tkinter import *
from dframe import *

def reg_server(root,frame1,name,sex,zone,city,passw):
    if(passw=='' or passw==' '):
        msg = Message(frame1, text="Error: Missing Fields", width=500, font=('Helvetica', 14))
        msg.grid(row = 10, column = 0, columnspan = 5, pady=10)
        return -1
```

```
    vid = df.taking_data_voter(name, sex, zone, city, passw)
    for widget in frame1.winfo_children():
        widget.destroy()

    # Create main container for centering
    main_container = Frame(frame1)
    main_container.pack(expand=True)

    txt = "Registered Voter with\n\n VOTER I.D. = " + str(vid)
    Label(main_container, text=txt, font=('Helvetica', 20, 'bold'), justify='center').grid(row = 0, column
= 0, pady=100)

    # Function to go back to Admin
    def go_back_to_admin():
        # Clear current frame
        for widget in frame1.winfo_children():
            widget.destroy()
        # Import and call AdminHome
        from Admin import AdminHome
        frame3 = root.winfo_children()[1]  # Get the frame3 from root
        AdminHome(root, frame1, frame3)

    # Add a working back button
    back_btn = Button(main_container, text="Back to Admin", width=15, height=2, font=('Helvetica',
14, 'bold'),
                      command=go_back_to_admin,
                      bg='#2196F3', fg='white', relief='raised', bd=3)
    back_btn.grid(row=1, column=0, pady=30)

def Register(root,frame1):

    root.title("Register Voter")
    for widget in frame1.winfo_children():
        widget.destroy()

    # Create main container for better layout
    main_container = Frame(frame1)
    main_container.pack(expand=True, pady=20)

    Label(main_container, text="Register Voter", font=('Helvetica', 25, 'bold')).grid(row = 0, column =
1, columnspan=2, pady=30)

    # Increased spacing between fields
    Label(main_container, text="", height=1).grid(row = 1, column = 0)

    # Labels with larger font
    Label(main_container, text="Name:", anchor="e", justify=LEFT, font=('Helvetica', 16)).grid(row =
```

```
2, column = 0, padx=20, pady=15, sticky='e')
    Label(main_container, text="Sex:", anchor="e", justify=LEFT, font=('Helvetica', 16)).grid(row = 3,
column = 0, padx=20, pady=15, sticky='e')
    Label(main_container, text="Zone:", anchor="e", justify=LEFT, font=('Helvetica', 16)).grid(row =
4, column = 0, padx=20, pady=15, sticky='e')
    Label(main_container, text="City:", anchor="e", justify=LEFT, font=('Helvetica', 16)).grid(row =
5, column = 0, padx=20, pady=15, sticky='e')
    Label(main_container, text="Password:", anchor="e", justify=LEFT, font=('Helvetica',
16)).grid(row = 6, column = 0, padx=20, pady=15, sticky='e')

    name = tk.StringVar()
    sex = tk.StringVar()
    zone = tk.StringVar()
    city = tk.StringVar()
    password = tk.StringVar()

    # Entry fields with larger font
    e2 = Entry(main_container, textvariable = name, font=('Helvetica', 14), width=20)
    e2.grid(row = 2, column = 1, padx=20, pady=15)

    e5 = Entry(main_container, textvariable = zone, font=('Helvetica', 14), width=20)
    e5.grid(row = 4, column = 1, padx=20, pady=15)

    e6 = Entry(main_container, textvariable = city, font=('Helvetica', 14), width=20)
    e6.grid(row = 5, column = 1, padx=20, pady=15)

    e7 = Entry(main_container, textvariable = password, font=('Helvetica', 14), width=20, show='*')
    e7.grid(row = 6, column = 1, padx=20, pady=15)

    # Combobox with larger font
    e4 = ttk.Combobox(main_container, textvariable = sex, width=18, font=('Helvetica', 14))
    e4['values'] = ("Male", "Female", "Transgender")
    e4.grid(row = 3, column = 1, padx=20, pady=15)
    e4.current()

    # Spacing before register button
    Label(main_container, text="", height=2).grid(row = 7, column = 0)

    # Function to handle registration with Enter key
    def perform_registration():
        reg_server(root, frame1, name.get(), sex.get(), zone.get(), city.get(), password.get())

    # Larger register button
    reg = Button(main_container, text="Register", width=15, height=2, font=('Helvetica', 16, 'bold'),
                 command=perform_registration,
                 bg='#4CAF50', fg='white', relief='raised', bd=4)
    reg.grid(row = 8, column = 1, columnspan = 2, pady=20)
```

```
    # Bind Enter key to registration function
    root.bind('<Return>', lambda event: perform_registration())
    e2.bind('<Return>', lambda event: perform_registration())
    e4.bind('<Return>', lambda event: perform_registration())
    e5.bind('<Return>', lambda event: perform_registration())
    e6.bind('<Return>', lambda event: perform_registration())
    e7.bind('<Return>', lambda event: perform_registration())

    frame1.pack()
    root.mainloop()

# if __name__ == "__main__":
#         root = Tk()
#         root.geometry('500x500')
#         frame1 = Frame(root)
#         Register(root,frame1)
```

server.py:

```
import socket
import threading
import dframe as df
from threading import Thread
from dframe import *

lock = threading.Lock()

def client_thread(connection):

    data = connection.recv(1024)     #receiving voter details           #2

    #verify voter details
    log = (data.decode()).split(' ')
    log[0] = int(log[0])
    if(df.verify(log[0],log[1])):                          #3 Authenticate
        if(df.isEligible(log[0])):
            print('Voter Logged in... ID:'+str(log[0]))
```

```
            connection.send("Authenticate".encode())
        else:
            print('Vote Already Cast by ID:'+str(log[0]))
            connection.send("VoteCasted".encode())
    else:
        print('Invalid Voter')
        connection.send("InvalidVoter".encode())

    data = connection.recv(1024)                          #4 Get Vote
    print("Vote Received from ID: "+str(log[0])+"  Processing...")
    lock.acquire()
    #update Database
    if(df.vote_update(data.decode(),log[0])):
        print("Vote Casted Sucessfully by voter ID = "+str(log[0]))
        connection.send("Successful".encode())
    else:
        print("Vote Update Failed by voter ID = "+str(log[0]))
        connection.send("Vote Update Failed".encode())
                                                  #5

    lock.release()
    connection.close()

def voting_Server():

    serversocket = socket.socket()
    host = socket.gethostname()
    port = 4001

    ThreadCount = 0

    try :
```

```
            serversocket.bind((host, port))
        except socket.error as e :
            print(str(e))
        print("Waiting for the connection")

        serversocket.listen(10)

        print( "Listening on " + str(host) + ":" + str(port))

        while True :
            client, address = serversocket.accept()

            print('Connected to :', address)

            client.send("Connection Established".encode())   ### 1
            t = Thread(target = client_thread,args = (client,))
            t.start()
            ThreadCount+=1
            # break

        serversocket.close()

    if __name__ == '__main__':
        voting_Server()

    video file.')
```

voting.py:

```
import tkinter as tk
import socket
```

```
from tkinter import *
from VotingPage import votingPg

def establish_connection():
    host = socket.gethostname()
    port = 4001
    client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    client_socket.connect((host, port))
    print(client_socket)
    message = client_socket.recv(1024)      #connection establishment message   #1
    if(message.decode()=="Connection Established"):
        return client_socket
    else:
        return 'Failed'

def failed_return(root,frame1,client_socket,message):
    for widget in frame1.winfo_children():
        widget.destroy()
    message = message + "... \nTry again..."
    Label(frame1, text=message, font=('Helvetica', 16, 'bold')).grid(row = 1, column = 1,
pady=20)
    client_socket.close()

def log_server(root,frame1,client_socket,voter_ID,password):
    message = voter_ID + " " + password
    client_socket.send(message.encode()) #2

    message = client_socket.recv(1024) #Authenticatication message
    message = message.decode()
```

```
    if(message=="Authenticate"):
        votingPg(root, frame1, client_socket)

    elif(message=="VoteCasted"):
        message = "Vote has Already been Cast"
        failed_return(root,frame1,client_socket,message)

    elif(message=="InvalidVoter"):
        message = "Invalid Voter"
        failed_return(root,frame1,client_socket,message)

    else:
        message = "Server Error"
        failed_return(root,frame1,client_socket,message)


def voterLogin(root,frame1):

    client_socket = establish_connection()
    if(client_socket == 'Failed'):
        message = "Connection failed"
        failed_return(root,frame1,client_socket,message)

    root.title("Voter Login")
    for widget in frame1.winfo_children():
        widget.destroy()

    # Create main container for better layout
```

```
    main_container = Frame(frame1)
    main_container.pack(expand=True, pady=20)

    Label(main_container, text="Voter Login", font=('Helvetica', 25, 'bold')).grid(row =
0, column = 1, columnspan=2, pady=30)

    # Spacing
    Label(main_container, text="", height=1).grid(row = 1, column = 0)

    # Labels with larger font
    Label(main_container, text="Voter ID:", anchor="e", justify=LEFT,
font=('Helvetica', 16)).grid(row = 2, column = 0, padx=20, pady=20, sticky='e')
    Label(main_container, text="Password:", anchor="e", justify=LEFT,
font=('Helvetica', 16)).grid(row = 3, column = 0, padx=20, pady=20, sticky='e')

    voter_ID = tk.StringVar()
    password = tk.StringVar()

    # Entry fields with larger font
    e1 = Entry(main_container, textvariable = voter_ID, font=('Helvetica', 14),
width=20)
    e1.grid(row = 2, column = 1, padx=20, pady=20)

    e3 = Entry(main_container, textvariable = password, show = '*', font=('Helvetica',
14), width=20)
    e3.grid(row = 3, column = 1, padx=20, pady=20)

    # Spacing before login button
    Label(main_container, text="", height=2).grid(row = 4, column = 0)
```

```
    # Larger login button
    sub = Button(main_container, text="Login", width=15, height=2, font=('Helvetica', 16, 'bold'),
                 command = lambda: log_server(root, frame1, client_socket, voter_ID.get(), password.get()),
                 bg='#4CAF50', fg='white', relief='raised', bd=4)
    sub.grid(row = 5, column = 1, columnspan = 2, pady=20)


    frame1.pack()
    root.mainloop()


# if __name__ == "__main__":
#         root = Tk()
#         root.geometry('500x500')
#         frame1 = Frame(root)
#         voterLogin(root,frame1)
```

VotingPage.py:

```
import tkinter as tk
import socket
from tkinter import *
from PIL import ImageTk,Image

def voteCast(root,frame1,vote,client_socket):

    for widget in frame1.winfo_children():
```

```
        widget.destroy()


    client_socket.send(vote.encode()) #4


    message = client_socket.recv(1024) #Success message
    print(message.decode()) #5
    message = message.decode()
    if(message=="Successful"):
        Label(frame1, text="Vote Casted Successfully", font=('Helvetica', 18,
'bold')).grid(row = 1, column = 1)
    else:
        Label(frame1, text="Vote Cast Failed... \nTry again", font=('Helvetica', 18,
'bold')).grid(row = 1, column = 1)


    client_socket.close()



def votingPg(root,frame1,client_socket):


    root.title("Cast Vote")
    for widget in frame1.winfo_children():
        widget.destroy()


    # Create main container for better centering
    main_container = Frame(frame1)
    main_container.pack(expand=True, pady=10)


    # Centered header
    Label(main_container, text="Cast Your Vote", font=('Helvetica', 25,
'bold')).grid(row=0, column=0, columnspan=3, pady=20)
```

```
    Label(main_container, text="").grid(row=1, column=0)


    vote = StringVar(main_container,"-1")


    # BJP - Adjusted sizes
    Radiobutton(main_container, text="BJP\n\nNarendra Modi", variable=vote,
value="bjp",
                indicator=0, height=3, width=18, font=('Helvetica', 11, 'bold'),
                command=lambda: voteCast(root,frame1,"bjp",client_socket)).grid(row=2,
column=2, pady=10)
    bjpLogo =
ImageTk.PhotoImage((Image.open("img/bjp.png")).resize((40,40),Image.Resampling.L
ANCZOS))
    bjpImg = Label(main_container, image=bjpLogo).grid(row=2, column=1, padx=15)


    # Congress - Adjusted sizes
    Radiobutton(main_container, text="Congress\n\nRahul Gandhi", variable=vote,
value="cong",
                indicator=0, height=3, width=18, font=('Helvetica', 11, 'bold'),
                command=lambda: voteCast(root,frame1,"cong",client_socket)).grid(row=3,
column=2, pady=10)
    congLogo =
ImageTk.PhotoImage((Image.open("img/cong.jpg")).resize((35,45),Image.Resampling.
LANCZOS))
    congImg = Label(main_container, image=congLogo).grid(row=3, column=1,
padx=15)


    # DMK - Adjusted sizes
    Radiobutton(main_container, text="DMK\n\nM. K. Stalin", variable=vote,
value="dmk",
                indicator=0, height=3, width=18, font=('Helvetica', 11, 'bold'),
                command=lambda: voteCast(root,frame1,"dmk",client_socket)).grid(row=4,
```

```
column=2, pady=10)
    dmkLogo =
ImageTk.PhotoImage((Image.open("img/dmk.jpg")).resize((40,35),Image.Resampling.
LANCZOS))
    dmkImg = Label(main_container, image=dmkLogo).grid(row=4, column=1,
padx=15)


    # ADMK - Adjusted sizes
    Radiobutton(main_container, text="ADMK\n\nE. Palani", variable=vote,
value="admk",
            indicator=0, height=3, width=18, font=('Helvetica', 11, 'bold'),
            command=lambda: voteCast(root,frame1,"admk",client_socket)).grid(row=5,
column=2, pady=10)
    admkLogo =
ImageTk.PhotoImage((Image.open("img/admk.jpg")).resize((40,40),Image.Resampling
.LANCZOS))
    admkImg = Label(main_container, image=admkLogo).grid(row=5, column=1,
padx=15)


    # TVK - Adjusted sizes
    Radiobutton(main_container, text="TVK\n\nVijay", variable=vote, value="tvk",
            indicator=0, height=3, width=18, font=('Helvetica', 11, 'bold'),
            command=lambda: voteCast(root,frame1,"tvk",client_socket)).grid(row=6,
column=2, pady=10)
    tvkLogo =
ImageTk.PhotoImage((Image.open("img/tvk.jpg")).resize((40,40),Image.Resampling.L
ANCZOS))
    tvkImg = Label(main_container, image=tvkLogo).grid(row=6, column=1, padx=15)


    # NOTA - Made clearly visible with proper sizing
    Radiobutton(main_container, text="NOTA\n\nNone of the Above", variable=vote,
value="nota",
```

```
            indicator=0, height=3, width=18, font=('Helvetica', 11, 'bold'),
            command=lambda: voteCast(root,frame1,"nota",client_socket)).grid(row=7, column=2, pady=10)
    notaLogo = ImageTk.PhotoImage((Image.open("img/nota.jpg")).resize((40,30),Image.Resampling.LANCZOS))
    notaImg = Label(main_container, image=notaLogo).grid(row=7, column=1, padx=15)

    # Instructions - Centered
    Label(main_container, text="Click on any party to cast your vote",
          font=('Helvetica', 12, 'italic'), fg='gray').grid(row=8, column=1, columnspan=2, pady=20)

    frame1.pack()
    root.mainloop()

# if __name__ == "__main__":
#         root = Tk()
#         root.geometry('500x500')
#         frame1 = Frame(root)
#         client_socket='Fail'
#         votingPg(root,frame1,client_socket)
```

Chapter 10

APPENDIX 2 – SAMPLE OUTPUT

10.1 Home Page

The project output screenshots are shown as follows:

10.1 Homepage

The home page serves as the authoritative and secure entry point for the electoral platform. **Visually**, it features a clean, professional design utilizing national or institutional colors to instill confidence and seriousness. The central focus is a prominent **Status Dashboard** detailing the current election phase—for example, "Voter Registration Open," "Voting Period Active (Ends: November 15th)," or "Results Pending Certification." This section provides real-time information to manage public expectations. **Navigation is clear and limited** to essential links: a primary **"VOTE NOW"** button prominently placed for authenticated voters, and links for "Voter Registration," "View Election Information," and "Security & Audit Details."

The page clearly demarcates access based on user role. A highly visible Voter Portal section

directs registered users to the login interface, often featuring a secure, multi-factor authentication (MFA) prompt (e.g., username, password, and a one-time code) to immediately reinforce the system's commitment to security. Concurrently, a discreet Administrator Login link is provided, leading to a secure, separate backend management system used for election setup, monitoring, and result certification. This separation of concerns is explicitly communicated to reassure the public that voter data and system management are strictly segregated.

10.2 Admin Details

10.2 Admin Details

The OVS Administrator Dashboard is a highly secure, compartmentalized interface accessible only through multi-factor authentication and role-based access control. Its primary design goal is to provide comprehensive, real-time oversight of the election process while ensuring strict separation of duties to prevent any single entity from compromising the results. The main screen presents a System Health and Status Overview, featuring critical metrics like total registered voters, current turnout percentage, server load/latency status, and a visible indicator of the current election phase (e.g., "Phase 2: Voting Active"). Any security alerts or anomalies (e.g., high failed login attempts, unexpected server traffic spikes) are prominently displayed at the top to facilitate immediate response.

The dashboard is logically organized into three primary management modules to control the election lifecycle. The **Pre-Election Management** module is used to import, verify, and finalize the secure voter registry list, set up the ballot structure, and formally initiate cryptographic key generation (often involving multiple trusted administrators). The **Active Election Monitoring** module provides tools for viewing **anonymized** voting activity (e.g., votes cast per hour, geographic distribution of voters) without accessing or decrypting individual ballots, thereby upholding vote secrecy. This module also includes controls for pausing or extending the voting period, which are heavily logged actions requiring a verifiable audit trail.

10.3 Vote Casting Page

10.3 Vote Casting Page

The Vote Casting Page, or the digital ballot, is launched only after the voter has successfully passed the multi-factor authentication process. The interface is designed for **maximum clarity and minimal distraction**, featuring a static, secure banner that continuously displays key election details, such as the voter's anonymized ID number, a countdown timer until the ballot expires (if applicable), and a clear indicator confirming the connection is

encrypted and secure. The entire page is strictly free of navigation links, advertisements, or external content to prevent coercion or tampering.information about the incident. It includes details such as the severity level of the accident, the associated probability, and most importantly, the exact location where the accident occurred.

The ballot itself is structured simply, presenting the offices or issues available for voting. Each choice uses a **large, high-contrast, universally accessible format** with clear candidate names, party affiliations, and accompanying information buttons for quick access to biographies or issue summaries. To prevent invalid ballots, the system employs **real-time validation**; for instance, if the election requires voting for exactly one candidate, the interface physically prevents the voter from selecting a second choice, providing immediate, non-intrusive feedback. A "Review Ballot" button is mandatory before submission, allowing the voter to double-check all selections before finalizing.

10.4 Vote Casted Page

10.4 Vote Casted Page

The "Vote Casted Successfully" page, which appears immediately after the voter submits their encrypted ballot, functions as the **final, critical acknowledgment** of the voter's participation. The immediate display of the clear message, **"Vote Casted Successfully!"** , assures the user that their interaction was complete and their vote was securely received by the election servers. This screen must immediately pivot the voter's focus from the act of voting to the **proof and verifiability** of their action.

Crucially, while the image displays a visual confirmation, a comprehensive Online Voting System requires this page to also prominently display the **Unique Voting Receipt (Confirmation Code)**. This is a cryptographic hash or a unique transaction ID generated for the encrypted ballot, which acts as the voter's personal token of participation. The description must emphasize that this receipt is the single most important piece of information the voter needs for audit purposes. The system should

encourage the voter to **note down, print, or securely copy** this code, as it is the key they will use later to verify that their encrypted vote was indeed counted in the final tally (End-to-End Verifiability).

Chapter 11

REFERENCES

[1] **Rubin, A. D., et al. (2004). Analysis of an electronic voting system.** *(Classic paper demonstrating vulnerabilities in early paperless DRE machines, emphasizing the need for robust security.)*

[2] Wagner, D., & Schneier, B. (1996). Analysis of the Secure Socket Layer (SSL) protocol. (Fundamental reference for secure communication protocols, which are essential for internet voting.)

[3] **Jefferson, D., et al. (2004). A Security Analysis of the New York City Voting System.** *(Highlights risks like Man-in-the-Middle attacks and the need for comprehensive end-to-end security.)*

[4] **Cranor, L. F. (2000). Voting After Florida: No Easy Answers.** *(A key early report that kicked off significant research into better voting technology and auditing.)*

[5] **Caltech/MIT Voting Technology Project (VTP). (2001). Voting—What Is, What Could Be.** *(An influential report setting benchmarks for reliability, usability, and transparency in voting technology.)*

[6] **Rivest, R. L., et al. (2001). A Modular Voting Architecture ("Frogs").** *(Proposes an early architecture designed for modularity and auditability.)*

[7] **Fujioka, A., Okamoto, T., & Ohta, K. (1992). A Practical Secret Voting Scheme for Large Scale Elections.** *(One of the earliest foundational papers proposing a practical blind voting protocol.)*.

[8] **Chaum, D. (1982). Blind signatures for untraceable payments.** *(Introduces the concept of blind signatures, a cryptographic tool used to issue voter credentials without linking the credential to the voter's identity, key for anonymity in Sensus/Scantegrity systems.)*

[9] **Paillier, P. (1999). Public-Key Cryptosystems Based on Composite Degree Residue Classes.** *(Defines the Paillier homomorphic encryption scheme, commonly used in E-Voting for securely tallying votes while they remain encrypted.)*

[10] **Benaloh, J. (1987). Verifiable Secret-Ballot Elections.** *(Early work on mix-nets and cryptographic proof that a tally is correct, forming the basis for End-to-End Verifiability (E2E-V).)*

[11] **Ryan, P., & Schürmann, C. (2005). Verifiable Voting Systems.** *(A general reference defining the concepts and requirements of E2E-V systems.)*

[12] **Yaga, D., Mell, P., Roby, N., & Scarfone, K. (2019). Blockchain technology overview.** *(A good general overview of blockchain principles relevant to its application in voting.)*

[13] **Wang, B., et al. (2018). Large-scale election based on blockchain.** *(Examines the feasibility and challenges of using blockchain for large-scale public elections.)*

[14] **Debant, A., & Hirschi, L. (2022). Reversing, Breaking, and Fixing the French Legislative Election E-Voting Protocol.** *(A recent case study highlighting vulnerabilities in a real-world system and proposing concrete cryptographic fixes.)*

[15] . **MDPI (2024). Blockchain-Based E-Voting Systems: A Technology Review.** *(A systematic literature review analyzing the benefits, challenges (scalability, privacy), and current state of research in 2023/2024.)*

[16] **Lauer, T. (2004). Voter-Verified Paper Audit Trail (VVPAT).** *(Defines the concept of a paper record that a voter can verify, a key requirement for many physical/electronic hybrid systems.)*

[17] **IDEA (2006). Introducing Electronic Voting: Essential Considerations.** *(An International IDEA report providing a high-level policy guide on the socio-political and technical foundations required for e-voting trust.)*

[18] **Gasser, U., & Gerlach, J. (2011). Electronic Voting: Approaches, Strategies, and Policy Issues - A Report from Switzerland.** *(A practical case study on internet voting implementation, addressing governance and public confidence.)*

[19] **European Parliament (2016). Potential and challenges of E-Voting in the European Union.** *(A policy review covering the political, constitutional, and technical hurdles for large-scale implementation.)*

[20] **Estonian National Electoral Committee. E-voting concept security: analysis and measures.** *(References to Estonia's long-standing internet voting system, often used as a key case study in implementation.)*

[21] Github link of the project: https://github.com/Murali-Krishna0/MINI-PROJECT

[22] Youtube link of the project: https://youtu.be/2I3EmYLEW2k