Final Internship Project Report

Secure Login System with RBAC

Adithya Prakash
Cybersecurity Intern

Date: 25 September 2025

GitHub Repository:
`https://github.com/adthyaprakash/secure-login-system`

GitHub: github.com/adthyaprakash
LinkedIn: linkedin.com/in/adithya-prakash-a729231b9/

Abstract

This project report documents the design and development of a secure login system with role-based access control (RBAC). It integrates key cybersecurity features such as input validation, CAPTCHA, and account lockout mechanisms. The system separates admin and user functionalities, ensuring strong access control.

Contents

Chapter 1

Introduction

1.1 Overview

In the digital era, user authentication and secure access management are critical for any web-based system. Unauthorized access, data breaches, and brute-force attacks are among the most common cybersecurity threats faced by organizations today. To mitigate these risks, strong authentication and role-based access control (RBAC) mechanisms are required.

This project, **Secure Login System with RBAC**, was developed as part of a cybersecurity internship. It focuses on creating a secure web-based system that not only provides traditional login and registration functionality but also integrates additional security features such as CAPTCHA verification, account lockout mechanisms, and secure password hashing. The system distinguishes between different types of users (Admins and standard Users) and ensures that each has access only to the features relevant to their role.

1.2 Objectives

The primary objectives of the project are as follows:

- To design and implement a secure user login and registration system.
- To incorporate **role-based access control (RBAC)**, allowing Admins and Users to have separate dashboards and privileges.
- To integrate modern security measures, such as:
  - Password hashing using **bcrypt**.
  - **Google reCAPTCHA** to mitigate brute-force attacks.

  - Account lockout functionality after repeated failed login attempts.
  - Input validation to reduce risks such as SQL Injection.

- To design a user-friendly interface using **HTML, CSS (Glassmorphism)**, and JavaScript.
- To connect the backend with a **PostgreSQL** database for secure and persistent storage of user data.

1.3 Motivation

The motivation behind this project is the growing need for secure authentication mechanisms in web applications. Traditional login systems are often vulnerable to attacks if not implemented properly. By incorporating RBAC, this system ensures that administrative tasks are restricted to privileged users, while standard users have a safe and controlled environment.

The project also serves as hands-on practice in applying cybersecurity principles in real-world software development, aligning with the responsibilities of a cybersecurity engineer or analyst.

1.4 Scope

The scope of the project includes:

- Development of login and registration pages with proper validations.
- Backend implementation using Flask framework with RESTful endpoints.
- PostgreSQL integration for database management.
- Secure authentication using JWT and session management.
- Role-based dashboards:
  - Admin Dashboard: manage users, disable/enable accounts, and monitor failed login attempts.
  - User Dashboard: personalized access for standard users.
- Security features: CAPTCHA, account lockout, and hashed credentials.

1.5 Expected Outcome

The expected outcome of this project is a fully functional, secure, and scalable web-based login system that demonstrates:

1. Secure user authentication and authorization.
2. Effective separation of roles (Admin vs. User).
3. Strong defense against brute-force and injection attacks.
4. A visually appealing, glassmorphic frontend.
5. Proper documentation, code structure, and deployment readiness.

Chapter 2

Technology Overview

2.1 Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is a widely used mechanism for restricting system access to authorized users based on their assigned roles. Instead of assigning permissions to individual users, roles are defined with specific privileges, and users are assigned to these roles. This makes management more scalable and less error-prone.

For example:

- An **Admin** role may have permissions to manage users, disable or enable accounts, and monitor login activity.
- A **User** role may only have access to personal dashboards and basic functionalities.

RBAC ensures the principle of **least privilege**, which reduces security risks by giving users only the access they need.

2.2 Flask Framework

Flask is a lightweight and flexible Python web framework used for developing web applications. It follows the **WSGI** standard and the **Jinja2** templating engine. Flask is chosen in this project because:

- It is simple to set up and extend.
- It integrates easily with databases like PostgreSQL.
- It supports RESTful APIs for backend communication.

- It allows easy integration of third-party libraries such as `passlib` for password hashing and `flask-cors` for cross-origin requests.

2.3 PostgreSQL Database

PostgreSQL is an open-source, object-relational database system known for its robustness and security. It is used in this project to store user credentials and manage login-related data. Key features include:

- Support for complex queries and transactions.
- ACID compliance (Atomicity, Consistency, Isolation, Durability).
- Strong user management and authentication controls.
- Ability to define constraints such as unique emails, ensuring data integrity.

The database schema includes fields such as:

- Username and Email (unique identifiers).
- Password (stored as a hashed value).
- Role (Admin/User).
- Account status (active/inactive).
- Failed login attempts (for lockout mechanism).

2.4 Password Hashing with bcrypt

Storing plain text passwords is a major security risk. Instead, passwords are hashed using algorithms such as **bcrypt**. Bcrypt automatically handles:

- Salting (adding random data before hashing).
- Iterative hashing, making brute-force attacks computationally expensive.

During login, the entered password is verified against the stored hash. This ensures that even if the database is compromised, the actual passwords remain secure.

2.5 JWT Authentication

JSON Web Tokens (JWT) are used for securely transmitting information between the client and server. They consist of three parts:

- Header
- Payload (contains user ID, role, and expiration time)
- Signature

JWTs are signed with a secret key, ensuring they cannot be tampered with. In this project:

- JWTs are generated during login and stored in the client's session.
- They expire after 12 hours to enhance security.
- Admin/User role information is embedded in the token to enforce RBAC.

2.6 Security Enhancements

Several security enhancements have been integrated into the project:

1. **Input Validation:** Prevents SQL Injection and malicious inputs by enforcing strict patterns on email, username, and role.
2. **Google reCAPTCHA:** Protects against automated brute-force login attempts by requiring human verification.
3. **Account Lockout:** After 5 failed login attempts, the account is temporarily disabled to prevent repeated attacks.
4. **HTTPS Deployment (future scope):** Ensures encrypted communication between client and server.

2.7 Frontend Design (Glassmorphism)

The frontend uses HTML, CSS, and JavaScript with a **glassmorphic** design approach. Glassmorphism provides a visually appealing interface with transparency, blur, and subtle shadows, enhancing user experience while maintaining readability. Bootstrap and custom CSS are also used to ensure responsiveness across devices.

Chapter 3

System Design and Implementation

3.1 System Architecture

The Secure Login System with RBAC is designed with a three-tier architecture:

1. **Frontend (Client-Side):** Developed using HTML, CSS, and JavaScript. Features login, registration, and dashboards with a glassmorphic design. Communicates with the backend via RESTful API calls.
2. **Backend (Server-Side):** Implemented using the Flask framework (Python). Handles authentication, authorization, session management, and interaction with the database. Provides JSON responses to frontend requests.
3. **Database Layer:** PostgreSQL stores user data, including:
    - Username, Email
    - Hashed Passwords
    - User Role (Admin/User)
    - Account Status (active/inactive)
    - Failed Login Attempts

Figure 3.1: Project Directory and File Structure of the Secure Login System

3.2 Workflow

The workflow of the system is as follows:

1. A user registers by providing username, email, password, and role (Admin/User). Input validation, password hashing, and CAPTCHA verification are applied before storing data in the database.

2. During login, the user enters credentials along with CAPTCHA verification. If valid, a JWT token is issued and stored in the session. Failed attempts are recorded, and account lockout is triggered after 5 failed logins.

3. Based on the role:

   - Admins are redirected to the **Admin Dashboard**, where they can view, enable/disable, and manage user accounts.
   - Users are redirected to the **User Dashboard**, which displays secure session details.

4. Security enhancements such as session expiration, input validation, and reCAPTCHA help protect against brute-force and injection attacks.

3.3 Implementation Details

3.3.1 Frontend

- **HTML Templates:** `index.html`, `register.html`, `dashboard.html`, and `admin_dashboard.html`.
- **CSS:** Glassmorphic styles defined in `style.css`.
- **JavaScript:** Handles form submissions, API calls, and dynamic DOM updates.

3.3.2 Backend

- **Flask Framework:** Provides API endpoints for login, registration, and admin management.
- **Authentication:** Passwords hashed with `bcrypt` before storage. JWT tokens issued for session handling.
- **Authorization:** Admin-only routes secured via role checks in Flask.

3.3.3 Database Schema

The PostgreSQL schema for users is as follows:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(20) DEFAULT 'user',
    is_active BOOLEAN DEFAULT TRUE,
    failed_attempts INTEGER DEFAULT 0
);
```

3.4 Screenshots

The following figures illustrate the system in action:

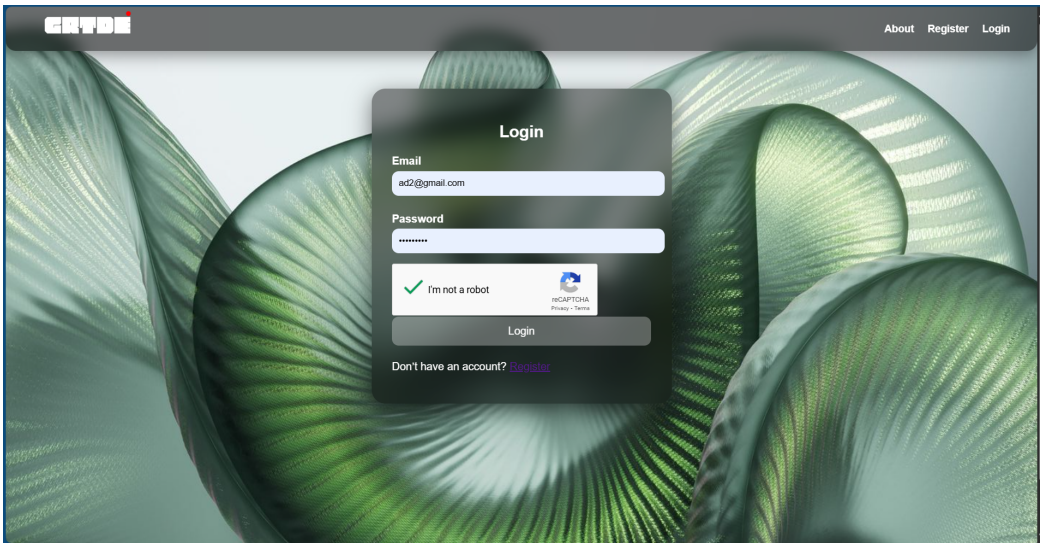

Figure 3.2: Login Page with CAPTCHA

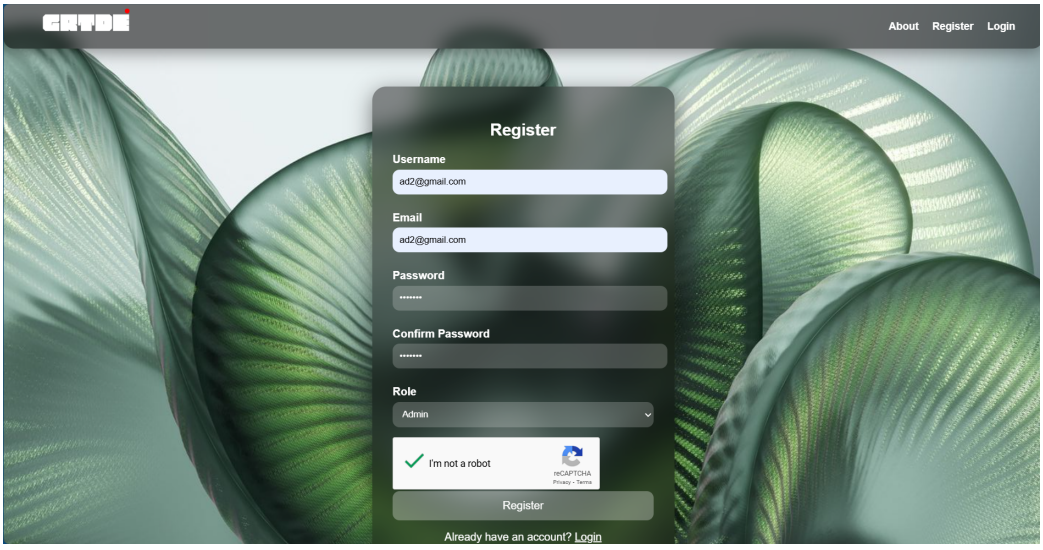

Figure 3.3: User Registration Page

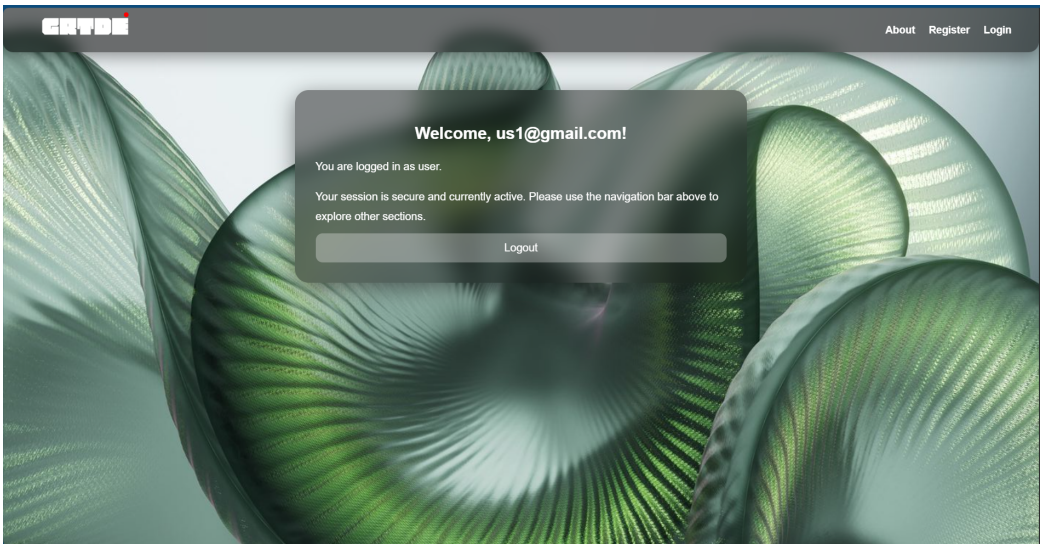

Figure 3.4: User Dashboard

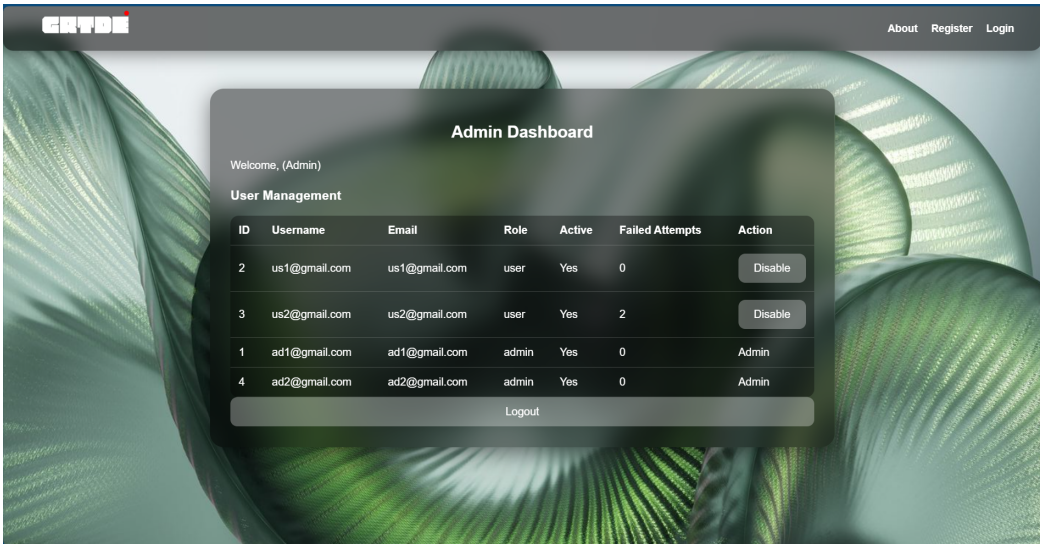

Figure 3.5: Admin Dashboard with User Management

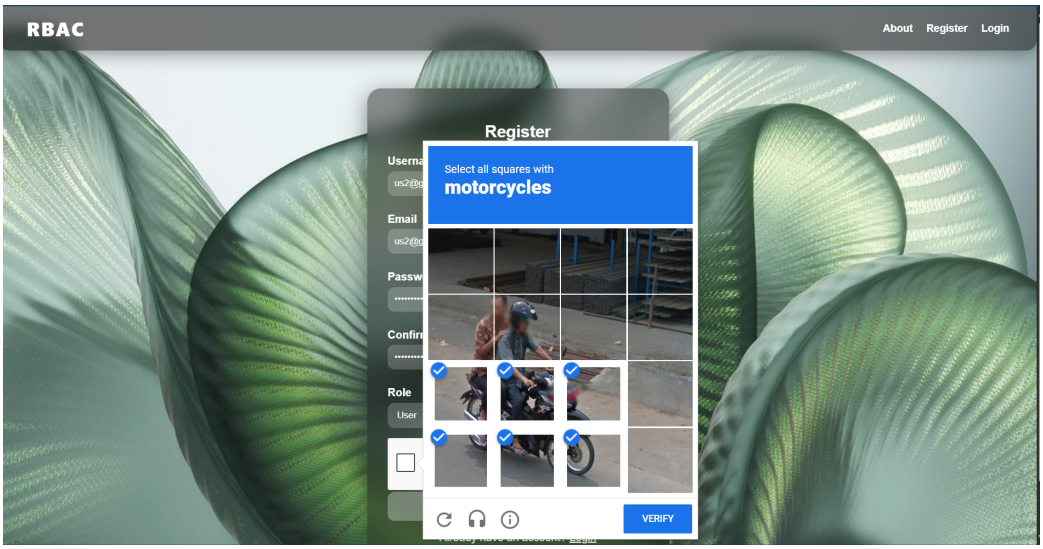

Figure 3.6: Login Protected by Google reCAPTCHA

```
vaultdb=# SELECT * FROM users
vaultdb-# ;
 id |   username    |     email     |                          password                          | role  | is_active | failed_attempts
----+---------------+---------------+------------------------------------------------------------+-------+-----------+-----------------
  2 | us1@gmail.com | us1@gmail.com | $2b$12$fy1/rxstjMKl5syjQS5zZeBcWqPwq5.Ig/gC3YRm1Vf.I2RBS/yGu | user  | t         |               0
  3 | us2@gmail.com | us2@gmail.com | $2b$12$LAEyKCvlQO9H4Q46s3lSAOolKOTNkCd9r8gNMrhMNoY1Wxd1diP5q | user  | t         |               2
  1 | ad1@gmail.com | ad1@gmail.com | $2b$12$VdRsJXnE9AX2oi3QtAmZEO3icgdl4tfxlBFhtgkC5MMJmTla3NX.C | admin | t         |               0
  4 | ad2@gmail.com | ad2@gmail.com | $2b$12$8f.tOgNiH2NRxzALLyCpCO9zlJzXXBvy7KjDCRN.CYvIQBnJUWd1G | admin | t         |               0
(4 rows)

vaultdb=#
```

Figure 3.7: PostgreSQL Database Records

Chapter 4

Code

This chapter presents the complete source code of the Secure Login System with RBAC. The implementation includes the backend written in Python (Flask), HTML templates for the frontend, CSS for styling (glassmorphic design), and JavaScript for interactivity and API communication.

4.1 Backend: app.py

The main Flask application that manages routing, authentication, role-based access control, and communication with the PostgreSQL database.

Listing 4.1: app.py

```
from flask import Flask, request, jsonify, render_template,
    redirect, url_for, session
from flask_cors import CORS
import psycopg2
from psycopg2.extras import RealDictCursor
from passlib.hash import bcrypt
import jwt
import datetime
import re

app = Flask(__name__, static_folder="static", template_folder
    ="templates")
CORS(app)
app.config['SECRET_KEY'] = '6LeeANQrAAAAANCERRtSdlwa6ODRSy-
    JuJJT5Ku_'  #        replace with strong random key

# ---------- Database Connection ----------
def get_db_connection():
    return psycopg2.connect(
        host="localhost",
```

```
        database="vaultdb",
        user="postgres",
        password="root"  #        replace with your postgres
            password
    )

# ---------- Routes ----------
@app.route('/about_page')
def about_page():
    return render_template("about.html")

@app.route('/')
def login_page():
    return render_template("index.html")

@app.route('/register_page')
def register_page():
    return render_template("register.html")

@app.route('/dashboard_page')
def dashboard_page():
    role = session.get("role")
    if not role:
        return redirect(url_for("login_page"))
    if role == "admin":
        return redirect(url_for("admin_dashboard_page"))
    return render_template("dashboard.html")

@app.route('/admin_dashboard_page')
def admin_dashboard_page():
    role = session.get("role")
    if role != "admin":
        return redirect(url_for("dashboard_page"))
    return render_template("admin_dashboard.html")

# ---------- Registration ----------
@app.route('/register', methods=['POST'])
def register():
    data = request.json
    username = data.get('username')
    email = data.get('email')
    password = data.get('password')
    role = data.get('role', 'user')

    if not re.match(r"[^@]+@[^@]+\.[^@]+", email):
        return jsonify({"error": "Invalid email format"}),
            400
    if not username or not password:
        return jsonify({"error": "Missing fields"}), 400
```

```
    hashed_password = bcrypt.hash(password)

    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute(
            "INSERT␣INTO␣users␣(username,␣email,␣password,␣
               role,␣is_active,␣failed_attempts)␣VALUES␣(%s,␣
               %s,␣%s,␣%s,␣TRUE,␣0)",
            (username, email, hashed_password, role)
        )
        conn.commit()
        cur.close()
        conn.close()
        return jsonify({"message": "User␣registered␣
           successfully"}), 201
    except Exception as e:
        return jsonify({"error": str(e)}), 500

# ---------- Login ----------
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    email = data.get('email')
    password = data.get('password')

    try:
        conn = get_db_connection()
        cur = conn.cursor(cursor_factory=RealDictCursor)
        cur.execute(
            "SELECT␣id,␣username,␣email,␣password,␣role,␣
               is_active,␣failed_attempts␣FROM␣users␣WHERE␣
               email␣=␣%s",
            (email,)
        )
        user = cur.fetchone()

        if not user:
            return jsonify({"error": "Invalid␣credentials"}),
                401
        if not user['is_active']:
            return jsonify({"error": "Account␣disabled"}),
               403

        if bcrypt.verify(password, user['password']):
            cur.execute("UPDATE␣users␣SET␣failed_attempts␣=␣0
               ␣WHERE␣id␣=␣%s", (user['id'],))
            conn.commit()
```

```
            token = jwt.encode(
                {
                    'user_id': user['id'],
                    'role': user['role'],
                    'exp': datetime.datetime.now(datetime.
                       timezone.utc) + datetime.timedelta(
                       hours=12)
                },
                app.config['SECRET_KEY'],
                algorithm="HS256"
            )

            session['role'] = user['role']
            session['username'] = user['username']

            cur.close()
            conn.close()
            return jsonify({
                "message": "Login successful",
                "user": {
                    "id": user['id'],
                    "username": user['username'],
                    "email": user['email'],
                    "role": user['role']
                },
                "token": token
            }), 200
        else:
            cur.execute("UPDATE users SET failed_attempts =
               failed_attempts + 1 WHERE id = %s", (user['id'
               ],))
            conn.commit()

            if user['failed_attempts'] + 1 >= 5:
                cur.execute("UPDATE users SET is_active =
                   FALSE WHERE id = %s", (user['id'],))
                conn.commit()

            cur.close()
            conn.close()
            return jsonify({"error": "Invalid credentials"}),
                401

    except Exception as e:
        print("Login error:", e)
        return jsonify({"error": str(e)}), 500

# ---------- Admin: Fetch Users ----------
```

```
@app.route('/admin/get_users', methods=['GET'])
def get_users():
    try:
        conn = get_db_connection()
        cur = conn.cursor(cursor_factory=RealDictCursor)
        cur.execute("SELECT id, username, email, role,
            is_active, failed_attempts FROM users")
        users = cur.fetchall()
        cur.close()
        conn.close()
        return jsonify({"users": users}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

# ---------- Admin: Toggle User ----------
@app.route('/admin/toggle_user', methods=['POST'])
def toggle_user():
    data = request.json
    user_id = data.get('user_id')
    is_active = data.get('is_active')

    try:
        conn = get_db_connection()
        cur = conn.cursor()
        if is_active:  # re-enable account
            cur.execute("UPDATE users SET is_active = TRUE,
                failed_attempts = 0 WHERE id = %s AND role !=
                'admin'", (user_id,))
        else:  # disable account
            cur.execute("UPDATE users SET is_active = FALSE
                WHERE id = %s AND role != 'admin'", (user_id,)
                )
        conn.commit()
        cur.close()
        conn.close()
        return jsonify({"message": "User status updated
            successfully"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

# ---------- Main ----------
if __name__ == '__main__':
    try:
        conn = get_db_connection()
        print("Database connected successfully!")
        conn.close()
    except Exception as e:
        print("DB connection failed:", e)
```

```
    app.secret_key = app.config['SECRET_KEY']
    app.run(debug=True)
```

4.2 Frontend Templates

4.2.1 Login Page (index.html)

Listing 4.2: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
     scale=1.0">
  <title>Login</title>
  <link rel="stylesheet" href="{{ url_for('static', filename
     ='style.css') }}">
  <script src="https://www.google.com/recaptcha/api.js" async
      defer></script>
</head>
<body>

  <!-- Navbar -->
  <nav class="navbar">
  <div class="logo-container-left">
    <img src="{{ url_for('static', filename='assets/logo.png
       ') }}" alt="Logo" class="logo-large" onclick="window.
       location.href='/'">
  </div>
  <ul>
    <li><a href="/about_page">About</a></li>
    <li><a href="/register_page">Register</a></li>
    <li><a href="/">Login</a></li>
  </ul>
</nav>


  <!-- Page Content -->
  <div class="page-content">
    <div class="glass-container login-container">
      <h2>Login</h2>
      <form id="loginForm">
        <label>Email</label>
        <input type="email" name="email" required>

        <label>Password</label>
```

```
        <input type="password" name="password" required>

        <!-- reCAPTCHA -->
        <div class="g-recaptcha" data-sitekey="6
           LeeANQrAAAAAHUaa3yyaDZj82UniF9D6sg2HgG5"></div>

        <button type="submit">Login</button>
      </form>
      <p> D o n t  have an account? <a href="/register_page">
         Register</a></p>
    </div>
  </div>

  <script src="{{ url_for('static', filename='script.js') }}"
     ></script>
</body>
</html>
```

4.2.2 Register Page (register.html)

Listing 4.3: register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
     scale=1.0">
  <title>Register</title>
  <link rel="stylesheet" href="{{ url_for('static', filename
     ='style.css') }}">
  <script src="https://www.google.com/recaptcha/api.js" async
      defer></script>
</head>
<body>

 <nav class="navbar">
  <div class="logo-container-left">
    <img src="{{ url_for('static', filename='assets/logo.png
       ') }}" alt="Logo" class="logo-large" onclick="window.
       location.href='/'">
  </div>
  <ul>
    <li><a href="/about_page">About</a></li>
    <li><a href="/register_page">Register</a></li>
    <li><a href="/">Login</a></li>
  </ul>
</nav>
```

```
  <!-- Page Content -->
  <div class="page-content">
    <div class="glass-container">
      <h2>Register</h2>
      <form id="registerForm">
        <label for="username">Username</label>
        <input type="text" name="username" id="username"
           placeholder="Enter␣username" required>

        <label for="email">Email</label>
        <input type="email" name="email" id="email"
           placeholder="Enter␣email" required>

        <label for="password">Password</label>
        <input type="password" name="password" id="password"
           placeholder="Enter␣password" required>

        <label for="confirm_password">Confirm Password</label
           >
        <input type="password" name="confirm_password" id="
           confirm_password" placeholder="Confirm␣password"
           required>

        <label for="role">Role</label>
        <select name="role" id="role">
          <option value="user">User</option>
          <option value="admin">Admin</option>
        </select>

        <!-- reCAPTCHA -->
        <div class="g-recaptcha" data-sitekey="6
           LeeANQrAAAAAHUaa3yyaDZj82UniF9D6sg2HgG5"></div>

        <button type="submit">Register</button>
      </form>
      <p class="redirect">Already have an account? <a href="/
         ">Login</a></p>
    </div>
  </div>

  <script src="{{␣url_for('static',␣filename='script.js')␣}}"
     ></script>
</body>
</html>
```

4.2.3 User Dashboard (dashboard.html)

Listing 4.4: dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
     scale=1.0">
  <title>User Dashboard</title>
  <link rel="stylesheet" href="{{ url_for('static', filename
     ='style.css') }}">
</head>
<body>

  <!-- Navbar -->
  <nav class="navbar">
  <div class="logo-container-left">
    <img src="{{ url_for('static', filename='assets/logo.png
       ') }}" alt="Logo" class="logo-large" onclick="window.
       location.href='/'">
  </div>
  <ul>
    <li><a href="/about_page">About</a></li>
    <li><a href="/register_page">Register</a></li>
    <li><a href="/">Login</a></li>
  </ul>
</nav>


  <!-- Page Content -->
  <div class="page-content">
    <div class="glass-container dashboard-container">
      <h2>Welcome, <span id="username"></span>!</h2>
      <p>You are logged in as <span id="userRole"></span>.</p
         >
      <p class="info-text">Your session is secure and
         currently active.
         Please use the navigation bar above to explore other
             sections.</p>

      <button id="logoutBtn" class="btn">Logout</button>
    </div>
  </div>

  <script>
    // Populate user data from sessionStorage
```

```
    const username = sessionStorage.getItem("username") || "
       User";
    const role = sessionStorage.getItem("role") || "user";

    document.getElementById("username").innerText = username;
    document.getElementById("userRole").innerText = role;

    // Logout
    document.getElementById("logoutBtn").addEventListener("
       click", () => {
      sessionStorage.clear();
      window.location.href = "/";
    });
  </script>
</body>
</html>
```

4.2.4 Admin Dashboard (admin_dashboard.html)

```
[language=HTML, caption={admin_dashboard.html}]
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
     scale=1.0">
  <title>Admin Dashboard</title>
  <link rel="stylesheet" href="{{ url_for('static', filename
     ='style.css') }}">
</head>
<body>
  <!-- Navbar -->
  <nav class="navbar">
  <div class="logo-container-left">
    <img src="{{ url_for('static', filename='assets/logo.png
       ') }}" alt="Logo" class="logo-large" onclick="window.
       location.href='/'">
  </div>
  <ul>
    <li><a href="/about_page">About</a></li>
    <li><a href="/register_page">Register</a></li>
    <li><a href="/">Login</a></li>
  </ul>
</nav>


  <!-- Page Content -->
```

```
  <div class="page-content">
    <div class="glass-container admin-dashboard-container">
      <h2>Admin Dashboard</h2>
      <p>Welcome, <span id="adminName"></span> (Admin)</p>

      <h3>User Management</h3>
      <table id="userTable">
        <thead>
          <tr>
            <th>ID</th>
            <th>Username</th>
            <th>Email</th>
            <th>Role</th>
            <th>Active</th>
            <th>Failed Attempts</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          <!-- Populated dynamically with admin_script.js -->
        </tbody>
      </table>

      <button id="logoutBtn" class="btn">Logout</button>
    </div>
  </div>

  <!-- Attach Script -->
  <script src="{{ url_for('static', filename='admin_script.js
     ') }}"></script>
</body>
</html>
```

4.2.5 About Page (about.html)

Listing 4.5: about.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
     scale=1.0">
  <title>About - RBAC Secure Login</title>
  <link rel="stylesheet" href="{{ url_for('static', filename
     ='style.css') }}">
</head>
<body>
```

```
  <!-- Navbar -->
 <nav class="navbar">
  <div class="logo-container-left">
    <img src="{{ url_for('static', filename='assets/logo.png
       ') }}" alt="Logo" class="logo-large" onclick="window.
       location.href='/'">
  </div>
  <ul>
    <li><a href="/about_page">About</a></li>
    <li><a href="/register_page">Register</a></li>
    <li><a href="/">Login</a></li>
  </ul>
</nav>


  <!-- Content -->
  <div class="page-content">
    <div class="glass-container">
      <h2>About This Project</h2>
      <p><strong>RBAC Secure Login System</strong> is a web
         app built with:</p>
      <ul>
        <li>Python (Flask)</li>

        <li>PostgreSQL</li>

        <li>JWT (JSON Web Tokens)</li>

        <li>bcrypt (Password Hashing)</li>
        <li>Google reCAPTCHA</li>
        <li>Role-Based Access Control (RBAC)</li>
      </ul>

      <h3>Developer Links</h3>
      <p>
        <a href="https://github.com/adthyaprakash" target="
           _blank">GitHub</a> |
        <a href="https://linkedin.com/in/adthyaprakash"
           target="_blank">LinkedIn</a>
      </p>
    </div>
  </div>

</body>
</html>
```

4.3 CSS: style.css

The stylesheet responsible for the glassmorphic design, navbar, and page layout.

Listing 4.6: style.css

```
/* ---------- General Page Styles ---------- */
body, html {
  height: 100%;
  margin: 0;
  font-family: 'Arial', sans-serif;
  background: url("assets/bg.jpeg") no-repeat center center
     fixed;
  background-size: cover;
  color: #fff;
}

/* ---------- Glassmorphic Containers ---------- */
.glass-container {
  background: rgba(0, 0, 0, 0.45); /* darker for visibility
     */
  backdrop-filter: blur(15px);
  -webkit-backdrop-filter: blur(15px);
  border-radius: 20px;
  padding: 30px;
  box-shadow: 0 8px 32px rgba(0,0,0,0.5);
  text-align: left;
  width: 380px;
  max-width: 90%;
  margin: auto;
  color: #fff;
}

/* Specific container sizes */
.dashboard-container {
  width: 600px;
}

.admin-dashboard-container {
  width: 850px; /* admin dashboard wider for table */
}

/* ---------- Headings ---------- */
.glass-container h2 {
  text-align: center;
  margin-bottom: 20px;
  color: #fff;
}
```

```
/* ---------- Form Elements ---------- */
.glass-container label {
  display: block;
  margin: 10px 0 5px;
  font-weight: bold;
}

.glass-container input,
.glass-container select {
  width: 100%;
  padding: 10px;
  margin-bottom: 15px;
  border-radius: 10px;
  border: none;
  background: rgba(255, 255, 255, 0.2);
  color: #fff;
}

.glass-container input::placeholder {
  color: #ddd;
}

.glass-container button {
  width: 100%;
  padding: 12px;
  border-radius: 10px;
  border: none;
  background: rgba(255,255,255,0.3);
  color: #fff;
  font-size: 16px;
  cursor: pointer;
  transition: 0.3s;
}

.glass-container button:hover {
  background: rgba(255,255,255,0.5);
}

/* ---------- Redirect Links ---------- */
.redirect {
  text-align: center;
  margin-top: 10px;
}

.redirect a {
  color: #fff;
  text-decoration: underline;
}
```

```
/* ---------- Tables (Admin Dashboard) ---------- */
table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 15px;
  color: #fff;
  border-radius: 12px;
  overflow: hidden;
  background: rgba(0, 0, 0, 0.35);
}

th, td {
  padding: 12px;
  border-bottom: 1px solid rgba(255,255,255,0.2);
  text-align: left;
}

.disable-btn, .enable-btn {
  padding: 6px 12px;
  border-radius: 8px;
  border: none;
  cursor: pointer;
  color: #fff;
}

.disable-btn {
  background: rgba(255,0,0,0.4);
}

.disable-btn:hover {
  background: rgba(255,0,0,0.7);
}

.enable-btn {
  background: rgba(0,200,0,0.4);
}

.enable-btn:hover {
  background: rgba(0,200,0,0.7);
}

/* ---------- Navbar ---------- */
.navbar {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  background: rgba(0, 0, 0, 0.55);
```

```
  backdrop-filter: blur(12px);
  -webkit-backdrop-filter: blur(12px);
  border-radius: 0 0 15px 15px;
  padding: 8px 40px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-shadow: 0 6px 24px rgba(0,0,0,0.4);
  z-index: 1000;
  height: 65px;
  box-sizing: border-box;
}

/* App Title in Navbar */
.app-title {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-
     serif;
  font-size: 28px;
  font-weight: 800;
  color: #fff;
  letter-spacing: 2px;
  text-transform: uppercase;
  cursor: pointer;
}

/* Nav links container */
.navbar ul {
  list-style: none;
  display: flex;
  gap: 20px;
  margin: 0;
  padding: 0;
  flex-wrap: wrap;
  justify-content: flex-end;
}

/* Nav links */
.navbar ul li a {
  color: #fff;
  text-decoration: none;
  font-weight: 600;
  font-size: 15px;
  transition: 0.3s;
  white-space: nowrap;
}

.navbar ul li a:hover {
  text-decoration: underline;
  color: #d1d1d1;
```

```
}

/* ---------- Page Content ---------- */
body {
  padding-top: 80px; /* prevent content overlap with navbar
     */
}

.page-content {
  margin-top: 40px;
  display: flex;
  justify-content: center;
  align-items: center;
}

/* For paragraphs and list items inside glass-container */
.glass-container p,
.glass-container li {
  line-height: 1.8;   /* increases vertical spacing */
  margin-bottom: 10px; /* extra spacing between items */
}
/* Extra spacing for dashboard text */
.info-text {
  line-height: 1.8;
  margin-top: 15px;
  margin-bottom: 20px;
}
/* Navbar logo replacement */
.logo-large {
  height: 160px;   /* fits neatly inside navbar */
  width: auto;
  cursor: pointer;
}
```

4.4 JavaScript Files

4.4.1 script.js

Handles registration, login requests, and client-side validation.

Listing 4.7: script.js

```
// -------- Register --------
const registerForm = document.getElementById("registerForm");
if (registerForm) {
  registerForm.addEventListener("submit", async (e) => {
```

```
    e.preventDefault();

    const username = registerForm.username.value;
    const email = registerForm.email.value;
    const password = registerForm.password.value;
    const confirmPassword = registerForm.confirm_password.
       value;
    const role = registerForm.role.value;
    const captcha = grecaptcha.getResponse();

    if (password !== confirmPassword) {
      alert("Passwords do not match!");
      return;
    }
    if (!captcha) {
      alert("Please complete CAPTCHA!");
      return;
    }

    try {
      const res = await fetch("/register", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ username, email, password,
           role, captcha })
      });

      const data = await res.json();
      alert(data.message || data.error);

      if (res.status === 201) {
        window.location.href = "/"; // go to login page
      }
    } catch (err) {
      console.error("Register error:", err);
      alert("Error connecting to server");
    }
  });
}

// -------- Login --------
const loginForm = document.getElementById("loginForm");
if (loginForm) {
  loginForm.addEventListener("submit", async (e) => {
    e.preventDefault();

    const email = loginForm.email.value;
    const password = loginForm.password.value;
    const captcha = grecaptcha.getResponse();
```

```
    if (!captcha) {
      alert("Please complete CAPTCHA!");
      return;
    }

    try {
      const res = await fetch("/login", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, password, captcha })
      });

      const data = await res.json();
      alert(data.message || data.error);

      if (res.status === 200) {
        // Save user info
        sessionStorage.setItem("username", data.user.username
            );
        sessionStorage.setItem("role", data.user.role);
        sessionStorage.setItem("token", data.token);

        // Redirect based on role
        if (data.user.role === "admin") {
          window.location.href = "/admin_dashboard_page";
        } else {
          window.location.href = "/dashboard_page";
        }
      }
    } catch (err) {
      console.error("Login error:", err);
      alert("Error connecting to server");
    }
  });
}
```

4.4.2 admin_script.js

Handles fetching users, enabling/disabling accounts, and admin-side actions.

Listing 4.8: admin$_s$*cript.js*

```
async function fetchUsers() {
  try {
    const res = await fetch("/admin/get_users");
    const data = await res.json();
    if (!res.ok) throw new Error(data.message || "Failed to
       fetch users");
```

```
    const tbody = document.querySelector("#userTable tbody");
    tbody.innerHTML = "";

    if (!data.users || data.users.length === 0) {
      tbody.innerHTML = `<tr><td colspan="7">No users found</
          td></tr>`;
      return;
    }

    data.users.forEach(u => {
      const tr = document.createElement("tr");
      tr.innerHTML = `
        <td>${u.id}</td>
        <td>${u.username}</td>
        <td>${u.email}</td>
        <td>${u.role}</td>
        <td>${u.is_active ? "Yes" : "No"}</td>
        <td>${u.failed_attempts}</td>
        <td>
          ${u.role !== "admin"
            ? `<button class="${u.is_active ? "disable-btn" :
                "enable-btn"}"
                       onclick="toggleUser(${u.id}, ${u.
                          is_active ? 'false' : 'true'})">
                 ${u.is_active ? "Disable" : "Enable"}
               </button>`
            : "Admin"}
        </td>
      `;
      tbody.appendChild(tr);
    });
  } catch (err) {
    console.error("Error fetching users:", err);
    alert("Error loading users");
  }
}

// Toggle user status
async function toggleUser(userId, isActive) {
  try {
    const res = await fetch("/admin/toggle_user", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ user_id: userId, is_active:
          isActive })
    });

    const data = await res.json();
```

```
    alert(data.message || "Updated");
    fetchUsers(); // refresh table
  } catch (err) {
    console.error("Error toggling user:", err);
    alert("Failed to update user");
  }
}

document.addEventListener("DOMContentLoaded", fetchUsers);
```

Chapter 5

Challenges Faced and Solutions Implemented

During the development of the Secure Login System with RBAC, several challenges were encountered. Each of these challenges provided valuable learning opportunities and helped in building a more secure and robust system. Below are the major challenges and the corresponding solutions:

5.1 Database Integration Issues

Challenge: Initially, there were difficulties in connecting Flask with PostgreSQL due to missing drivers and misconfigured credentials. **Solution:** The issue was resolved by installing the `psycopg2-binary` package and ensuring that database credentials in `app.py` matched the PostgreSQL setup. Additionally, a robust error-handling mechanism was implemented for failed connections.

5.2 Password Hashing and Verification

Challenge: At first, login attempts failed because password hashes were not being verified correctly. This caused login errors even for valid users. **Solution:** The `passlib.hash.bcrypt` library was used for secure password hashing and verification, ensuring consistency across registration and login.

5.3 Role-Based Redirection

Challenge: Both Admin and User accounts were initially redirected to the same dashboard. **Solution:** Conditional logic was added to `app.py` and JavaScript to differentiate between roles. Admins are now redirected to the Admin Dashboard, while regular users are redirected to the User Dashboard.

5.4 CAPTCHA Implementation

Challenge: Integrating Google reCAPTCHA resulted in an "Invalid site key" error, preventing proper login and registration form submissions. **Solution:** The correct site key and secret key were generated from Google's reCAPTCHA console, and validation was implemented both on the client side (via HTML) and server side (via Flask).

5.5 Account Lockout Mechanism

Challenge: Users could attempt unlimited failed logins, exposing the system to brute-force attacks. **Solution:** An account lockout feature was implemented. After 5 failed attempts, the account is automatically disabled. Admins can later re-enable the account through the Admin Dashboard.

5.6 Frontend Styling and Alignment

Challenge: The glassmorphic design was not displaying correctly, with misaligned containers and inconsistent text colors. **Solution:** A consistent `style.css` file was created with dark glassmorphic effects, ensuring proper alignment, responsiveness, and a modern look for all pages.

5.7 GitHub Synchronization

Challenge: Pushing code to GitHub caused conflicts because the local repository was out of sync with the remote branch. **Solution:** This was fixed by performing a `git pull --rebase` followed by a forced push (`git push --force`) after verifying all local changes were correct.

Overall, these challenges significantly improved the understanding of secure coding practices, version control, and web application security mechanisms.

Chapter 6

Documentation

This chapter provides the necessary documentation for understanding, setting up, and running the Secure Login System with RBAC. It includes setup instructions, database schema, and a list of implemented features.

6.1 Project Overview

The Secure Login System with RBAC is a Flask-based web application with PostgreSQL as the backend database. It provides user registration, secure login, role-based dashboards, CAPTCHA integration, and security features such as password hashing and account lockouts.

6.2 Setup Instructions

6.2.1 Clone the Repository

```
git clone https://github.com/adthyaprakash/secure-login-system.git
cd secure-login-system
```

6.2.2 Create and Activate Virtual Environment

Windows:

```
python -m venv venv
venv\Scripts\activate
```

Linux/Mac:

```
python3 -m venv venv
source venv/bin/activate
```

6.2.3 Install Dependencies

```
pip install -r requirements.txt
```

6.2.4 Setup PostgreSQL Database

Login to PostgreSQL and create a database:

```
CREATE DATABASE vaultdb;
```

Create the users table:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(20) DEFAULT 'user',
    is_active BOOLEAN DEFAULT TRUE,
    failed_attempts INTEGER DEFAULT 0
);
```

6.2.5 Configure Database in `app.py`

Update with your PostgreSQL credentials:

```
def get_db_connection():
    return psycopg2.connect(
        host="localhost",
        database="vaultdb",
        user="postgres",
        password="your_password_here"
    )
```

6.2.6 Run the Application

```
python src/backend/app.py
```

The app will run at:

```
http://127.0.0.1:5000/
```

6.3 Database Schema

The application uses a single table named `users` with the following structure:

- **id** — Unique identifier for each user (Primary Key).
- **username** — The chosen username.
- **email** — Unique email address for login.
- **password** — Hashed password using bcrypt.
- **role** — Either `admin` or `user`.
- **is_active** — Boolean flag to enable/disable accounts.
- **failed_attempts** — Tracks consecutive failed login attempts.

6.4 Features Implemented

1. User registration with hashed passwords.
2. Secure login with JWT session management.
3. Role-based access control (Admin/User).
4. Separate dashboards for Admins and Users.
5. Admins can enable/disable user accounts.
6. Account lockout after repeated failed login attempts.
7. Google reCAPTCHA integration to prevent brute-force attacks.
8. Glassmorphic styled frontend for modern UI.
9. Secure input validation to prevent SQL injection.

6.5 Screenshots Reference

Screenshots of the system in action are included in Chapter 3. These demonstrate the login page, registration page, user dashboard, admin dashboard, CAPTCHA integration, and database view.

Chapter 7

Conclusion

The Secure Login System with Role-Based Access Control (RBAC) was successfully implemented as part of the internship project. The system demonstrates the practical application of core cybersecurity concepts such as secure authentication, authorization, password hashing, input validation, and brute-force attack prevention.

Through the development of this project, the following goals were achieved:

- Implementation of a secure registration and login mechanism with bcrypt password hashing.
- Integration of Google reCAPTCHA for enhanced protection against automated attacks.
- Enforcement of account lockouts after multiple failed login attempts to mitigate brute-force risks.
- Development of role-based dashboards for Admin and User roles, ensuring proper access control.
- Creation of an Admin panel with functionalities to manage user accounts.
- Use of PostgreSQL for reliable data storage and Flask for a secure backend framework.
- Enhancement of user experience with a modern glassmorphic user interface.

This project provided valuable hands-on experience in secure software development practices and strengthened the understanding of both backend security measures and frontend usability. It also reinforced the importance

of combining technical safeguards with user-friendly design in real-world applications.

The Secure Login System is now fully functional, meeting the project objectives and internship requirements.