

1.INTRODUCTION

1.1 Cybersecurity



Cybersecurity is the practice of protecting computer systems, networks, and data from cyber threats such as hacking, malware, phishing, and ransomware attacks. It involves implementing technologies, processes, and best practices to safeguard sensitive information from unauthorized access and potential damage. Key areas of cybersecurity include network security, which focuses on defending against cyber threats using firewalls and encryption; information security, which ensures data protection; and application security, which prevents vulnerabilities in software. Additionally, cloud security protects cloud-based services, while endpoint security secures individual devices like computers and smartphones. Identity and access management (IAM) controls who can access systems, and incident response involves detecting and mitigating cyber incidents. Cybersecurity is essential for individuals, businesses, and governments to prevent financial losses, data breaches, and reputational harm in an increasingly digital world.

1.2 Advantages of Cybersecurity:

1.**Data Protection** – Safeguards sensitive information from unauthorized access and breaches.

2.**Privacy Enhancement** – Prevents personal and confidential data from being exposed.

3.**Protection Against Cyber Threats** – Defends against malware, ransomware, phishing, and hacking.

4.**Maintaining Customer Trust** – Ensures businesses protect customer data, improving reputation.

5.**Regulatory Compliance** – Helps organizations meet legal and industry security standards.

6. **Business Continuity** – Prevents cyberattacks from disrupting operations and causing financial losses.

7.**Reduced Financial Losses** – Protects businesses from the high costs of data breaches and cyber incidents.

8.**Improved System Performance** – Reduces the risk of slowdowns caused by viruses and malware.

9.**Safeguarding National Security** – Protects government data and critical infrastructure from cyber threats.

10.**Prevention of Identity Theft** – Helps individuals secure personal data from fraudsters and hackers.

11.**Early Threat Detection** – Identifies and mitigates cyber threats before they cause harm.

12.**Protection from Financial Fraud** – Prevents unauthorized transactions and financial scams.

13.**Securing Online Transactions** – Ensures safe online payments through encryption and authentication.

14.**Enhanced Employee Awareness** – Encourages cybersecurity training to prevent human errors.

1.3 Disadvantages of Cybersecurity:

- 1.**High Implementation Costs** – Setting up strong cybersecurity measures can be expensive for businesses.
- 2.**Complexity** – Managing cybersecurity systems requires technical expertise and regular updates.
- 3.**Maintenance and Upgrades** – Continuous monitoring, patching, and software updates are necessary.
- 4.**User Inconvenience** – Strict security measures, like multi-factor authentication (MFA), can slow down user access.
- 5.**Possibility of False Positives** – Security systems may block legitimate activities, causing disruptions.
- 6.**Risk of Insider Threats** – Even with strong cybersecurity, employees or insiders can still cause data breaches.
- 7.**Dependence on Technology** – Over-reliance on automated security tools can lead to gaps in protection.
- 8.**Cybersecurity Skills Shortage** – Finding qualified professionals to manage security systems is challenging.
- 9.**No Absolute Protection** – Even the best cybersecurity measures cannot guarantee 100% security.
- 10.**Privacy Concerns** – Some security measures, like monitoring user activity, may invade privacy.
- 11.**Slow System Performance** – Security software, like antivirus and firewalls, can slow down computers and networks.
- 12.**Risk of Misconfiguration** – Incorrect security settings can create vulnerabilities instead of protection.
- 13.**Legal and Compliance Challenges** – Businesses must keep up with evolving cybersecurity laws and regulations.
- 14.**Increased Complexity for Small Businesses** – Smaller companies may struggle to implement advanced security measures.

2. Different types of attacks in cybersecurity:

1. Brute force
2. Broken Authentication
3. Sensitive data exposure
4. SQL Injection

2.1 Brute Force



A brute force attack is a hacking method where an attacker systematically tries all possible combinations of passwords, encryption keys, or login credentials until they find the correct one. This technique relies on computational power and persistence rather than exploiting software vulnerabilities. There are several types of brute force attacks, including simple brute force attacks, which try every possible combination; dictionary attacks, which use a list of commonly used passwords; and hybrid attacks, which combine dictionary words with numbers or symbols. Other variations include credential stuffing, where attackers use leaked username-password pairs, and reverse brute force attacks, which apply a common password to multiple usernames. To prevent brute force attacks, users and organizations should implement strong, complex passwords, enable multi-factor authentication (MFA), limit login attempts, use CAPTCHA to block bots, and monitor authentication logs for unusual activity. These measures help strengthen security and reduce the risk of unauthorized access.

Impact of Brute Force Attacks

- **Unauthorized Access** – Attackers can gain control over accounts, systems, or databases.
- **Data Breaches** – Sensitive personal or business data can be stolen, leading to privacy violations.
- **Financial Losses** – Stolen banking credentials or fraud can result in direct financial damage.
- **Identity Theft & Fraud** – Compromised personal data can be used for fraudulent activities.
- **System Downtime** – Continuous login attempts can overwhelm servers, causing service disruptions.
- **Reputational Damage** – Businesses can lose customer trust and credibility.
- **Legal & Regulatory Consequences** – Companies may face fines or legal actions for failing to protect user data.
- **Credential Stuffing Risks** – If passwords are reused, multiple accounts can be compromised across platforms.
- **Increased Security Costs** – Organizations must invest in cybersecurity measures to prevent and mitigate attacks.
- **Ransomware & Malware Deployment** – Attackers can use access to install malicious software.

Examples of Brute Force Attacks

1. **Attack on Yahoo (2012-2016)** – One of the largest data breaches in history, where attackers used credential stuffing (a type of brute force attack) to gain access to user accounts, affecting billions of users.
2. **Mirai Botnet (2016)** – The Mirai malware used brute force attacks on IoT devices with weak or default passwords, turning them into a massive botnet that launched distributed denial-of-service (DDoS) attacks.
3. **Facebook Hack (2019)** – Attackers used brute force techniques to exploit vulnerabilities in Facebook's system, compromising millions of user accounts.
4. **Tesla Cloud Attack (2018)** – Hackers gained access to Tesla's cloud environment through brute force methods, allowing them to mine cryptocurrency using Tesla's computing resources.
5. **WordPress Website Attacks** – Hackers often target WordPress sites using automated brute force tools to guess admin login credentials, leading to website defacement or data theft.
6. **Instagram Account Hacks** – Attackers use automated brute force attacks to guess weak passwords of Instagram users, often leading to account takeovers and scams.
7. **Windows Remote Desktop Protocol (RDP) Attacks** – Cybercriminals frequently use brute force techniques to crack RDP login credentials, gaining access to corporate networks and deploying ransomware.
8. **Gmail and Email Account Breaches** – Hackers use brute force methods to access email accounts, leading to phishing attacks, identity theft, and financial fraud.

Literature Survey

Brute force:

Invented:

Brute force attacks were not invented by a specific individual but evolved as a natural method of breaking codes and solving cryptographic challenges. The concept dates back to ancient times when early civilizations, such as the Romans and Greeks, used simple ciphers that could be deciphered by systematically testing different letter substitutions. During World War II, Alan Turing and his team at Bletchley Park used systematic techniques to crack the Enigma machine, which involved testing various encryption keys, a form of brute force. With the advancement of computing technology, brute force attacks became more sophisticated, allowing hackers and security professionals to automate the process and test billions of password combinations in a short time. Today, brute force attacks remain a common cybersecurity threat, used by both attackers and ethical hackers to test system vulnerabilities

First Brute force attack:

The first known brute force attack dates back to **World War II**, when Alan Turing and his team at **Bletchley Park** worked to break the German **Enigma machine** by systematically testing encryption keys. In modern computing, brute force attacks became common in the **1970s and 1980s** with the rise of password-protected systems. A major example occurred in **1997**, when the **Electronic Frontier Foundation (EFF)** cracked the **Data Encryption Standard (DES)** using brute force, proving its vulnerability.

Brute force:

Practical example on our project:

Scenario 1:

If we use Django authentication system and encryption techniques

Using **Django's authentication system** along with strong encryption techniques significantly enhances security and helps prevent brute force attacks.

Django provides built-in mechanisms for secure authentication, including **hashed passwords, authentication middleware, and user session management.**

Views.py for register and login page

```
def register(request):
    if request.method == 'POST':
        email = request.POST['email']
        password = request.POST['password']
        user = User.objects.create(email=email, password=password)
        return HttpResponseRedirect('User created successfully')
    return render(request, 'register.html')

def login(request):
    if request.method == 'POST':
        email = request.POST['email']
        password = request.POST['password']
        user = User.objects.filter(email=email, password=password).first()

        if user:
            user.session_id = str(random.randint(1000, 9999)) #weak session id
            user.save()
            response= redirect('dashboard',user_id=user.id)
            response.set_cookie('sessionid', user.session_id) # weak cookie
            return response
        else:
            return HttpResponseRedirect('Invalid Credentials. <a href="/register/">Register here</a>')

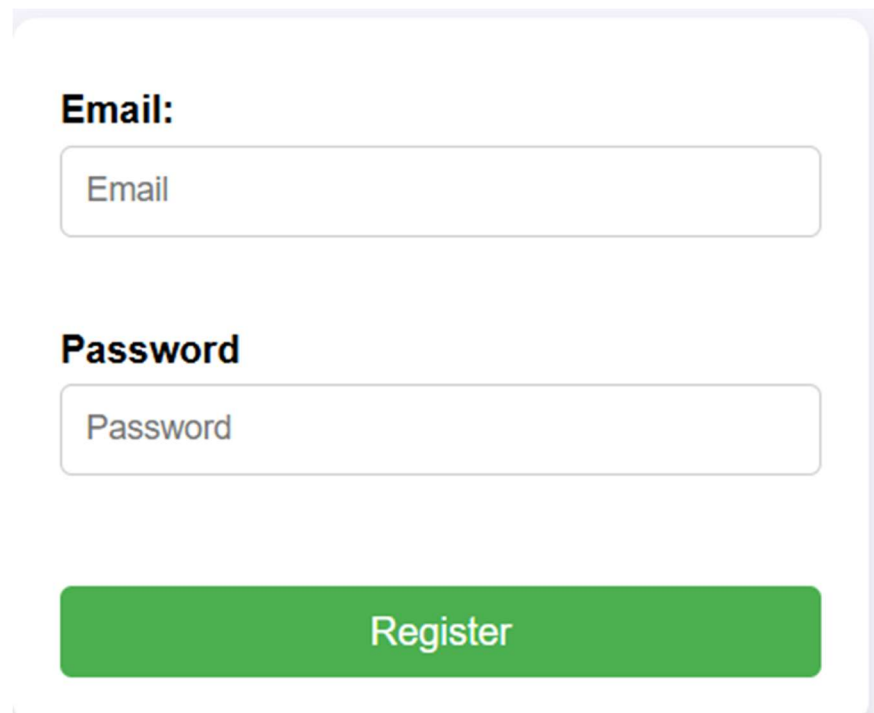
    return render(request, 'login.html')
```


In Database

<u>id</u>	password	email	session_id
Fil...	Filter	Filter	Filter
1	password@123	admin@gmail.com	<i>NULL</i>
2	password@123	manju@gmail.com	3251
3	mateen	manoj@gmail.com	6883
4	12344321	ram@gmail.com	3882

How does it work:

1.First we should register new user in register page. use new email and password.

A registration form with a light purple border. It contains two input fields: one for 'Email' and one for 'Password'. Below these fields is a green 'Register' button. The labels 'Email:' and 'Password' are in bold black text.

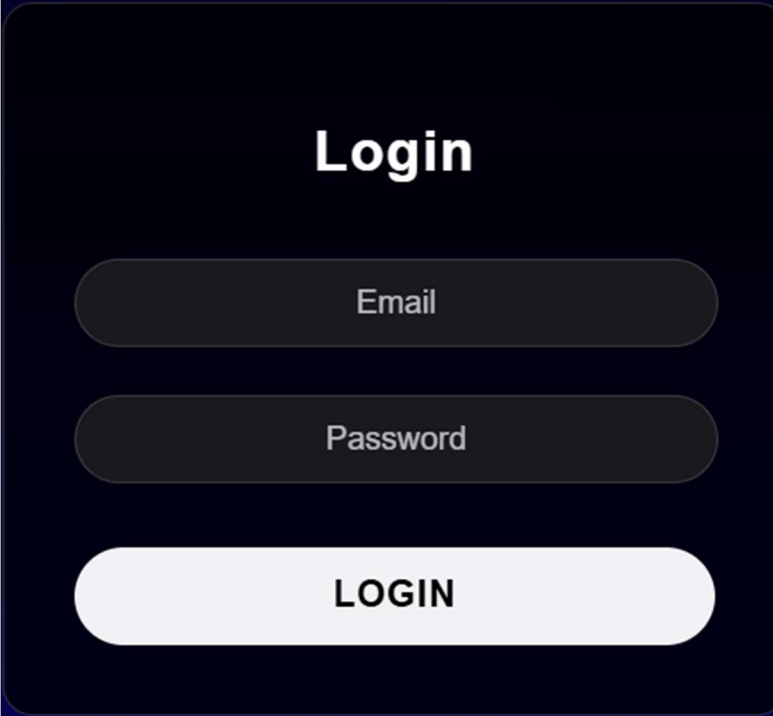
Email:

Password

Register

A new user can register by filling out the registration form with their email and password. The form consists of two input fields: one for the email and another for the password. Once the user enters the required information, they must click the "Register" button to submit the form. The system then processes the details, validates the email format and password strength, and checks for duplicate accounts. If all conditions are met, a new account is successfully created, and the user may receive a confirmation email or be redirected to the login page.

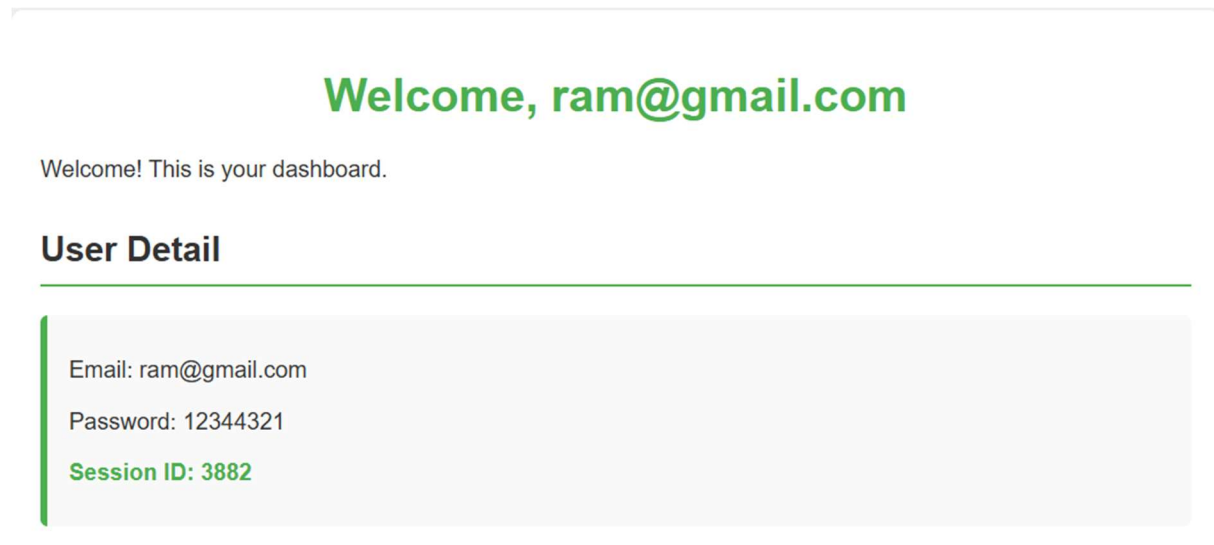
2. Then login with the email and password which you have created while registering a new user.



The image shows a login interface. It has a dark blue background. At the top, the word "Login" is centered in a white, bold font. Below this, there are two input fields. The first is labeled "Email" and the second is labeled "Password". Both labels are in a light gray font. Below these fields is a large, white, rounded rectangular button with the word "LOGIN" in a dark blue, bold font.

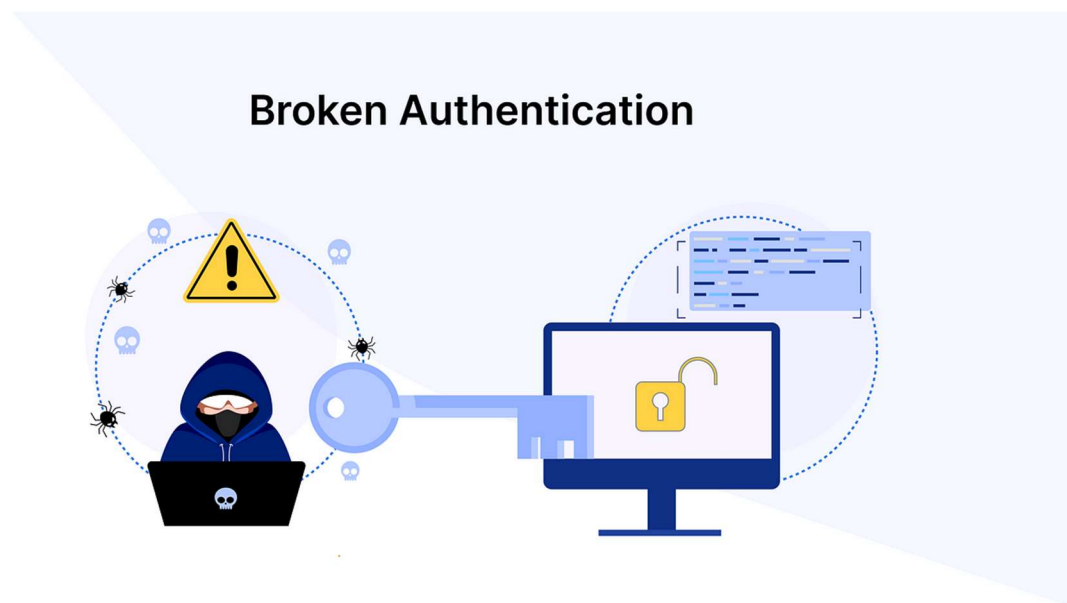
To log in, a user must enter their registered email and password into the respective input fields on the login form. Once the credentials are entered, they need to click the "LOGIN" button to proceed. The system then verifies the provided information by checking it against stored user data. If the credentials match an existing account, the user is granted access to their dashboard or homepage. If the login attempt fails due to incorrect credentials, the system may display an error message prompting the user to re-enter their details or reset their password.

3. After login a dashboard appears information of users



4. open vs code and open terminal run code `py bruteforce1.py` after it checks the password list and identifies the correct password. After identifying the correct password for email it shows success

2.2 Broken authentication



Broken authentication is a security vulnerability that occurs when authentication mechanisms are weak or improperly implemented, allowing attackers to gain unauthorized access to accounts. This can happen due to weak password policies, credential stuffing, brute force attacks, session hijacking, or insecure storage of credentials. For example, if users are allowed to set weak passwords or if authentication systems do not limit failed login attempts, attackers can easily guess or use stolen credentials to access accounts. Additionally, if session tokens are not properly managed or secured, an attacker could hijack an active session and impersonate a legitimate user. To prevent broken authentication, organizations should enforce strong password policies, implement multi-factor authentication (MFA), limit login attempts, use secure session management techniques, and store passwords securely using strong hashing algorithms like bcrypt or Argon2. Additionally, ensuring all authentication-related communications are encrypted with HTTPS and monitoring login activities for suspicious behavior can help mitigate the risks. By strengthening authentication mechanisms, organizations can prevent unauthorized access and protect sensitive user data.

Impact of Broken Authentication

1. **Account Takeover** – Attackers can gain unauthorized access to user accounts, leading to identity theft or fraudulent transactions.
2. **Data Breach** – Sensitive user or organizational data can be exposed, leading to financial and reputational damage.
3. **Unauthorized System Access** – Attackers may gain control over administrative accounts, compromising the entire system.
4. **Financial Loss** – Stolen credentials can be used for fraudulent transactions, causing direct financial harm.
5. **Reputation Damage** – Companies lose customer trust and credibility after security breaches caused by weak authentication.
6. **Compliance Violations** – Failure to secure authentication mechanisms can lead to non-compliance with regulations like GDPR, HIPAA, or PCI-DSS, resulting in legal penalties.
7. **Session Hijacking Consequences** – Attackers who hijack sessions can act as legitimate users, potentially making unauthorized changes or stealing information.
8. **Increased Phishing and Social Engineering Attacks** – If authentication is weak, attackers can trick users into revealing credentials, leading to further exploitation.
9. **Service Disruptions** – Attackers may use compromised accounts to disrupt services, delete data, or perform malicious actions within an organization.
10. **Credential Reuse Attacks** – Stolen credentials from one system can be used to access other accounts if users reuse passwords across multiple platforms.

Examples of Broken Authentication

1. Yahoo Data Breach (2013-2014)

– Over **3 billion user accounts** were compromised due to weak authentication and stolen credentials. Attackers gained access to email accounts, personal details, and security answers, leading to widespread identity theft.

2. Facebook Access Token Leak (2018)

– A vulnerability in Facebook's authentication system exposed **50 million user access tokens**, allowing attackers to take over user sessions without needing passwords. This enabled them to access private messages, photos, and connected third-party services.

3. Uber Data Breach (2016)

– Hackers used **stolen credentials** from a private GitHub repository to access Uber's cloud storage, exposing **57 million user and driver records**. The breach remained undisclosed for a year, damaging Uber's reputation.

4. Marriott Hotel Breach (2018)

– Attackers gained unauthorized access to Marriott's database using **compromised login credentials**, exposing sensitive data of **500 million guests**, including passport numbers and payment details.

5. Tesla API Authentication Issue (2020)

– Security researchers found that Tesla's API did not enforce strong session management, allowing attackers to **remotely control vehicles** if they had access to a user's authentication tokens.

6. Instagram Account Takeover Vulnerability (2019)

– A flaw in Instagram's password reset process allowed attackers to **bypass authentication** and take control of user accounts, leading to unauthorized access and privacy violations.

7. Twitter Password Storage Issue (2018)

– Twitter mistakenly stored user passwords in **plaintext logs** instead of hashing them, making them vulnerable to internal leaks and potential cyberattacks.

Literature survey

Invented:

Broken authentication is not something that was "invented" by a specific person but rather a category of security vulnerabilities that emerged as authentication systems evolved. As computer systems and the internet grew, early authentication mechanisms relied on simple username-password combinations. Over time, weaknesses in these systems were exploited by attackers, leading to the identification of broken authentication as a critical security issue. Organizations like OWASP (Open Web Application Security Project) have played a key role in defining and raising awareness about broken authentication, listing it as one of the **OWASP Top 10** security risks. The concept has developed as cybersecurity experts, researchers, and ethical hackers have discovered vulnerabilities in authentication methods, prompting improvements such as multi-factor authentication (MFA), secure session management, and strong password hashing techniques.

First Broken authentication attack:

The **first recorded broken authentication attack** is difficult to pinpoint because authentication weaknesses have existed since the early days of computing. However, one of the earliest known incidents related to broken authentication was the **Morris Worm** in 1988.

The **Morris Worm**, created by Robert Tappan Morris, exploited weak authentication mechanisms in Unix systems by using password guessing techniques. It attempted to log in using common and default passwords, leveraging weak authentication as a means of spreading across networks. This attack highlighted the dangers of poor password security and led to increased awareness of authentication vulnerabilities.

If we use Django authentication system and encryption techniques

If you use Django's authentication system along with proper encryption techniques, you can significantly reduce the risk of **broken authentication** vulnerabilities. Django provides built-in security mechanisms to protect user credentials, manage sessions securely, and enforce best authentication practices.

Views.py for custom login and dashboard

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login, logout
from django.http import HttpResponseRedirect
from django.contrib.auth.decorators import login_required

# Create your views here.
def custom_login(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)

            session_key=request.session.session_key
            if session_key is None:
                request.session.create()
                session_key=request.session.session_key

            print(f"User{user.username} logged in with session key: {session_key}")
            return redirect('/dashboard')
        else:
            return HttpResponseRedirect('Invalid Credentials', status=401)
    return render(request, 'login.html')

@login_required(login_url='/login')
def dashboard(request):
    return render(request, 'dashboard.html',{'user':request.user})

def custom_logout(request):
    logout(request)
    return redirect('/login')
```

DB

	id	password	last_login	is_superuser	username	last_name	email	is_staff	is_active	date_joined	first_name
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	pbkdf2_sha256\$870000\$adjMI0d35r5FkA...	2025-03-09 05:28:14.562396	1	haegl		haegl@gmail.com	1	1	2025-03-05 05:46:47.116549	
2	2	pbkdf2_sha256\$870000\$14cEQCplyFu19r...	2025-03-09 12:28:01.294912	1	admin		admin@gmail.com	1	1	2025-03-05 05:47:23.950750	
3	3	pbkdf2_sha256\$870000\$e47gA81VrMVXQi...	2025-03-09 05:29:46.812397	0	manju			0	1	2025-03-09 05:29:24.815106	

How it works

- 1 Run the project & open it in two browser
- 2 Login on both browser with different user
- 3 Right click Inspect Applications Cookies Copy Session id value
- 4 Paste Session id value in another logged in browser and refresh

Run the project & open it in two browser

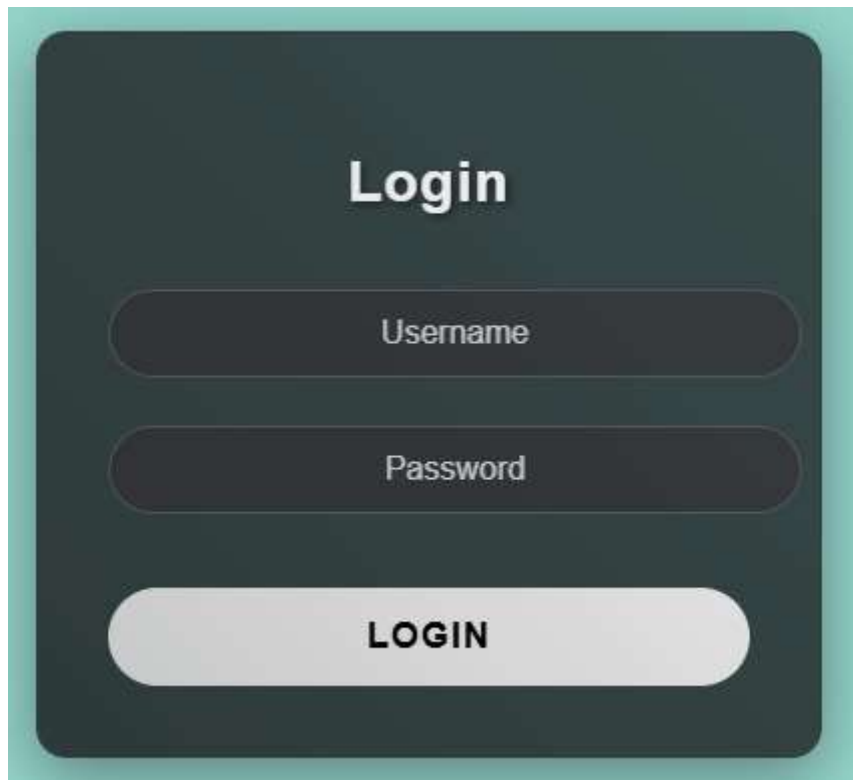
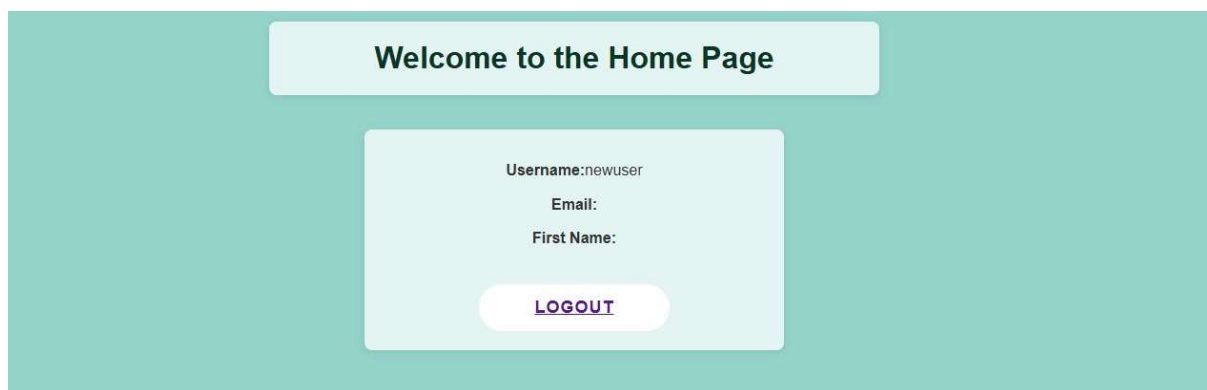
```
PS C:\Users\Manjunath\OneDrive\Desktop\broken_auth_demo> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 09, 2025 - 19:34:35
Django version 5.1.6, using settings 'broken_auth_demo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

When you run a Django project using `python manage.py runserver` and open it in two different browsers, Django treats each browser session separately. When you access the application in the first browser, Django creates a session for that visit, storing a session ID in a secure cookie. If you log in, Django associates that session with your user account. When you open the same URL in a second browser, Django creates a new session, treating it as a separate visit, even if you are the same user. If you log in from the second browser, Django allows multiple active sessions unless restricted by settings. Logging out in one browser may or may not log you out in the other, depending on session management settings like `SESSION_EXPIRE_AT_BROWSER_CLOSE`. If Django is configured to enforce strict session handling, logging out in one browser might invalidate the session across all devices. This behavior ensures that authentication and session security are maintained across multiple instances of the application.

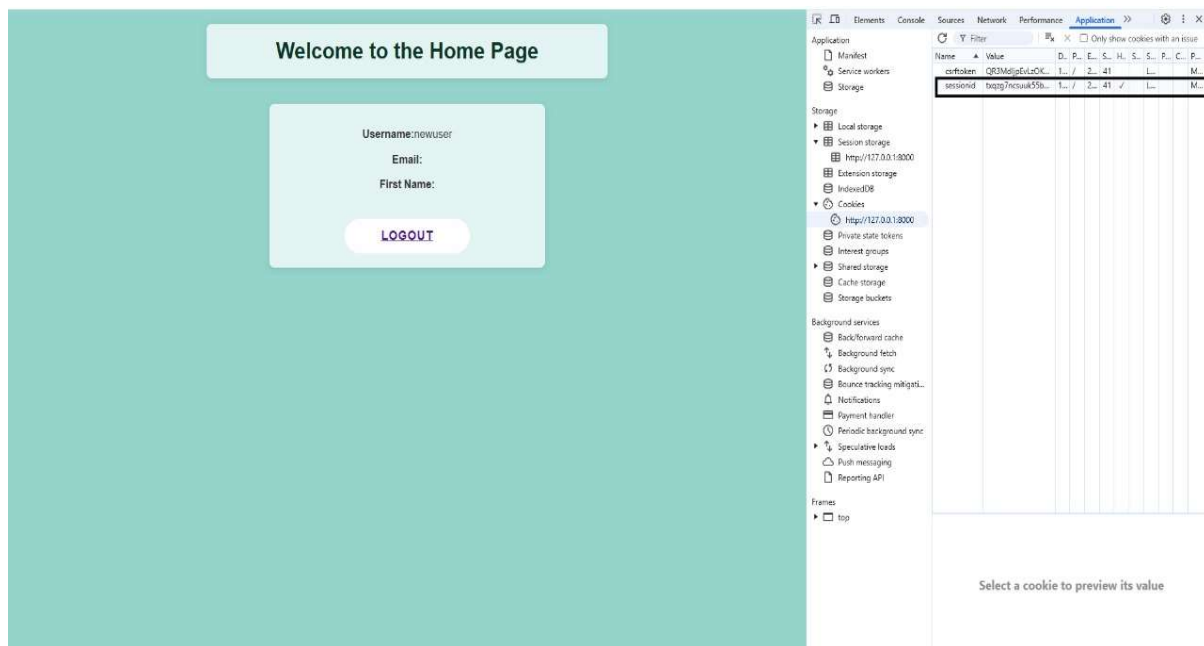
Login on both browser with different user

Login

A login form with a dark teal background. At the top, the word "Login" is written in white. Below it are two rounded rectangular input fields: the first is labeled "Username" and the second is labeled "Password". At the bottom is a large, light gray rounded rectangular button with the word "LOGIN" in bold black capital letters.A home page with a light teal background. At the top, a white rounded rectangular button contains the text "Welcome to the Home Page". Below this is a white rounded rectangular box containing the text "Username:newuser", "Email:", and "First Name:". At the bottom of this box is a white rounded rectangular button with the word "LOGOUT" in purple capital letters.

When you log in to a Django website on two different browsers using two different user accounts, Django creates separate sessions for each user. Each browser maintains its own session, ensuring that User A in Browser 1 and User B in Browser 2 have independent access to their respective accounts. This means that each user can navigate the site, view their personal data, and perform actions without interfering with the other. If one user logs out, it only affects their session in that specific browser, while the other user remains logged in. Django's session management ensures that user authentication remains isolated and secure across different browsers.

Right click ,Inspect ,Applications ,Cookies ,Copy Session id value



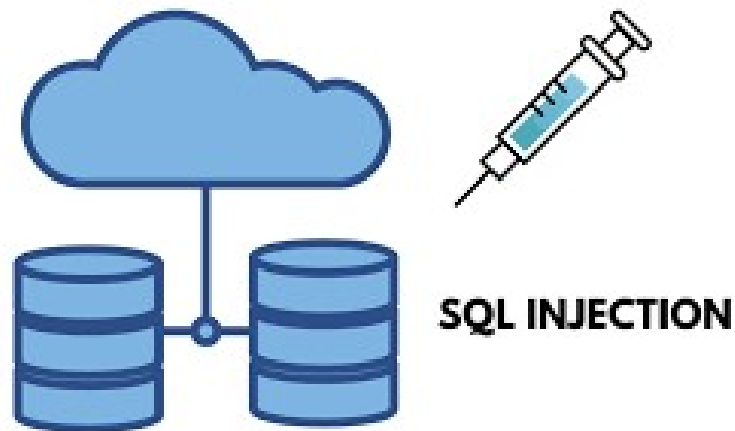
When you right-click on a Django web page, select **Inspect**, go to the **Application** tab, and navigate to **Cookies**, you can find the session ID stored by Django. Under the **Storage** section, clicking on **Cookies** and selecting your website's URL (e.g., <http://127.0.0.1:8000/>) will display all stored cookies. Among them, the **sessionid** cookie holds the session identifier that Django uses to track user authentication. You can copy the session ID value from the **Value** column, but it is crucial to handle it securely. If an attacker gains access to this session ID, they could potentially hijack a user's session and access their account. To prevent such risks, Django provides security features like `SESSION_COOKIE_SECURE = True` to restrict session cookies to HTTPS, and `SESSION_EXPIRE_AT_BROWSER_CLOSE = True` to clear sessions upon closing the browser.

Paste Session id value in another logged in browser and refresh



If you copy the **session ID value** from one logged-in browser and paste it into another logged-in browser's cookies, then refresh the page, the second browser will take over the session of the first one. This happens because Django identifies users based on their **session ID stored in cookies**. When you replace the session ID in **Browser 2** with the one from **Browser 1** and refresh the page, Django assumes that Browser 2 is the same as Browser 1 and logs in as that user. This technique, known as **session hijacking**, highlights the importance of securing session management. To prevent such attacks, Django provides security features like `SESSION_COOKIE_SECURE = True` to enforce HTTPS-only cookies, `SESSION_COOKIE_HTTPONLY = True` to prevent JavaScript access, and `SESSION_COOKIE_AGE` to expire sessions after a set time.

2.3 SQL injection



SQL Injection (SQLi) is a critical web security vulnerability that allows attackers to manipulate a website's database by injecting malicious SQL code. This occurs when user input is not properly validated and is directly included in SQL queries, allowing attackers to modify the intended behavior of the query. For example, in a login form where the application constructs a query like `SELECT * FROM users WHERE username = 'user' AND password = 'password';`, an attacker could input `"admin' --"` as the username, effectively bypassing authentication. SQL injection can be used to steal sensitive data, modify or delete records, and even gain administrative control over the database. There are different types of SQL injection, such as Union-based, Error-based, and Blind SQL injection, each with varying methods of extracting data. To prevent SQL injection, developers should use prepared statements and parameterized queries, validate and sanitize user input, implement Web Application Firewalls (WAFs), and enforce the principle of least privilege in database access. Failure to prevent SQL injection can lead to severe consequences, including data breaches, financial losses, and reputational damage for businesses.

Impacts of SQL Injection

1. **Data Theft** – Attackers can steal sensitive data, including usernames, passwords, credit card details, and personal information.
2. **Unauthorized Access** – SQL Injection can allow attackers to bypass authentication and gain admin-level access to a website or application.
3. **Data Manipulation** – Hackers can modify, delete, or insert data, leading to corruption or loss of critical information.
4. **Website Defacement** – Attackers can alter website content, damaging a company's reputation and credibility.
5. **Financial Loss** – Businesses may suffer financial losses due to data breaches, legal penalties, and compensation to affected users.
6. **System Takeover** – In severe cases, SQL Injection can be used to execute system commands, allowing full control of the server.
7. **Denial of Service (DoS)** – Attackers can overload the database with malicious queries, causing the application to crash or become unresponsive.
8. **Regulatory Non-Compliance** – A data breach due to SQL Injection can result in violations of GDPR, HIPAA, or other data protection laws, leading to legal consequences.
9. **Loss of Customer Trust** – Users may lose confidence in a company's security practices, leading to a decline in customer retention and brand reputation.
10. **Backdoor Creation** – Attackers can inject persistent malicious code into the database, allowing future unauthorized access even after initial fixes.

Examples of SQL Injection Attacks

1. **Login Bypass** – An attacker enters ' OR '1'='1 as the username and any password, making the SQL query always true, bypassing authentication.
2. **Extracting Database Information (Union-Based Attack)** – Using UNION SELECT to retrieve sensitive data from other tables.
3. **Dumping Database Tables (Error-Based Injection)** – Injecting a payload that forces the database to return error messages containing table names.
4. **Blind SQL Injection (Boolean-Based)** – An attacker checks if a query returns true or false without seeing the data directly.
5. **Modifying Data (Update Query Injection)** – An attacker injects SQL to change user passwords.
6. **Deleting Records (Destructive SQL Injection)** – Malicious input that deletes all user data.
7. **Creating an Admin Account (Privilege Escalation)** – Injecting SQL to insert a new admin user.
8. **Extracting Database Version and System Info** – Using system functions to gather details about the database.
9. **Denial of Service (DoS) Attack** – Injecting queries that overload the database with resource-heavy operations.
10. **Command Execution (Advanced SQL Injection)** – If a database allows it, an attacker can execute system commands (only possible in certain configurations).

Literature survey

Invented

SQL Injection was not intentionally "invented" but rather discovered as a vulnerability. The first public mention of SQL Injection appeared in **1998**, when security researcher **Jeff Forristal (Rain Forest Puppy)** published an advisory detailing how attackers could exploit improperly sanitized SQL queries to manipulate databases. His work highlighted the risks of user input being directly included in SQL statements, leading to data breaches, unauthorized access, and other security threats. Since then, SQL Injection has remained one of the most critical and well-known web vulnerabilities, often ranking high in OWASP's **Top 10 Web Security Risks**.

First attack of SQL injection

The **first recorded SQL Injection attack** occurred in **1998**, discovered by security researcher **Jeff Forristal (Rain Forest Puppy)**. He documented how attackers could manipulate SQL queries by injecting malicious input into web applications that failed to properly sanitize user input.

One of the **most well-known early attacks** happened in **2002**, when an SQL Injection vulnerability was exploited against **Microsoft's SQL Server-based websites**. However, one of the **most significant large-scale attacks** was in **2008**, when **Heartland Payment Systems**, a financial services company, was breached using SQL Injection, exposing over **130 million credit card numbers**.

Since then, SQL Injection has remained a **top security threat**, frequently used in cyberattacks against companies, governments, and organizations worldwide. It is still listed as one of the most critical vulnerabilities in OWASP's **Top 10 Web Security Risks**.

If we use Django authentication system and encryption techniques

Using Django's authentication system and encryption techniques significantly enhances security by protecting against SQL injection and other vulnerabilities. Django's **Object-Relational Mapping (ORM)** automatically escapes SQL queries, preventing direct execution of malicious input, making it inherently safe from SQL injection. Additionally, Django's authentication system securely handles user credentials by hashing passwords using strong algorithms like PBKDF2, bcrypt, or Argon2 instead of storing them in plain text. It also provides built-in features such as user authentication, session management, and permission handling, reducing the risk of unauthorized access. Furthermore, encryption techniques like Django's cryptography module or third-party libraries can be used to encrypt sensitive data such as user information, API keys, and financial details. By leveraging Django's security features, developers can create secure applications with minimal risk of common vulnerabilities like SQL injection, data breaches, and credential leaks.

Views.py for login and user

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from django.db import connection

# Create your views here.
def vulnerable_login(request):
    if request.method == 'GET':
        return render(request, 'login.html')

    username = request.POST.get('username')
    password = request.POST.get('password')

    query = f"select * from authentication_user where username='{username}' and password='{password}'"
    print("Executing Query:", query)

    with connection.cursor() as cursor:
        cursor.execute(query)
        user = cursor.fetchone()

    if user:
        return HttpResponseRedirect("Login Successful")
    return HttpResponseRedirect("invalid credentials")
```

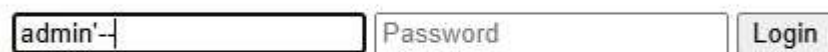
DB

id	username	password
Filter	Filter	Filter
1	admin	12345

How it works

1. At first give any name in login page, which possibly exist in database.
2. At the password column, write a malicious SQL injection code.
Ex: "OR 1=1"
3. If the user's password is not encrypted, it directly redirects to main patients dashboard.
4. Here attacker is successful in performing SQL injection attack.

Login from



The screenshot shows a login interface with the heading "Login from". Below the heading are two input fields and a "Login" button. The first input field, which is for the username, contains the text "admin'--". The second input field is labeled "Password" and is empty. The "Login" button is to the right of the password field. The entire form is enclosed in a light gray border.

Here, when he click on login. If the SQL command satisfies the condition it can easily redirect to patients dashboard, which may lead in sensitive information leakage. It will result in Data integrity failure, lose of customers trust, data breach, also affect organizations growth etc.

So, it is very necessary to follow good coding practices, such that these types of attacks can be easily defended and our website will be protected.

2.4 Sensitive data exposure



Sensitive data exposure occurs when confidential or personal information, such as passwords, financial details, health records, or personally identifiable information (PII), is inadequately protected and becomes accessible to unauthorized users. This can result from weak encryption, improper access controls, insecure data storage, or transmission over unprotected channels. Attackers can exploit these vulnerabilities to steal, manipulate, or misuse sensitive data, leading to financial loss, identity theft, or regulatory penalties. To prevent sensitive data exposure, organizations should implement strong encryption, enforce access controls, use secure communication protocols, and regularly assess their security measures.

Impact

1. **Financial Loss** – Organizations may face fines, legal fees, and compensation costs due to data breaches.
2. **Identity Theft** – Exposed personal information can be used for fraudulent activities, such as opening unauthorized accounts.
3. **Reputation Damage** – Loss of customer trust and brand credibility can negatively impact business growth.
4. **Legal and Regulatory Consequences** – Violations of data protection laws (e.g., GDPR, CCPA) can lead to hefty penalties.
5. **Operational Disruptions** – Organizations may need to shut down services temporarily to address the breach.
6. **Competitive Disadvantage** – Exposure of trade secrets or proprietary data can benefit competitors.
7. **Increased Cybersecurity Risks** – Attackers may exploit leaked data to launch further attacks, such as phishing or ransomware.
8. **Loss of Customer Trust** – Customers may stop using services if they feel their data is not secure.
9. **Legal Liabilities** – Organizations may face lawsuits from affected individuals or businesses.
10. **National Security Threats** – Exposure of sensitive government or defense-related data can lead to serious security risks.

Examples of Sensitive Data Exposure:

1. **Unencrypted Database Leak** – A company stores customer credit card details in plaintext, and a hacker gains access to the database.
2. **Misconfigured Cloud Storage** – Sensitive business documents are left publicly accessible due to incorrect cloud storage settings.
3. **Insecure API Exposure** – A healthcare app exposes patient records due to improper API authentication.
4. **Data Breach from Phishing** – An employee falls for a phishing attack, leading to the exposure of login credentials.
5. **Lack of HTTPS Encryption** – A website transmits login details over HTTP, allowing attackers to intercept sensitive information.
6. **Accidental Email Disclosure** – A company mistakenly sends confidential client data to the wrong email recipient.
7. **Third-Party Breach** – A vendor handling payroll data is breached, exposing employee financial information.
8. **Printing and Disposal Errors** – Sensitive documents are improperly discarded, leading to unauthorized access.
9. **Publicly Shared Code Repositories** – Developers accidentally upload API keys or database credentials to public repositories like GitHub.
10. **Lost or Stolen Devices** – A company laptop without encryption is stolen, exposing sensitive corporate data.

Literature survey

Invented

Sensitive data exposure is not an invention but a cybersecurity vulnerability that has existed since the early days of digital data storage and communication. It became a recognized security issue as technology advanced and organizations began storing sensitive information electronically.

However, it was formally categorized as a security vulnerability by organizations like **OWASP (Open Web Application Security Project)**, which included it in their **OWASP Top 10** list of web application security risks. OWASP has played a major role in raising awareness about sensitive data exposure and providing best practices for preventing it.

First attack of Sensitive data exposure

One of the earliest recorded cases of sensitive data exposure occurred in 1984 with the **TRW Information Systems data breach** (now part of Experian). A hacker discovered an administrative password that was left exposed online, granting unauthorized access to TRW's credit reporting database, which contained the financial records of **90 million Americans**. This breach highlighted the dangers of weak security practices, such as poor password management and inadequate access controls. It served as an early warning about the risks of sensitive data exposure, emphasizing the need for encryption, secure authentication, and stricter cybersecurity measures to protect sensitive information.

If we use Django authentication system and encryption techniques

Using Django's authentication system and encryption techniques helps prevent **sensitive data exposure** by securing user credentials and sensitive information. Django hashes passwords using **PBKDF2**, supports encryption with **Fernet**, and enforces **HTTPS** for secure data transmission. It also includes security middleware to protect against **XSS, SQL injection, and clickjacking** while providing robust **access controls and permissions**. Storing credentials in **environment variables** further enhances security. Proper implementation of these features ensures a **secure web application** with minimal risk of data leaks.

Views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth import login, logout
from django.contrib.auth import authenticate
from .models import CustomUser

def register_staff(request):
    if request.method == 'POST':
        first_name = request.POST['first_name']
        last_name = request.POST['last_name']
        username = request.POST['username']
        password = request.POST['password']
        phone_number = request.POST['phone_number']
        aadhar_number = request.POST['aadhar_number']
        address = request.POST['address']
        account_number = request.POST['account_number']
        ifsc_code = request.POST['ifsc_code']

        # Create a new staff user
        user = CustomUser.objects.create_user(
            username=username, first_name=first_name, last_name=last_name,
            password=password, user_type='staff', phone_number=phone_number,
            aadhar_number=aadhar_number, address=address, account_number=account_number,
            ifsc_code=ifsc_code)

        return redirect('admin_dashboard') #redirect to admin dashboard
    return render(request, 'register_staff.html')
```



```
def admin_dashboard(request):
    if request.user.is_authenticated and request.user.user_type == 'admin':
        staff_users = CustomUser.objects.filter(user_type='staff') #fetch all staff users
        return render(request, 'admin_dashboard.html', {'staff_users': staff_users})
    else:
        return redirect('user_login')

def staff_dashboard(request):
    user=request.user

    if request.user.is_authenticated:
        return render(request, 'staff_dashboard.html',{'user': user})
    else:
        return redirect('user_login')

def logout_view(request):
    logout(request)
    return redirect('user_login')
```

```
def add_staff(request):
    if not request.user.is_authenticated or request.user.user_type != 'admin':
        return redirect('user_login')

    if request.method == 'POST':
        first_name = request.POST['first_name']
        last_name = request.POST['last_name']
        username = request.POST['username']
        password = request.POST['password']
        phone_number = request.POST['phone_number']
        aadhar_number = request.POST['aadhar_number']
        address = request.POST['address']
        account_number = request.POST['account_number']
        ifsc_code = request.POST['ifsc_code']

        try:
            user = CustomUser.objects.create_user(
                username=username,
                first_name=first_name,
                last_name=last_name,
                password=password,
                user_type='staff',
                phone_number=phone_number,
                aadhar_number=aadhar_number,
                address=address,
                account_number=account_number,
                ifsc_code=ifsc_code
            )
            return redirect('admin_dashboard')
        except Exception as e:
            return render(request, 'add_staff.html', {'error': str(e)})

    return render(request, 'add_staff.html')
```

How it Works

1. Start the Django Project using command `python manage.py runserver`
2. Open your web browser,
2. login to admin dashboard and then add the staff
3. Login to the Staff Account
4. try to Access the Admin Dashboard using Debug Mode Enabled in landing page.

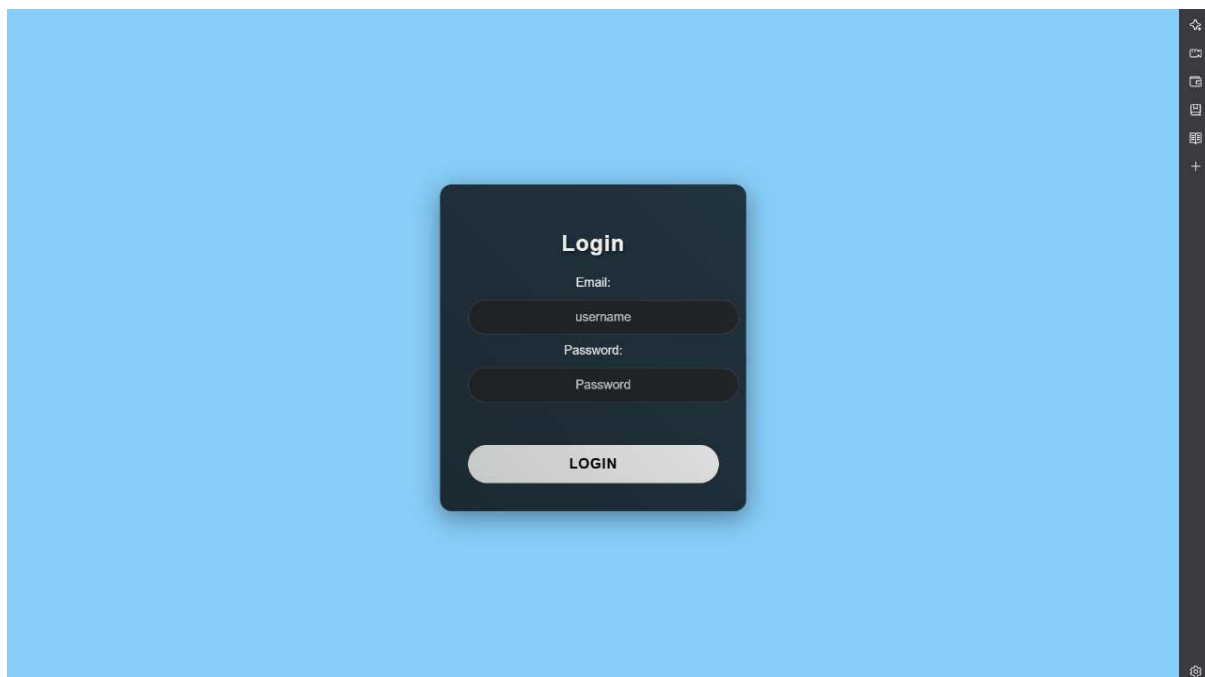
Runserver

```
PS C:\Users\haree\Videos\sensitive_data_exposure\cs\sensitive_data_exposure\Sensitive_Data_Exposure> py manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 09, 2025 - 23:11:38
Django version 5.1.6, using settings 'Sensitive_Data_Exposure.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[09/Mar/2025 23:11:38] "GET /add-staff/ HTTP/1.1" 302 0
[09/Mar/2025 23:11:38] "GET /login/ HTTP/1.1" 200 3758
data hareesh password@123
[09/Mar/2025 23:11:48] "POST /login/ HTTP/1.1" 302 0
[09/Mar/2025 23:11:48] "GET /admin-dashboard/ HTTP/1.1" 200 3474
[09/Mar/2025 23:11:53] "GET /add-staff/ HTTP/1.1" 200 3907
[09/Mar/2025 23:18:31] "GET /add-staff/ HTTP/1.1" 200 3907
Not Found: /favicon.ico
[09/Mar/2025 23:18:31] "GET /favicon.ico HTTP/1.1" 404 3545
[09/Mar/2025 23:18:31] "GET /add-staff/ HTTP/1.1" 200 3907
```

Open browser



login to admin dashboard and then add the staff

Phone: 9019725714

Aadhaar: 456123789456

Address: jsbckjs

Bank Details: 45679846 (IFSC: 456)

Name: gm gm

Username: gm1

Phone: 12

Aadhaar: dsabdkjb

Address: jbdkjsf

Bank Details: cssjdbckdbj (IFSC: dskbjcksdj)

Name: staff_name

Username: staff

Phone: 1234567890

Aadhaar: 123456789012

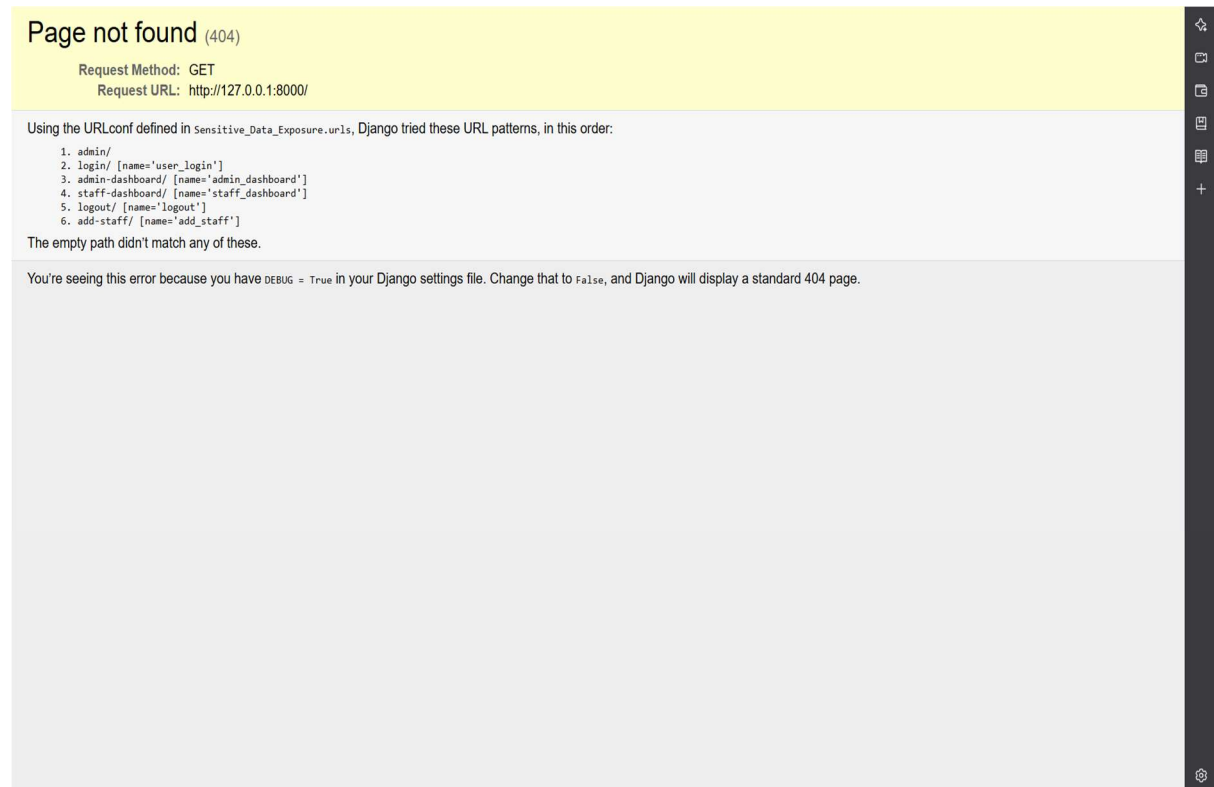
Address: hubli

Bank Details: 1234567890 (IFSC: 12345)

[ADD NEW STAFF](#) [LOGOUT](#)



try to Access the Admin Dashboard using Debug Mode Enabled in landing page.



3.System Specification

3.1 Software Specification:

- Operating System - Windows 10,11

Operating System: The system is compatible with both Windows 10 and Windows 11 operating systems. Users can choose to run the software on either of these versions of Windows.

- Web Browser-Google Chrome, Microsoft Edge

Web Browser: The software is optimized to work with Google Chrome and Microsoft Edge web browsers. Users can access the system using either of these browsers for an optimal experience.

3.2Hardware Specification

- Processor Intel Core i5

The system requires an **Intel Core i5 processor or higher** to ensure optimal performance. This provides sufficient processing power to run the software smoothly without lag. A higher-end processor enhances multitasking capabilities, improving efficiency. Choosing a powerful CPU ensures seamless execution of tasks and better overall system performance.

- Memory (RAM)-8 GB

The system requires a minimum of **8 GB of RAM** to ensure optimal performance. Sufficient RAM is crucial for smooth multitasking and efficient operation of the software, especially when handling complex computations, high-performance applications, or large datasets. With adequate RAM, users can experience faster data processing, seamless multitasking, and enhanced system stability, reducing the risk of slowdowns or crashes during resource-intensive tasks. For demanding workflows such as software development, graphic design, data analysis, or virtualization, **16 GB or more** is recommended to further improve responsiveness and overall efficiency.

4.Developments Tools used

4.1Django (Python Framework):



Django is a high-level Python web framework designed to encourage rapid development and clean, pragmatic design. It follows the **Model-View-Template (MVT)** architectural pattern, making it easy to separate business logic, user interface, and data handling. Django comes with built-in features such as authentication, URL routing, database management using Object-Relational Mapping (ORM), and security measures like protection against SQL injection, XSS, and CSRF attacks. It promotes the "**Don't Repeat Yourself**" (**DRY**) principle, reducing redundant code and improving maintainability. With its scalability and extensive ecosystem of third-party packages, Django is widely used for building secure, scalable, and feature-rich web applications.

4.2SQLite:



SQLite is a lightweight, serverless, and self-contained relational database management system (RDBMS). Unlike traditional databases such as MySQL or PostgreSQL, SQLite does not require a separate server process; instead, it stores data in a single file on disk, making it easy to set up and use for small to medium-scale applications. It supports **ACID (Atomicity, Consistency, Isolation, Durability)** transactions, ensuring data integrity. Due to its minimal configuration and low overhead, SQLite is commonly used for development, testing, mobile applications, and embedded systems. However, for large-scale applications with high concurrent users, a more robust database like PostgreSQL or MySQL is preferred.

4.3 HTML, CSS, JavaScript:

HTML, CSS, and JavaScript are the core technologies for building web applications:

- **HTML (HyperText Markup Language)** is the backbone of web pages, defining the structure and content using elements like headings, paragraphs, forms, and tables. It provides the foundation upon which a webpage is built.
- **CSS (Cascading Style Sheets)** is used to style and design web pages, controlling layout, colours, fonts, and responsiveness. Frameworks like **Bootstrap** help speed up development with pre-designed styles and components.
- **JavaScript** is a programming language that adds interactivity and dynamic behaviour to web pages. It enables features like form validation, animations, and real-time content updates. Libraries and frameworks like **React, Vue.js, and jQuery** enhance JavaScript's capabilities.

4.4 VS code:



Visual Studio Code (VS Code) is a lightweight yet powerful code editor developed by Microsoft. It supports multiple programming languages, including Python, JavaScript, HTML, and CSS, making it a popular choice for web development. VS Code offers **intelligent code completion (IntelliSense)**, **syntax highlighting**, **debugging tools**, **Git integration**, and **extensions** that enhance productivity. It also has built-in support for **Django, SQLite, and JavaScript frameworks**, making it ideal for developing applications like the Clinic Management System. With features like **Live Server**, **Docker integration**, and **customizable themes**, VS Code is widely used for both beginner and professional developers.

4.5 Email OTP (Two-Factor Authentication):

Email OTP (One-Time Password) Two-Factor Authentication (2FA) is a security mechanism that enhances user authentication by requiring a second layer of verification. In this method, when a user attempts to log in, an OTP (a randomly generated code) is sent to their registered email. The user must enter this code within a limited time to complete the login process.

This approach improves security by ensuring that even if a password is compromised, unauthorized access is prevented without access to the user's email. Django provides built-in support for email-based authentication, and additional libraries like **django-otp** or custom implementations using **Django's** Email Backend can be used to implement 2FA. It is commonly used in applications requiring high security, such as banking systems, healthcare platforms, and enterprise applications.

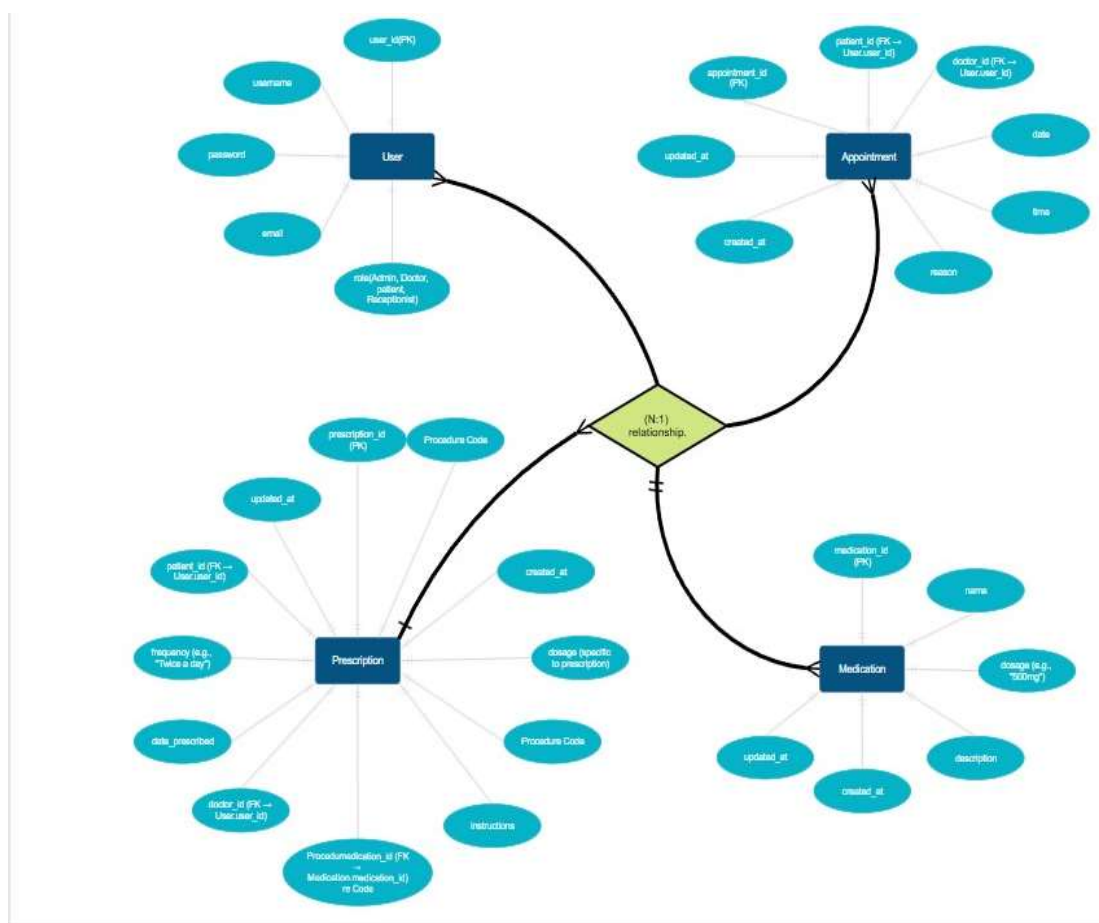
4.6 Django's built-in authentication system:

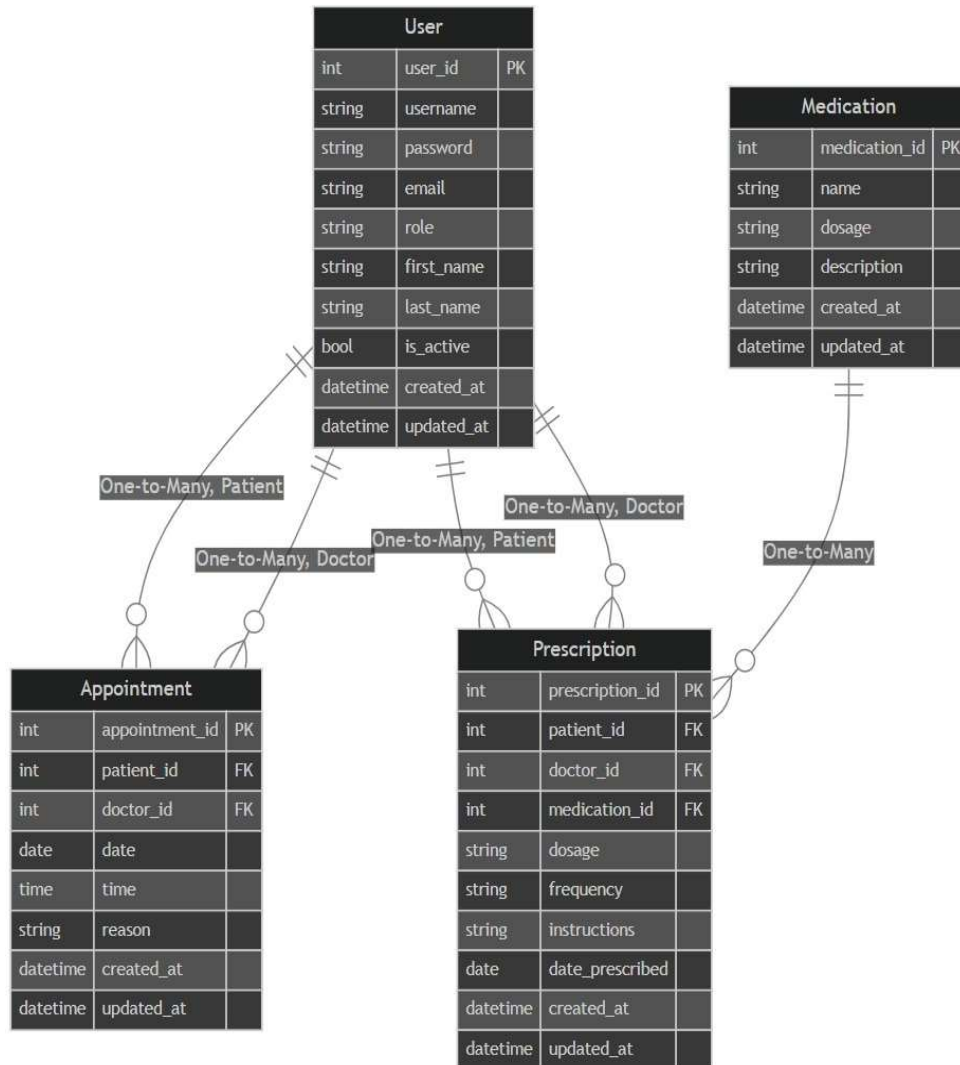
Django's built-in authentication system provides a secure and flexible way to handle user authentication, including login, logout, password management, and user permissions. It uses Django's **User model**, which securely stores passwords using hashing (PBKDF2 by default) and supports session-based or token-based authentication. The system includes features such as password reset, user registration, and Role-Based Access Control (RBAC) using groups and permissions. It also allows integration with third-party authentication methods like Google or Facebook logins and supports Two-Factor Authentication (2FA). With built-in security protections against brute-force attacks, CSRF, and session hijacking, Django's authentication system ensures a reliable and secure authentication process for web applications.

5 Design

5.1 ER Diagram

An **ER (Entity-Relationship) Diagram** visually represents a database's structure by showing entities (tables), their attributes, and relationships. Entities like **Users**, **Appointments**, and **Medications** are connected through relationships such as **patients booking appointments** or **doctors prescribing medications**. It helps in designing an efficient database by ensuring proper structure, reducing redundancy, and maintaining data integrity.





5.2 Project Architecture

The design phase is a pivotal stage for our artisan marketplace project. This phase is essential for translating the requirements gathered in the previous stages into a well-defined architecture and design that forms the basis for development.

- **Requirement Analysis:**

We thoroughly examine the gathered requirements to understand what our marketplace needs to do and who it's for.

- **System Architecture:**

This is like designing the blueprint of our marketplace's structure. We define how different parts of our platform will work together to ensure it can grow smoothly and be easy to manage.

- **Design Documentation:**

Think of this as our project's handbook. We write down all our design decisions and technical details to keep everyone on the same page throughout the project.

- **Prototyping:**

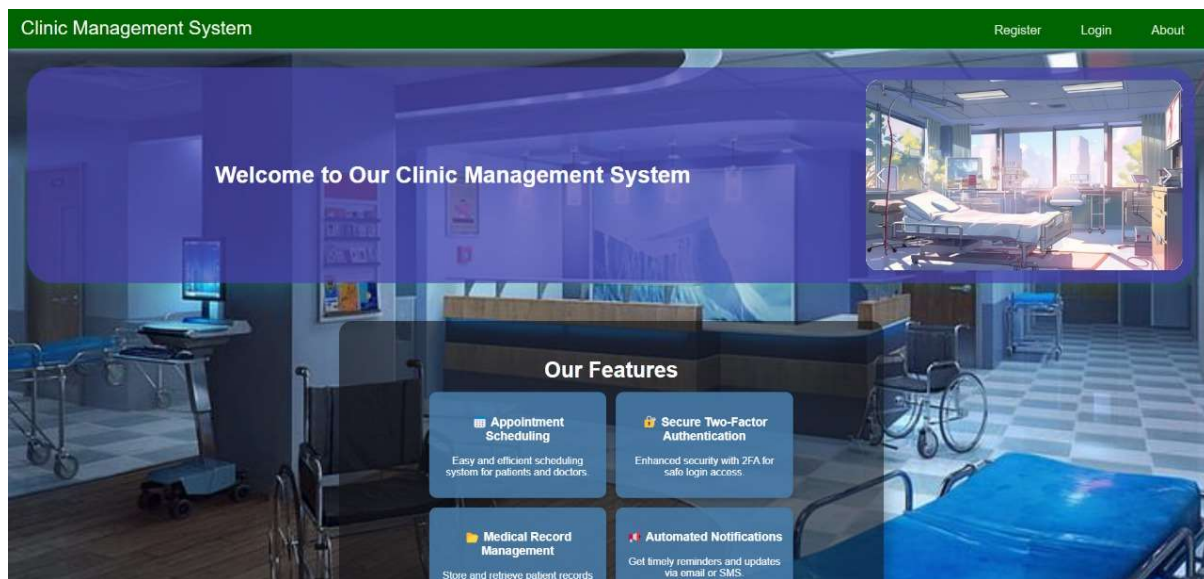
Prototypes are like rough drafts of our marketplace. We create them to see how things will look and work before building the final product, making sure we're on the right track.

- **Design Reviews:**

We gather feedback from stakeholders to make sure our designs meet their needs and are feasible to implement. This step helps us catch any issues early on and make necessary adjustments.

- **Risk Analysis:**

Here, we identify potential problems that could arise during development and come up with plans to tackle them. It's like having a backup plan for any bumps we might encounter along the way.



Clinic Management System

Register Login About

Welcome to Our Clinic Management System

Our Features

- Appointment Scheduling**
Easy and efficient scheduling system for patients and doctors.
- Secure Two-Factor Authentication**
Enhanced security with 2FA for safe login access.
- Medical Record Management**
Store and retrieve patient records.
- Automated Notifications**
Get timely reminders and updates via email or SMS.

Clinic Management Home About Login Register

Join Today

Email
Email address

Role: Patient

First Name

Last Name

Password

Password (again)

- Your password can't be too similar to your other personal information.
- Your password must contain at least 6 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Sign Up

Already Have An Account? [Log In](#)

Our Sidebar

- Latest Posts
- Announcements
- Calendars
- etc

Clinic Management Home About Login Register

Log in
to continue to Your App

Username
patient2@gmail.com

Password

Login

[Forgot password?](#)

[Create New account](#)

Our Sidebar

- Latest Posts
- Announcements
- Calendars
- etc.

Clinic Management Home About Logout

Two-Factor Authentication
Please check your email for the verification code

Enter verification code:

Verify

Our Sidebar

- Latest Posts
- Announcements
- Calendars
- etc.

Subject: Your 2FA Verification Code
From: hareesh@gmail.com
To: patient2@gmail.com
Date: Thu, 20 Mar 2025 13:04:48 -0000
Message-ID: <174247588892.6964.16090517392805342463@manju>

Your verification code is: 453742

