**Internship Report**

(Project Work)

**On**

**USER AUTHENTICATION SYSTEM**

*Submitted to*

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR, ANANTHAPURAMU**

*In Partial Fulfillment of the Requirements for the Award of the Degree of*

### BACHELOR OF TECHNOLOGY

**In**

### COMPUTER SCIENCE & ENGINEERING (CYBER SECURITY)

**Submitted By**

### PURAM NARASIMHULU - (21691A3732)

**Under the Guidance of**

**SLASH MARK INTERNSHIP**

**Department of Computer Science & Engineering (Cyber Security)**

****

### MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE (UGC – AUTONOMOUS)

**(Affiliated to JNTUA, Ananthapuramu) Accredited by NBA, Approved by AICTE, New Delhi)**

**AN ISO 9001:2008 Certified Institution**

**P. B. No: 14, Angallu, Madanapalle, Annamayya – 517325**

### MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE

(UGC-AUTONOMOUS INSTITUTION)

Affiliated to JNTUA, Ananthapuramu &Approved by AICTE, New Delhi NAAC Accredited with A+ Grade

NBA Accredited -B.Tech. (CIVIL, CSE, ECE, EEE, MECH), MBA&MCA

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (CYBER SECURITY)**

**BONAFIDE CERTIFICATE**

This is to certify that the **SUMMER INTERNSHIP-1 (20CSC701)** entitled **“USER AUTHENTICATION SYSTEM”** is a bonafide work carried out by

### PURAM NARASIMHULU - (21691A3732)

Submitted in partial fulfillment of the requirements for the award of degree **Bachelor of Technology** in the stream of **Computer Science & Engineering (Cyber Security)** in **Madanapalle Institute of Technology & Science, Madanapalle,** affiliated to **Jawaharlal Nehru Technological University Anantapur, Ananthapuramu** during the academic year 2023-2024

**Head of the Department Dr.S.V.S.Gangadevi Professor and Head Department of**

**CSE(CS)**

### DECLARATION

We hereby declare that results embodied in this **SUMMER INTERNSHIP-1(20CSC701) “USER AUTHENTICATION SYSTEM”** by us under the guidance of **SLASH MARK** in partial fulfillment of the award of **Bachelor of Technology** in **Computer Science & Engineering (Cyber Security)** from **Jawaharlal Nehru Technological University Anantapur, Ananthapuramu** and we have not submitted the same to any other University/institute for award of any other degree.

**Date : 23 – 06 – 2024 Place : MADANAPALLE**

### PROJECT MEMBER PURAM NARASIMHULU

**(21691A3732)**

I certify that above statement made by the students is correct to the best of my knowledge.

**Date:23-06-2024 Guide:**

SLASH MARK

## TABLE OF CONTENT

### S.NO TOPIC PAGE NO.

1. INTRODUCTION 01
   1. About Industry or Organization Details 02
   2. [My Personal Benefits](#_TOC_250009) 03
   3. Objective of the Project 04
   4. Limitations of Project 05
2. SYSTEM ANALYSIS 08
   1. [Introduction](#_TOC_250008) 09
   2. [Existing System](#_TOC_250007) 09
   3. [Advantages over Existing System 1](#_TOC_250006)2
3. SYSTEM SPECIFICATION 16
   1. Hardware Requirement Specification 17
   2. Software Requirement Specification 19
4. SYSTEM DESIGN 24
   1. [System Architecture](#_TOC_250005) 28
   2. [Modules Flow diagrams](#_TOC_250004) 29
   3. Use case diagram 30
5. IMPLEMENTATION AND RESULTS 31
   1. Introduction 32
   2. Implementation of key functions 33
   3. Method of Implementation(CODING) 34
   4. Testing and validation 59
6. CONCLUSION 62
   1. [Conclusion 62](#_TOC_250000)
7. REFERENCES 67

**ABSTRACT**

Title: "USER AUTHENTICATION USING BRAIN WAVES"

Abstract: User Authentication System

In the digital age, ensuring secure access to online services is paramount. User authentication systems play a critical role in verifying the identity of users before granting access to sensitive information or functionalities. This abstract discusses the key components and functionalities of a robust user authentication system.

The system typically involves multiple layers of security measures, such as passwords, biometric data, two-factor authentication (2FA), and encryption protocols. These measures collectively ensure that only authorized individuals can access protected resources.

Key challenges in designing such systems include balancing security with usability, minimizing vulnerabilities to cyber threats, and accommodating diverse user needs across various platforms and devices. Advances in technology, such as machine learning for behavior analysis and blockchain for decentralized authentication, continue to shape the evolution of user authentication systems.

This abstract explores current trends, challenges, and future directions in user authentication, highlighting the importance of adaptive and resilient security measures in safeguarding digital identities.

---

Feel free to adjust the specifics based on the exact focus and technologies used in your user authentication system.

# INTRODUCTION

### ABOUT INDUSTRY / ORGANIZATION DETAILS:

**SLASH MARK INTERNSHIP**

It seems like you're asking about the details related to "Slash," which could refer to various entities. Here are a few possibilities:

1. Slash (Musician):

Full Name: Saul Hudson

Born: July 23, 1965

Known For: Guitarist of Guns N' Roses and Velvet Revolver

2. Slash (Programming Language):

Type: A domain-specific language (DSL) for describing character sets and lexical tokens.

Usage: Often used in compiler construction and parsing tasks.

3. Slash (Organization/Company):

Details: If this refers to a specific organization or company named "Slash," please provide more context or details about the industry or sector they operate in, as there are several entities that could potentially be referred to by this name.

If you could clarify which specific "Slash" you're interested in, I can provide more detailed information accordingly!

#### My Personal Benefits

Implementing a user authentication system offers numerous personal benefits, whether you're developing it for a personal project, enhancing your skillset, or adding security to your applications. Here are some key advantages:

Enhanced Security:

Data Protection: Ensures that personal and sensitive information is accessible only to authorized users, protecting against data breaches.

Access Control: Enables fine-grained control over who can access what resources, enhancing overall security.

Multi-Factor Authentication (MFA): Adds an extra layer of security, making it harder for unauthorized users to gain access.

2. User Management

User Tracking: Helps in tracking user activity, which can be useful for analyzing behavior and improving the user experience.

Role-Based Access: Allows assigning different roles to users, providing access to specific parts of an application based on their role.

Password Management: Facilitates secure password storage and recovery mechanisms.

3. Improved User Experience:

Single Sign-On (SSO): Simplifies the login process for users by allowing them to use one set of credentials to access multiple applications.

Personalization: Enables personalized experiences by identifying users and tailoring content based on their preferences and past behavior.

4. Compliance:

Regulatory Compliance: Helps meet legal and regulatory requirements for data protection and user privacy (e.g., GDPR, HIPAA).

5.Professional Development:

Skill Enhancement: Developing an authentication system enhances your programming skills, particularly in areas such as security, cryptography, and user interface design.

Marketability: Knowledge and experience with authentication systems make you more valuable in the job market, as security is a critical aspect of most applications.

6.Trust Building:

User Trust: Increases user trust in your application by demonstrating a commitment to protecting their data.

Reputation: Builds a positive reputation for your application as being secure and reliable.

7. Operational Benefits:

Automation: Automates the process of user authentication and authorization, reducing the manual effort needed to manage users.

### Practical Examples of Personal Benefits

1. Developing a Portfolio: Creating a secure authentication system can be an impressive addition to your portfolio, showcasing your technical abilities.

2. Protecting Personal Projects: If you have personal projects, such as a blog or an online service, implementing an authentication system ensures that only you or designated users can access sensitive sections.

3.Learning and Experimentation: Provides a platform to experiment with and learn about various authentication methods (e.g., OAuth, JWT, biometrics).

By implementing a user authentication system, you not only safeguard your applications and data but also gain a competitive edge in the tech industry through enhanced knowledge and skills.

#### 1.3 Objective:

User authentication systems on objects involve managing access to specific resources or objects within an application based on the user's identity and permissions. This approach ensures that only authorized users can access or modify certain objects, enhancing security and providing fine-grained access control. Here are the key components and benefits of implementing a user authentication system on objects:

### 1.4 LIMITATION OF PROJECT:

Implementing a user authentication system in a project, while essential for security, comes with several limitations and challenges. Being aware of these can help in planning and mitigating potential issues. Here are some key limitations:

**### 1. Complexity and Maintenance**

Implementation Complexity: Developing a robust authentication system requires significant effort and expertise in security protocols, cryptography, and secure coding practices.

Maintenance: Keeping the authentication system up to date with the latest security patches and best practices can be resource-intensive. This includes regular updates to cryptographic algorithms, security libraries, and handling new security threats.

**### 2. Usability vs. Security Trade-Off**

User Experience: Implementing strong security measures, such as multi-factor authentication (MFA), can sometimes complicate the user experience, leading to frustration and potential user drop-off.

Password Management: Users often struggle with managing complex passwords, which can lead to weak passwords or the reuse of passwords across multiple platforms, increasing vulnerability.

**### 3. Performance**

Latency: Authentication processes, especially those involving MFA or external authentication services, can introduce latency, affecting the overall performance and user experience.

Scalability: As the number of users grows, the authentication system must scale accordingly. This can require significant infrastructure and optimization to handle large volumes of authentication requests without degrading performance.

**### 4. Security Risks**

Vulnerabilities: Even with best practices, authentication systems can have vulnerabilities such as SQL injection, cross-site scripting (XSS), or vulnerabilities in third-party libraries.

Brute Force Attacks: Protecting against brute force attacks requires implementing rate limiting, CAPTCHAs, and other protective measures, which add to the complexity.

Phishing and Social Engineering: No matter how secure the system, users can still fall prey to phishing attacks or social engineering, compromising their credentials.

**### 5. Integration Challenges**

Legacy Systems: Integrating modern authentication methods with legacy systems can be challenging and may require significant refactoring or bridging solutions.

Third-Party Services: Reliance on third-party authentication services (e.g., OAuth providers) can introduce dependencies and potential points of failure.

**### 6. Legal and Compliance Issues**

Data Protection Laws: Compliance with data protection regulations such as GDPR, CCPA, or HIPAA requires careful handling of user data, secure storage of credentials, and transparent privacy policies.

Audit Requirements: Meeting audit requirements for security and compliance can be burdensome and require detailed logging and reporting mechanisms.

**### 7. Cost**

Development and Maintenance: Developing and maintaining an authentication system can be costly in terms of both time and resources.

Infrastructure: Scaling the authentication infrastructure to handle high loads can incur significant costs, especially if high availability and redundancy are required.

**### 8. User Management**

Account Recovery: Implementing secure and user-friendly account recovery mechanisms is complex and essential to prevent unauthorized access.

User Support: Providing support for users facing authentication issues (e.g., forgotten passwords, locked accounts) requires additional resources.

**### Mitigation Strategies**

1. Use Established Frameworks: Utilize well-established authentication frameworks and libraries to avoid common pitfalls and reduce implementation complexity.

2. Regular Audits and Penetration Testing: Conduct regular security audits and penetration testing to identify and fix vulnerabilities.

3. User Education: Educate users on best practices for password management and recognizing phishing attempts.

4. Scalable Architecture: Design the authentication system with scalability in mind, using cloud services and load balancing to handle increased loads.

5. Compliance Automation: Use tools and services that help automate compliance with legal and regulatory requirements.

By understanding these limitations and implementing strategies to mitigate them, you can build a more secure and reliable user authentication system.

# SYSTEM ANALYSIS

## Introduction

System analysis in user authentication involves a comprehensive examination and evaluation of the mechanisms and processes used to verify the identity of users within a system. This analysis is crucial for developing robust and secure authentication systems that safeguard sensitive information and ensure only authorized users have access to specific resources.

#### Importance of User Authentication

User authentication is a critical component of information security. It protects systems and data from unauthorized access, ensuring that only legitimate users can perform actions within a system. Effective authentication systems prevent security breaches, protect user privacy, and maintain the integrity and confidentiality of data.

#### Existing System:

User authentication systems are integral to modern digital security, and there are several well-established methods and frameworks employed across various platforms and applications. These systems range from simple password-based methods to advanced multi-factor authentication (MFA) and biometric solutions. Below are some of the most commonly used user authentication systems:

**#### 1. Password-Based Authentication**

Description: Users authenticate by entering a unique password associated with their username or email.

Features:

Hashing: Passwords are stored as hashes to enhance security.

Salting: Additional random data (salt) is added to passwords before hashing to prevent dictionary attacks.

Examples:

- Web applications, email services, and most online accounts use password-based authentication.

Limitations:

- Vulnerable to brute force attacks, phishing, and poor password practices by users.

**#### 2. Multi-Factor Authentication (MFA)**

Description: Combines two or more independent credentials: what the user knows (password), what the user has (security token), and what the user is (biometric verification).

Features:

Two-Factor Authentication (2FA): The most common form of MFA, typically combining a password with a one-time code sent via SMS or generated by an app.

- \*\*Biometric Factors\*\*: Fingerprints, facial recognition, or retinal scans.

\*\*Examples\*\*:

- Google Authenticator, Microsoft Authenticator, Authy.

- Hardware tokens like YubiKey.

\*\*Limitations\*\*:

- Can be more complex and expensive to implement.

- Potential user resistance due to increased steps in the authentication process.

#### 3. \*\*Single Sign-On (SSO)\*\*

\*\*Description\*\*: Allows users to authenticate once and gain access to multiple applications or systems without re-authenticating.

\*\*Features\*\*:

- \*\*Centralized Authentication\*\*: Users authenticate through a central identity provider (IdP).

- \*\*Token-Based\*\*: Uses tokens like SAML (Security Assertion Markup Language) or OAuth to manage authentication across systems.

\*\*Examples\*\*:

- Google Workspace, Microsoft Azure Active Directory, Okta.

\*\*Limitations\*\*:

- Single point of failure: if the SSO provider is compromised, all connected services are at risk.

#### 4. \*\*OAuth and OpenID Connect\*\*

\*\*Description\*\*: OAuth is an authorization framework that allows third-party services to exchange information without exposing user credentials. OpenID Connect is an authentication layer on top of OAuth 2.0.

\*\*Features\*\*:

- \*\*Token-Based\*\*: Uses access tokens and refresh tokens.

- \*\*Scopes\*\*: Defines the level of access granted to the third-party application.

\*\*Examples\*\*:

- Logging into third-party services using Google, Facebook, or LinkedIn accounts.

\*\*Limitations\*\*:

- Complexity in implementation and configuration.

- Dependency on the third-party provider’s uptime and security.

#### 5. \*\*Biometric Authentication\*\*

\*\*Description\*\*: Uses unique biological characteristics (fingerprints, facial recognition, retinal scans) for authentication.

\*\*Features\*\*:

- \*\*High Security\*\*: Difficult to replicate or forge.

- \*\*Convenience\*\*: Users do not need to remember passwords or carry tokens.

\*\*Examples\*\*:

- Apple Face ID, Touch ID, Windows Hello.

\*\*Limitations\*\*:

- Privacy concerns and potential misuse of biometric data.

- False positives or negatives can occur, affecting reliability.

#### 6. \*\*Certificate-Based Authentication\*\*

\*\*Description\*\*: Uses digital certificates issued by a trusted certificate authority (CA) to authenticate users.

\*\*Features\*\*:

- \*\*Public Key Infrastructure (PKI)\*\*: Uses a pair of cryptographic keys (public and private).

- \*\*High Security\*\*: Strong encryption makes it difficult to intercept or forge credentials.

\*\*Examples\*\*:

- SSL/TLS certificates for secure web browsing.

- Client certificates for authenticating users in enterprise environments.

\*\*Limitations\*\*:

- Complexity in managing certificates.

- Cost associated with acquiring and maintaining certificates.

Each of these systems has its own strengths and weaknesses, and the choice of an authentication system often depends on the specific needs and constraints of the application or organization. The trend in modern security is to use a combination of these methods to create a layered defense that balances security and usability.

#### Advantages over Existing System:

Implementing a robust user authentication system offers numerous advantages that enhance security, improve user experience, and streamline operations. Here are some key advantages:

### 1. \*\*Enhanced Security\*\*

- \*\*Data Protection\*\*: Authentication systems ensure that only authorized users can access sensitive information, protecting against unauthorized access and data breaches.

- \*\*Prevention of Unauthorized Access\*\*: Reduces the risk of malicious attacks such as phishing, brute force attacks, and credential stuffing by verifying user identities before granting access.

- \*\*Multi-Factor Authentication (MFA)\*\*: Adds an extra layer of security by requiring multiple forms of verification, making it more difficult for attackers to gain access.

### 2. \*\*Improved User Experience\*\*

- \*\*Single Sign-On (SSO)\*\*: Simplifies the login process by allowing users to access multiple applications with one set of credentials, reducing the need to remember multiple passwords.

- \*\*Personalized Experience\*\*: Enables personalized user experiences by securely identifying users and tailoring content and services to their preferences and history.

- \*\*Convenience\*\*: Advanced authentication methods such as biometrics (e.g., facial recognition, fingerprint scanning) provide quick and seamless access, enhancing user convenience.

### 3. \*\*Operational Efficiency\*\*

- \*\*Automated Access Control\*\*: Streamlines the management of user access rights and permissions, reducing the administrative burden on IT staff.

- \*\*Reduced Help Desk Load\*\*: Minimizes the number of password-related help desk calls by implementing features like self-service password recovery and SSO.

- \*\*Scalability\*\*: Authentication systems can scale with the growth of an organization, accommodating an increasing number of users and access points without compromising security.

### 4. \*\*Compliance and Auditability\*\*

- \*\*Regulatory Compliance\*\*: Helps organizations comply with data protection regulations such as GDPR, HIPAA, and CCPA by ensuring secure access to sensitive data.

- \*\*Audit Trails\*\*: Provides detailed logs of user access and activities, which are essential for auditing and forensic analysis. This helps in identifying and responding to security incidents and ensuring accountability.

### 5. \*\*Risk Management\*\*

- \*\*Risk-Based Authentication\*\*: Adjusts the authentication requirements based on the risk profile of the access attempt, such as requiring additional verification for high-risk activities or unusual login locations.

- \*\*Behavioral Analytics\*\*: Monitors user behavior to detect anomalies that could indicate compromised accounts, allowing for proactive security measures.

### 6. \*\*Trust Building\*\*

- \*\*User Trust\*\*: Increases user confidence in the security of the application or service, as they see measures in place to protect their data and identity.

- \*\*Reputation Management\*\*: Helps maintain and enhance the reputation of the organization by demonstrating a commitment to security and protecting user information.

### 7. \*\*Flexibility and Integration\*\*

- \*\*Integration with Modern Technologies\*\*: Modern authentication systems often support integration with a wide range of technologies and services, including cloud applications, mobile devices, and third-party services.

- \*\*Adaptability\*\*: Easily adapts to new security requirements and threats, ensuring long-term protection and compliance.

### 8. \*\*Cost Savings\*\*

- \*\*Reduction in Fraud\*\*: By preventing unauthorized access, authentication systems help reduce the financial losses associated with fraud and cyberattacks.

- \*\*Efficiency Gains\*\*: Automated and streamlined authentication processes reduce the need for extensive IT support, lowering operational costs.

### Practical Examples of Advantages

1. \*\*E-commerce Platforms\*\*:

- Enhanced security through MFA and SSO helps protect user accounts and financial transactions, building trust and reducing fraud.

- Personalized experiences based on secure user identification increase customer satisfaction and loyalty.

2. \*\*Corporate Environments\*\*:

- SSO and automated access control improve productivity by reducing login times and administrative overhead.

- Compliance with industry regulations through secure authentication practices protects sensitive corporate data and ensures legal compliance.

3. \*\*Healthcare Systems\*\*:

- Strong authentication methods protect patient data, ensuring confidentiality and compliance with HIPAA.

- Secure access to medical records for authorized personnel only enhances patient trust and operational efficiency.

By leveraging the advantages of a robust user authentication system, organizations can not only protect their assets and data but also provide a better user experience, achieve regulatory compliance, and improve overall operational efficiency.

# System Specifications

#### Hardware requirements:

Implementing a user authentication system involves various hardware requirements to ensure the system is secure, reliable, and scalable. These requirements vary depending on the size of the organization, the complexity of the authentication methods, and the level of security needed. Here are the typical hardware requirements for a robust user authentication system:

### 1. \*\*Servers\*\*

\*\*Authentication Server(s)\*\*:

- \*\*Purpose\*\*: Handles the authentication requests and manages user credentials.

- \*\*Specifications\*\*:

- \*\*CPU\*\*: Multi-core processors (e.g., Intel Xeon or AMD EPYC).

- \*\*RAM\*\*: 16GB or more, depending on the load.

- \*\*Storage\*\*: SSDs for faster read/write operations, with sufficient capacity for storing user credentials and logs.

- \*\*Network\*\*: High-speed network interfaces (1Gbps or higher).

\*\*Database Server(s)\*\*:

- \*\*Purpose\*\*: Stores user information, authentication logs, and configuration data.

- \*\*Specifications\*\*:

- \*\*CPU\*\*: Multi-core processors with high clock speeds.

- \*\*RAM\*\*: 32GB or more, especially for large databases.

- \*\*Storage\*\*: High-capacity SSDs or HDDs with RAID configuration for redundancy.

- \*\*Network\*\*: High-speed network interfaces.

\*\*Redundancy and Load Balancing\*\*:

- \*\*Purpose\*\*: Ensures high availability and distributes the load across multiple servers.

- \*\*Specifications\*\*: Multiple servers configured with load balancers (hardware or software) to handle failover and load distribution.

### 2. \*\*Networking Equipment\*\*

\*\*Firewalls\*\*:

- \*\*Purpose\*\*: Protects the authentication system from external threats.

- \*\*Specifications\*\*: Enterprise-grade firewalls with advanced security features such as intrusion detection/prevention systems (IDS/IPS).

\*\*Switches and Routers\*\*:

- \*\*Purpose\*\*: Manages internal network traffic and connects different components of the authentication system.

- \*\*Specifications\*\*: High-speed, managed switches and routers with support for VLANs and QoS.

\*\*VPN Gateways\*\*:

- \*\*Purpose\*\*: Provides secure remote access for users.

- \*\*Specifications\*\*: VPN appliances or servers with support for secure protocols (e.g., IPSec, SSL/TLS).

### 3. \*\*Security Appliances\*\*

\*\*Hardware Security Modules (HSMs)\*\*:

- \*\*Purpose\*\*: Provides secure key management and cryptographic operations.

- \*\*Specifications\*\*: HSMs with FIPS 140-2 Level 3 or higher certification for critical environments.

### 4. \*\*Biometric Devices\*\*

\*\*Fingerprint Scanners\*\*:

- \*\*Purpose\*\*: Captures and verifies fingerprint data.

- \*\*Specifications\*\*: High-resolution optical or capacitive scanners.

\*\*Facial Recognition Cameras\*\*:

- \*\*Purpose\*\*: Captures and processes facial images for authentication.

- \*\*Specifications\*\*: High-definition cameras with infrared capability for low-light environments.

\*\*Other Biometric Readers\*\*:

- \*\*Purpose\*\*: Captures other biometric data such as iris or voice.

- \*\*Specifications\*\*: Devices with high accuracy and reliability.

### 5. \*\*Client Devices\*\*

\*\*Workstations and Mobile Devices\*\*:

- \*\*Purpose\*\*: End-user devices for accessing the authentication system.

- \*\*Specifications\*\*: Desktops, laptops, and mobile devices with current operating systems and security patches.

### 6. \*\*Backup and Recovery\*\*

\*\*Backup Servers\*\*:

- \*\*Purpose\*\*: Stores backups of critical data and system configurations.

- \*\*Specifications\*\*: Servers with large storage capacity and high-speed network interfaces.

\*\*Tape Drives or NAS/SAN Systems\*\*:

- \*\*Purpose\*\*: Provides additional backup storage and redundancy.

- \*\*Specifications\*\*: High-capacity, reliable storage solutions with automated backup software.

### 7. \*\*Environmental Controls\*\*

\*\*Uninterruptible Power Supplies (UPS)\*\*:

- \*\*Purpose\*\*: Ensures continuous power supply to critical components during outages.

- \*\*Specifications\*\*: UPS units with sufficient capacity to support servers, networking equipment, and security appliances.

\*\*Cooling Systems\*\*:

- \*\*Purpose\*\*: Maintains optimal operating temperatures for hardware.

- \*\*Specifications\*\*: Precision cooling systems for data centers or server rooms.

### 8. \*\*Monitoring and Management\*\*

\*\*Monitoring Tools\*\*:

- \*\*Purpose\*\*: Monitors system performance, security, and availability.

- \*\*Specifications\*\*: Hardware or software solutions for real-time monitoring and alerting.

\*\*Management Consoles\*\*:

- \*\*Purpose\*\*: Provides centralized management of authentication components.

- \*\*Specifications\*\*: Workstations or servers with management software installed.

### Example Hardware Setup for a Medium-Sized Organization

1. Authentication Servers:

- 2x Dell PowerEdge R740 (or equivalent) with Intel Xeon Silver processors, 32GB RAM, 1TB SSD, and dual 1Gbps NICs.

2. \*\*Database Servers\*\*:

- 2x HP ProLiant DL380 (or equivalent) with Intel Xeon Gold processors, 64GB RAM, 2TB SSD RAID, and dual 1Gbps NICs.

3. \*\*Load Balancer\*\*:

- 1x F5 BIG-IP or equivalent for distributing traffic and ensuring high availability.

4. \*\*Networking Equipment\*\*:

- Cisco Catalyst 9300 Series switches.

- Cisco ASA 5500-X Series firewalls.

5. \*\*Biometric Devices\*\*:

- 50x Digital Persona 4500 fingerprint scanners for desktop authentication.

- 10x Logitech BRIO 4K webcams for facial recognition.

6. \*\*Backup and Recovery\*\*:

- 1x Dell EMC Data Domain backup server.

- 1x Quantum Scalar tape library for offsite backups.

By carefully planning and investing in the appropriate hardware, organizations can ensure their user authentication systems are secure, efficient, and capable of meeting current and future demands.

#### Sowftware Requirements:

Implementing a user authentication system requires specific software components and configurations to ensure it is secure, reliable, and scalable. Here are the essential software requirements for a robust user authentication system:

### 1. \*\*Operating Systems\*\*

\*\*Server OS\*\*:

- \*\*Windows Server\*\*: Suitable for environments using Active Directory.

- \*\*Linux Distributions\*\*: Common choices include Ubuntu Server, CentOS, Red Hat Enterprise Linux (RHEL), and Debian.

\*\*Client OS\*\*:

- \*\*Windows\*\*: Windows 10/11 for desktops and laptops.

- \*\*macOS\*\*: macOS for Apple devices.

- \*\*Linux\*\*: Various distributions for developer environments.

- \*\*Mobile OS\*\*: iOS and Android for mobile authentication.

### 2. \*\*Database Management Systems (DBMS)\*\*

- \*\*Relational Databases\*\*: MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database.

- \*\*NoSQL Databases\*\*: MongoDB, Cassandra for more scalable and flexible data storage.

### 3. \*\*Authentication Software and Libraries\*\*

\*\*Authentication Frameworks and Protocols\*\*:

- \*\*OAuth 2.0\*\*: For secure authorization.

- \*\*OpenID Connect\*\*: For authentication built on OAuth 2.0.

- \*\*SAML (Security Assertion Markup Language)\*\*: For Single Sign-On (SSO) between identity providers and service providers.

- \*\*LDAP (Lightweight Directory Access Protocol)\*\*: For directory services.

- \*\*Kerberos\*\*: For network authentication.

\*\*Libraries and SDKs\*\*:

- \*\*Auth0\*\*: Provides authentication and authorization as a service.

- \*\*Okta\*\*: Identity management and SSO solutions.

- \*\*Firebase Authentication\*\*: For mobile and web applications.

- \*\*Passport.js\*\*: Middleware for Node.js providing various authentication strategies.

- \*\*Spring Security\*\*: For Java-based applications.

### 4. \*\*Web Servers and Application Servers\*\*

- \*\*Web Servers\*\*: Apache HTTP Server, Nginx, Microsoft IIS.

- \*\*Application Servers\*\*: Apache Tomcat, JBoss, WebLogic, WebSphere.

### 5. \*\*Encryption and Security Software\*\*

\*\*Encryption Libraries\*\*:

- \*\*OpenSSL\*\*: Toolkit for SSL/TLS.

- \*\*Bcrypt, Argon2\*\*: For password hashing.

- \*\*AES (Advanced Encryption Standard)\*\*: For data encryption.

\*\*Security Tools\*\*:

- \*\*Fail2Ban\*\*: Intrusion prevention software that protects against brute-force attacks.

- \*\*CAPTCHA Services\*\*: Google reCAPTCHA to prevent automated abuse.

### 6. \*\*Monitoring and Logging Tools\*\*

- \*\*Monitoring Software\*\*: Nagios, Zabbix, Prometheus for monitoring system health and performance.

- \*\*Logging Tools\*\*: ELK Stack (Elasticsearch, Logstash, Kibana), Splunk for log management and analysis.

- \*\*Audit Tools\*\*: OSSEC for host-based intrusion detection and auditing.

### 7. \*\*Backup and Recovery Software\*\*

- \*\*Backup Solutions\*\*: Veeam, Acronis, Bacula for regular backups and disaster recovery.

- \*\*Database Backup Tools\*\*: Native tools like `mysqldump` for MySQL, `pg\_dump` for PostgreSQL.

### 8. \*\*Development Tools\*\*

- \*\*IDEs\*\*: Visual Studio Code, IntelliJ IDEA, PyCharm for writing and maintaining authentication-related code.

- \*\*Version Control\*\*: Git, GitHub, GitLab for source code management and collaboration.

- \*\*CI/CD Pipelines\*\*: Jenkins, Travis CI, CircleCI for continuous integration and deployment.

### 9. \*\*Integration Tools\*\*

- \*\*API Gateways\*\*: Kong, Apigee for managing and securing APIs.

- \*\*Identity Providers\*\*: Microsoft Azure Active Directory, Google Identity Platform for SSO and user management.

### 10. \*\*Client-Side Software\*\*

\*\*Browsers\*\*:

- \*\*Web Authentication API\*\*: Support for modern authentication methods like WebAuthn in browsers like Chrome, Firefox, Safari, and Edge.

\*\*Mobile SDKs\*\*:

- \*\*iOS\*\*: Swift and Objective-C libraries for integrating authentication.

- \*\*Android\*\*: Java and Kotlin libraries for integrating authentication.

### Example Software Stack for a Medium-Sized Organization

1. \*\*Server OS\*\*: Ubuntu Server 20.04 LTS

2. \*\*Database\*\*: PostgreSQL for storing user data

3. \*\*Authentication Framework\*\*:

- OAuth 2.0 and OpenID Connect implemented using Auth0

- SSO using SAML with Okta

4. \*\*Web Server\*\*: Nginx as a reverse proxy and load balancer

5. \*\*Application Server\*\*: Apache Tomcat for hosting Java-based applications

6. \*\*Encryption\*\*:

- OpenSSL for SSL/TLS

- Argon2 for password hashing

7. \*\*Monitoring\*\*: Prometheus for metrics and Grafana for visualization

8. \*\*Logging\*\*: ELK Stack (Elasticsearch, Logstash, Kibana)

9. \*\*Backup\*\*: Veeam for server and database backups

10. \*\*Development Tools\*\*:

- Visual Studio Code for code development

- GitHub for version control

- Jenkins for CI/CD

11. \*\*Client-Side\*\*:

- Web Authentication API in modern browsers

- Mobile SDKs for iOS and Android apps

By selecting the appropriate software components and ensuring they are correctly configured and maintained, organizations can build a secure and efficient user authentication system that meets their security requirements and operational needs.

# SYSTEM DESIGN

#### System Architecture:

**Use Case Diagram:**

Creating a use case diagram for a user authentication system helps illustrate the interactions between users (actors) and the system, identifying key functionalities and their relationships. Here is a detailed use case diagram and its components for a user authentication system.

### Use Case Diagram Components

#### Actors

1. \*\*User\*\*: An individual who wants to access the system.

2. \*\*Admin\*\*: An individual who manages the user authentication system.

3. \*\*Identity Provider (IdP)\*\*: External service providing identity verification (e.g., Auth0, Okta).

4. \*\*System\*\*: The user authentication system itself.

#### Use Cases

1. \*\*Register\*\*: Users can register for a new account.

2. \*\*Login\*\*: Users can log in to the system.

3. \*\*Logout\*\*: Users can log out of the system.

4. \*\*Reset Password\*\*: Users can reset their password if forgotten.

5. \*\*Change Password\*\*: Users can change their current password.

6. \*\*Two-Factor Authentication (2FA) Setup\*\*: Users can set up two-factor authentication.

7. \*\*Two-Factor Authentication (2FA) Verification\*\*: Users verify their identity using 2FA.

8. \*\*Manage Users\*\*: Admins can manage user accounts (create, update, delete).

9. \*\*Monitor System\*\*: Admins can monitor the authentication system's performance and logs.

10. \*\*Authenticate User\*\*: Identity Provider verifies user credentials.

11. \*\*Issue Token\*\*: The system issues authentication tokens upon successful login.

### Use Case Diagram

Below is a visual representation of the use case diagram for a user authentication system:

### Descriptions of Use Cases

1. \*\*Register\*\*:

- \*\*Description\*\*: Allows a user to create a new account by providing necessary information (e.g., username, password, email).

- \*\*Actor\*\*: User

- \*\*Steps\*\*:

1. User enters registration details.

2. System validates and stores user details.

3. User account is created and confirmed.

2. \*\*Login\*\*:

- \*\*Description\*\*: Allows a user to log in to the system using credentials.

- \*\*Actor\*\*: User

- \*\*Steps\*\*:

1. User enters username and password.

2. System authenticates the credentials.

3. If enabled, prompts for 2FA.

4. Upon successful authentication, issues a session token.

3. \*\*Logout\*\*:

- \*\*Description\*\*: Allows a user to log out from the system.

- \*\*Actor\*\*: User

- \*\*Steps\*\*:

1. User requests to log out.

2. System invalidates the session token.

4. \*\*Reset Password\*\*:

- \*\*Description\*\*: Allows a user to reset their password if forgotten.

- \*\*Actor\*\*: User

- \*\*Steps\*\*:

1. User requests a password reset.

2. System sends a reset link to the registered email.

3. User clicks the link and enters a new password.

4. System updates the password.

5. \*\*Change Password\*\*:

- \*\*Description\*\*: Allows a user to change their current password.

- \*\*Actor\*\*: User

- \*\*Steps\*\*:

1. User enters the current password and the new password.

2. System validates and updates the password.

6. \*\*Two-Factor Authentication (2FA) Setup\*\*:

- \*\*Description\*\*: Allows a user to set up 2FA for additional security.

- \*\*Actor\*\*: User

- \*\*Steps\*\*:

1. User selects 2FA setup.

2. System presents available 2FA methods (e.g., SMS, Authenticator app).

3. User completes the setup.

7. \*\*Two-Factor Authentication (2FA) Verification\*\*:

- \*\*Description\*\*: Users verify their identity using 2FA during login.

- \*\*Actor\*\*: User

- \*\*Steps\*\*:

1. After entering username and password, user is prompted for 2FA code.

2. User enters the code.

3. System verifies the code and grants access.

8. \*\*Manage Users\*\*:

- \*\*Description\*\*: Allows an admin to manage user accounts (create, update, delete).

- \*\*Actor\*\*: Admin

- \*\*Steps\*\*:

1. Admin logs in to the admin panel.

2. Admin performs user management tasks.

9. \*\*Monitor System\*\*:

- \*\*Description\*\*: Allows an admin to monitor the authentication system’s performance and logs.

- \*\*Actor\*\*: Admin

- \*\*Steps\*\*:

1. Admin accesses monitoring tools.

2. Admin reviews performance metrics and logs.

10. \*\*Authenticate User\*\*:

- \*\*Description\*\*: Verifies user credentials against the identity provider.

- \*\*Actor\*\*: Identity Provider

- \*\*Steps\*\*:

1. System forwards user credentials to IdP.

2. IdP validates credentials and responds with the result.

11. \*\*Issue Token\*\*:

- \*\*Description\*\*: Issues authentication tokens upon successful login.

- \*\*Actor\*\*: System

- \*\*Steps\*\*:

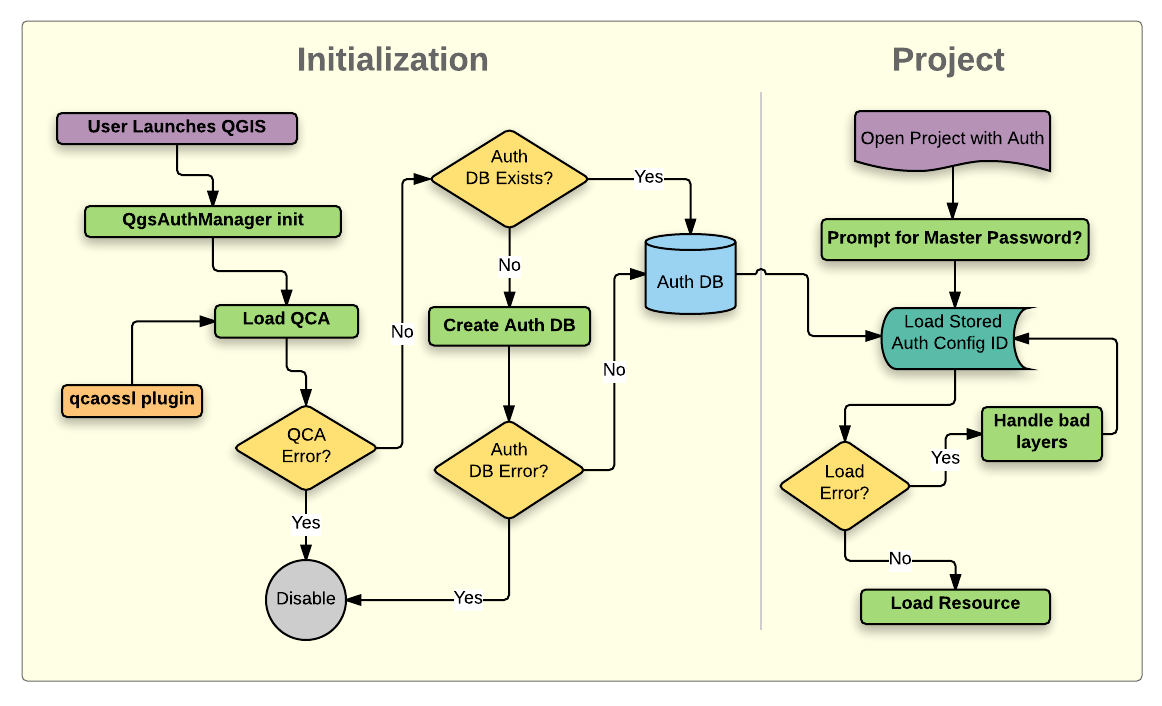
1. After successful authentication, system generates access and refresh tokens.

2. Tokens are sent to the client.

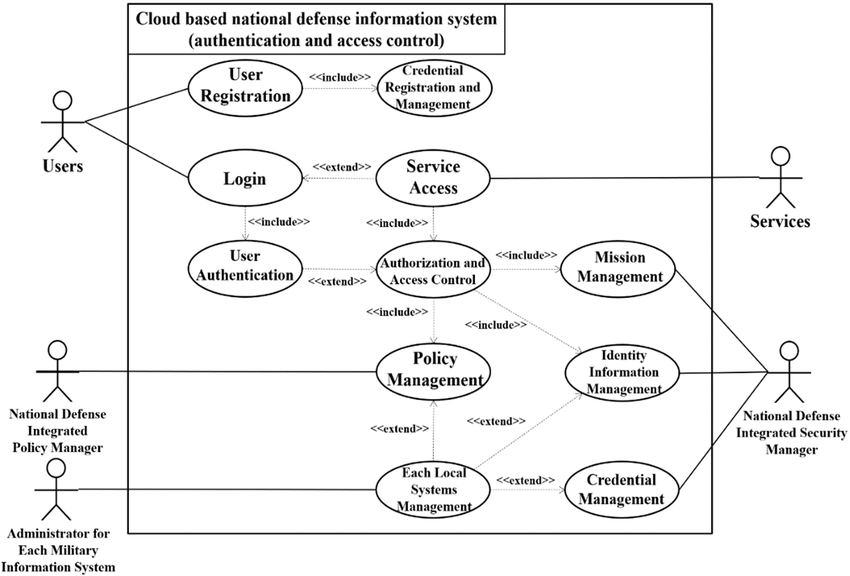
### Summary

This use case diagram provides a comprehensive view of the functionalities involved in a user authentication system and the interactions between users, admins, the system, and external identity providers. Each use case addresses a specific aspect of user authentication, ensuring a secure and efficient process for managing user identities and access.

#### Modules Flow Diagrams



**4.3.Use Case Diagram between USER and SYSTEM**

****

# IMPLEMENTATION

AND RESULTS

### INTRODUCTION FOR IMPLEMENTATION

Implementing a user authentication system involves several steps, from planning and design to coding and deployment. Below is a comprehensive guide to implementing a user authentication system, including best practices and technologies.

### Steps to Implement a User Authentication System

#### 1. **Requirements Gathering and Planning**

Objective: Identify the requirements and plan the architecture of the authentication system.

* Define user roles and permissions.
* Identify security requirements (e.g., password policies, multi-factor authentication).
* Determine integration needs with existing systems (e.g., LDAP, OAuth).
* Plan the architecture (refer to previous architecture section).

#### 2. **Design the System**

Objective: Create a detailed design of the authentication system.

* Create use case diagrams and flowcharts.
* Design the database schema for storing user information.
* Choose the authentication protocols (e.g., OAuth 2.0, OpenID Connect).
* Design user interfaces for registration, login, password reset, and other interactions.

#### 3. **Set Up the Development Environment**

Objective: Prepare the environment for development.

* Install necessary software (e.g., web server, database, programming languages).
* Set up version control (e.g., Git).
* Configure development tools (e.g., IDE, build tools).

### IMPLEMENTATION FOR KEY FUNCTIONS

Implementing key functions in a user authentication system involves handling user registration, login, password management, multi-factor authentication (MFA), and session management. Below is a detailed guide on how to implement these functions.

#### Key Functions

1. **User Registration**
2. **User Login**
3. **Password Management**
4. **Multi-Factor Authentication (MFA)**
5. **Session Management**

### SOURCE CODE:

**Here,The below python code represents the User authentication system:**

# Author Shoolpani Dubey

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.figure as \_figg

from sklearn.pipeline import Pipeline

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.model\_selection import ShuffleSplit, cross\_val\_score

from mne import Epochs, pick\_types, events\_from\_annotations, pick\_channels, preprocessing, concatenate\_epochs

from mne.channels import make\_standard\_montage

from mne.io import concatenate\_raws, read\_raw\_edf

from mne.datasets import eegbci

from mne.decoding import CSP

from mne.decoding import Scaler

import csv

print(\_\_doc\_\_)

# cue onset.

tmin, tmax = -1., 4.

event\_id = dict(hands=2, feet=3)

subject = 1

runs = [3,4,5,6,7,8,9,10,11,12,13,14] # motor imagery: hands vs feet

raw\_fnames = eegbci.load\_data(subject, runs)

raw = concatenate\_raws([read\_raw\_edf(f, preload=True) for f in raw\_fnames])

print("RAW Concatenated EDF")

print(raw)

print(raw.info)

# print(raw.info['ch\_names'])

# print(pick\_channels(raw.info['ch\_names'], include=['Fp1.']))

eog\_epochs = preprocessing.create\_eog\_epochs(raw, ch\_name="Fp1.")

print(eog\_epochs)

rect = 0, .5, 1, 100

\_fig = \_figg.Figure(dpi=200.0)

\_fig.add\_axes([0,.5,0,0])

\_fig.add\_axes([0,0,0,100], frameon=False, facecolor='g')

eog\_epochs.plot\_image()

# eog\_epochs.plot\_image(combine='mean')

data = eog\_epochs.get\_data()

print("EOG DATA EX")

print(data[0].shape)

#printing shape of all the artifacts obtained.

# for x in data:

# print(x.shape)

print("Number of detected EOG artifacts : %d" % len(data))

# Version 1

# We are going to create the dataset in csv format from the raw eeg signals from the users.

# Constants

no\_of\_subjects = 1

def \_getEOGForSub(subject):

# runs = [3,4,5,6,7,8,9,10,11,12,13,14]

runs = [3]

raw\_fnames = eegbci.load\_data(subject, runs)

raw = concatenate\_raws([read\_raw\_edf(f, preload=True) for f in raw\_fnames])

eog\_epochs = preprocessing.create\_eog\_epochs(raw, ch\_name="Fp1.")

data = eog\_epochs.get\_data()

return data

def \_createDataTillSub(subjects):

subListTest=[]

subListTrain=[]

for x in range(0,subjects):

data = \_getEOGForSub(x+1)

print('This is the raw data')

print(data.shape)

# Lets flatten the data

dataFlattened = np.ravel(data)

print(dataFlattened)

print("Data at index 0:",dataFlattened[0])

dataList=dataFlattened.tolist()

dataList.insert(0,x+1)

dataArrayFinal = np.array(dataList)

print("Data1 at index 0:",dataArrayFinal[0])

print(dataArrayFinal)

subListTrain.append(dataArrayFinal)

subListTest.append(dataFlattened)

print("Len of Train Data:",len(subListTrain[0]))

print("Len of Test Data:",len(subListTest[0]))

return subListTrain,subListTest

def \_createCSVFromArrayNHeaders(array,headers,type):

print("HeaderLen",len(headers))

print("Headers",headers[:10])

if type is "train":

name="train001.csv"

else:

name="test001.csv"

print("CSV NAME:",name)

with open(name, "w", newline='') as f:

writer = csv.writer(f, delimiter=',')

writer.writerow(headers) # write the header

# write the actual content line by line

for l in array:

writer.writerow(l)

# or we can write in a whole

# writer.writerows(lines)

def \_createCSVFromArray(array,type):

arrayLen= len(array[0])

print("ArrayLen:",arrayLen)

headers=[]

if type is "train":

headers.append("Subject")

arrayLen= arrayLen-1

for i in range(0,arrayLen):

headers.append("SignalPixel"+str(i))

\_createCSVFromArrayNHeaders(array,headers,type)

def \_createDataframeTrainNTest():

df\_train = pd.read\_csv('train001.csv')

df\_test = pd.read\_csv('test001.csv')

return df\_train,df\_test

def \_main():

# Create EOG data for subject from 1 to 10

dataTrain,dataText= \_createDataTillSub(no\_of\_subjects)

print("Data FP1:")

print(dataTrain[0])

\_createCSVFromArray(dataText,"test")

\_createCSVFromArray(dataTrain,"train")

\_main()

# Version 2 : CNN : More than 90 %

import tensorflow as tf

import keras

import matplotlib.pyplot as plt

from keras.layers import Dense,Flatten, Dropout,Conv2D,MaxPooling2D,Activation

from keras.models import Sequential

from keras.regularizers import l2

import sklearn.preprocessing as preprocessing1

from keras.optimizers import SGD

from sklearn.model\_selection import train\_test\_split

# Constants

no\_of\_subjects = 2

def \_getEOGEpochForSub(subject,channel):

runs = [3,4,5,6,7,8,9,10,11,12,13,14]

verbose = False

# runs = [3]

raw\_fnames = eegbci.load\_data(subject, runs,verbose)

raw = concatenate\_raws([read\_raw\_edf(f, preload=True) for f in raw\_fnames])

eog\_epochs = preprocessing.create\_eog\_epochs(raw, ch\_name=channel)

return eog\_epochs

def \_createDataTillSub(subjects):

X=()

y=()

for x in range(0,subjects):

epoch1 = \_getEOGEpochForSub(x+1,"Fp1.")

epoch2 = \_getEOGEpochForSub(x+1,"Fp2.")

combinedEpoch = concatenate\_epochs((epoch1,epoch2))

combinedData = combinedEpoch.get\_data()

data = Scaler(scalings='median').fit\_transform(combinedData)

print('Shape of the raw data:')

print(data.shape)

print(type(data))

if X is not None:

X=X+tuple(data)

\_y=np.asarray(y)

for i in range(data.shape[0]):

\_y=np.append(\_y,x)

y=tuple(\_y)

print("Len of Train Data:",len(X))

print("Len of Test Data:",len(y))

return X,y

def \_plotRandomTrainData(X\_train,y\_train):

fig = plt.figure()

for i in range(9):

plt.subplot(3,3,i+1)

plt.tight\_layout()

plt.imshow(X\_train[i], cmap='gray', interpolation='none')

plt.title("Subject: {}".format(y\_train[i]))

plt.xticks([])

plt.yticks([])

fig

def \_main():

# Create EOG data for subject from 1 to 10

X,y= \_createDataTillSub(no\_of\_subjects)

# Lets test train split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(np.array(X),np.array(y), test\_size=0.30, random\_state=100)

# random plotting of train data

\_plotRandomTrainData(X\_train,y\_train)

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], X\_train.shape[2], 1).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], X\_test.shape[2], 1).astype('float32')

print("X\_train:",X\_train.shape)

y\_train = keras.utils.to\_categorical(y\_train, no\_of\_subjects)

y\_test = keras.utils.to\_categorical(y\_test, no\_of\_subjects)

print("Y\_test shape:",y\_test.shape)

# Next, Design a basic Conv 3d model

model = Sequential()

model.add(Conv2D(8, (1, 1),padding='same' ,activation='relu',input\_shape=(X\_train.shape[1], X\_train.shape[2], 1), kernel\_regularizer=l2(0.0001)))

model.add(MaxPooling2D((2, 2)))

model.add(Dropout(0.25))

# model.add(Conv2D(16, (1, 1), activation='relu'))

# model.add(MaxPooling2D((2, 2)))

# model.add(Dropout(0.25))

# model.add(Conv2D(64, (1, 1), activation='relu'))

# model.add(MaxPooling2D((2, 2)))

# model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(8, activation='relu',kernel\_regularizer=l2(0.0001)))

model.add(Dropout(0.25))

model.add(Dense(no\_of\_subjects, activation='softmax'))

model.summary()

opt = SGD(lr=0.0001, momentum=0.9)

model.compile(loss='categorical\_crossentropy', optimizer=opt ,metrics=['accuracy'])

history = model.fit(X\_train, y\_train, epochs=5, validation\_data=(X\_test, y\_test))

plt.figure(1)

# summarize history for accuracy

plt.subplot(211)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Training', 'Validation'], loc='lower right')

# summarize history for loss

plt.subplot(212)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Training', 'Validation'], loc='upper right')

plt.tight\_layout()

plt.show()

# evaluate the model

train\_mse = model.evaluate(X\_train, y\_train, verbose=0)

test\_mse = model.evaluate(X\_test, y\_test, verbose=0)

print('Train: %.3f, Test: %.3f' % (train\_mse, test\_mse))

# plot loss during training

pyplot.title('Loss / Mean Squared Error')

pyplot.plot(history.history['loss'], label='train')

pyplot.plot(history.history['val\_loss'], label='test')

pyplot.legend()

pyplot.show()

print('Test loss:', test\_mse[0])

print('Test accuracy:', test\_mse[1])

\_main()

# Version 3 : CNN :

import tensorflow as tf

import keras

from keras import backend as K

import matplotlib.pyplot as plt

from keras.layers import Dense,Flatten, Dropout,Conv2D,MaxPooling2D,Activation

from keras.models import Sequential

from keras.regularizers import l2

import sklearn.preprocessing as preprocessing1

from keras.optimizers import SGD

from sklearn.model\_selection import train\_test\_split

import statistics

from sklearn.metrics import confusion\_matrix

import seaborn as sns

print(tf.\_\_version\_\_)

# Constants

no\_of\_subjects = 2

cap\_limit\_per\_class = 1300

def \_getEOGEpochForSub(subject,channel):

# runs = [3,4,5,6,7,8,9,10,11,12,13,14]

runs = [3]

raw\_fnames = eegbci.load\_data(subject, runs)

raw = concatenate\_raws([read\_raw\_edf(f, preload=True) for f in raw\_fnames])

eog\_epochs = preprocessing.create\_eog\_epochs(raw, ch\_name=channel,baseline=(-0.5, -0.2))

return eog\_epochs

def \_createDataTillSub(subjects):

X=()

y=()

for x in range(0,subjects):

epoch1 = \_getEOGEpochForSub(x+1,"Fp1.")

epoch2 = \_getEOGEpochForSub(x+1,"Fp2.")

epoch3 = \_getEOGEpochForSub(x+1,"Fpz.")

combinedEpoch = concatenate\_epochs((epoch1,epoch2,epoch3))

# combinedEpoch.plot\_image(combine='mean')

# print("Plot power spectrum")

# combinedEpoch.plot\_psd()

# combinedEpoch.plot()

combinedData = combinedEpoch.get\_data()

data = Scaler(scalings='median').fit\_transform(combinedData)

print('Shape of the raw data for subject :',x)

print(data.shape)

print(type(data))

# plot the sample

# fig = plt.figure

# plt.imshow(data[0], cmap='gray\_r')

# plt.show()

# Some eda on data

print('Min and Max of all data')

print(np.min(data), np.max(data))

print(data.std())

if X is not None:

X=X+tuple(data[:cap\_limit\_per\_class])

\_y=np.asarray(y)

for i in range(data[:cap\_limit\_per\_class].shape[0]):

\_y=np.append(\_y,x)

y=tuple(\_y)

print("Len of Train Data:",len(X))

print("Len of Test Data:",len(y))

return X,y

def \_plotRandomTrainData(X\_train,y\_train):

fig = plt.figure()

for i in range(9):

plt.subplot(3,3,i+1)

plt.tight\_layout()

plt.imshow(X\_train[i], cmap='gray', interpolation='none')

plt.title("Subject: {}".format(y\_train[i]))

plt.xticks([])

plt.yticks([])

fig

def recall(y\_true, y\_pred):

y\_true = K.ones\_like(y\_true)

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

all\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

recall = true\_positives / (all\_positives + K.epsilon())

return recall

def precision(y\_true, y\_pred):

y\_true = K.ones\_like(y\_true)

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

precision = true\_positives / (predicted\_positives + K.epsilon())

return precision

def f1\_score(y\_true, y\_pred):

pre = precision(y\_true, y\_pred)

rec = recall(y\_true, y\_pred)

return 2\*((pre\*rec)/(pre+rec+K.epsilon()))

def \_main():

# Create EOG data for subject from 1 to 10

X,y= \_createDataTillSub(no\_of\_subjects)

# Distribution of data for each subject

plt.hist(y)

plt.show()

# Lets test train split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(np.array(X),np.array(y), test\_size=0.50, random\_state=100)

# printing their shapes

print("\_X\_train",X\_train.shape)

print("\_X\_test",X\_test.shape)

print("\_y\_train",y\_train.shape)

print("\_y\_test",y\_test.shape)

# random plotting of train data

# \_plotRandomTrainData(X\_train,y\_train)

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], X\_train.shape[2], 1).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], X\_test.shape[2], 1).astype('float32')

# print("X\_train:",X\_train.shape)

y\_train = keras.utils.to\_categorical(y\_train, no\_of\_subjects)

y\_test = keras.utils.to\_categorical(y\_test, no\_of\_subjects)

# print("Y\_test shape:",y\_test.shape)

print("+\_X\_train",X\_train.shape)

print("+\_X\_test",X\_test.shape)

print("+\_y\_train",y\_train.shape)

print("+\_y\_test",y\_test.shape)

# Next, Design a basic Conv 3d model

model = Sequential()

model.add(Conv2D(8, (1, 1),padding='same' ,activation='relu',input\_shape=(X\_train.shape[1], X\_train.shape[2], 1), kernel\_regularizer=l2(0.0001)))

model.add(MaxPooling2D((2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(8, activation='relu',kernel\_regularizer=l2(0.0001)))

model.add(Dropout(0.25))

model.add(Dense(no\_of\_subjects, activation='softmax'))

model.summary()

opt = SGD(lr=0.0001, momentum=0.9)

model.compile(loss='kullback\_leibler\_divergence', optimizer=opt ,metrics=['accuracy',f1\_score, precision, recall])

history = model.fit(X\_train, y\_train, epochs=1, validation\_data=(X\_test, y\_test))

plt.figure(1)

# summarize history for accuracy

plt.subplot(211)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Training', 'Validation'], loc='lower right')

# summarize history for loss

plt.subplot(212)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Training', 'Validation'], loc='upper right')

plt.tight\_layout()

plt.show()

score = model.evaluate(X\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

print('F1:', score[2])

print('Precision:', score[3])

print('Recall:', score[4])

# Compute confusion matrix

\_pred=model.predict(X\_test, verbose=0)

y\_pred = np.argmax(\_pred, axis=1)

yy\_test = np.argmax(y\_test, axis=1)

print("+\_y\_test",yy\_test)

print("+y\_pred",y\_pred)

confusion\_mtx = tf.math.confusion\_matrix(yy\_test, y\_pred)

print(confusion\_mtx)

plt.figure(figsize=(10, 8))

sns.heatmap(confusion\_mtx, xticklabels=[0,1,2,3,4,5,6,7,8,9], yticklabels=[0,1,2,3,4,5,6,7,8,9],

annot=True, fmt='g')

plt.xlabel('Prediction')

plt.ylabel('Label')

plt.show()

# yy\_test = yy\_test.astype(np.int64)

# y\_pred = y\_pred.astype(np.int64)

# cat\_acc = keras.metrics.categorical\_accuracy(yy\_test, y\_pred)

# print("Recall",recall(yy\_test, y\_pred))

\_main()

from sklearn.metrics import roc\_curve

y\_pred\_keras = model.predict(X\_test).ravel()

fpr\_keras, tpr\_keras, thresholds\_keras = roc\_curve(y\_test, y\_pred\_keras)

# Version 3 : LSTM

import tensorflow as tf

import keras

import matplotlib.pyplot as plt

from matplotlib import pyplot

from keras.layers import Dense,Flatten, Dropout,Conv2D,MaxPooling2D,Activation

from keras.models import Sequential

from keras.regularizers import l2

from numpy import mean

from numpy import std

from sklearn.model\_selection import KFold

from keras.models import Sequential

from keras.layers import LSTM

from keras.layers import MaxPooling2D

from keras.layers import Dense

from keras.layers import Flatten

from keras.optimizers import SGD

from numpy import array

# Constants

no\_of\_subjects = 2

def \_getEOGForSub(subject,channel):

# runs = [3,4,5,6,7,8,9,10,11,12,13,14]

runs = [3]

raw\_fnames = eegbci.load\_data(subject, runs)

raw = concatenate\_raws([read\_raw\_edf(f, preload=True) for f in raw\_fnames])

eog\_epochs = preprocessing.create\_eog\_epochs(raw, ch\_name=channel)

epochs\_data = eog\_epochs.get\_data()

print('Shape of the raw data:')

print(epochs\_data[0])

# data = Scaler(info=epochs.info).fit\_transform(epochs\_data)

# Some how the scalings='mean' is working so that the validation accuracy is not constant anymore.

data = Scaler(scalings='mean').fit\_transform(epochs\_data)

print('Shape of the Scaled raw data:')

print(data[0])

return data

def \_createDataTillSub(subjects):

X=()

y=()

for x in range(0,subjects):

data1 = \_getEOGForSub(x+1,"Fp1.")

data2 = \_getEOGForSub(x+1,"Fp2.")

data = np.concatenate((data1,data2))

print('Shape of the raw data:')

print(data.shape)

print(type(data))

if X is not None:

X=X+tuple(data)

\_y=np.asarray(y)

for i in range(data.shape[0]):

\_y=np.append(\_y,x)

y=tuple(\_y)

print("Len of Train Data:",len(X))

print("Len of Test Data:",len(y))

return X,y

# def \_plotRandomTrainData(X\_train,y\_train):

# fig = plt.figure()

# for i in range(9):

# plt.subplot(3,3,i+1)

# plt.tight\_layout()

# plt.imshow(X\_train[i], cmap='gray', interpolation='none')

# plt.title("Subject: {}".format(y\_train[i]))

# plt.xticks([])

# plt.yticks([])

# fig

def \_main():

# Create EOG data for subject from 1 to 10

X,y= \_createDataTillSub(no\_of\_subjects)

# Lets test train split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(np.array(X),np.array(y), test\_size=0.30, random\_state=100)

# random plotting of train data

# \_plotRandomTrainData(X\_train,y\_train)

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], X\_train.shape[2], 1).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], X\_test.shape[2], 1).astype('float32')

# print("X\_train:",X\_train.shape)

# y\_train = keras.utils.to\_categorical(y\_train, no\_of\_subjects)

# y\_test = keras.utils.to\_categorical(y\_test, no\_of\_subjects)

# print("Y\_test shape:",y\_test.shape)

trainX =X\_train.reshape(X\_train.shape[0],X\_train.shape[1],X\_train.shape[2])

testX =X\_test.reshape(X\_test.shape[0],X\_test.shape[1],X\_test.shape[2])

trainY = keras.utils.to\_categorical(y\_train, no\_of\_subjects)

testY = keras.utils.to\_categorical(y\_test, no\_of\_subjects)

model = Sequential()

model.add(LSTM(8, activation='relu', kernel\_initializer='he\_uniform', input\_shape=(trainX.shape[1],trainX.shape[2])))

# model.add(MaxPooling2D((2, 2)))

# model.add(Flatten())

model.add(Dense(16, activation='relu', kernel\_initializer='he\_uniform'))

model.add(Dropout(0.24))

model.add(Dense(16, activation='relu', kernel\_initializer='he\_uniform'))

model.add(Dropout(0.24))

model.add(Dense(8, activation='relu', kernel\_initializer='he\_uniform'))

model.add(Dense(no\_of\_subjects, activation='softmax'))

model.summary()

# compile model

# opt = SGD(lr=0.001, momentum=0.9)

opt = keras.optimizers.Adam(learning\_rate=0.001)

model.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=['accuracy'])

scores, histories = list(), list()

history = model.fit(trainX, trainY, epochs=1,shuffle=True, batch\_size=100, validation\_data=(testX, testY))

# evaluate model

\_, acc = model.evaluate(testX, testY, verbose=0)

print('> %.3f' % (acc \* 100.0))

# append scores

scores.append(acc)

histories.append(history)

for i in range(len(histories)):

# plot loss

pyplot.subplot(211)

pyplot.title('Cross Entropy Loss')

pyplot.plot(histories[i].history['loss'], color='blue', label='train')

pyplot.plot(histories[i].history['val\_loss'], color='orange', label='test')

# plot accuracy

pyplot.subplot(212)

pyplot.title('Classification Accuracy')

pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')

pyplot.plot(histories[i].history['val\_accuracy'], color='orange', label='test')

pyplot.show()

\_main()

# Version 4 : LSTM :

import tensorflow as tf

import keras

import matplotlib.pyplot as plt

from keras.layers import Dense,Flatten, Dropout,Conv2D,MaxPooling2D,Activation

from keras.models import Sequential

from keras.regularizers import l2

import sklearn.preprocessing as preprocessing1

from keras.optimizers import SGD

from sklearn.model\_selection import train\_test\_split

from matplotlib import pyplot

from keras.regularizers import l2

from numpy import mean

from numpy import std

import statistics

from sklearn.model\_selection import KFold

from keras.models import Sequential

from keras.layers import LSTM

from keras.layers import MaxPooling2D

from keras.layers import Dense

from keras.layers import Flatten

from keras.optimizers import SGD

from numpy import array

from keras import backend as K

from sklearn.metrics import confusion\_matrix

import seaborn as sns

# Constants

no\_of\_subjects = 10

cap\_limit\_per\_class = 1300

def \_getEOGEpochForSub(subject,channel):

runs = [3,4,5,6,7,8,9,10,11,12,13,14]

# runs = [3]

raw\_fnames = eegbci.load\_data(subject, runs)

raw = concatenate\_raws([read\_raw\_edf(f, preload=True) for f in raw\_fnames])

eog\_epochs = preprocessing.create\_eog\_epochs(raw, ch\_name=channel,baseline=(-0.5, -0.2))

return eog\_epochs

def recall(y\_true, y\_pred):

y\_true = K.ones\_like(y\_true)

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

all\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

recall = true\_positives / (all\_positives + K.epsilon())

return recall

def precision(y\_true, y\_pred):

y\_true = K.ones\_like(y\_true)

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

precision = true\_positives / (predicted\_positives + K.epsilon())

return precision

def f1\_score(y\_true, y\_pred):

pre = precision(y\_true, y\_pred)

rec = recall(y\_true, y\_pred)

return 2\*((pre\*rec)/(pre+rec+K.epsilon()))

def \_createDataTillSub(subjects):

X=()

y=()

for x in range(0,subjects):

epoch1 = \_getEOGEpochForSub(x+1,"Fp1.")

epoch2 = \_getEOGEpochForSub(x+1,"Fp2.")

epoch3 = \_getEOGEpochForSub(x+1,"Fpz.")

combinedEpoch = concatenate\_epochs((epoch1,epoch2,epoch3))

# combinedEpoch.plot\_image(combine='mean')

# print("Plot power spectrum")

# combinedEpoch.plot\_psd()

# combinedEpoch.plot()

combinedData = combinedEpoch.get\_data()

data = Scaler(scalings='median').fit\_transform(combinedData)

print('Shape of the raw data for subject :',x)

print(data.shape)

print(type(data))

# plot the sample

# fig = plt.figure

# plt.imshow(data[0], cmap='gray\_r')

# plt.show()

# Some eda on data

print('Min and Max of all data')

print(np.min(data), np.max(data))

print(data.std())

if X is not None:

X=X+tuple(data[:cap\_limit\_per\_class])

\_y=np.asarray(y)

for i in range(data[:cap\_limit\_per\_class].shape[0]):

\_y=np.append(\_y,x)

y=tuple(\_y)

print("Len of Train Data:",len(X))

print("Len of Test Data:",len(y))

return X,y

def \_plotRandomTrainData(X\_train,y\_train):

fig = plt.figure()

for i in range(9):

plt.subplot(3,3,i+1)

plt.tight\_layout()

plt.imshow(X\_train[i], cmap='gray', interpolation='none')

plt.title("Subject: {}".format(y\_train[i]))

plt.xticks([])

plt.yticks([])

fig

def \_main():

# Create EOG data for subject from 1 to 10

X,y= \_createDataTillSub(no\_of\_subjects)

# Distribution of data for each subject

plt.hist(y)

plt.show()

# Lets test train split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(np.array(X),np.array(y), test\_size=0.30, random\_state=100)

# printing their shapes

# print("\_X\_train",X\_train.shape)

# print("\_X\_test",X\_test.shape)

# print("\_y\_train",y\_train.shape)

# print("\_y\_test",y\_test.shape)

# random plotting of train data

# \_plotRandomTrainData(X\_train,y\_train)

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], X\_train.shape[2], 1).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], X\_test.shape[2], 1).astype('float32')

# print("X\_train:",X\_train.shape)

# y\_train = keras.utils.to\_categorical(y\_train, no\_of\_subjects)

# y\_test = keras.utils.to\_categorical(y\_test, no\_of\_subjects)

trainX =X\_train.reshape(X\_train.shape[0],X\_train.shape[1],X\_train.shape[2])

testX =X\_test.reshape(X\_test.shape[0],X\_test.shape[1],X\_test.shape[2])

trainY = keras.utils.to\_categorical(y\_train, no\_of\_subjects)

testY = keras.utils.to\_categorical(y\_test, no\_of\_subjects)

# print("Y\_test shape:",y\_test.shape)

# print("+\_X\_train",X\_train.shape)

# print("+\_X\_test",X\_test.shape)

# print("+\_y\_train",y\_train.shape)

# print("+\_y\_test",y\_test.shape)

model = Sequential()

# model.add(LSTM(8, activation='relu', kernel\_initializer='he\_uniform', input\_shape=(trainX.shape[1],trainX.shape[2])))

model.add(LSTM(8, return\_sequences=True, input\_shape=(trainX.shape[1],trainX.shape[2])))

model.add(Dropout(0.30))

model.add(LSTM(8))

model.add(Dropout(0.30))

model.add(Dense(no\_of\_subjects, activation='softmax'))

model.summary()

opt = keras.optimizers.Adam(learning\_rate=0.0001)

model.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=['accuracy',f1\_score, precision, recall])

scores, histories = list(), list()

history = model.fit(trainX, trainY, epochs=1,shuffle=True, batch\_size=100, validation\_data=(testX, testY))

# evaluate model

# \_, acc = model.evaluate(testX, testY, verbose=0)

# print('> %.3f' % (acc \* 100.0))

# append scores

# scores.append(acc)

histories.append(history)

for i in range(len(histories)):

# plot loss

pyplot.subplot(211)

pyplot.title('Cross Entropy Loss')

pyplot.plot(histories[i].history['loss'], color='blue', label='train')

pyplot.plot(histories[i].history['val\_loss'], color='orange', label='test')

# plot accuracy

pyplot.subplot(212)

pyplot.title('Classification Accuracy')

pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')

pyplot.plot(histories[i].history['val\_accuracy'], color='orange', label='test')

pyplot.show()

score = model.evaluate(testX, testY, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

print('F1:', score[2])

print('Precision:', score[3])

print('Recall:', score[4])

# Compute confusion matrix

\_pred=model.predict(testX, verbose=0)

y\_pred = np.argmax(\_pred, axis=1)

yy\_test = np.argmax(testY, axis=1)

print("+\_y\_test",yy\_test)

print("+y\_pred",y\_pred)

confusion\_mtx = tf.math.confusion\_matrix(yy\_test, y\_pred)

print(confusion\_mtx)

plt.figure(figsize=(10, 8))

sns.heatmap(confusion\_mtx, xticklabels=[0,1,2,3,4,5,6,7,8,9], yticklabels=[0,1,2,3,4,5,6,7,8,9],

annot=True, fmt='g')

plt.xlabel('Prediction')

plt.ylabel('Label')

plt.show()

\_main()

# K NN implementation on the EOG Data.

import tensorflow as tf

import keras

import matplotlib.pyplot as plt

from keras.layers import Dense,Flatten, Dropout,Conv2D,MaxPooling2D,Activation

from keras.models import Sequential

from keras.regularizers import l2

import sklearn.preprocessing as preprocessing1

from keras.optimizers import SGD

from sklearn.model\_selection import train\_test\_split

from matplotlib import pyplot

from keras.regularizers import l2

from numpy import mean

from numpy import std

import statistics

from sklearn.model\_selection import KFold

from keras.models import Sequential

from keras.layers import LSTM

from keras.layers import MaxPooling2D

from keras.layers import Dense

from keras.layers import Flatten

from keras.optimizers import SGD

from numpy import array

from keras import backend as K

from sklearn.metrics import accuracy\_score,confusion\_matrix

import seaborn as sns

from sklearn.multiclass import OneVsRestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import roc\_curve, auc, classification\_report

# Constants

no\_of\_subjects = 10

cap\_limit\_per\_class = 1300

def \_getEOGEpochForSub(subject,channel):

runs = [3,4,5,6,7,8,9,10,11,12,13,14]

# runs = [3]

raw\_fnames = eegbci.load\_data(subject, runs)

raw = concatenate\_raws([read\_raw\_edf(f, preload=True) for f in raw\_fnames])

eog\_epochs = preprocessing.create\_eog\_epochs(raw, ch\_name=channel,baseline=(-0.5, -0.2))

return eog\_epochs

def recall(y\_true, y\_pred):

y\_true = K.ones\_like(y\_true)

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

all\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

recall = true\_positives / (all\_positives + K.epsilon())

return recall

def precision(y\_true, y\_pred):

y\_true = K.ones\_like(y\_true)

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

precision = true\_positives / (predicted\_positives + K.epsilon())

return precision

def f1\_score(y\_true, y\_pred):

pre = precision(y\_true, y\_pred)

rec = recall(y\_true, y\_pred)

return 2\*((pre\*rec)/(pre+rec+K.epsilon()))

def \_createDataTillSub(subjects):

X=()

y=()

for x in range(0,subjects):

epoch1 = \_getEOGEpochForSub(x+1,"Fp1.")

epoch2 = \_getEOGEpochForSub(x+1,"Fp2.")

epoch3 = \_getEOGEpochForSub(x+1,"Fpz.")

combinedEpoch = concatenate\_epochs((epoch1,epoch2,epoch3))

# combinedEpoch.plot\_image(combine='mean')

# print("Plot power spectrum")

# combinedEpoch.plot\_psd()

# combinedEpoch.plot()

combinedData = combinedEpoch.get\_data()

data = Scaler(scalings='median').fit\_transform(combinedData)

print('Shape of the raw data for subject :',x)

print(data.shape)

print(type(data))

# plot the sample

# fig = plt.figure

# plt.imshow(data[0], cmap='gray\_r')

# plt.show()

# Some eda on data

print('Min and Max of all data')

print(np.min(data), np.max(data))

print(data.std())

if X is not None:

X=X+tuple(data[:cap\_limit\_per\_class])

\_y=np.asarray(y)

for i in range(data[:cap\_limit\_per\_class].shape[0]):

\_y=np.append(\_y,x)

y=tuple(\_y)

print("Len of Train Data:",len(X))

print("Len of Test Data:",len(y))

return X,y

def \_plotRandomTrainData(X\_train,y\_train):

fig = plt.figure()

for i in range(9):

plt.subplot(3,3,i+1)

plt.tight\_layout()

plt.imshow(X\_train[i], cmap='gray', interpolation='none')

plt.title("Subject: {}".format(y\_train[i]))

plt.xticks([])

plt.yticks([])

fig

def \_main():

# Create EOG data for subject from 1 to 10

X,y= \_createDataTillSub(no\_of\_subjects)

# Distribution of data for each subject

# plt.hist(y)

# plt.show()

X = np.array(X)

y = np.array(y)

print("X shape:",X.shape)

print("y shape",y.shape)

nsamples, nx, ny = X.shape

X = X.reshape((nsamples,nx\*ny))

print("X shape 1:",X.shape)

# Lets test train split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

# knn = OneVsRestClassifier(KNeighborsClassifier())

knn = KNeighborsClassifier(n\_neighbors=no\_of\_subjects, metric='minkowski', p=2)

knn.fit(X\_train,y\_train)

# predict for one observation

knn.predict(X\_test[0].reshape(1,-1))

# predict for multiple observation (images) at once

knn.predict(X\_test[0:10])

# make prediction on entire test data

predictions = knn.predict(X\_test)

%time

cm = confusion\_matrix(y\_test,predictions)

plt.figure(figsize=(9,9))

sns.heatmap(cm,annot=True, fmt='.3f', linewidths=.5, square=True,cmap='Blues\_r')

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

all\_sample\_title = 'Accuracy Score: {0}'.format(accuracy\_score(y\_test,predictions))

plt.title(all\_sample\_title,size=15)

print('KNN Accuracy: %.3f' % accuracy\_score(y\_test,predictions))

print(classification\_report(y\_test,predictions))

# predict probabilities for X\_test using predict\_proba

probabilities = knn.predict\_proba(X\_test)

# select the probabilities for label 1.0

y\_proba = probabilities[:, 1]

# calculate false positive rate and true positive rate at different thresholds

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_proba, pos\_label=1)

# calculate AUC

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

plt.figure(figsize=(9,9))

plt.title('Receiver Operating Characteristic')

# plot the false positive rate on the x axis and the true positive rate on the y axis

roc\_plot = plt.plot(false\_positive\_rate,

true\_positive\_rate,

label='AUC = {:0.2f}'.format(roc\_auc))

plt.legend(loc=0)

plt.plot([0,1], [0,1], ls='--')

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate');

\_main()

# SVM implementation on the EOG Data.

import tensorflow as tf

import keras

import matplotlib.pyplot as plt

from keras.layers import Dense,Flatten, Dropout,Conv2D,MaxPooling2D,Activation

from keras.models import Sequential

from keras.regularizers import l2

import sklearn.preprocessing as preprocessing1

from keras.optimizers import SGD

from sklearn.model\_selection import train\_test\_split

from matplotlib import pyplot

from keras.regularizers import l2

from numpy import mean

from numpy import std

import statistics

from sklearn.model\_selection import KFold

from keras.models import Sequential

from keras.layers import LSTM

from keras.layers import MaxPooling2D

from keras.layers import Dense

from keras.layers import Flatten

from keras.optimizers import SGD

from numpy import array

from keras import backend as K

from sklearn.metrics import accuracy\_score,confusion\_matrix

import seaborn as sns

from sklearn.multiclass import OneVsRestClassifier

from sklearn.svm import SVC

from sklearn.metrics import roc\_curve, auc, classification\_report

# Constants

no\_of\_subjects = 10

cap\_limit\_per\_class = 1300

def \_getEOGEpochForSub(subject,channel):

runs = [3,4,5,6,7,8,9,10,11,12,13,14]

# runs = [3]

raw\_fnames = eegbci.load\_data(subject, runs)

raw = concatenate\_raws([read\_raw\_edf(f, preload=True) for f in raw\_fnames])

eog\_epochs = preprocessing.create\_eog\_epochs(raw, ch\_name=channel,baseline=(-0.5, -0.2))

return eog\_epochs

def recall(y\_true, y\_pred):

y\_true = K.ones\_like(y\_true)

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

all\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

recall = true\_positives / (all\_positives + K.epsilon())

return recall

def precision(y\_true, y\_pred):

y\_true = K.ones\_like(y\_true)

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

precision = true\_positives / (predicted\_positives + K.epsilon())

return precision

def f1\_score(y\_true, y\_pred):

pre = precision(y\_true, y\_pred)

rec = recall(y\_true, y\_pred)

return 2\*((pre\*rec)/(pre+rec+K.epsilon()))

def \_createDataTillSub(subjects):

X=()

y=()

for x in range(0,subjects):

epoch1 = \_getEOGEpochForSub(x+1,"Fp1.")

epoch2 = \_getEOGEpochForSub(x+1,"Fp2.")

epoch3 = \_getEOGEpochForSub(x+1,"Fpz.")

combinedEpoch = concatenate\_epochs((epoch1,epoch2,epoch3))

# combinedEpoch.plot\_image(combine='mean')

# print("Plot power spectrum")

# combinedEpoch.plot\_psd()

# combinedEpoch.plot()

combinedData = combinedEpoch.get\_data()

data = Scaler(scalings='median').fit\_transform(combinedData)

print('Shape of the raw data for subject :',x)

print(data.shape)

print(type(data))

# plot the sample

# fig = plt.figure

# plt.imshow(data[0], cmap='gray\_r')

# plt.show()

# Some eda on data

print('Min and Max of all data')

print(np.min(data), np.max(data))

print(data.std())

if X is not None:

X=X+tuple(data[:cap\_limit\_per\_class])

\_y=np.asarray(y)

for i in range(data[:cap\_limit\_per\_class].shape[0]):

\_y=np.append(\_y,x)

y=tuple(\_y)

print("Len of Train Data:",len(X))

print("Len of Test Data:",len(y))

return X,y

def \_plotRandomTrainData(X\_train,y\_train):

fig = plt.figure()

for i in range(9):

plt.subplot(3,3,i+1)

plt.tight\_layout()

plt.imshow(X\_train[i], cmap='gray', interpolation='none')

plt.title("Subject: {}".format(y\_train[i]))

plt.xticks([])

plt.yticks([])

fig

def \_main():

# Create EOG data for subject from 1 to 10

X,y= \_createDataTillSub(no\_of\_subjects)

# Distribution of data for each subject

# plt.hist(y)

# plt.show()

X = np.array(X)

y = np.array(y)

print("X shape:",X.shape)

print("y shape",y.shape)

nsamples, nx, ny = X.shape

X = X.reshape((nsamples,nx\*ny))

print("X shape 1:",X.shape)

# Lets test train split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(np.array(X), np.array(y), test\_size=0.25, random\_state=0)

svm = SVC(kernel='linear', probability=True, random\_state=42,verbose=True)

# knn = OneVsRestClassifier(svm)

svm.fit(X\_train,y\_train)

# predict for one observation

svm.predict(X\_test[0].reshape(1,-1))

# predict for multiple observation (images) at once

svm.predict(X\_test[0:10])

# make prediction on entire test data

predictions = svm.predict(X\_test)

%time

cm = confusion\_matrix(y\_test,predictions)

plt.figure(figsize=(9,9))

sns.heatmap(cm,annot=True, fmt='.3f', linewidths=.5, square=True,cmap='Blues\_r')

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

all\_sample\_title = 'Accuracy Score: {0}'.format(accuracy\_score(y\_test,predictions))

plt.title(all\_sample\_title,size=15)

print('SVM Accuracy: %.3f' % accuracy\_score(y\_test,predictions))

print(classification\_report(y\_test,predictions))

# predict probabilities for X\_test using predict\_proba

probabilities = svm.predict\_proba(X\_test)

# select the probabilities for label 1.0

y\_proba = probabilities[:, 1]

# calculate false positive rate and true positive rate at different thresholds

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_proba, pos\_label=1)

# calculate AUC

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

plt.figure(figsize=(9,9))

plt.title('Receiver Operating Characteristic')

# plot the false positive rate on the x axis and the true positive rate on the y axis

roc\_plot = plt.plot(false\_positive\_rate,

true\_positive\_rate,

label='AUC = {:0.2f}'.format(roc\_auc))

plt.legend(loc=0)

plt.plot([0,1], [0,1], ls='--')

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate');

\_main()

#### 5.4Testing and validation:

User authentication systems:are crucial components of any software application, ensuring that users are who they claim to be and protecting sensitive data from unauthorized access. Given their importance, it is essential to rigorously test and validate these systems to ensure they function correctly, securely, and efficiently. This introduction outlines the fundamental concepts and the significance of testing and validation in the context of user authentication systems.

### Importance of Testing and Validation

1. \*\*Security Assurance\*\*:

- \*\*Protection Against Threats\*\*: Authentication systems are prime targets for attackers. Rigorous testing helps identify and mitigate vulnerabilities such as SQL injection, cross-site scripting (XSS), and brute force attacks.

- \*\*Compliance\*\*: Ensures adherence to industry standards and regulations (e.g., GDPR, HIPAA, PCI DSS) that mandate strict security controls for authentication systems.

2. \*\*Functional Accuracy\*\*:

- \*\*Correct User Verification\*\*: Ensures that the system accurately verifies user identities and grants access to authorized users only.

- \*\*Reliability\*\*: Validates that the system functions correctly under various conditions, including edge cases and high-load scenarios.

3. \*\*User Experience\*\*:

- \*\*Usability\*\*: Ensures that the authentication process is user-friendly, reducing friction and improving user satisfaction.

- \*\*Accessibility\*\*: Confirms that the system is accessible to all users, including those with disabilities.

4. \*\*Performance Optimization\*\*:

- \*\*Efficiency\*\*: Validates that the system performs authentication quickly and efficiently, even under peak load conditions.

- \*\*Scalability\*\*: Ensures the system can scale to accommodate growing user bases without degradation in performance.

### Key Aspects of Testing and Validation

1. \*\*Functional Testing\*\*:

- \*\*Unit Tests\*\*: Verify the functionality of individual components (e.g., login, registration, password reset).

- \*\*Integration Tests\*\*: Ensure that different components of the system work together correctly (e.g., integration with databases, external authentication providers).

- \*\*End-to-End Tests\*\*: Simulate real-world scenarios to validate the entire authentication workflow from the user's perspective.

2. \*\*Security Testing\*\*:

- \*\*Penetration Testing\*\*: Simulate attacks to identify vulnerabilities and assess the system's resilience against them.

- \*\*Vulnerability Scanning\*\*: Use automated tools to detect known vulnerabilities.

- \*\*Threat Modeling\*\*: Analyze potential threats and design tests to ensure the system can defend against them.

3. \*\*Performance Testing\*\*:

- \*\*Load Testing\*\*: Assess how the system performs under expected user loads.

- \*\*Stress Testing\*\*: Determine the system's breaking point by pushing it beyond normal operational capacity.

- \*\*Latency Testing\*\*: Measure the response time of the authentication process to ensure it meets performance requirements.

4. \*\*Usability Testing\*\*:

- \*\*User Feedback\*\*: Collect feedback from actual users to identify usability issues and areas for improvement.

- \*\*Accessibility Testing\*\*: Ensure that the system complies with accessibility standards (e.g., WCAG) and is usable by people with disabilities.

5. \*\*Compliance Testing\*\*:

- \*\*Regulatory Compliance\*\*: Verify that the system meets the requirements of relevant regulations and standards.

- \*\*Policy Adherence\*\*: Ensure that the system adheres to internal security and privacy policies.

**### Conclusion**

Testing and validation are integral to the development and maintenance of robust, secure, and user-friendly user authentication systems. By employing a comprehensive testing strategy that includes functional, security, performance, usability, and compliance testing, organizations can ensure their authentication systems meet the highest standards of security and functionality. This, in turn, helps protect sensitive data, enhance user trust, and comply with regulatory requirements.

# CONCLUSION

### CONCLUSION

### Introduction to Testing and Validation in User Authentication Systems

\*\*User authentication systems\*\* are crucial components of any software application, ensuring that users are who they claim to be and protecting sensitive data from unauthorized access. Given their importance, it is essential to rigorously test and validate these systems to ensure they function correctly, securely, and efficiently. This introduction outlines the fundamental concepts and the significance of testing and validation in the context of user authentication systems.

### Importance of Testing and Validation

1. \*\*Security Assurance\*\*:

- \*\*Protection Against Threats\*\*: Authentication systems are prime targets for attackers. Rigorous testing helps identify and mitigate vulnerabilities such as SQL injection, cross-site scripting (XSS), and brute force attacks.

- \*\*Compliance\*\*: Ensures adherence to industry standards and regulations (e.g., GDPR, HIPAA, PCI DSS) that mandate strict security controls for authentication systems.

2. \*\*Functional Accuracy\*\*:

- \*\*Correct User Verification\*\*: Ensures that the system accurately verifies user identities and grants access to authorized users only.

- \*\*Reliability\*\*: Validates that the system functions correctly under various conditions, including edge cases and high-load scenarios.

3. \*\*User Experience\*\*:

- \*\*Usability\*\*: Ensures that the authentication process is user-friendly, reducing friction and improving user satisfaction.

- \*\*Accessibility\*\*: Confirms that the system is accessible to all users, including those with disabilities.

4. \*\*Performance Optimization\*\*:

- \*\*Efficiency\*\*: Validates that the system performs authentication quickly and efficiently, even under peak load conditions.

- \*\*Scalability\*\*: Ensures the system can scale to accommodate growing user bases without degradation in performance.

### Key Aspects of Testing and Validation

1. \*\*Functional Testing\*\*:

- \*\*Unit Tests\*\*: Verify the functionality of individual components (e.g., login, registration, password reset).

- \*\*Integration Tests\*\*: Ensure that different components of the system work together correctly (e.g., integration with databases, external authentication providers).

- \*\*End-to-End Tests\*\*: Simulate real-world scenarios to validate the entire authentication workflow from the user's perspective.

2. \*\*Security Testing\*\*:

- \*\*Penetration Testing\*\*: Simulate attacks to identify vulnerabilities and assess the system's resilience against them.

- \*\*Vulnerability Scanning\*\*: Use automated tools to detect known vulnerabilities.

- \*\*Threat Modeling\*\*: Analyze potential threats and design tests to ensure the system can defend against them.

3. \*\*Performance Testing\*\*:

- \*\*Load Testing\*\*: Assess how the system performs under expected user loads.

- \*\*Stress Testing\*\*: Determine the system's breaking point by pushing it beyond normal operational capacity.

- \*\*Latency Testing\*\*: Measure the response time of the authentication process to ensure it meets performance requirements.

4. \*\*Usability Testing\*\*:

- \*\*User Feedback\*\*: Collect feedback from actual users to identify usability issues and areas for improvement.

- \*\*Accessibility Testing\*\*: Ensure that the system complies with accessibility standards (e.g., WCAG) and is usable by people with disabilities.

5. \*\*Compliance Testing\*\*:

- \*\*Regulatory Compliance\*\*: Verify that the system meets the requirements of relevant regulations and standards.

- \*\*Policy Adherence\*\*: Ensure that the system adheres to internal security and privacy policies.

### Conclusion

Testing and validation are integral to the development and maintenance of robust, secure, and user-friendly user authentication systems. By employing a comprehensive testing strategy that includes functional, security, performance, usability, and compliance testing, organizations can ensure their authentication systems meet the highest standards of security and functionality. This, in turn, helps protect sensitive data, enhance user trust, and comply with regulatory requirements.

# REFERENCE

## REFERENCES:

* https://www.kaggle.com/code/puramnarasimhulu/user-authentication-using-brainwaves-analysis-of-f/edit