Contents

1. Introduction: 4

1.1. Purpose 4

1.2. Scope 4

1.3. Definitions, Acronyms, and Abbreviations 5

1.4. References 5

1.5. Overview 6

2. System Overview 7

2.1. System Context 7

2.2. User personas 7

End User (Regular User): 8

Administrator: 8

Third-Party User (Google OAuth User): 8

Security Administrator: 8

2.3. System Architecture 9

Frontend (User Interface): 9

Backend (Application Logic): 9

Database (User Data Storage): 9

Email Service (OTP Delivery): 10

OAuth Service (Google Login Integration): 10

Session Management: 10

Security Mechanisms: 10

3. System features 11

3.1. User authentication 11

3.1.1. User registration: 11

3.1.2. User Login 11

3.1.3. Google OAuth Login 12

3.2. Multi-Factor Authentication (MFA) 12

3.2.1. Email OTP for Login 12

3.2.2. Email OTP for registration 13

3.2.3. Email OTP for password reset 13

3.3. Session management 13

3.3.1. Session creation and expiration 13
3.3.2. Secure session cookie 14
3.4. Account Security Features 14
3.4.1. Password Hashing and Encryption 14
3.4.2. Account Lockout After Failed Logins 14
3.4.3. Rate Limiting for OTP Requests 15
3.4.4. Logout & Session Expiry 15
3.5. Email communication 15
3.5.1. OTP Email Communication 15
3.5.2. Password Reset Email Communication 16
4. Functional Requirements 17
4.1. User Registration 17
4.2. User Login 17
4.3. Google Login Integration 18
4.4. OTP Generation & Verification 18
4.5. Password Reset Process 19
4.6. Account Lockout & Rate Limiting 19
4.7. Session Expiry & Logout 20
4.8. Secure Email Communication 20
5. Non-Functional Requirements 22
5.1. Performance Requirements 22
5.2. Security Requirements 22
5.3. Availability and Reliability 23
5.4. Scalability 23
5.5. Usability 24
5.6. Compatibility 25
6. Security Requirements 26
6.1. Data Protection & Encryption 26
6.2. Secure Storage of Sensitive Data 26
6.3. Preventing SQL Injection 27
6.4. Preventing Cross-Site Scripting (XSS) 27
6.5. Multi-Factor Authentication (MFA) 28
6.6. Secure Password Storage 28

6.7. Rate Limiting and Anti-Brute Force 29
6.8. Secure Communication (TLS & HTTPS) 29
7. External Interface Requirements 30
7.1. User Interface (UI) Requirements 30
7.2. System Interface Requirements 30
7.3. Hardware Interface Requirements 31
7.4. Software Interface Requirements 31
7.5. Communication Interface Requirements 32
8. Feature enhancements 34
8.1. OAuth2 for Email Sending 34
8.2. Redis Session Storage 34
8.3. Implementing Two-Factor Authentication (TOTP) 35
8.4. Rate-Limiting for Login & OTP Requests 35
9. Appendices 37
9.1. Glossary 37
9.2. Acronyms and Abbreviations 37
9.3. References 38

Authentication app (Software Requirements specification).

1. Introduction:

1.1. Purpose

This Software Requirements Specification (SRS) document outlines the design and functionality of the user authentication system. The primary goal is to provide a secure method for users to register, log in, and manage their accounts with enhanced security measures, including multi-factor authentication (MFA). The system will ensure that users can authenticate their identities through email-based One-Time Passwords (OTPs), protecting sensitive data and preventing unauthorized access. Passwords will be securely hashed using bcrypt.js, and login sessions will be managed to maintain user authentication.

This document defines the system's requirements, security measures, and the expected outcomes to guide developers, testers, and stakeholders through the implementation and validation process. By ensuring clear and precise specifications, this SRS will help create a reliable and secure authentication system.

1.2. Scope

The Authentication System enables users to securely register, log in, and manage their accounts. Key features include user registration with email verification, login via email or username, and secure session management. Multi-factor authentication (MFA) via email OTP enhances security by requiring users to confirm their identity before accessing their account.

Additional functionalities include account lockout after multiple failed login attempts, rate-limited OTP requests to prevent abuse, and integration with Google OAuth for alternative login options. The system will be integrated with the web application, ensuring it functions securely across all user interactions. It will scale with increasing user demands, supporting future enhancements such as Google Authenticator for two-factor authentication (TOTP).

The scope focuses on building a secure authentication mechanism while implementing best practices for password storage, session management, and protection against common web security threats like SQL injection and XSS.

1.3. Definitions, Acronyms, and Abbreviations

i. **MFA (Multi-Factor Authentication)**: A security measure that requires users to provide two or more verification factors (e.g., password and OTP) to access their account.
ii. **OTP (One-Time Password)**: A password valid for a short period, used as a second layer of security for logging in, password resets, and email verification.
iii. **OAuth (Open Authorization)**: A protocol that allows users to log in through third-party services like Google, bypassing the need to create new login credentials.
iv. **bcrypt.js**: A library for securely hashing passwords to protect user data by ensuring passwords are stored in an irreversible format.
v. **SQL Injection**: A web security vulnerability that allows attackers to interfere with the queries a web application makes to its database, potentially gaining unauthorized access to sensitive data.
vi. **XSS (Cross-Site Scripting)**: A vulnerability that allows attackers to inject malicious scripts into webpages, which can then execute in the context of other users' browsers.
vii. **TLS (Transport Layer Security)**: A protocol used to encrypt data during transmission, ensuring that sensitive information is securely sent over the internet.

1.4. References

OAuth 2.0 Authorization Framework (IETF RFC 6749)

This reference outlines the OAuth framework used to integrate third-party login options like Google OAuth.

bcrypt.js Documentation

Official documentation for bcrypt.js, detailing how to securely hash user passwords for storage.

OWASP Top Ten Security Risks

A guide to the top security risks that web applications face, which will be referenced to ensure best security practices are followed.

Password Storage Cheat Sheet (OWASP)

A guideline for securely storing passwords and avoiding common mistakes that compromise password security.

1.5. Overview

This document provides the detailed requirements for the Authentication System. It describes the functionality for secure user registration, login, and session management, along with multi-factor authentication (MFA) via email OTP. Security features such as account lockout after failed login attempts, password hashing with bcrypt.js, and TLS encryption for email communications are also specified.

Further sections will detail the technical specifications, user interfaces, and security mechanisms that ensure the system is both functional and secure. By adhering to these requirements, the system will ensure that user data is protected and that only authorized users gain access to sensitive information.

2. System Overview

2.1. System Context

The authentication system is a core component of a web application designed to manage user access and ensure security across various features. It is integrated into the web application and provides a robust login mechanism, session management, and multi-factor authentication (MFA) capabilities. The system operates within the broader context of the application by interacting with databases for user information storage, email services for OTP verification, and third-party OAuth services like Google for alternative login methods.

The authentication system handles all user access requests, ensuring that only authenticated and authorized users can access secure resources and perform sensitive actions. The system also enforces password policies, including strength requirements and expiration rules, and provides secure password reset functionality to protect against unauthorized access due to forgotten or compromised credentials.

The system is designed to operate in conjunction with the web application, ensuring secure communication and data exchange between different modules and services. Key external components that interact with the system include the email server for sending OTPs and password reset emails, the OAuth service for third-party login integration, and the database for storing user credentials, session tokens, and other security-related data.

2.2. User personas

The authentication system is built to cater to various types of users, each with distinct needs and behaviors. The following are the primary user personas:

End User (Regular User):

The typical user who will use the application to access secure areas. They will register, log in, and manage their account settings. They may request a password reset or verify their email address through OTPs. Their main goal is to use the system securely and easily, with minimal friction during the login process.

Administrator:

Administrators have elevated privileges within the system. They manage user accounts, monitor login activity, and handle account-related issues, such as password resets or account lockouts. They ensure the integrity of the authentication system by reviewing security logs and enforcing policy changes when necessary.

Third-Party User (Google OAuth User):

Users who prefer logging in using their Google account rather than creating a new password. This persona focuses on ease of access and will be able to authenticate using OAuth credentials from Google, bypassing the need for traditional email/password combinations.

Security Administrator:

Security administrators are responsible for overseeing the overall security of the authentication system. They ensure that all security measures, such as rate-limiting for OTP requests, account lockouts, and encryption protocols, are properly implemented and functioning. They also track user activity to identify potential threats and breaches.

2.3. System Architecture

The architecture of the authentication system is designed to be modular, scalable, and secure. It consists of several interconnected components, each responsible for a specific part of the authentication process.

Frontend (User Interface):

The user interface (UI) is the entry point for users interacting with the system. It includes the login form, registration page, password reset page, and Google OAuth login button. The frontend is responsible for collecting user inputs, validating them, and sending them to the backend for processing. It also displays appropriate error or success messages, such as "OTP sent to email" or "Password reset successful."

Backend (Application Logic):

The backend handles the core business logic of the authentication system. It processes user registrations, login attempts, password resets, and OTP verifications. It also manages session creation and expiration using secure session tokens. The backend is built with Node.js and Express, serving as the API layer that communicates with both the frontend and the database.

Database (User Data Storage):

The database is where user data, such as email, username, password hash, and session tokens, are securely stored. It also stores information related to OTP expiration times and password reset tokens. The database uses SQLite or any relational database that ensures efficient querying and secure data storage. SQL queries are parameterized to prevent SQL injection attacks.

Email Service (OTP Delivery):

The email service is responsible for sending OTPs to users for email verification, login, and password reset. It ensures that OTPs are delivered securely and within a time window, typically 5 minutes. The system uses an SMTP server, secured via TLS encryption, to send the OTP emails to the users.

OAuth Service (Google Login Integration):

The OAuth service enables users to log in using their Google account credentials. This component communicates with the Google OAuth API, securely handling authentication tokens and ensuring that users' personal data is not exposed. The OAuth service is integrated with Passport.js to facilitate the login flow.

Session Management:

Session management is crucial for maintaining authenticated user states across different pages of the application. The system uses express session to generate and manage secure session cookies, which persist across page refreshes. These sessions expire after a defined period (e.g., 24 hours), ensuring that users are required to log in again for security purposes.

Security Mechanisms:

The authentication system implements several security mechanisms, such as password hashing using bcrypt.js, account lockout after multiple failed login attempts, rate-limiting for OTP requests, and input validation to protect against SQL injection and cross-site scripting (XSS) attacks. These mechanisms ensure that users' sensitive data is protected and that malicious activities, such as brute-force attacks, are mitigated.

3. System features

3.1. User authentication

3.1.1. User registration:

User registration is the first step in creating an account within the system. Users will be required to provide basic personal details, including first name, last name, email, username, and a password. Upon submission, the system will perform several actions to ensure the accuracy and security of the user's data:

- Email Verification: An OTP is sent to the provided email address. The user must enter the OTP to confirm ownership of the email and complete the registration process. This step prevents fraudulent or erroneous registrations and ensures that users have access to a valid email address.
- Password Hashing: The password entered by the user will be hashed using bcrypt.js before being stored in the database. This ensures that the password is stored in a secure, irreversible format and protects against unauthorized access even if the database is compromised.
- Duplicate Account Prevention: The system will check if the provided email or username already exists in the database. If either is found, the user will be informed and asked to choose another value.

3.1.2. User Login

Once registered, users can log in to the system using either their email or username, along with their password. The login process includes the following steps:

- Credential Validation: The system will validate the email/username and password entered by the user. If the credentials are correct, the system proceeds to generate an OTP for further verification.
- Multi-Factor Authentication (MFA): Upon successful credential validation, the system will send a 6-digit OTP to the user's registered email address. The user will need to enter this OTP within 5 minutes to complete the login process.

- Session Creation: If the OTP is verified successfully, a session will be created, and the user will be logged in. The system will generate a secure session cookie that persists throughout the user's session until they log out or the session expires.

3.1.3. Google OAuth Login

The system integrates with Google OAuth to provide an alternative login method, allowing users to log in using their Google credentials. This option simplifies the authentication process for users who prefer not to remember additional login details. The OAuth login will be facilitated by Passport.js. The process involves:

- OAuth Integration: When users select the Google Login option, they will be redirected to Google's OAuth authorization page.
- Token Exchange: After the user grants access, an authorization token will be exchanged for an access token, which allows the system to retrieve the user's Google account information.
- Session Creation: Upon successful authentication, the system will create a session and allow the user to access the application securely.

3.2. Multi-Factor Authentication (MFA)

3.2.1. Email OTP for Login

Multi-factor authentication (MFA) is an additional layer of security added to the login process. After the user enters their email or username and password, they will receive an OTP via email. This OTP ensures that even if the user's password is compromised, unauthorized access is still prevented. The MFA steps are as follows:

- OTP Generation: The system generates a unique 6-digit OTP and sends it to the user's registered email.
- OTP Expiry: The OTP will expire after 5 minutes, and any attempt to verify the OTP after this time will result in failure.
- OTP Verification: The user must enter the OTP to complete the login process. Once verified, the user gains access to the system.

3.2.2. Email OTP for registration

To ensure secure registration, the system requires an OTP to verify the user's email before finalizing their account. After the user submits their registration form:

- OTP Generation: An email containing a unique 6-digit OTP is sent to the user's registered email address.
- OTP Verification: The user must enter the OTP into the registration page to confirm their email address. The registration will only be completed if the OTP is verified.
- OTP Expiry: The OTP expires after 5 minutes to prevent misuse.

3.2.3. Email OTP for password reset

In the event that a user forgets their password or wishes to reset it, the system provides an OTP verification step to confirm the user's identity:

- OTP Generation: A 6-digit OTP is sent to the user's email address when a password reset request is initiated.
- OTP Verification: The user must enter the OTP to verify their identity before being allowed to reset their password.
- OTP Expiry: The OTP expires after 5 minutes to prevent unauthorized use.

3.3. Session management

3.3.1. Session creation and expiration

Session management ensures that users remain authenticated across different pages of the application without needing to log in again. The system will:

- Session Creation: After successful login (via email/username + password and OTP), a secure session will be created using express session. This session will store information that identifies the user across multiple requests.

- Session Expiration: The session will automatically expire after 24 hours, requiring users to log in again for security purposes. Sessions can also be manually invalidated when users log out.

3.3.2. Secure session cookie

To maintain a secure connection during a user's session, the system will use secure cookies:

- Secure Cookies: Cookies will be set with the `HttpOnly` and `Secure` flags to prevent session hijacking and ensure that the cookie is only transmitted over secure HTTPS connections.
- Session Management: The session cookie is linked to the user's authentication state and is automatically destroyed upon logout.

3.4. Account Security Features

3.4.1. Password Hashing and Encryption

To protect user credentials, the system uses bcrypt.js to hash passwords before storing them in the database. This process ensures:

- One-way Hashing: Passwords are hashed using bcrypt's key derivation function, making it computationally infeasible to reverse the hashed password.
- Salted Hashing: A random salt is added to each password before hashing, further increasing security by ensuring that even identical passwords produce unique hashes.

3.4.2. Account Lockout After Failed Logins

To prevent brute-force attacks, the system enforces an account lockout policy:

- Failed Login Attempts: If a user fails to log in 3 times within a short period, their account will be temporarily locked for 15 minutes.
- Temporary Suspension: During the lockout period, any further login attempts will be rejected until the lockout expires, providing additional protection against automated attacks.

3.4.3. Rate Limiting for OTP Requests

The system implements rate limiting to prevent abuse of the OTP functionality:

- OTP Requests: Users are restricted to a limited number of OTP requests within a defined time window (e.g., 5 OTP requests per hour).
- Prevention of Abuse: Excessive OTP requests will trigger warnings, and further requests may be temporarily blocked to prevent spam and malicious activities.

3.4.4. Logout & Session Expiry

To ensure the security of user accounts:

- **Logout**: Users can log out of their account at any time. When they log out, the system destroys their session, ensuring that unauthorized access is not possible.
- **Session Expiry**: Sessions automatically expire after **24 hours** of inactivity, prompting the user to log in again.

3.5. Email communication

3.5.1. OTP Email Communication

The system uses email communication to send OTPs for various purposes:

- **Email Security**: Emails containing OTPs are sent using a secure email service with **TLS encryption** to protect against eavesdropping.
- **OTP Message**: The email contains a **6-digit OTP** and a message explaining its purpose (e.g., for login, registration, or password reset).
- **Expiry**: The OTP expires after **5 minutes**, and users are informed of the expiry in the email to encourage timely use.

3.5.2. Password Reset Email Communication

When users request a password reset, the system sends an email with the following:

- Password Reset Link: A link to the password reset page, which is valid for 1 hour from when it was sent.
- Email Security: The reset email is securely transmitted using TLS encryption, ensuring that sensitive information is protected during transit.

4. Functional Requirements

4.1. User Registration

The user registration process is the initial step in the creation of a new user account. The functional requirements for this feature include:

- **Input Fields**: Users must be required to fill in the following fields: **first name**, **last name**, **email**, **username**, and **password**.
- **Email Verification**: Upon registration, an OTP will be sent to the user's email address. The user must enter this OTP to verify their email address and complete the registration process.
- **Password Hashing**: The user's password will be securely hashed using **bcrypt.js** before being stored in the database.
- **Duplicate Account Prevention**: The system will check the database for any existing **email** or **username**. If a match is found, the user will be prompted to choose a different value.
- **Error Handling**: The system will validate user inputs and provide appropriate error messages for invalid fields, including **invalid email format**, **password strength requirements**, and **username already taken**.

4.2. User Login

The user login process enables registered users to access their accounts securely. The functional requirements include:

- **Input Fields**: Users must provide either their **email** or **username** and their **password** to authenticate.
- **Credential Validation**: The system will compare the entered credentials with the stored credentials in the database. If the credentials match, the system will proceed to the next step.
- **OTP Verification**: Upon successful credential validation, an OTP will be sent to the user's registered email address. The user must enter this OTP to proceed with the login.

- **Login Success**: If the OTP is correct, the system will create a secure session for the user, storing their authentication state across multiple requests.
- **Login Failure**: If the credentials are incorrect or the OTP is not validated, the system will return an error message, prompting the user to re-enter their details.

4.3. Google Login Integration

The system supports Google login as an alternative method of user authentication. The functional requirements include:

- **OAuth Integration**: The system will integrate **Passport.js** to handle OAuth login via Google.
- **Redirection to Google Login**: Users must be able to click on a **Google Login** button, which redirects them to the Google OAuth page for authentication.
- **Token Retrieval**: Once the user authenticates via Google, the system will retrieve the user's **Google profile information** (e.g., name, email) and use it to create an account or log the user in.
- **Session Management**: Upon successful authentication via Google, the system will create a session for the user, and they will be redirected to the application's main page.

4.4. OTP Generation & Verification

The system will utilize **One-Time Passwords (OTPs)** for additional security during the login, registration, and password reset processes. The functional requirements include:

- **OTP Generation**: After credential validation (login or registration), the system will generate a unique **6-digit OTP**.
- **Email Delivery**: The OTP will be sent to the user's **registered email address** securely using TLS encryption.
- **OTP Expiry**: The OTP will expire after **5 minutes** to ensure it cannot be reused.

- **OTP Verification**: The user must input the OTP they received to complete the verification process. If the OTP matches and is within the time limit, the user will be granted access.
- **Error Handling**: If the OTP is incorrect or expired, the system will return an error message, prompting the user to request a new OTP.

4.5. Password Reset Process

Users can reset their password in the event of forgetting it. The functional requirements for the password reset process include:

- **Password Reset Request**: Users can initiate a password reset by entering their **email** on the password reset page.
- **OTP for Verification**: After receiving the password reset request, an OTP will be sent to the user's **registered email address**. The user must enter the OTP to verify their identity.
- **New Password Input**: Once the OTP is verified, the user will be prompted to input a new password. The new password will be hashed using **bcrypt.js** before being stored in the database.
- **Password Strength Validation**: The new password must meet the defined **security requirements** (e.g., minimum length, special characters).
- **Error Handling**: If the user enters an incorrect OTP or fails to meet password criteria, the system will return appropriate error messages.

4.6. Account Lockout & Rate Limiting

To prevent brute-force and other automated attacks, the system includes account lockout and rate-limiting features. The functional requirements include:

- **Account Lockout After Failed Logins**: If a user fails to log in **3 times** consecutively, their account will be locked for **15 minutes**. During this time, any further login attempts will be blocked.

- **OTP Rate Limiting**: Users will be limited to a set number of **OTP requests per hour** (e.g., 5 OTP requests). Excessive requests will trigger a warning, and further requests will be temporarily blocked.
- **Rate-Limiting Middleware**: The system will implement **rate-limiting middleware** to control the frequency of login and OTP requests, protecting against abuse.

4.7. Session Expiry & Logout

Session management ensures that users are securely authenticated and that their sessions do not remain active indefinitely. The functional requirements include:

- **Session Creation**: Upon successful login, a session will be created using **express session**. The session will store the user's authentication state.
- **Session Expiry**: The session will automatically expire after **24 hours** or after a specified period of inactivity.
- **Manual Logout**: Users can log out of the system at any time by clicking the **Logout** button. This will destroy the user's session, ensuring that unauthorized access is prevented.
- **Session Timeout Warning**: Users will receive a **warning message** if their session is about to expire, allowing them to extend the session if needed.

4.8. Secure Email Communication

Email communication plays a critical role in the authentication process, particularly for OTPs and password resets. The functional requirements for secure email communication include:

- **TLS Encryption**: All emails containing OTPs or password reset links will be transmitted using **TLS encryption** to ensure the security of the email contents during transmission.
- **Email Templates**: Emails will be formatted with clear instructions, including the OTP or password reset link, and will include a message about the expiration of the OTP.

- **Email Validity**: The system will verify the email address provided during registration to ensure it is valid and belongs to the user.
- **Error Handling**: If the email cannot be sent due to an issue (e.g., network problem), the system will inform the user and provide troubleshooting steps.

5. Non-Functional Requirements

5.1. Performance Requirements

Performance requirements are critical to ensuring the system operates efficiently and meets the needs of users. The system must meet the following performance standards:

- Response Time: The system should respond to user requests (e.g., login, registration, password reset) within 2 seconds under normal load conditions.
- Scalability Under Load: The system should handle at least 100 concurrent users performing actions such as logging in, OTP verification, and password resets, with response times not exceeding 3 seconds.
- Database Performance: SQL queries, particularly for authentication, OTP verification, and password reset, should execute within 500 milliseconds.
- High Availability: The system should maintain 99.9% uptime during regular operating hours and be capable of recovering from minor failures within 5 minutes.

5.2. Security Requirements

Security is paramount for the authentication system, as it handles sensitive user data and credentials. The security requirements are as follows:

- Data Encryption: All sensitive data, such as passwords and OTPs, should be encrypted at rest using strong encryption algorithms, such as AES-256 for data storage.
- Session Management Security: Session cookies should use Secure, HttpOnly, and SameSite flags to prevent session hijacking and cross-site scripting (XSS) attacks.
- Password Hashing: User passwords must be hashed using bcrypt.js with a sufficiently high salt rounds value (e.g., 12) to prevent password cracking.
- Two-Factor Authentication: The system must implement a multi-factor authentication process for logging in and password resets to ensure a higher level of security.

- TLS/SSL Encryption: All communication, including emails and user interactions with the web app, must use TLS (Transport Layer Security) encryption to prevent data interception.
- Protection Against Attacks: The system must be protected against common security threats, including SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Brute Force attacks.

5.3. Availability and Reliability

The system must ensure that it is highly available and reliable for users, especially for critical functionalities such as logging in and resetting passwords. The following requirements are necessary:

- Uptime: The system must ensure an availability rate of 99.9% over a 12-month period, meaning the system should be down for no more than 8.77 hours per year.
- Failover Capabilities: The system must have failover mechanisms in place, including database replication and backup systems to ensure continued operation during failures.
- Error Handling and Recovery: The system should be able to recover from unexpected errors and resume normal operation within 5 minutes. It should include logging and monitoring to detect system failures and trigger automatic recovery processes.
- Backup and Disaster Recovery: The system must perform daily backups of user data and configuration to ensure recovery in the event of data loss. Backups must be stored securely and be accessible for restoration in case of system failure.

5.4. Scalability

The system must be designed to scale efficiently as user traffic increases. Scalability requirements include:

- Horizontal Scaling: The system must support horizontal scaling to distribute traffic and load across multiple servers as the number of users grows. This includes the ability to scale the web application and the database layer.

- Database Scaling: The database must be able to scale as the number of users increases, ensuring that queries and authentication processes continue to perform efficiently. Techniques such as sharding or partitioning may be employed to improve database performance.
- Cloud Infrastructure: The system must be deployable on cloud infrastructure such as AWS, Azure, or Google Cloud, allowing for dynamic resource allocation based on traffic demands.
- Load Balancing: The system should implement load balancers to evenly distribute user requests across multiple servers, ensuring that no single server becomes a bottleneck.

5.5. Usability

Usability is a key factor in ensuring that users can navigate and interact with the authentication system easily. The following usability requirements are essential:

- User Interface (UI): The system should provide an intuitive and user-friendly interface for registration, login, OTP verification, and password reset. Form fields should be clearly labeled, and error messages should be descriptive and helpful.
- Accessibility: The system must be accessible to users with disabilities. It should comply with WCAG 2.1 Level AA guidelines, including screen reader support and keyboard navigation.
- Mobile Compatibility: The system must be fully responsive and usable on mobile devices, ensuring that users can log in and manage their accounts from smartphones and tablets without issues.
- Error Handling and Feedback: The system should provide immediate feedback to users about the success or failure of their actions (e.g., registration, login, OTP verification) and offer clear instructions for troubleshooting common issues.

5.6. Compatibility

The system must be compatible with various browsers, operating systems, and third-party services to ensure smooth operation for all users. The following compatibility requirements are outlined:

- Browser Compatibility: The system must work seamlessly with the latest versions of Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, and Opera. It should also provide backward compatibility for previous versions of these browsers as much as possible.
- Operating System Compatibility: The system must be compatible with popular operating systems, including Windows, macOS, and Linux for desktop users. For mobile users, the system should support Android and iOS platforms.
- Email Providers Compatibility: The system must ensure compatibility with popular email services such as Gmail, Yahoo Mail, Outlook, and ProtonMail, ensuring OTPs and password reset emails are delivered reliably.
- Third-Party Integrations: The system must integrate with third-party services such as Google OAuth and SMTP email providers for authentication and email delivery. Compatibility with these services must be ensured for smooth operation.

6. Security Requirements

6.1. Data Protection & Encryption

Data protection is a critical requirement to ensure that all sensitive user data, including passwords, personal information, and OTPs, is securely protected from unauthorized access. The system must meet the following data protection standards:

- Encryption at Rest: All sensitive data, including passwords, OTPs, and user details, must be stored in an encrypted format using strong encryption algorithms like AES-256. This ensures that even if the data is accessed by unauthorized entities, it cannot be read or tampered with.
- Encryption in Transit: All communication between users and the system, including login requests, OTP verifications, and password resets, must be encrypted using TLS (Transport Layer Security) protocols to prevent interception and unauthorized access during transmission.
- Key Management: Encryption keys must be securely managed using a Key Management System (KMS) or a similar service to prevent unauthorized access or compromise of sensitive data.

6.2. Secure Storage of Sensitive Data

Sensitive data, including user credentials and OTPs, must be stored securely to prevent leakage or unauthorized access. The following guidelines should be followed:

- Password Hashing: Passwords must never be stored in plaintext. Instead, they should be securely hashed using a strong hashing algorithm such as bcrypt.js, which provides salting and multiple iterations to make brute-force attacks impractical.
- OTP Storage: OTPs should be stored temporarily in a secure, hashed format, and only kept for the duration of their validity. Once the OTP expires or is used, it must be securely removed from the system to avoid unauthorized reuse.

- Database Access Control: Access to sensitive data in the database must be tightly controlled. Only authorized users or services should have access to this data, and database access should be monitored and logged.

6.3. Preventing SQL Injection

SQL Injection is a common attack vector that can compromise the security of the system. To prevent this, the following measures must be implemented:

- Parameterized Queries: The system must always use parameterized queries when interacting with the database to prevent malicious input from altering the structure of SQL queries. This ensures that user input is treated as data rather than executable code.
- ORMs: If applicable, use Object-Relational Mapping (ORM) libraries such as Sequelize or TypeORM
- that handle query construction securely and prevent raw SQL execution.
- Input Validation and Sanitization: All user inputs must be validated for expected formats (e.g., no special characters in usernames or email addresses) and sanitized to remove any potentially malicious code before being processed.

6.4. Preventing Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks occur when malicious scripts are injected into web pages viewed by other users. The system must take the following measures to prevent XSS:

- Output Encoding: All user-generated content that is displayed on the webpage must be properly encoded using HTML escape functions to prevent injected scripts from being executed by the browser.
- Content Security Policy (CSP): Implement a Content Security Policy (CSP) to restrict the types of content (scripts, images, etc.) that can be loaded and executed on the web pages, thereby reducing the risk of executing malicious code.

- Sanitization Libraries: Use trusted libraries like DOMPurify to sanitize user-generated HTML or input before rendering it on the page.

6.5. Multi-Factor Authentication (MFA)

Multi-factor authentication (MFA) is essential for securing user accounts. The system must enforce MFA for sensitive operations, such as login and password reset. This can be achieved by:

- Email OTP (One-Time Password): After the user enters their credentials (email/username + password), an OTP is sent to their registered email address. The user must enter the OTP to complete the login process.
- SMS-based MFA: If needed, MFA can also be extended to SMS-based OTPs for additional security. This can be integrated through a third-party service.
- Google Authenticator: For stronger security, the system can support Time-based One-Time Passwords (TOTP) through services like Google Authenticator, which generate an additional layer of authentication beyond email or SMS.

6.6. Secure Password Storage

Passwords are a critical part of user security, and it’s essential to store them securely. The system must adhere to the following password storage guidelines:

- Strong Hashing Algorithms: Passwords should be hashed using a modern, cryptographically secure hashing algorithm such as bcrypt.js, Argon2, or PBKDF2. The hashing process must involve salting to ensure that each password is unique, even if two users have the same password.
- Password Strength Enforcement: The system should enforce strong password policies, requiring users to create passwords with a minimum length and complexity, such as a mix of uppercase letters, lowercase letters, numbers, and special characters.
- Password Salting: A unique salt should be applied to each password before hashing to prevent the use of precomputed hash tables (e.g., rainbow tables) for cracking passwords.

6.7. Rate Limiting and Anti-Brute Force

To prevent brute-force attacks on login, password reset, and OTP verification endpoints, the system must implement the following:

- Rate Limiting: Use rate-limiting mechanisms to restrict the number of login attempts, OTP requests, and password reset requests a user can make within a specified time period. This helps mitigate brute-force and credential stuffing attacks.
- Account Lockout: After a user exceeds the maximum number of failed login attempts (e.g., 5), their account should be locked for a short period (e.g., 15 minutes) or until an administrator manually unlocks it.
- Captcha Integration: For sensitive operations like login or password reset, the system can integrate CAPTCHA challenges (e.g., Google reCAPTCHA) to distinguish between human users and automated bots.

6.8. Secure Communication (TLS & HTTPS)

Ensuring secure communication between the client and the server is crucial to protect user data in transit. The following security requirements must be adhered to:

- TLS/SSL Encryption: All communication between the user's browser and the server must be encrypted using TLS (Transport Layer Security) or SSL (Secure Sockets Layer) to protect data from being intercepted or tampered with by malicious actors.
- HTTPS for All Pages: The system must use HTTPS across all pages and endpoints to ensure that user data, including passwords, OTPs, and personal information, is always transmitted securely.
- Strict Transport Security (HSTS): Implement HTTP Strict Transport Security (HSTS) to enforce the use of HTTPS for all communications, thereby preventing downgrade attacks and cookie hijacking.

7. External Interface Requirements

7.1. User Interface (UI) Requirements

The user interface (UI) must be intuitive, responsive, and accessible to ensure a seamless and positive user experience. The UI should be designed with the following features:

- Login & Registration Forms: These forms should allow users to enter their credentials securely. The user registration form should capture first name, last name, email, username, and password, with validation for each field. The login form should provide fields for either email or username and password, with a button for submitting credentials.
- OTP Verification: After entering the correct login credentials, the user should be prompted to enter the One-Time Password (OTP) sent via email. This form should be clear and concise, with a timer indicating the OTP expiration.
- Password Reset Forms: The password reset form should allow users to request a reset link by entering their email. The reset link sent via email should direct users to a form where they can securely set a new password.
- Error Handling & Validation: The UI should provide real-time feedback for user input validation (e.g., password strength, valid email format). In case of errors (e.g., incorrect credentials, expired OTP), appropriate error messages should be displayed to guide the user.
- Responsive Design: The UI must be responsive and adapt to various screen sizes, such as desktop, tablet, and mobile devices. This will ensure users have an optimal experience regardless of the device they are using.
- Accessibility: The UI should meet accessibility standards, including appropriate contrast ratios, font sizes, and navigation options for users with disabilities.

7.2. System Interface Requirements

The system should interact seamlessly with external systems, including email services, OAuth authentication providers, and databases. The following system interface requirements must be met:

- Email Service Integration: The system must be able to send email-based OTPs for registration, login, and password resets. Integration with an email service provider (such as Gmail SMTP, SendGrid, or Mailgun) is required to send these emails securely.
- OAuth Providers: For Google OAuth login integration, the system must interact with the Google OAuth 2.0 API for authentication. This requires secure interaction with Google's authentication endpoints and handling of OAuth tokens.
- Database Interaction: The system must interact with a SQLite database to store user data securely. This includes handling CRUD operations for user accounts, session management, and OTP generation. The system must use parameterized queries to prevent SQL injection.

7.3. Hardware Interface Requirements

The system has minimal hardware interface requirements, as it is a web-based application. However, the following hardware requirements should be considered:

- Server Requirements: The application should be hosted on a secure server with adequate processing power to handle user authentication requests, email verifications, and session management. A cloud server (e.g., AWS, Azure, or Google Cloud) is recommended for scalability and reliability.
- User Device Requirements: The system must be accessible from devices with internet access, including desktops, laptops, and mobile phones. Users should be able to interact with the application from web browsers such as Chrome, Firefox, Safari, and Edge.
- Storage Requirements: The system should have sufficient storage capacity for user data and logs. Database storage must be able to handle a growing number of user accounts, login attempts, and session data.

7.4. Software Interface Requirements

The system must interface with several software components and services, including:

- Web Browsers: The system must be compatible with modern web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. The application should be optimized for both desktop and mobile web browsers.
- Node.js and Express: The backend system will use Node.js and Express for server-side logic, routing, and handling HTTP requests. The system must be compatible with the required Node.js version (e.g., v14.x or higher) and related packages (e.g., express-session, bcrypt.js).
- Passport.js: For Google OAuth login functionality, the system will utilize Passport.js, a Node.js authentication middleware. The system must ensure proper handling of OAuth tokens and securely store session information.
- SQLite: The system will use SQLite as the database management system to store user data, session information, and OTPs. The system must implement SQLAlchemy or similar tools for database access.
- Email API Service: The system will integrate with an email API service (e.g., SendGrid, Mailgun) to send OTPs and password reset emails securely. The service must support SMTP and allow for seamless email sending.

7.5. Communication Interface Requirements

The system must communicate with external services and users using secure and reliable communication protocols. These include:

- HTTP/HTTPS Protocols: All communications between the client and server should be conducted over HTTPS to ensure encryption and prevent man-in-the-middle (MITM) attacks. The system should reject HTTP connections and enforce secure communication with SSL/TLS encryption.
- SMTP for Email Communication: The system must send emails (such as OTPs and password reset links) using a secure SMTP (Simple Mail Transfer Protocol) server. The email communication must be encrypted using TLS to ensure that email content cannot be intercepted.

- OAuth API Communication: The system must use OAuth 2.0 to authenticate users via Google. This communication involves exchanging authorization codes, access tokens, and refresh tokens with Google’s OAuth endpoints. All communications with Google’s API should be secured using TLS.

8. Feature enhancements

8.1. OAuth2 for Email Sending

Currently, the system uses SMTP credentials to send email OTPs and password reset links. However, for improved security and easier management of email accounts, the system could be enhanced by integrating OAuth2 for email sending. OAuth2 provides more secure token-based authentication and eliminates the need for storing and managing sensitive email account passwords.

Benefits:

- Improved Security: OAuth2 allows access to email services without exposing email credentials, reducing the risk of account compromise.
- Better Token Management: OAuth2 tokens can be easily revoked or rotated, making the system more secure.
- Scalability: OAuth2 supports better integration with multiple email service providers (e.g., Gmail, SendGrid), making the system more adaptable to future changes.

8.2. Redis Session Storage

Currently, the system stores user sessions in memory using express-session. As the system scales, this approach may face limitations in terms of reliability and scalability. Redis is an in-memory data store that can provide faster and more scalable session management. By implementing Redis for session storage, we can ensure that sessions are preserved across multiple servers and instances.

Benefits:

- Scalability: Redis allows the system to scale horizontally across multiple servers by storing sessions in a distributed cache.
- Performance: Redis offers low-latency access to session data, improving the overall performance of the authentication system.
- Persistence: Redis offers optional persistence, ensuring that session data is not lost even in the case of server crashes.

8.3. Implementing Two-Factor Authentication (TOTP)

Currently, the system uses email-based OTP for multi-factor authentication (MFA). While this is a good solution for security, it can be vulnerable to phishing or email account compromises. Two-Factor Authentication (TOTP) using an app like Google Authenticator or Authy can offer a more secure and user-friendly alternative.

Benefits:

- Increased Security: TOTP-based MFA is more secure than email-based OTP as it does not rely on email delivery and is not subject to the same risks.
- User Control: Users have more control over their authentication by using their mobile devices to generate the authentication code.
- Widely Adopted: TOTP is a widely adopted standard for securing accounts, and many users are familiar with it.

8.4. Rate-Limiting for Login & OTP Requests

To protect against brute-force attacks and excessive OTP requests, the system currently limits the number of login attempts and OTP requests. However, it can be enhanced further by adding rate-limiting middleware such as express-rate-limit. This would limit the number of requests a user can make within a specified time window, thereby reducing the likelihood of successful brute-force or denial-of-service (DoS) attacks.

Benefits:

- Brute-Force Protection: Rate-limiting ensures that attackers cannot flood the login or OTP request endpoints with automated requests.
- Better User Experience: Users who make legitimate requests will experience fewer delays as the system efficiently handles requests.

- Prevention of Abuse: Rate-limiting prevents the system from being overloaded with requests, ensuring stability and availability.

9. Appendices

9.1. Glossary

This glossary provides definitions for key terms and concepts used in the document.

- Authentication: The process of verifying the identity of a user by checking their credentials (e.g., username and password).
- MFA (Multi-Factor Authentication): A security process that requires two or more verification methods from independent categories of credentials, such as something the user knows (password) and something the user has (OTP).
- OTP (One-Time Password): A password that is valid for only one login session or transaction, typically used as part of MFA.
- OAuth2: An open standard for authorization that allows third-party services to exchange tokens on behalf of the user without sharing their credentials.
- Rate Limiting: A technique used to control the amount of incoming traffic to a server by limiting the number of requests a user can make in a given time period.
- Session: A temporary connection between a user and a server that allows the user to perform multiple actions without re-authenticating.
- TOTP (Time-based One-Time Password): A type of one-time password that is generated based on the current time and is used in Two-Factor Authentication (2FA) systems.
- SQL Injection: A code injection technique that exploits a vulnerability in a web application's software by manipulating SQL queries to execute arbitrary SQL code.

9.2. Acronyms and Abbreviations

The following is a list of acronyms and abbreviations used throughout the document.

- API – Application Programming Interface
- CRUD – Create, Read, Update, Delete
- DB – Database

- EJS – Embedded JavaScript
- HTTPS – Hypertext Transfer Protocol Secure
- MFA – Multi-Factor Authentication
- OTP – One-Time Password
- OAuth2 – Open Authorization 2.0
- SQL – Structured Query Language
- TOTP – Time-based One-Time Password
- TLS – Transport Layer Security

9.3. References

This section includes references to standards, documentation, and other resources used to create the system.

1. **Express.js Documentation** – Official documentation for the Express web framework used in the backend. Available at: https://expressjs.com
2. **Passport.js Documentation** – Official documentation for Passport.js, the authentication middleware used in the system. Available at: http://www.passportjs.org
3. **Google OAuth 2.0 Documentation** – Official documentation for integrating OAuth 2.0 with Google services. Available at: https://developers.google.com/identity/protocols/oauth2
4. **bcrypt.js Documentation** – Official documentation for bcrypt.js, a library used for password hashing. Available at: https://www.npmjs.com/package/bcryptjs
5. **SQLite Documentation** – Official documentation for SQLite, the database management system used for data storage. Available at: https://www.sqlite.org/docs.html
6. **OWASP Top 10 Security Risks** – A list of the top 10 security risks for web applications, provided by the Open Web Application Security Project (OWASP). Available at: https://owasp.org/www-project-top-ten/