**COMPUTER SYSTEMS SECURITY**

|  |  |
| --- | --- |
| TITLE | PERSONAL ASSIGNMENT (PART 1) |
| STUDENT NAME | PEDRO FABIAN OWONO ONDO MANGUE |

**Table of Contents**

[1. Functions And Importance of Security Features 3](#_Toc190428114)

[Authentication and Authorization 3](#_Toc190428115)

[Data Encryption 4](#_Toc190428116)

[Input Validation 5](#_Toc190428117)

[Captcha 5](#_Toc190428118)

[OTP (One-Time Password) 6](#_Toc190428119)

[2. Discussion of Algorithm or Methods 6](#_Toc190428120)

[Authentication and Authorization 6](#_Toc190428121)

[Data Encryption 9](#_Toc190428122)

[Input Validation 9](#_Toc190428123)

[Captcha 11](#_Toc190428124)

[OTP (One-Time Password) 13](#_Toc190428125)

[3. Implementation and Critical Analysis 15](#_Toc190428126)

[Authentication and Authorization 15](#_Toc190428127)

[Data Encryption 15](#_Toc190428128)

[Input Validation 16](#_Toc190428129)

[Captcha 17](#_Toc190428130)

[OTP (One-Time Password) 18](#_Toc190428131)

[4. References 20](#_Toc190428132)

**List of Figures**

[Figure 1: Admin Login Form 6](#_Toc190302152)

[Figure 2: Merchant Login Form 6](#_Toc190302153)

[Figure 3: Login View Algorithm 7](#_Toc190302154)

[Figure 4: Merchant Login View Algorithm 8](#_Toc190302155)

[Figure 5: User Role 8](#_Toc190302156)

[Figure 6: Idea Concept Authentication and Authorization 8](#_Toc190302157)

[Figure 7: Secret Keys for Encryption 9](#_Toc190302158)

[Figure 8: Encryption and Decryption Methods 9](#_Toc190302159)

[Figure 9: Clean Password for Input Validation 10](#_Toc190302160)

[Figure 10: Clean Email for Input Validation 10](#_Toc190302161)

[Figure 11: Clean Mobile Number 10](#_Toc190302162)

[Figure 12: Parameters Captcha 11](#_Toc190302163)

[Figure 13: Captcha Login Form 11](#_Toc190302164)

[Figure 14: Customer User Creation with Captcha 12](#_Toc190302165)

[Figure 15: Merchant Login View with Captcha 12](#_Toc190302166)

[Figure 16: EmailOTP Model 13](#_Toc190302167)

[Figure 17: Send OTP and Generate OTP 14](#_Toc190302168)

[Figure 18: Verify OTP 14](#_Toc190302169)

[Figure 19: Implementation of Authorization and Authentication of Users 15](#_Toc190302170)

[Figure 20: Order List with Data Encryption 16](#_Toc190302171)

[Figure 21: Encryption Data for Phone Number and Address 16](#_Toc190302172)

[Figure 22: Failed Login 17](#_Toc190302173)

[Figure 23: Too Many Tries 17](#_Toc190302174)

[Figure 24: Captcha from Login Page 18](#_Toc190302175)

[Figure 25: Forgot Password 19](#_Toc190302176)

[Figure 26: OTP Code 19](#_Toc190302177)

[Figure 27: OTP Confirmation 19](#_Toc190302178)

# Functions And Importance of Security Features

## Authentication and Authorization

Security is one of the key aspects for any system and when users access different types of data and service via application, it becomes mandatory to make Authentication and Authorization as major pillars of the application.

* Authentication is when the identity of a user is verified. This way, it makes sure that the system can be accessed only by verified users.
* Authorization occurs after authentication. It specifies what resources and actions can be accessed or performed by the authenticated user. So, while an admin can manage all users and orders, a customer can only see their own orders.

Simply stated, when you are trying to get to a website, there is almost always a point in time when you need to "log in." when you are verifying who you are (authentication). Once logged in, what you can see and do is permitted by your role (authorization). Without proper controls, unauthorized users could access sensitive areas of the system, a potential path for data breaches. Imagine a concert. The entrance is where you present your ticket to indicate that you are authorized to be there (authentication). However, backstage access is only granted to VIP ticket holders (authorization).

**Function:** Authentication helps to verify the identity of the user, whereas authorization grants access to specific resources and functionalities.

**Importance:**

* Avoids Unauthorized Access: If not implemented, anyone would have access to critical sections of the application.
* Secures Sensitive Data: Only trusted users can access sensitive data such as user profiles, orders, and payments
* Boosts User Trust: Users can feel safe with their data being secured.
* Role Management Support: Different users have different requirements. An admin handles the whole platform, while a customer needs to see their own orders only. And that is where role management comes into play.

## Data Encryption

Encryption is a security process that converts data into a jumbled and unreadable form (ciphertext). This means that even if someone who is not authorized gains access to the data, they do not have the ability to interpret or utilize that data unless they also have the decryption key to revert the data to its original state.

Encryption is a very important mechanism when it comes to protecting sensitive information such as passwords, payment information, and other personal user data contained in your application. To put it simply, encryption can be compared to putting your valuable data in a safe, which can only be opened by authorized users with the correct key.

Encryption is the process of using an algorithm to encode data. Most of the encryptions that we work on today fall under one of the two umbrella types:

* **Symmetric encryption:** One key is used to encrypt and decrypt.
* **Asymmetric encryption:** Two keys (one public and one private) are used, where one of the keys encrypt the data, and the other one decrypts it.

In my application, a symmetric encryption has been implemented with the Fernet cipher from the cryptography library. Fernet makes sure encrypted data is authenticated and secure, so the system knows when the data is tampered with.

**Function**: Encryption supports sensitive information by encoding it to prevent unauthorized access. This is especially important when dealing with sensitive information such as user credentials, financial transactions, and personal information.

**Importance**:

* Protects Data if Breached: Even if an intruder is able to access the database, any encrypted data will be unintelligible to them without a valid decryption key.
* Prohibit Data Leakage: Encryption plays a pivotal role in reducing data leak as well as insider threat related risks.
* Compliant: Many regulations on data protection require sensitive data to be encrypted, e.g. GDPR (General Data Protection Regulation), etc.
* Builds Trust: Users will trust an application that concerns their data security.

## Input Validation

Input validation consists of making sure the data that an user provides, fulfils the expected knowledge of the application before processing or storing it. For instance, when the user fills a form on the website, he must verify whether the email address is in the correct format, whether the password and confirm passwords are same, whether the fields like phone number, OTP code is valid.

This is the most important step to follow to avoid the system accepting malicious input that can be used to compromise the application. Input validation helps mitigate several common security attacks, including:

* SQL Injection: An attacker inputs SQL commands to make changes to or access the database.
* Cross Site Scripting (XSS): Attack scripts are added to web pages which will be seen by other users.
* Brute Force Attacks: Automated tries to guess user credentials.

Input validation in a Django web application usually takes place through forms and models. It is important for us to validate the data to verify the user entry quality. This means that, instead of taking the action, the application returns the error message when input does not pass the validation.

A password confirmation form may verify that the two entered passwords match, or whether an email from an email form already exists in a certain database, as an example.

**Function:** Input validation prevents bad data from entering the system. It helps verify that:

* The data format is correct (i.e., phone numbers start with a + and consist of digits only).
* Data constraints are observed (e.g., email addresses are unique).
* It reduces security risks such as injection attacks.

**Importance:**

* Protects from Security Attacks: Prevents SQL injection, XSS, and other input-based attacks
* Monitors and Ensures Data Integrity: Only high-quality, structured data gets to the database; this minimizes errors and data inconsistencies.
* Users Get Immediate Feedback: Whenever users enter data that doesn't match the system standards, they get instant feedback.

## Captcha

CAPTCHA is an acronym that stands for "Completely Automated Public Turing test to tell Computers and Humans Apart." It is a security feature to ensure that automated programs (bots) do not carry out such actions as submitting forms, logging in, or registering fake accounts. A CAPTCHA typically involves a simple task like picking some images, typing out distorted text, or doing basic math. These tasks are trivial for humans but very hard for automated bots to solve.

In response to a user submitting a form, the CAPTCHA field verifies whether the user can complete the challenge. A form submission is refused if the CAPTCHA is not solved correctly. This helps you to avoid spam, brute-force attacks, and other machine-generated attempts on your application. And bots typically try to guess usernames and passwords on login forms by attempting the same action repeatedly (brute-force attacks). Requiring a CAPTCHA prevents automated attempts because the bot cannot solve the challenge.

**Function**: A CAPTCHA is a challenge-response test, used to ensure that the response is not generated by a computer. It’s particularly helpful for any login, registration, and password reset forms.

**Importance:**

* Prevents Automated Attack: CAPTCHA is used to protect against brute-force login attempts and spam registrations in websites.
* Enhances System Security: CAPTCHA improves system security by filtering out bots from authentication or other processes requiring human interaction.
* Less load on server: Automated bots keep sending repeated requests to your server; CAPTCHA mitigation prevents those requests from being processed.

## OTP (One-Time Password)

OTP (One Time Password) It is a security measure that aims to add an extra layer of security to critical operations such as password reset, account verification or multi-factor authentication. An OTP is a one-time use password that is only valid for a short period of time unlike a regular password.

Here's how it works:

* An OTP is generated and sent to the user's email or phone when a user attempts a sensitive action, such as a password reset.
* The user should enter the valid OTP before the expiry time to complete the action.
* Access is denied if the OTP is invalid or expired.

This helps in securing the system whereby even if, the user password is compromised, still no unauthorized access is granted without the OTP.

**Function:** The OTP mechanism requires a temporary, one-time code to protect actions such as password resets.

**Importance:**

* Prevents All Attacks: Attackers cannot change the passwords without users OTP.
* Improves security: A second verification method in addition to password
* Reduces Attack Window: OTPs expire relatively quickly, thus giving any attacker a very short time frame to use them.
* Increases User Trust: It gives peace to users that sensitive operations demand extra security.

# Discussion of Algorithm or Methods

## Authentication and Authorization

In the ecommerce Django application, when a user goes to the login page and enters his username and password to log in, the authentication process starts. For added security, Merchants can only login via their own Merchant Login page, which adds an additional CAPTCHA verification. Django then takes the username and password from the form and checks them against records in the user’s database. If yes, then it creates a session and authenticates the user if valid. But if wrong credentials, error message says that and lets user try again. When authenticated, the system plugs in with role-based redirection for admin-to-admin redirects, merchants to merchants redirects and customers to customers redirects. This allows for both secure authentication and easy access to what the user needs based on their role.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 1: Admin Login Form

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 2: Merchant Login Form

As you can observed in both figures 1 and 2, there is a potion of code from the login HTML where users such as admin and merchant have a link that directs them into their respective login forms. Basically, the three main users which are customers, merchants, and admin have different login forms which is a validation for authentication and authorization because it would have been a vulnerability if all users can access from the same login form that is why, the idea of different login forms has been created to strengthen the security in the application based on the user.

A computer screen shot of text

AI-generated content may be incorrect.

Figure 3: Login View Algorithm

In the Django web application example, the login\_view function illustrates how active sessions and authorization are managed via role-based access control and secure web form handling. All previous failed login attempts are removed upon visiting the login page to allow one last fresh attempt at logging in. When first performing a POST request (that is when a login form is submitted), the Django authentication system attempts to authenticate the form via CustomLoginForm. Then it checks the existence of the user and if the credentials are correct. After a successful authentication, the function checks the user role from figure 5. This route is gated for logging in for only users with the customer role. It should not allow a non-customer (admin, merchant) to sign in; instead, it should deny the sign-in and return an error message to prevent unauthorized access to the customer resource. If either the credentials are invalid or the input to a CAPTCHA is wrong, then the function sets a failed\_login flag in the session and redirects the user to the same page to enter their credentials again. Notice how this implementation demonstrates the double security; authentication (proving your identity) followed by authorization (declaring what you should be able to get access to).

In the next figure 4, you can appreciate a different login view for the merchant users from the ecommerce application:

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 4: Merchant Login View Algorithm

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 5: User Role

The Pedro Shop platform implements authentication and authorization using Django's built-in tools, enhanced with custom logic for role-based access control. Here’s a quick recap:

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6: Idea Concept Authentication and Authorization

## Data Encryption

Security is of immense importance whenever dealing with sensitive user data such as addresses, phone numbers, or passwords, that may be collected from Django web application. Encryption keeps unauthorized users from interpreting raw data, even if the database loses and gets hacked. It uses the Fernet encryption algorithm from the cryptography library, which performs symmetric encryption, or encrypts and decrypts using the same key.

For encrypting and decrypting we make an Encryption helper class. The user information is saved in the database is encrypted. Also, even when it is retrieved, it is always decrypted before showing to a permissible user.

**Generating an Encryption Key:**

An encryption key is a secret key used to encrypt and decrypt data. It should be kept safe to restrict unauthorized access.

A black background with white text

AI-generated content may be incorrect.

Figure 7: Secret Keys for Encryption

An Encryption Helper class is implemented as well as shown in the next figure 8, which implements two methods and exposes security functions to the rest of the application to encrypt and decrypt data. The encrypt method takes plaintext and returns it in a secure, encoded format, obscuring sensitive information from prying eyes. The decrypt method, on the other hand, possesses the capability to convert encrypted data back into its readable format, making it possible for authorized users to access information as required. Through this encryption-decryption system, the data is secured, and sensitive information is not exposed by an unauthorized entity that needs the information for rightful use.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 8: Encryption and Decryption Methods

## Input Validation

Input validation is an important security measure used to ensure that user-provided data meets the required criteria before processing or storage in a database. This mechanism safeguards against invalid, malformed, or malicious data, thereby protecting against security vulnerabilities.

**Common Security Threats Prevented by Input Validation**

Various security threats exist that can put a web application's integrity into question, such as: SQL Injection, Cross-Site Scripting (XSS), and Brute Force Attacks. SQL Injection: Attackers insert malicious SQL queries into the input fields of a web application, allowing them to manipulate database operations and potentially expose, modify, or delete sensitive data. This includes injection of malicious scripts in webpages, which can be executed in other users’ browsers and steal data or hijack account. Brute Force – These attacks involve the use of bots repeatedly entering user credentials. The Application Code Vulnerability DoS (Denial of Service) attacks being a prime example of an issue that can be eliminated or significantly reduced by implementing input validation.

**Algorithm and Methods Used for Input Validation**

Django provides built-in form validation through forms.Form and forms.ModelForm. Custom validation logic is implemented using clean methods, which allow checking and modifying user input before processing.

**Password Validation**

This is the function which make sure the user enters the same password twice during registering or resetting the password If they do not match, an error is shown.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 9: Clean Password for Input Validation

Verity whether both password fields in a submitted form match, thereby making sure that users set the password they intend to. It first takes the passwords entered and verifies if both fields are entered. An error is raised if passwords mismatch, and the system asks the user to re-type them. If they match, the password is accepted. This ensures that a user does not unknowingly create an invalid password, which can lead to problems down the line when they try to recover their account, not to mention it helps to increase security overall!

**Email Validation**

This prevents them from creating the same email twice and thus avoid scams to create unique accounts. The logic works by fetching the email input from the registration form and verifying that an existing user had already registered with that email. It then checks at least one matching error message is displayed, to prevent adding multiple accounts. If there is no account associated with the email, it is permitted, and they can continue to register. As this validation is essential for preserving data integrity, preventing unpredictable behaviour which could potentially break your application, and providing a secure authentication process.

A computer code with colorful text

AI-generated content may be incorrect.

Figure 10: Clean Email for Input Validation

**Forgot Password**

The Forgot password form uses input validation to ensure users enter a valid mobile number to proceed with password recovery. Given this is to be an international service, this checks that the phone number entered is in the correct format for international delivery and users are not entering in the wrong type or incorrectly formatted phone numbers. This click-to-confirm operation validates the email address or the phone number of the user, which allows only legitimate contacts to be available for the system and provides fewer errors for the user and better security for their accounts. This procedure aids in simplifying password recovery, guaranteeing users can consistently re-access their accounts in the event of an emergency.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 11: Clean Mobile Number

Pedro Shop implements input validation as a fundamental component of its security architecture, verifying that inputs are in a valid, safe, and properly formatted state before being processed by the platform. Django's built-in form validation and robust custom clean methods are employed in the platform to create strong validation rules, mitigating risks from malicious inputs such as SQL injection, cross-site scripting (XSS) and brute-force attacks. As an example, the password validation will allow users to enter new passwords but will compare them to ensure they match, preventing someone from entering repeated passwords accidentally or forgetting, thus securing their account. Thus, it helps the designer in avoiding duplication of the records and reducing fraudulent attempts by preventing the user from entering the same email id again. Moreover, while testing we were introduced to the international E.164 format and hence mobile number validation ensures that the phone number is well formatted for OTP based authentication and notifications. The platform achieves data integrity, user account protection and a smoother user experience through proactive compliance with provisions of these legislative and regulatory requirements. For instance, when users input invalid data, such as a wrong password or a phone number in the wrong format, they receive immediate feedback, and such incorrect entries are not allowed through for processing. By doing this, it not only protects the system from security issues but also earns user trust by showing responsibility in maintaining data accuracy and safety. Input validation, therefore, is a key component of the platform's security model, preventing bad data from entering and iteratively corrupting the platform.

## Captcha

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a security mechanism used to differentiate between humans and bots, to prevent automated bot activities like spam, brute force attacks, and the creation of fake accounts. It poses problems that humans readily solve but bots find challenging, like typing distorted text from an image, answering a simple math question or clicking on particular images from a group. CAPTCHA test protects the online platforms from unauthorized access and fraudulent activities as automated scripts cannot bypass these tests.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 12: Parameters Captcha

In Django web application, we must use some settings that can define the complexity and appearance of the CAPTCHA which will help to keep it more secure against the automated bots. From figure 12, settings include font size, character limit, rotation angle, background and foreground colours, and noise functions that put distortions to help the CAPTCHA bypassing be more complicated. This is called adjusting these parameters, so we make sure that the human user could solve it with ease, whereas the automated script will find it very hard to decipher. This implement ensures only authenticated user can make changes and access protected resources, reducing the chance of spam and other malicious behaviour.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 13: Captcha Login Form

Adding CAPTCHA to forms increases security since it prevents automated bot submissions in sensitive areas like login and user registration forms. This is done with the use of the Captcha Field which won't successfully validate until the form generates CAPTCHA. This helps protect against abusive behaviour such as spamming, brute-force attacks or fake account creations by only allowing legitimate users through after answering a CAPTCHA challenge. This add-on also ensures better authentication processes and maintains the integrity of user data in the application.

There was implementation of captcha security in various places throughout the ecommerce app on merchant and on the user creation or registration as you can see on the following figures.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 14: Customer User Creation with Captcha

Form with CAPTCHA validation during user registration in CustomUserCreationForm in Figure 14 In the above form, only users with the role of "customer" can sign up and it contains a CaptchaField, which helps actually ensuring a human user is filling in the registration. If the CAPTCHA input is incorrect, the field itself provides custom error messages. In case of a failure to input the required value, the users are also presented with an error message saying they have to try again while the CAPTCHA remains uncompleted. This creates barriers in the way of automated bot registrations, making the process of creating an account much more secure.

A computer screen shot of text

AI-generated content may be incorrect.

Figure 15: Merchant Login View with Captcha

If the user is of "merchant" role, the authentication process will be handled by as stated in merchant\_login\_view function as shown in Figure 15. In the above function, first it checks if the user is already logged in before granting access to the merchant dashboard. In the case of a login, the form verifies the username and password, then makes sure it's a merchant and allows access, in addition, if the form has a CAPTCHA validation and it doesn't pass, it shows an error preventing it from brute-forcing the login by automated bots. This login view enhances its security against bot-driven login attempts by using CAPTCH verification to avoid brute force attacks and limit unauthorized access.

## OTP (One-Time Password)

OTP is a Security Mechanism to protect the User from Password Reset, 2FA, Account Verification. An OTP is unique and is generated for a single real-time session compared to traditional passwords that are valid for longer duration. Hence, even if an attacker gained access to a user’s credentials, he still could not log in or reset the password without access to the OTP, making it an ideal layer of defence against cyber threats like phishing or credential theft.

A well-structured process of implementation of OTP authentication is depicted here. Initially, a random 6-digit OTP is created and stored in the database associated with the user's email id. The OTP is sent to the user through Django’s email and secure way. The system checks this OTP to finalize authentication, meaning the user must enter the code correctly before the set expiration time. The OTP expires after 10 minutes, so that it cannot be reused or used by an unauthorized access. All of these things create a reliable and timely authentication mechanism to keep your user secure.

**Algorithm and Methods Used for OTP**

This EmailOTP model from figure 16 helps us to store OTP codes sent to users in a secure manner and in an organized manner. With OTP, a single-use password that is valid for that operation and for a specific email address is linked to it, which allows the system to verify the user's identity in case of sensitive operations. The timestamp indicates when the OTP was created, allowing it to be validated based on the time. Also, this model has an is\_expired() which automatically checks whether the OTPCODE has expired, thus preventing the expiry from being used. This way, OTPs would stay secure, time-sensitive, and protected from unauthorized access attempts.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 16: EmailOTP Model

In the following where when a user requests a password reset, a random 6-digit OTP is generated to confirm the user identity. This OTP will be stored safely in the database. After generating the OTP, the system sends it through email so that only the respective owner of that email account can see it. That means if some hacker gets your account info, he won't be able to change or access the account till he has your registered mail, because the OTP will be sent there. The method of delivery being email for example gives the system a mechanism to check who is accessing it and preventing robbery.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 17: Send OTP and Generate OTP

Then, once when a user provides the OTP for verification, the OTP registered to that email in the database is fetched by the system. Afterwards, it verifies that the OTP is correct and not expired, because only active codes must be used to authenticate. Practically What happens if the OTP is Valid Time-Sensitive and Accurate? The user can then securely allow the password to reset. The OTP is marked as used, this prevents it from being reused to elicit a successful authentication. Following this process allows user accounts to be protected from unauthorized access, while also keeping a secure password recovery system in place.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 18: Verify OTP

# Implementation and Critical Analysis

## Authentication and Authorization

Pedro Shop e-commerce site handles the authentication and authorisation depending on role level because it’s the main building block of secure user access. These security features guarantee that users gain access only to resources appropriate to their roles, while restricting other users from entering sensitive portions of the system.

A screenshot of a login page

AI-generated content may be incorrect. A screenshot of a login screen

AI-generated content may be incorrect.A screenshot of a login form

AI-generated content may be incorrect.

Figure 19: Implementation of Authorization and Authentication of Users

In order to improve authentication, Role-Based Access Control (RBAC) is established to specifically ensure that only customers can access the Customer Login Page( The image at very left) Retrieving the user's credentials: When a user tries logging into the system, the system first validates the user using the username and password. Upon which if credentials are verified and user is in a customer role access is permitted. Otherwise, the merchants or admins are redirected to their login pages. By isolating these roles, we can avoid privilege escalation which keeps customer data locked away from other user roles. Likewise, the Merchant Login Page (the image in the middle) implements RBAC by allowing access only to users that hold the merchant role. In the event that a non-merchant user, like a customer or admin accesses the system, the system doesn't allow that user to log in. After successful authentication, an authenticated user is redirected to their merchant dashboard, where they can manage products, sales analytics, and transactions. Enforcing these roles tightly guarantees that customers cannot "mess up" the business and that merchants cannot access areas reserved for admin users. Finally, In Django Admin Panel (the rightest side image), RBAC is implemented at the top level with the access available only to superusers. Django includes an advanced user authentication system that handles groups, permissions, and accounts; as such, admins must log in through the application before they can access those features. Also, it prevents lower-privileged users from modifying settings that affect the platform, such as altering other user roles or changing system settings, as RBAC only gives this type of access to superusers and admin. Such restriction of access secures important functions of system administration and ensures proper functioning of the platform.

## Data Encryption

For example, storing sensitive customer information is done using data encryption, which is one of the most critical security implementations in the Pedro Shop e-commerce platform. The following two images are all the evidence you need of data encryption at work in real-time, letting you know for sure no sensitive fields (address, phone, etc.) are stored in plain text in the admin dashboard as well as the Django database panel.

The page show in Figure 20 is the Order List from the Django admin site. This page includes Order ID (a unique identifier for each transaction), Customer Name, Total Price, Order Status, and administrative Actions (viewing, editing, and deleting an order). The "Address" and "Phone" columns are notably left blank to avoid exposing sensitive user data without authorization. This security step prevents personally identifiable information (PII) from being seen in plain text, which is an obligation under data protection laws, for example, GDPR and CCPA. This data can only be accessed by authorized users with decryption permissions, thus minimizing the chance of leaks. This mechanism enhances security for sensitive information, ensuring that unauthorized access is blocked, customer privacy is preserved, and data breaches are mitigated, even if an admin account is compromised.

A screen shot of a phone number

AI-generated content may be incorrect.

Figure 20: Order List with Data Encryption

In figure 21 you can see the raw encrypted values of the customer Address and Phone Number in Django admin panel database. So rather than showing plaintext data, the system encrypts these sensitive fields into ciphertext, and only users with the correct decryption key can retrieve the original values. The encryption process results in an encrypted string that is lengthy, and its contents appear random, rendering any attempt to guess or reverse-engineer impossible without access to the appropriate cryptographic key.

As shown in figure 21, the system uses Fernet encryption from the cryptography library, with AES based symmetric encryption to secure customer data. Moreover, it uses Base64 encoding to save the encrypted data in a database-friendly manner. Decryption, which requires an appropriately stored encryption key, serves as preventive action against unauthorized access, and is thus the backbone of this security model, relying heavily on key management.

This significantly improves security by encrypting sensitive data. In case the database is hacked, attackers are not able to read the actual address or phone number without the key to decrypt. It safeguards customer privacy and minimizes the chances of identity theft, and makes sure compliance with data protection laws, further strengthening the security framework of the platform.

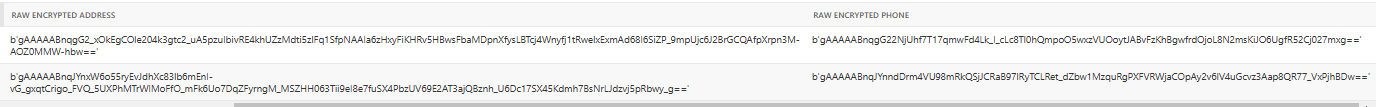


Figure 21: Encryption Data for Phone Number and Address

## Input Validation

It is simply a process of ensuring that only well-formed data gets to pass to the application. This helps prevent malicious attacks such as SQL Injection, Cross-Site Scripting (XSS), and brute-force attempts. In the Pedro Shop e-commerce platform, input validation is applied at the various levels like user login, registration forms, and forms submissions.

A screenshot of a login form

AI-generated content may be incorrect.

Figure 22: Failed Login

The system shows a failed login attempt in figure 22, where the user CustomerUser01 has provided invalid credentials. The system prompts an error message instead, “Login failed. Fix any issues and try again.” This message indicates to the user there was a problem without revealing whether it was the username or password.

This validation error helps secure the API by preventing brute-force attacks and ensuring that only valid credentials can be authenticated. The absence of valid usernames helps to prevent attackers and automated bots from guessing passwords, as they will have no insight into potential usernames. Form validation before submission also helps to trap incorrect credentials, avoiding wasted processing time in the back end and reduces server load as it prevents accessing the server with invalid credentials.

A white background with black text

AI-generated content may be incorrect.

Figure 23: Too Many Tries

Another functionality that is implemented arising from input validation is that after three attempts if a user has entered incorrect login details, then the account will be locked out of as you can see from the figure 22 which logs a security mechanism that due to too many failed logins attempts and account has been locked. This is especially important in preventing brute forcing, since brute forcing works by trying to guess passwords repeatedly in a rapid manner. An account lockout enables the system to prevent further attempts after a set number of wrong entries, successfully shielding the account from repeated access by unauthorized parties. This mechanism supports security by preventing unauthorized access to user accounts, as well as credential stuffing attacks, a type of attack where a bad actor takes usernames and passwords stolen from one breach and tries them with other services.

## Captcha

The CAPTCHA solution helps prevent bots from automatically guessing credentials by allowing only human users to continue with a login attempt. This has ensured that error handling in the system is clear and sophisticated, as incorrect CAPTCHA entries prompt users to retry, allowing for an uninterrupted flow in an end-user experience. Integrating CAPTCHA into the login process for merchants that deal with sensitive data, like transactions and customer orders, helps reinforce security by adding another line of defence to prevent unauthorized access. In the user authentication flow, users must enter the CAPTCHA correctly before their credentials are verified, preventing automated bots from circumventing this security measure. So, practically speaking, when a trying-to-login merchant enters their username/password, they also have to fill in the right CAPTCHA code. Well, if the value of CAPTCHA input changes then the login request is rejected and user is prompted to try again. This process ensures that only intentions to validate one of the local access credentials come through to the other side, thus avoiding fixed threat automation, while further ensuring that being automated is not something a legitimate user will experience in this system while attempting to gain secure access to his account.

A screenshot of a login screen

AI-generated content may be incorrect.

Figure 24: Captcha from Login Page

The figure 24 shows the example when the merchant tries to logout from the login portal which works perfectly as the only time was invalid seconds with an error message ("CAPTCHA verification failed; this has been integrated in the merchant login portal. Please try again." It should aim to block bots that try to brute-check password combinations by requiring a CAPTCHA (which can only be solved by a human) when logging in.

## OTP (One-Time Password)

Make sure that figures are presented in a logical order to show each step of the OTP (One-Time Password) implementation and a very important part of analysis. This is the recommended organization:

**Requesting an OTP**

From the next figure below it can be seen a scenarion where the user or the customer forgot its password therefore this is a vulnerability for user, specifically because there is no way that user can proceed without going again through the registration and signing up on new account, which is unneccessary. But there is a button which pop up in the login form where the user can press the “forgot your password” that will direct it to a page that will require an email so that from there an OTP code will be sent.

A screenshot of a login screen

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 25: Forgot Password

**Receiving OTP via Email**

This proves that the system successfully generates and sends a unique, time-sensitive OTP to the registered email. The OTP delivery is a key security feature to prevent unauthorized access.

**A close up of a computer screen

AI-generated content may be incorrect.**

Figure 26: OTP Code

A screenshot of a login form

AI-generated content may be incorrect.

Figure 27: OTP Confirmation

Once a valid OTP and new password are provided, it updates the user’s credentials in a secure manner, ensuring a safe system recovery.

By utilizing OTP for password resets, the user needs to obtain the OTP in addition to their email and username, providing additional security to the process, as OTPs are usually time-sensitive, unique, and temporarily valid. The OTP is validated against records stored in the system, and expiration limits are enforced, making it very less likely that an unauthorized person can gain access. OTPs have a short validity period, preventing brute-force attacks, as attackers cannot control when an OTP will be used, and thus cannot have a time window in which they can attack. This also means that the OTP will be only received on the registered email address, therefore only the right account holder can reset his password. This process further reduces the chances of phishing risks, as even if a hacker withholds the login credentials through phishing, they still require access to the email account of the victim. The system is designed with user experience in mind as it provides real-time feedback on incorrect OTP inputs, minimizing user frustration during the reset process. The (implementation) also checks the system is responsive, (ensuring that OTPs are generated and delivered) will not get delayed. Finally, handling expired OTP is correctly tested, with the overarching goal of preventing expired OTP from being accepted, further enriching the securely of password reset.

# References

Alzomai, M. &. ((2013)). Authentication vulnerabilities in web applications: Attacks and defense mechanisms. *International Journal of Computer Science and Security*, 7(4), 155-167.

Bonneau, J. H. (2012). The quest to replace password: A framework for comparative evaluation of web authentication schemes. *IEE Symposium on Security and Privacy*, pp. 553-567.

Halfond, W. G. (2006). A classification of SQL-injection attacks and countermeasures. *IEEE International Symposium on Secure Software Engineering*, pp. 13-15.

Jovanovic, N. K. (2006). Preventing cross-site request forgery attacks. *IEE*, pp. 1-10.

Kim, D. C. ((2018)). A study on data encryption for secure database protection. *Journal of Information Security and Application*, 42, 40-50.

Kuhn, D. R. (2010). Introduction to role-based access control. National Institute of Standard and Technology (NIST). *Special Publication*, 800-95.

Makanju, A. B.-H. (2009). Log analysis for intrusion detection: A machine learning approach. *Journal of Information Assurance and Security*, 4(2), 141-150.

Mutchler, L. P. (2015). Security misconfiguration detection in web applications using machine learning. . *10th International Conference on Securiy and Criptography*, (pp. 423-432).