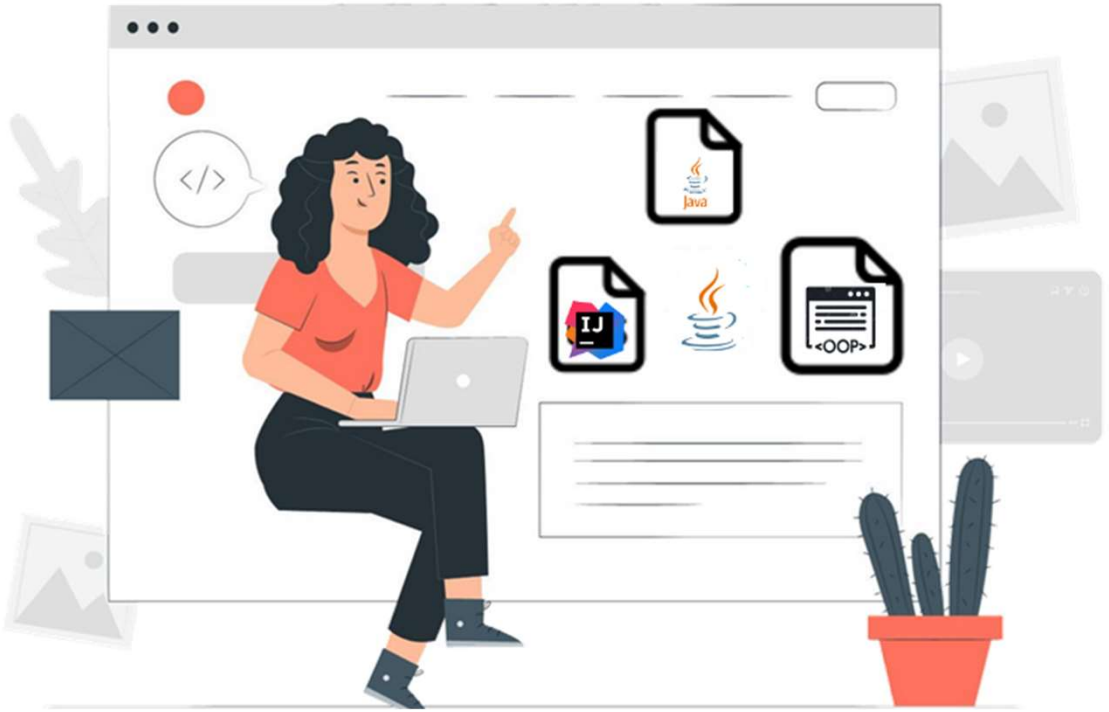


Domain Driven Design

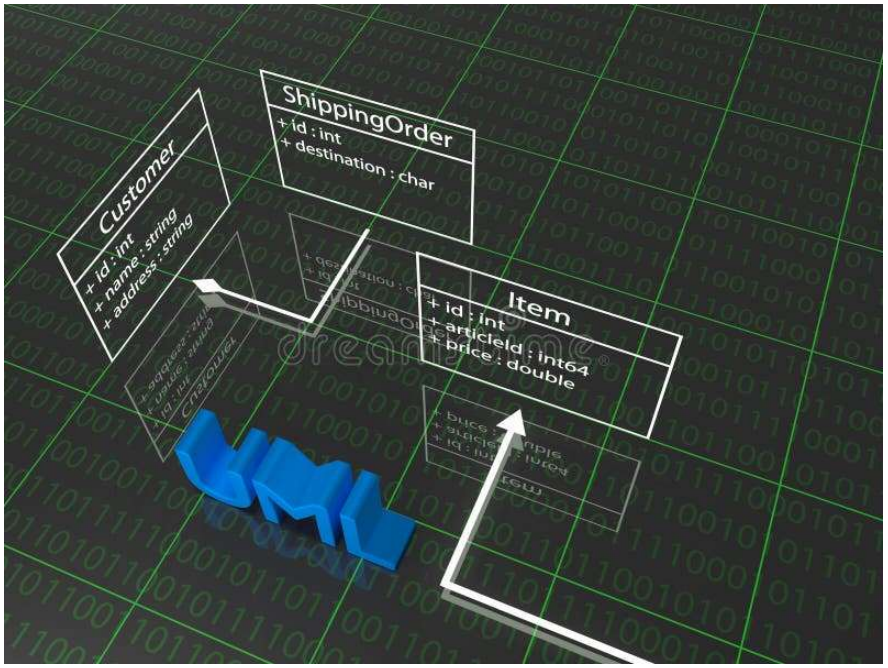
Eliane Marion

FIAP

2024

AGENDA DE TRABALHO

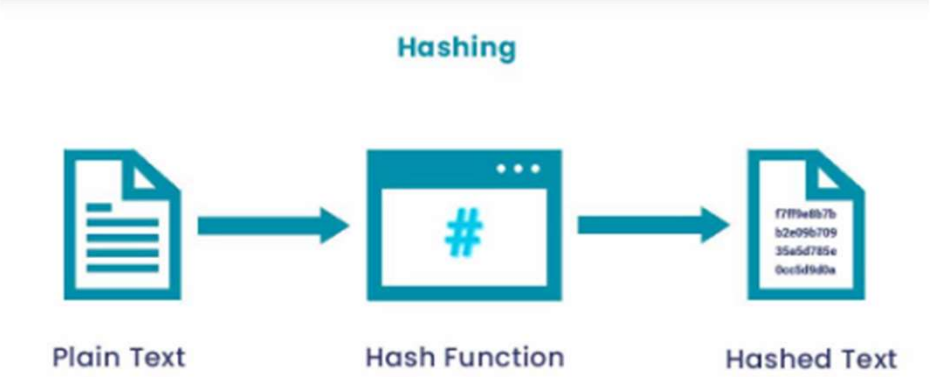
CRIPTOGRAFIA



01

CRIPTOGRAFIA DE SENHA

CRIPTOGRAFIA



É fundamental para garantir a segurança dos dados dos usuários, dificultando o acesso não autorizado a essas informações.

A melhor escolha é utilizar algoritmos de hash como bcrypt ou PBKDF2 que geram uma representação única e irreversível da senha, tornando a recuperação da senha original praticamente impossível.

Implementando criptografia no JAVA

Vamos utilizar o **bcrypt** um algoritmo de hash.

O método **BCrypt.gensalt(n)**: gera um salt (valor) aleatório gerado para cada senha, para evitar ataques de rainbow table.

O método **BCrypt.hashpw** combina a senha com o salt e gera o hash da senha.

O método **BCrypt.checkpw** compara a senha fornecida com o hash armazenado e retorna **true** se a senha estiver correta e **false** caso contrário.

Implementando criptografia no JAVA

```java
public static String hashPassword(String password) {

    String generatedSalt = BCrypt.gensalt(10);

    String hashedPassword = BCrypt.hashpw(password, generatedSalt);

    return hashedPassword;
}
```

```java
public static boolean checkPassword(String password, String
storedHash) {
    // Verifica se a senha fornecida corresponde ao hash armazenado
    return BCrypt.checkpw(password, storedHash);
}
```

Implementando criptografia no JAVA

Vamos utilizar o **PBKDF2WithHmacSHA512** um algoritmo de hash derivado de senha forte e amplamente utilizado.

Salt: Um valor aleatório gerado para cada senha, para evitar ataques de rainbow table.

PBKDF2WithHmacSHA512: Deriva uma chave de uma senha, usando o algoritmo HMAC-SHA512, um algoritmo criptográfico resistente a colisões.

Iterações: O número de iterações para tornar a derivação de chave mais lenta e aumentar a segurança contra ataques de força bruta.

Validação: Ao validar a senha, o hash da senha de entrada é gerado novamente com o mesmo salt e comparado com o hash armazenado.

Implementando criptografia no JAVA

```java
public class PasswordHasherPBKDF2WithHmacSHA512 {

    private static final int SALT_LENGTH = 16;
    private static final int ITERATIONS = 65536;
    private static final int KEY_LENGTH = 512;

    // Método para gerar um salt seguro
    public static byte[] generateSalt() {
        SecureRandom random = new SecureRandom();
        byte[] salt = new byte[SALT_LENGTH];
        random.nextBytes(salt);
        return salt;
    }
```

Implementando criptografia no JAVA

```
// Método para gerar o hash da senha
public static String hashPassword(String password, byte[] salt)
        throws NoSuchAlgorithmException, InvalidKeySpecException {

  KeySpec spec = new PBEKeySpec(password.toCharArray(), salt, ITERATIONS,
KEY_LENGTH);

    SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA512");
    byte[] hash = factory.generateSecret(spec).getEncoded();

     return Base64.getEncoder().encodeToString(hash);
}
```

Implementando criptografia no JAVA

```
// Método para verificar se a senha corresponde ao hash
public static boolean validatePassword(String password, String
storedHash, byte[] salt)
        throws NoSuchAlgorithmException, InvalidKeySpecException {

         String newHash = hashPassword(password, salt);

         return newHash.equals(storedHash);
}
```