May 03, 2023

PHP Password Manager Report

Titas Lukaitis
Vilnius Univeristy

Contents

1 UML Diagrams

Figure 1: *db model* of the ppm table
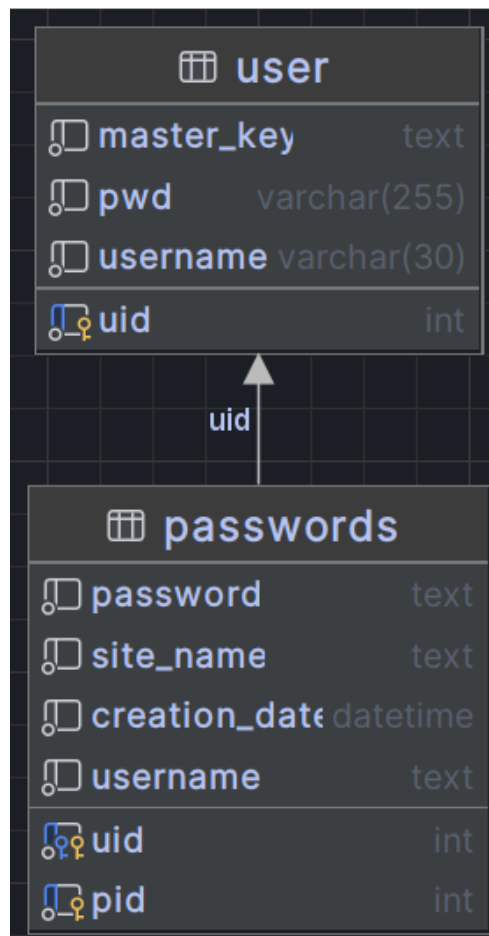
2 Responses

For more detailed code snippets please view the attached source code.

2.1 Index

```php
<?php
if (isset($_SESSION['uid'])) {
    echo '<h1 class="text-center">You are logged in! 👍</h1>';
} else {
    echo '<h1 class="text-center">You should log in or register to access the app!</h1>';
}
```
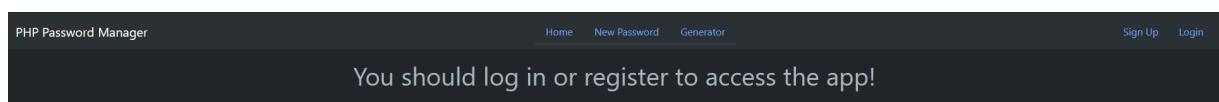
2.1.1 Logged Out

Figure 2: *index* when logged out
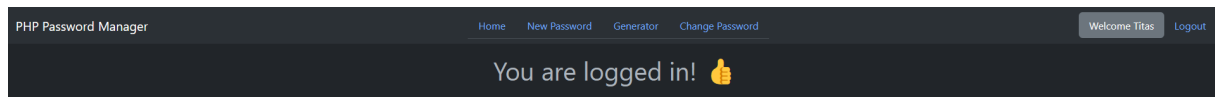
2.1.2 Logged In

Figure 3: *index* when logged in

2.2 Regstration

```php
<?php
if (isset($_POST['submit'])) {
    if (empty($_POST['username'])) {
        print_messages::printError("Empty Username");
    } elseif (empty($_POST['password'])) {
        print_messages::printError("Empty Password");
    } elseif (empty($_POST['repeatPassword'])) {
        print_messages::printError("Empty Repeated Password");
    } elseif ($repeatPassword !== $password) {
        print_messages::printError("Passwords do not match");
    } elseif (strlen($name) > 30) {
        print_messages::printError("Username too long");
    } elseif ($db_utils->usernamesExists($name)) {
        print_messages::printError("Username already exists");
    } else {
        register_user($password, $name);
    }
}


/**
 * Registers a new user in the database
 * @param string $password Their plain-text password
 * @param string $name Username
 * @return void
 */
function registerUser(string $password, string $name): void {
    // DB vars
    $password_hash = password_hash($password, PASSWORD_DEFAULT);
    $master_key_encode = PwdUtils::generateMasterKey();
    $master_key = PwdUtils::encryptMaster($master_key_encode, $password);

    // Database stuff
    try {
        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("INSERT INTO user (username, pwd, master_key) VALUES
(:name, :pwd, :master)");
        $stmt->bindParam(":name", $name);
        $stmt->bindParam(":pwd", $password_hash);
        $stmt->bindParam(":master", $master_key);

        // execute
        $stmt->execute();
        unset($stmt);
        unset($db);
    } catch (PDOException $e) {
        // Silently fail
        PrintMessages::printError("Database Error");
        die();
    }
```

```
        PrintMessages::printInfo("Registration Complete");
}
```

2.2.1 Missing Fields

Figure 4: *register* missing username

Figure 5: *register* missing password

Figure 6: *register* missing repeated password

2.2.2 Password and Repeated Password Don't Match

Inputs: Username: hello123, Password: 123, Repeated Password: 1234

Figure 7: *register* password and repeated password don't match

2.2.3 Username Length More Than 30 Characters

Inputs:

- Username: 1234567890_1234567890_1234567890
- Password: 123
- Repeated Password: 123

Figure 8: *register* Username exceeds 30 characters

2.2.4 Username Already Exists

Figure 9: *register* Existing usernames

Inputs:

- Username: 123
- Password: 123
- Repeated Password: 123

Figure 10: *register* Username exists

2.2.5 Successful Registration

Figure 11: *register* DB before registration

Figure 12: *register* Before registration
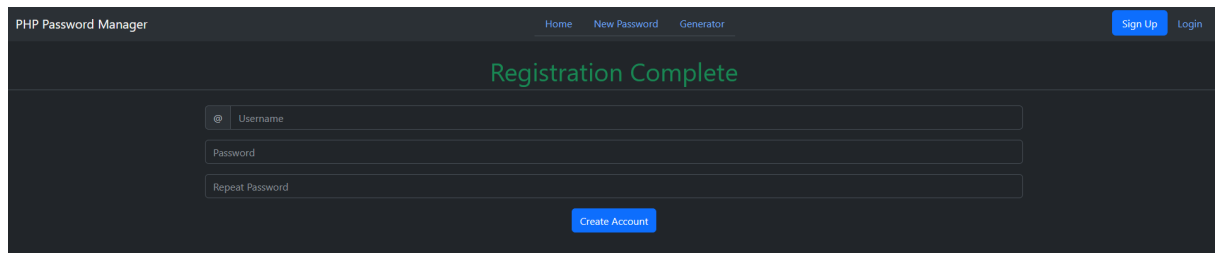
Figure 13: *register* After registration
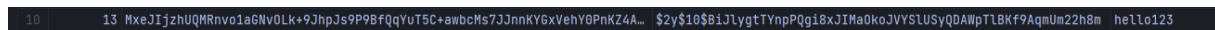
Figure 14: *register* New db entry

2.3 Login

```php
<?php
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PrintMessages.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/Database.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PwdUtils.php");

if (isset($_POST['submit'])) {
    if (empty($_POST['username'])) {
        PrintMessages::printError("Empty Username");
    } elseif (empty($_POST['password'])) {
        PrintMessages::printError("Empty Password");
    } else {
        $name = $_POST['username'];
        $password = $_POST['password'];
        loginUser($name, $password);
        header("Location: /app/home.php");
    }
}


/**
 * Logs the user in, and sets the session variables
 * @param string $username The username of the user
 * @param string $password Plain text password of the user
 * @return void
 */
function loginUser(string $username, string $password): void {
    try {
        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("SELECT * FROM user WHERE username = ? LIMIT 1;");
        $stmt->bindParam(1, $username);

        $stmt->execute();
        if ($stmt->rowCount() != 1) {
            PrintMessages::printError("Invalid Username Or Password");
            return;
        }

        $result = $stmt->fetch();

        $uid = $result['uid'];
        $master_key_result = $result['master_key'];
        $pwd = $result['pwd'];
```

7

```php
        $username = $result['username'];

        if (!password_verify($password, $pwd)) {
            PrintMessages::printError("Invalid Username Or Password");
            return;
        }

        // Apparently sessions are actually pretty secure already, though
        // session_regenerate_id() should be called after each login attempt
        session_regenerate_id();
        $master_key_decrypted = PwdUtils::decryptMaster($master_key_result,
$password);

        $_SESSION['uid'] = $uid;
        $_SESSION['master_key'] = $master_key_decrypted;
        $_SESSION['username'] = $username;

        PrintMessages::printInfo("Login Successful");
    } catch (PDOException $e) {
        // Silently fail
        PrintMessages::printError("Database Error");
        die();
    }

    unset($stmt);
    unset($db);
}
```
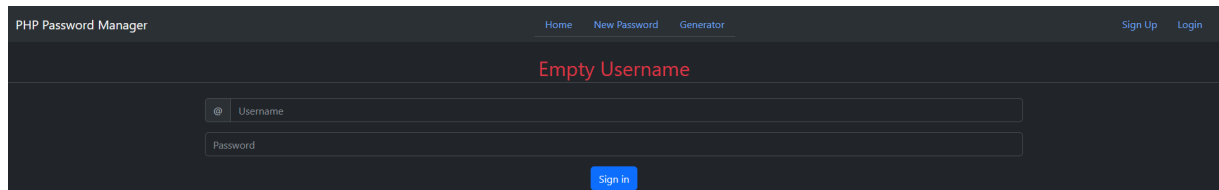
2.3.1 Missing Fields

Input: Nothing
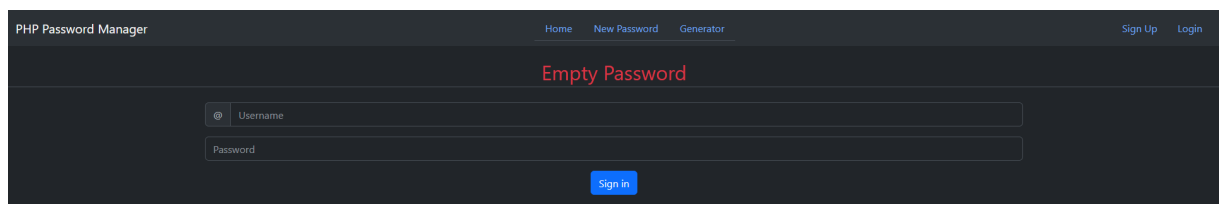
Figure 15: *login* Missing username

Input:

- Username: 123

Figure 16: *login* Missing password

2.3.2 Username Does Not Exist

Input:

- Username: DoesNotExist
- Password: 123
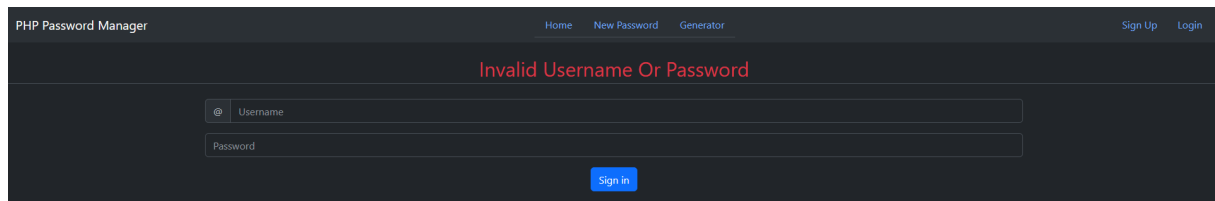
Figure 17: *login* Username does not exist

2.3.3 Invalid Password

Input:

- Username: Titas
- Password: Titas
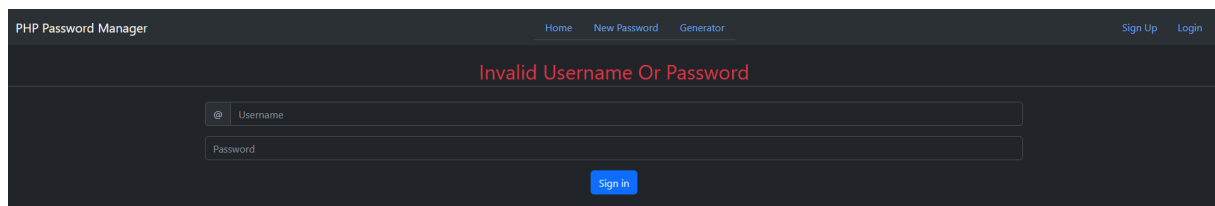
Figure 18: *login* Password is incorrect

2.3.4 Successful Login

Input:

- Username: Titas
- Password: Titas2003
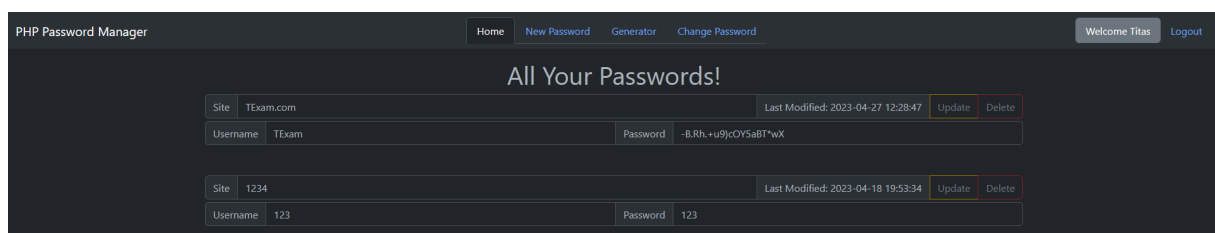
Figure 19: *login* Success!

You are automatically redirected to the Home page.

2.4 Logout

Before:
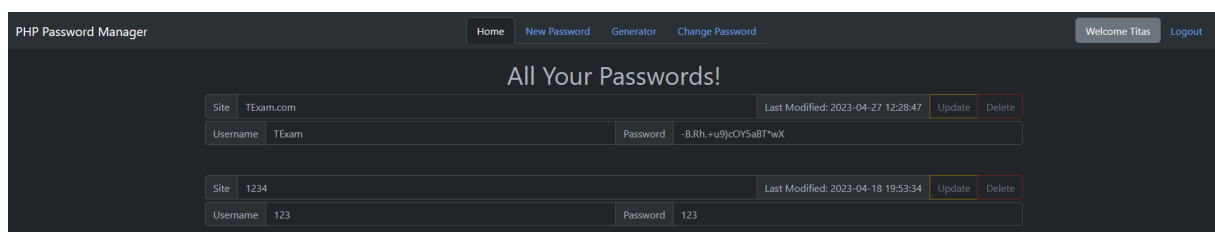
Figure 20: *logout* Before pressing button

After:
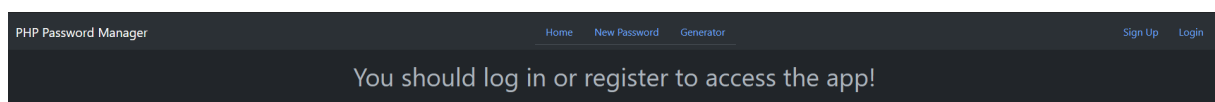
Figure 21: *logout* After pressing button

2.5 Password Generator

generator.php

```php
<?php
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PrintMessages.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/generator/PasswordGenerator.php");

$chars = 20;
if (isset($_POST['chars'])) {
    $chars = $_POST['chars'];
}

$specialChars = true;
if (!isset($_POST['specialChars'])) {
    $specialChars = false;
}

$numbers = true;
if (!isset($_POST['numbers'])) {
    $numbers = false;
}

$uppercase = true;
if (!isset($_POST['uppercase'])) {
    $uppercase = false;
}

$lowercase = true;
if (!isset($_POST['lowercase'])) {
    $lowercase = false;
}

if (isset($_POST['submit'])) {
    print ("<h2 class='text-center'>" . generatePassword($chars, $specialChars,
$numbers, $uppercase, $lowercase) . "</h2>");
}

/**
 * Generates a password according to the given parameters. Returns an error message
if the parameters are invalid.
 * @param int $chars A number between 1 and 128
 * @param bool $specialChars True if special characters should be included
 * @param bool $numbers True if numbers should be included
 * @param bool $uppercase True if uppercase letters should be included
 * @param bool $lowercase True if lowercase letters should be included
 * @return string The generated password
 */
function generatePassword(int $chars, bool $specialChars, bool $numbers, bool
$uppercase, bool $lowercase): string {
    $gen = new PasswordGenerator($chars, $numbers, $specialChars, $uppercase,
$lowercase);
    try {
        return $gen->generate();
    } catch (Exception $e) {
        return "<div class='text-danger'><p>" . $e->getMessage() . "</p></div>";
```

```php
    }
}
```

PasswordGenerator.php

```php
<?php
class PasswordGenerator {
    private int $passwordLength;
    private bool $hasUpperCase;
    private bool $hasLowerCase;
    private bool $hasSymbols;
    private bool $hasNumbers;

    public function __construct(int $passwordLength, bool $hasNumbers, bool
$hasSymbols, bool $hasUpperCase, bool $hasLowerCase) {
        $this->passwordLength = $passwordLength;
        $this->hasNumbers = $hasNumbers;
        $this->hasSymbols = $hasSymbols;
        $this->hasUpperCase = $hasUpperCase;
        $this->hasLowerCase = $hasLowerCase;
    }

    /**
     * Generates a password based on the previous parameters
     * @return string The password
     * @throws Exception
     */
    public function generate(): string {
        if ($this->passwordLength <= 0) {
            throw new Exception("Password length cannot be less than 1");
        } elseif ($this->passwordLength > 128) {
            throw new Exception("Password length cannot be more than 128");
        }

        if (!$this->hasLowerCase && !$this->hasUpperCase && !$this->hasSymbols && !
$this->hasNumbers) {
            throw new Exception("At least 1 option must be enabled");
        }

        $pwd = "";

        for ($i = 0; $i < $this->passwordLength;) {
            $switch = rand(0, 3);

            // lowercase character
            if (($switch == 0) && $this->hasLowerCase) {
                $pwd .= chr(rand(97, 122));
                $i++;
            }

            // uppercase character
            if (($switch == 1) && $this->hasUpperCase) {
                $pwd .= chr(rand(65, 90));
                $i++;
            }
```

```php
        // number
        if (($switch == 2) && $this->hasNumbers) {
            $pwd .= chr(rand(48, 57));
            $i++;
        }

        // symbol
        if (($switch == 3) && $this->hasSymbols) {
            $pwd .= chr(rand(37, 47));
            $i++;
        }
    }

    return $pwd;
    }
}
```

2.5.1 Invalid Char Length

Figure 22: *password generator* Length less than 0

Figure 23: *password generator* Length more than 128

2.5.2 No Options Selected

Figure 24: *password generator* No options selected with valid length

2.5.3 Options

Figure 25: *password generator* Only special characters

Figure 26: *password generator* Only numbers

Figure 27: *password generator* Only uppercase letters

Figure 28: *password generator* Only lowercase letters

Figure 29: *password generator* All options selected with a length of 30

2.6 New Password

```php
<?php
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PrintMessages.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/Database.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PwdUtils.php");

if (isset($_POST["submit"])) {
    if (empty($_POST["site"])) {
        PrintMessages::printError("Missing site name");
    } elseif (empty($_POST["username"])) {
        PrintMessages::printError("Missing username");
    } elseif (empty($_POST["password"])) {
        PrintMessages::printError("Missing password");
    } else {
        createNewPassword($_POST["site"], $_POST["username"], $_POST["password"]);
        unset($_POST["site"]);
        unset($_POST["username"]);
        unset($_POST["password"]);
    }
}

function createNewPassword(string $site, string $username, string $password): void {
    $current_time = date("Y-m-d H:i:s");
    $encrypted_password = PwdUtils::encryptPassword($password);
    try {
        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("INSERT INTO passwords (uid, password, username,
site_name, creation_date) VALUES (:uid, :pwd, :user, :site, :date)");
        $stmt->bindParam(':uid', $_SESSION['uid']);
        $stmt->bindParam(':pwd', $encrypted_password);
        $stmt->bindParam(':user', $username);
        $stmt->bindParam(':site', $site);
        $stmt->bindParam(':date', $current_time);

        $stmt->execute();

        PrintMessages::printInfo("Password created!");
        unset($stmt);
        unset($db);
    } catch (PDOException $e) {
        PrintMessages::printError("Database Error");
    }
}
```

2.6.1 Missing Fields

Figure 30: *new password* Missing site name

Figure 31: *new password* Missing username

Figure 32: *new password* Missing password

2.6.2 Success

Figure 33: *new password* Before entering a new password

Input:

- Site: site
- Username: user
- Password: pass

Figure 34: *new password* Success on new password
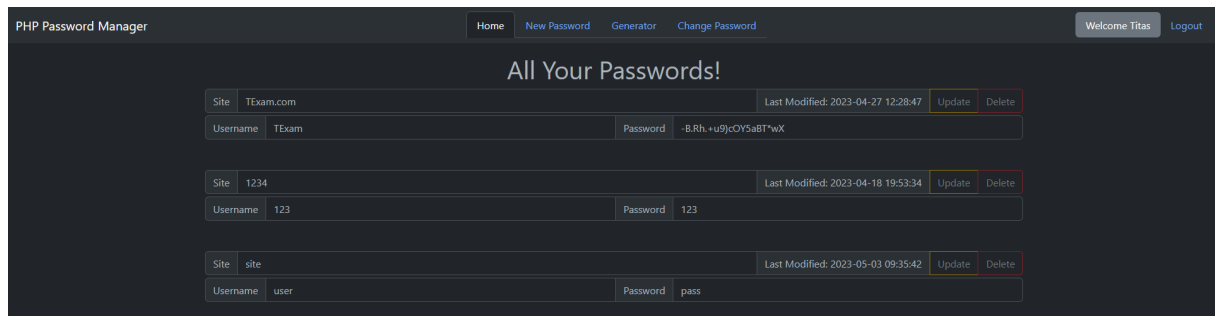
Figure 35: *new password* Home with the new password

Figure 36: *new password* Database with new password

2.7 Home Page

```php
<?php
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PrintMessages.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/Database.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PwdUtils.php");

if (isset($_POST['update'])) {
    $pid = $_POST['pid'];
    $site = $_POST['site'];
    $username = $_POST['username'];
    $password = PwdUtils::encryptPassword($_POST['password']);

    updatePassword($pid, $site, $username, $password);
} elseif (isset($_POST['delete'])) {
    $pid = $_POST['pid'];
    deletePassword($pid);
}

loadPasswords();

/**
 * Loads all of the passwords of the specific user
 * @return void
 */
function loadPasswords(): void {
    $uid = $_SESSION['uid'];

    try {
        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("SELECT * FROM ppm.passwords WHERE uid = :uid");
        $stmt->bindParam(":uid", $uid);

        $stmt->execute();

        if ($stmt->rowCount() <= 0) {
            print("<h2 class='text-center'>You don't have any passwords yet!</h2>");
            return;
        }

        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
```
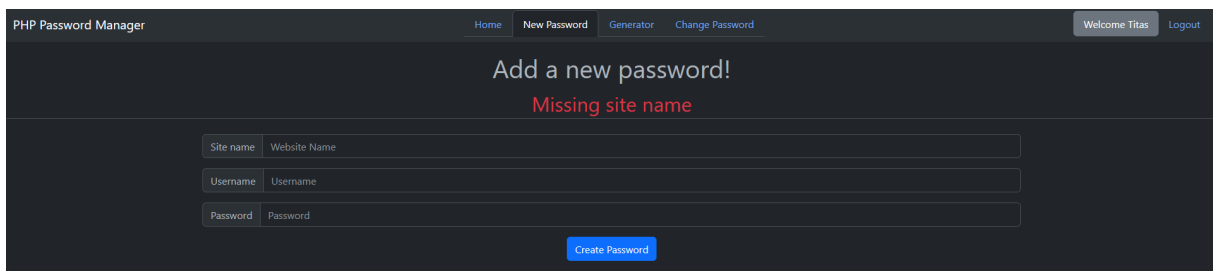
```php
            $pid = $row['pid'];
            $site = $row['site_name'];
            $username = $row['username'];
            $password = PwdUtils::decryptPassword($row['password']);
            $creation_date = $row['creation_date'];
            createPasswordForm($pid, $site, $username, $password, $creation_date);
        }
    } catch (PDOException $e) {
        PrintMessages::printError("Database Error");
    }
}

/**
 * Generates a form for a password that can be edited or deleted
 * @param int $pid Password ID
 * @param string $site Site or app name
 * @param string $username Username for the password
 * @param string $password Password
 * @param string $creation_date Date the password was created or modified
 * @return void
 */
function createPasswordForm(int $pid, string $site, string $username, string
$password, string $creation_date): void {
    echo '
        <div class="container mb-5">
            <form role="form" method="post" id="passwordForm" action="' .
htmlspecialchars($_SERVER["PHP_SELF"]) . '">
                <input type="hidden" id="pid" name="pid" value="' . $pid . '">
                <div class="input-group mb-1">
                    <span class="input-group-text">Site</span>
                    <input type="text" aria-label="Website" class="form-control"
id="site" name="site"
                        value="' . $site . '">
                    <span class="input-group-text" id="lastModified">Last Modified:
' . $creation_date . '</span>
                    <button class="btn btn-outline-secondary border-warning-subtle"
type="submit" name="update">Update</button>
                    <button class="btn btn-outline-secondary border-danger-subtle"
type="submit" name="delete">Delete</button>
                </div>
                <div class="input-group mb-1">
                    <span class="input-group-text">Username</span>
                    <input type="text" aria-label="Username" class="form-control"
id="username"
                        name="username" value="' . $username . '">
                    <span class="input-group-text">Password</span>
                    <input type="text" aria-label="password" class="form-control"
id="password"
                        name="password" value="' . $password . '">
                </div>
            </form>
        </div>';
}

/**
 * Updates an existing password in the database
```

```php
 * @param int $pid The password ID
 * @param string $site The site or app name
 * @param string $username The username for the password
 * @param string $password The password
 * @return void
 */
function updatePassword(int $pid, string $site, string $username, string $password):
void {
    $current_time = date("Y-m-d H:i:s");

    try {
        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("UPDATE passwords SET site_name = :site, username
= :user, password = :pwd, creation_date = :date WHERE uid = :uid AND pid = :pid");
        $stmt->bindParam(":uid", $_SESSION['uid']);
        $stmt->bindParam(":pid", $pid);
        $stmt->bindParam(":site", $site);
        $stmt->bindParam(":user", $username);
        $stmt->bindParam(":pwd", $password);
        $stmt->bindParam(":date", $current_time);

        $stmt->execute();
        unset($stmt);
        unset($db);
    } catch (PDOException $e) {
        PrintMessages::printError("Database Error");
    }
}

/**
 * Deletes a password from the database
 * @param int $pid The password ID
 * @return void
 */
function deletePassword(int $pid): void {
    try {
        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("DELETE FROM passwords WHERE pid = :pid AND uid
= :uid");
        $stmt->bindParam(":uid", $_SESSION['uid']);
        $stmt->bindParam(":pid", $pid);

        $stmt->execute();
        unset($stmt);
        unset($db);
    } catch (PDOException $e) {
        PrintMessages::printError("Database Error");
    }
}
```

2.7.1 No Passwords
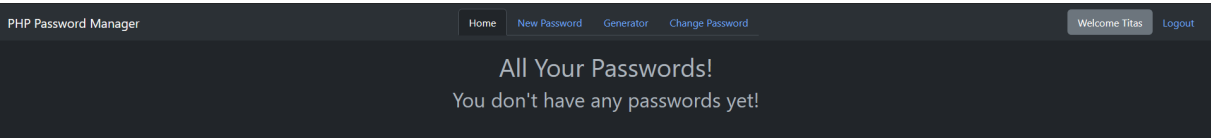
Figure 37: *home* Home with no passwords

2.7.2 Passwords
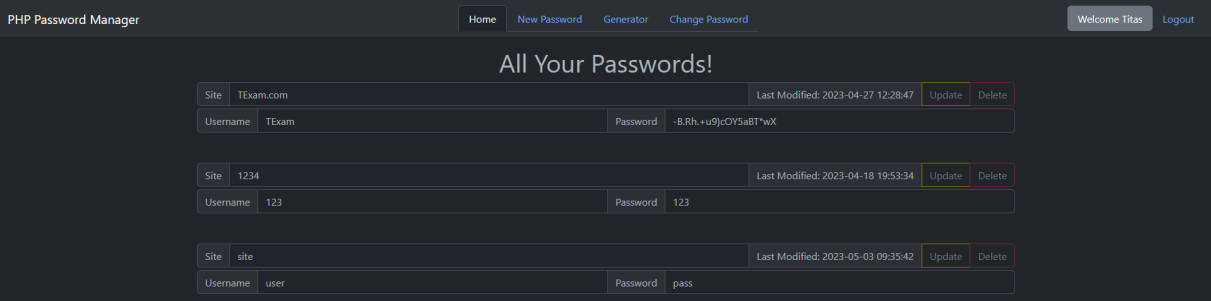
Figure 38: *home* Home with passwords

2.7.3 Change Password Details

Changing site and username first.
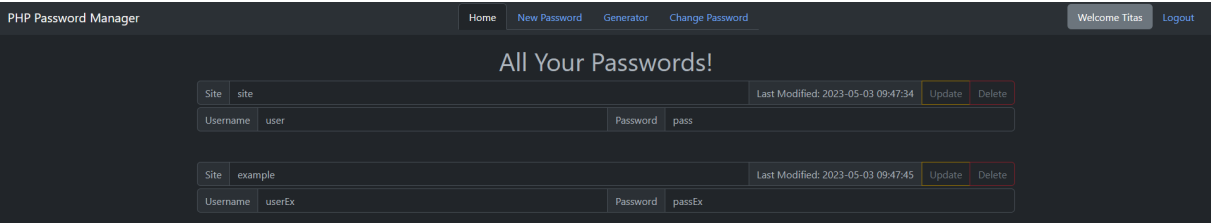
Figure 39: *home* Home before change first password
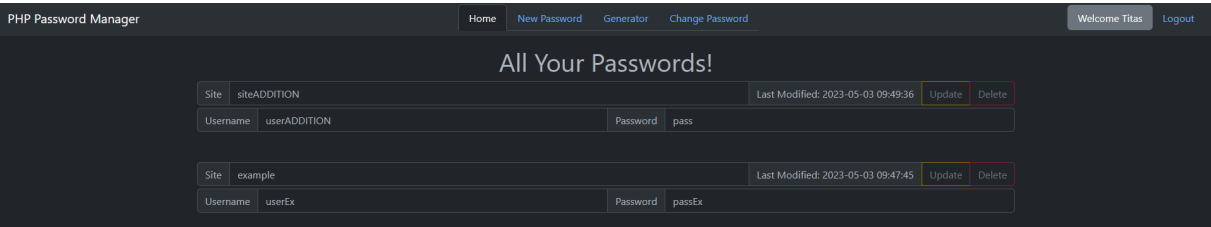
Figure 40: *home* Home after change first password
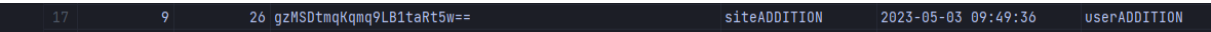
Figure 41: *home* Home after change first password in db

Changing password.
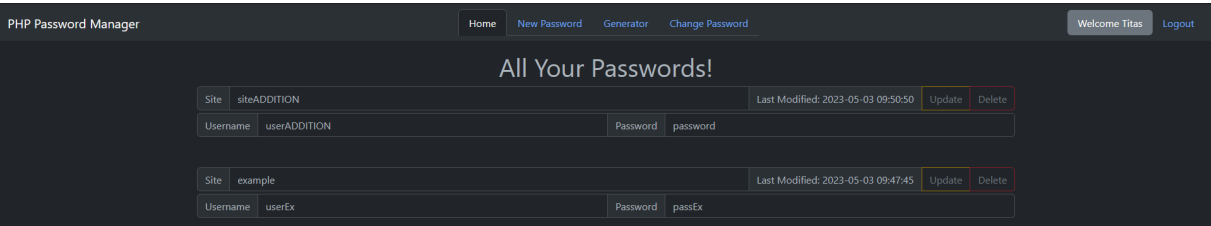
Figure 42: *home* Home after change password from 'pass' to 'password'

Figure 43: *home* Home after change password from 'pass' to 'password' in the db

2.7.4 Delete Password

Figure 44: *home* Home before deleting first password

Figure 45: *home* Home before deleting first password in the db

Figure 46: *home* Home after deleting first password

Figure 47: *home* Home after deleting first password in the db

2.8 Change Password

```php
<?php
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PrintMessages.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/Database.php");
include_once($_SERVER['DOCUMENT_ROOT'] . "/app/utils/PwdUtils.php");

if (isset($_POST['submit'])) {
    if (empty($_POST['password'])) {
        PrintMessages::printError("Empty Password");
    } elseif (empty($_POST['repeatPassword'])) {
        PrintMessages::printError("Empty Repeated Password");
    } elseif ($_POST['repeatPassword'] !== $_POST['password']) {
        PrintMessages::printError("Passwords do not match");
    } else {
        $password = $_POST['password'];
        changePassword($password);
    }
}

function changePassword($password): void {
    $new_master = PwdUtils::generateMasterKey();
    $passwords = array();

    try {
        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("SELECT pid, password FROM passwords WHERE uid
= :uid");
```

```php
        $stmt->bindParam(":uid", $_SESSION['uid']);

        $stmt->execute();

        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $pid = $row['pid'];
            $old_password = PwdUtils::decryptPassword($row['password']);
            $passwords[$pid] = $old_password;
        }
    } catch (PDOException $e) {
        PrintMessages::printError("Database Error");
    }

    $_SESSION['master_key'] = $new_master;

    foreach ($passwords as $pid => $old_password) {
        updatePassword($pid, $old_password);
    }

    try {
        $new_master_encrypted = PwdUtils::encryptMaster($new_master, $password);
        $new_password_hashed = password_hash($password, PASSWORD_DEFAULT);

        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("UPDATE user SET master_key = :master, pwd = :pwd
WHERE uid = :uid");
        $stmt->bindParam(":uid", $_SESSION['uid']);
        $stmt->bindParam(":master", $new_master_encrypted);
        $stmt->bindParam(":pwd", $new_password_hashed);

        $stmt->execute();
    } catch (PDOException $e) {
        PrintMessages::printError("Database Error");
    }
}

/**
 * Updates the password in the database
 * @param string $pid Password ID
 * @param string $password Plain-text Password
 * @return void
 */
function updatePassword(string $pid, string $password): void {
    $current_time = date("Y-m-d H:i:s");
    $encrypted_password = PwdUtils::encryptPassword($password);

    try {
        $db = new Database();
        $conn = $db->getConnection();
        $stmt = $conn->prepare("UPDATE passwords SET password = :pwd, creation_date
= :date WHERE uid = :uid AND pid = :pid");
        $stmt->bindParam(":uid", $_SESSION['uid']);
        $stmt->bindParam(":pid", $pid);
        $stmt->bindParam(":pwd", $encrypted_password);
        $stmt->bindParam(":date", $current_time);
```

```
        $stmt->execute();
    } catch (PDOException $e) {
        PrintMessages::printError("Database Error");
    }
}
```

2.8.1 Missing Fields

Figure 48: *change password* Missing password

Figure 49: *change password* Missing repeated password

2.8.2 Passwords Do Not Match

Figure 50: *change password* Passwords do not match

2.8.3 Success

Figure 51: *change password* User table before changing password

Figure 52: *change password* Passwords table before changing password

Input:
- Password: Titas
- Repeated Password: Titas

Figure 53: *change password* After successful change

WHERE uid=9   × =▾ ORDER BY

uid ⇕ master_key   ⇕ pwd   ⇕ username ⇕
9 388TgGiSfoDihl0Do6VVAycRuF4dQC63zxCOQBl5Lg1/HOfthNa7rgvC/1PtPoIlml5aK… $2y$10$Yk4FYOyPQi.rKdA9tbiT..hhDOOoDWV7zt8CzK/8v.yOvn426x.9C Titas

Figure 54: *change password* User table after changing password

WHERE uid=9   × =▾ ORDER BY

uid ⇕   pid ⇕ password   ⇕ site_name ⇕ creation_date   ⇕ username ⇕
9   27 DqSCWXk+1YiQ2EpE0Pbegw==   example   2023-05-03 10:02:05   userEx

Figure 55: *change password* Passwords table after changing password