# **SNSシステム開発 - タイムライン機能その１**

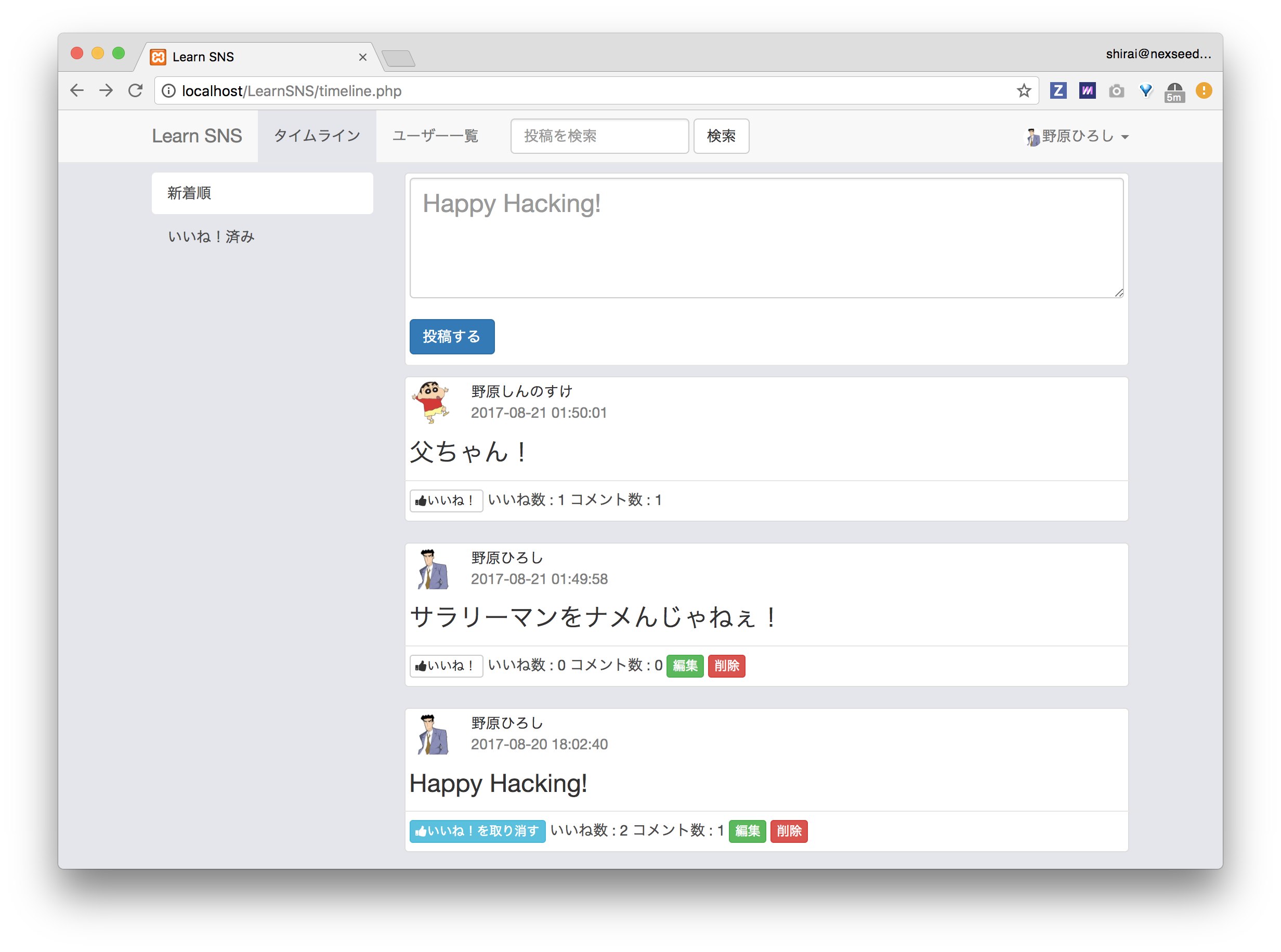
前回、ログイン機能が完成しました。

ここからは、**タイムライン**の開発に入っていきます。

## **概要**

このページでは、タイムライン画面のうちの以下１、２、３を題材に学習していきます。

1. ユーザ情報の表示
2. サインアウト
3. 投稿機能



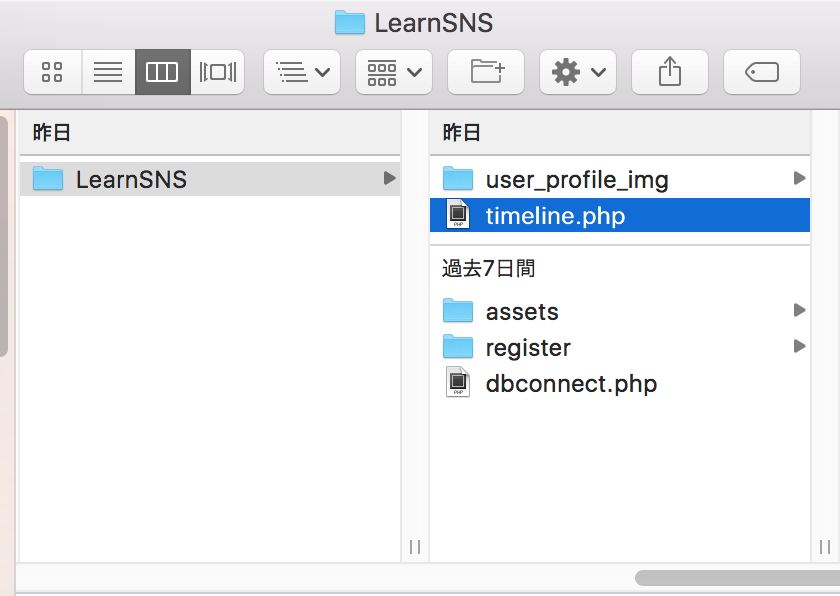
## **要件確認**

続いて、タイムライン画面の要件を確認していきます。

### **要件リスト**

1. **ログインしているユーザ名と画像が確認できる**
2. **ログアウトできる**
3. **投稿ができる**
   * **空チェックのバリデーションがある**
4. ユーザーは投稿の全てを見ることができる
   * 各投稿には投稿したユーザ名と画像が表示されている
5. ユーザーは投稿にいいねできる
6. 編集画面へ移動できる
   * 編集ボタンから詳細画面へ遷移する
7. 投稿の削除ができる
8. ページネーションできる
9. いいねした記事だけにフィルタができる

## **タイムライン画面の作成**

要件が確認できたので、timeline.phpの実装に入っていきます。

## **共通部分の切り分け**

今回も画面の基礎は出来ているので、PHPを使った機能実装に入っていきますが、ここでinclude()という組み込み関数を見ておきます。

Webサービスを作成しているとページ全体で共通して利用したい箇所が出てくると思います。例えば、上部のメニュー部分や下部のコピーライト部分などは、ページ毎にデザインを変える必要がなく、全ページから一貫した内容を表示されていることがほとんどです。

このような複数のページで同じレイアウトを使う際にinclude()が使えます。

include()はrequire()と同様に、指定した箇所に読み込んだファイルの内容をそのまま差し込みます。

そのため、読み込み元で利用可能な変数については、読み込まれたファイルの中でも利用することができます。

**※requireとincludeの違い**

どちらもほとんど同様の挙動をしますが、エラーが発生した際に異なる挙動になります。

requireの場合は、エラー発生時点でFatal Errorとなり、処理がそこで中断されます。

includeの場合は、エラーが発生してもWarningとなり、処理は続行されます。

初学者の方にとっては、処理が続行されるならincludeのほうが使いやすそうだ、と思われるかも知れませんが、通常、**プログラムにエラーが発生した際には正しく処理が止まること**が期待されます。

そのため、読み込みに失敗しても重大な問題とならないHTMLなどのテキストを読み込むのにはincludeを、DB接続などの重要な機能に関してはrequireを利用していきましょう。

## **ナビゲーションバーのログインユーザ情報表示機能実装**

それでは、サインアップ、サインインで実装したデータを表示させます。

どのユーザでログインしているか確認できるように、ナビゲーションバーにログインしているユーザを表示させましょう。

**1. ログインしているユーザのユーザ情報をデータベースから取得**

ユーザ名と画像はどのように表示できたでしょうか？表示するために必要な情報はどこに保存されていたでしょうか？

データベースのusersテーブルですね。

情報を取り出しましょう。データベースから取り出す時は、CRUD処理のRです。取り出しに必要なキーワードは、サインインした時に取得しています。

サインイン機能を実装した時に、$\_SESSION[‘LearnSNS’][‘id’]にユーザのIDを追加しました。このIDを使って情報を取り出します。

##### **timeline.php**

<?php

session\_start();

require('dbconnect.php');

$sql = 'SELECT \* FROM `users` WHERE `id`=?';

$data = [$\_SESSION['LearnSNS']['id']];

$stmt = $dbh->prepare($sql);

$stmt->execute($data);

$signin\_user = $stmt->fetch(PDO::FETCH\_ASSOC);

?>

きちんと取り出せているかは、var\_dumpなどで表示させて確認しましょう。

**2. ユーザ情報をHTML内に表示**

取得した情報をナビゲーションバーに表示させましょう。

##### **navbar.php**

・・・

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false"><img src="user\_profile\_img/<?php echo $signin\_user['img\_name']; ?>" width="18" class="img-circle"><?php echo $signin\_user['name']; ?> <span class="caret"></span></a>

<ul class="dropdown-menu">

・・・

## **サインアウト機能実装**

サインアウトがいつでもできるように実装しておきましょう。

サインアウトに必要な処理は、

1. $\_SESSIONを空にして、セッションを破棄する
2. サインイン画面に転送

$\_SESSIONは、サインアウトするときにはサーバからきちんと削除しておきましょう。$\_SESSION変数を空にして、session\_destroy()関数を実行することで完全に破棄されます。

session\_destroy()でサーバー内からは削除されますが、ブラウザにも残っているので空の配列を代入します。

##### **signout.php**

<?php

session\_start();

//SESSION変数の破棄

$\_SESSION = [];

//サーバー内の$\_SESSION変数のクリア

session\_destroy();

// signin.phpへ移動

header("Location: signin.php");

exit();

これで、ナビゲーションバーからすぐにサインアウトできるようになりました。

念のため、signin.phpでvar\_dumpなどを用いて$\_SESSIONが空になっていることを確認しておきましょう。

## **不正遷移制御**

サインアウトの実装が完了しましたが、このままだとURLの直入力で**timeline.php**に遷移できてしまいます。試しにサインアウトしたのちに、**timeline.php**にアクセスしてみると、Undefined indexのエラー文言が表示され、ユーザーの画像と名前が非表示になっています。

不正な遷移、つまり未ログインユーザーの遷移を許さないように修正しましょう。

実装方法はcheck.phpで行なった流れと同じです。

1. ログイン状態をチェックする
2. ログイン状態でなければsignin.phpに強制遷移する

##### **timeline.php**

if (!isset($\_SESSION['LearnSNS']['id'])) {

header('Location: signin.php');

exit();

}

この時、if文の条件に気を付けてください。!isset($\_SESSION['LearnSNS'])だけでは条件が足りません。

なぜなら、ログイン状態を保持しているのは、$\_SESSION['LearnSNS']['id']であって、$\_SESSION['LearnSNS']ではありません。登録画面などでも$\_SESSION['LearnSNS']に値を保持するしていましたよね。

実装できたら未ログイン状態で**timeline.php**に遷移が出来ないことを確認しましょう。

## **投稿機能実装**

次は、投稿機能を実装します。

まずは、要件で確認したとおり下記バリデーションを実装していきます。

* 投稿欄が空だった場合のバリデーション（blank）がある

### **空チェック**

**空チェック**を実装しましょう。

実装の流れはサインアップ、ログイン画面と同じです。以下は流れのおさらいです。

1. エラーだった場合に何エラーかを保存するための$errors配列を定義
2. 送信されたデータと''（空文字）とを比較
3. 一致する場合は$errors配列にfeedキーをセットし、blankという値を保存
4. HTMLの投稿フォーム下にもし$errors配列のfeedキーが存在し、blankという値が入っていた場合はエラーメッセージ投稿データを入力してくださいを出力

#### 

#### **▼練習問題1**

**投稿**について空文字チェックを実装してください。

**※すぐ下に解答プログラムを記載していますが、まずは今までのプログラムを参考に自分の力で実装しましょう。実践は応用の連続です。**

|  |  |
| --- | --- |
| **項目** | **使用する値** |
| 投稿内容の変数名 | $feed |
| 投稿のエラーキー名 | feed |
| 表示するエラーメッセージ | 投稿データを入力してください |

#### 

#### **▼練習問題1の解答プログラム**

##### **timeline.php**

<?php

….

// 初期化

$errors = [];

// ユーザーが投稿ボタンを押したら発動

if (!empty($\_POST)) {

// バリデーション

$feed = $\_POST['feed']; // 投稿データ

// 投稿の空チェック

if ($feed != '') {

// 投稿処理

} else {

$errors['feed'] = 'blank';

}

}

?>

<!DOCTYPE html>

….

<form method="POST" action="">

<div class="form-group">

<textarea name="feed" class="form-control" rows="3" placeholder="Happy Hacking!" style="font-size: 24px;"></textarea><br>

<?php if (isset($errors['feed']) && $errors['feed'] == 'blank') { ?>

<p class="text-danger">投稿データを入力してください</p>

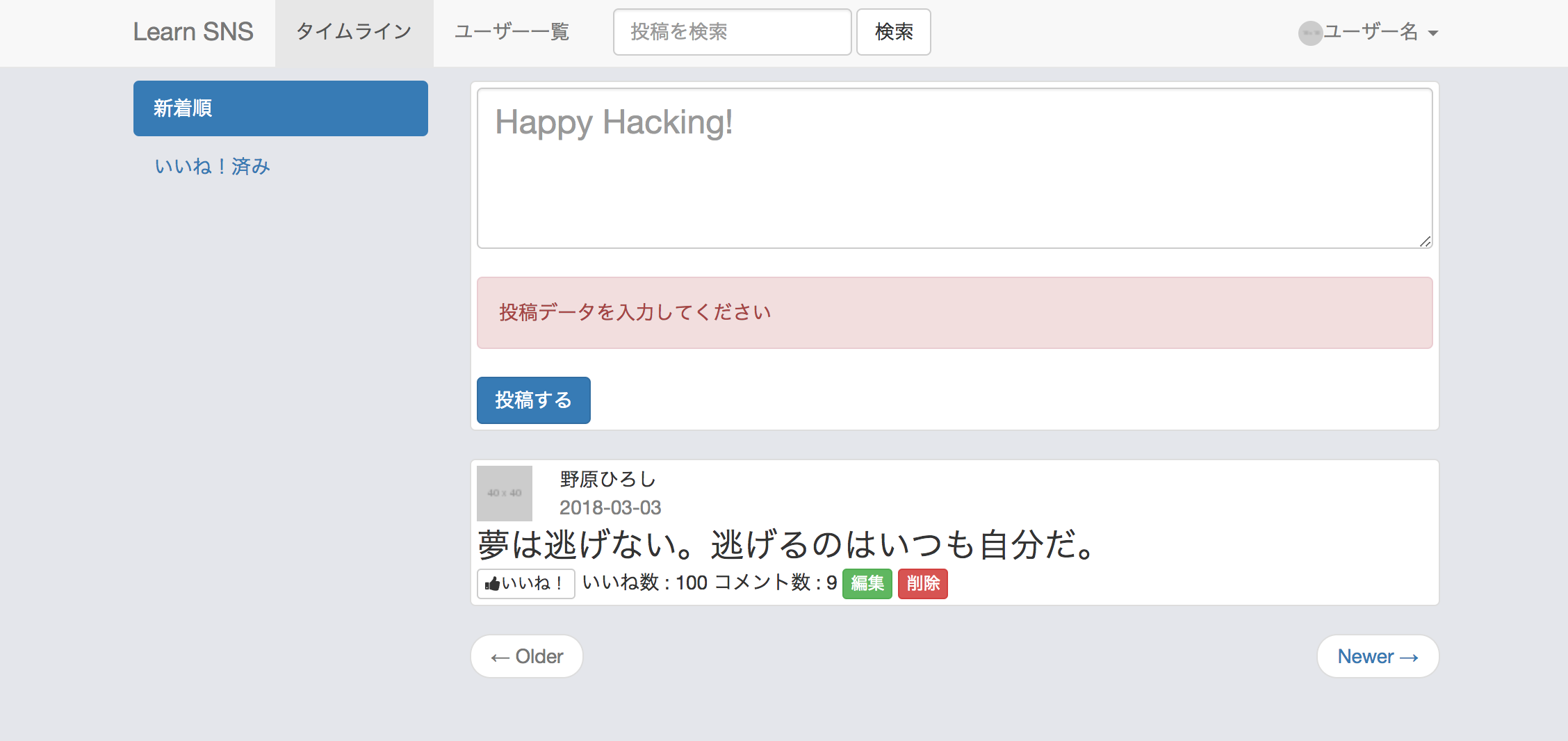
<?php } ?>

</div>

<input type="submit" value="投稿する" class="btn btn-primary">

</form>

….



### **投稿処理**

次はバリデーションを通過した（成功）場合の処理について進めていきます。

投稿機能は、CRUD処理のCです。

signup.phpでも行いました。サインアップでは、usersテーブルに追加しましたが、今回は投稿を管理するfeedsテーブルに追加します。

feedsテーブルのテーブル定義書を確認し、バリデーション通過した場合のif文の中に、Create処理を書いてみましょう。

* テーブル定義書を確認して、必要な内容なデータをSQL文に追加する
* 追加後は、header関数で自身のページを読み込む

#### 

#### **▼練習問題2**

**投稿**の**Create処理**を実装してください。投稿処理では以下のカラムだけ登録対象とします。

**※すぐ下に解答プログラムを記載していますが、今までのプログラムを参考に自分の力で実装しましょう。自分で考えたプログラムを検証することが学習に繋がります。**

**feedsテーブル定義書（抜粋）**

|  |  |  |
| --- | --- | --- |
| **論理名** | **物理名** | **型** |
| 投稿内容 | feed | TEXT |
| 投稿したユーザID | user\_id | INT |
| 投稿した日時 | created | DATETIME |

#### 

#### **▼練習問題2の解答プログラム**

<?php

….

**require**('dbconnect.php');

// 初期化

$errors = [];

…

if ($feed != '') {

// 投稿処理

$sql = 'INSERT INTO `feeds` (`feed`, `user\_id`, `created`) VALUES (?, ?, NOW())';

$data = [$feed, $signin\_user['id']];

$stmt = $dbh->prepare($sql);

$stmt->execute($data);

header('Location: timeline.php');

exit();

} else {

...

#### 

投稿できたかは、phpMyAdminで確認しましょう。

feedsテーブルに、投稿内容が追加されていれば、投稿機能の完了です。