# **SNSシステム開発 - ユーザー登録機能**

ここまで、Learn SNSを開発するための事前準備がすべて整いました。

ここからは、**ユーザー認証機能**の開発に入っていきます。

## **概要**

このページでは、ユーザー登録機能の最初の画面である**ユーザー情報入力画面**の作成を題材に下記について学習していきます。

1. フォームの作成
2. バリデーション
3. 画像のアップロード
4. スーパーグローバル変数$\_SESSION

今回、手を加えていくのは/LearnSNS/register/signup.phpになります。

サイトマップ上では下記①のページになります。 

## **要件確認**

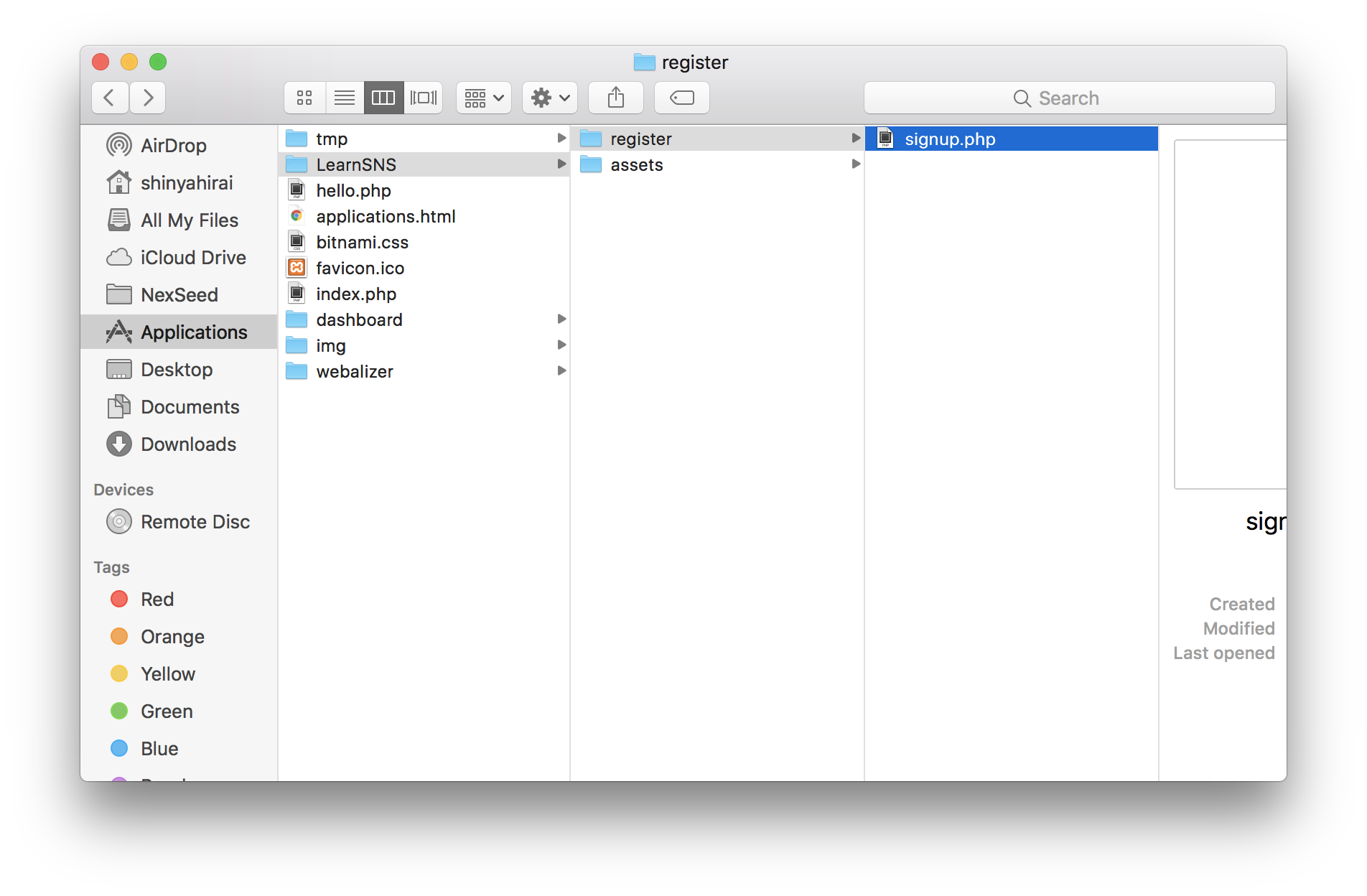
続いて、ユーザー情報入力画面の要件を確認していきます。

### **要件リスト**

1. ユーザーは「ユーザー名」「メールアドレス」「パスワード」を入力することができる
2. ユーザーは「プロフィール画像」を選択することができる
3. 各種入力項目には入力チェック（バリデーション）がある
   * ユーザー名が空だった場合エラーメッセージが出る
   * メールアドレスが空だった場合エラーメッセージが出る
   * パスワードが空だった場合エラーメッセージが出る
   * パスワードが4文字以下だった場合エラーメッセージが出る
   * パスワードが16文字以上だった場合エラーメッセージが出る
   * プロフィール画像が未選択だった場合エラーメッセージが出る
   * プロフィール画像の拡張子が「jpg」「png」「gif」以外だった場合エラーメッセージが出る
4. すべての入力チェックを通過した場合
   * セッション($\_SESSION)に入力された情報を保存する
   * プロフィール画像をuser\_profile\_imgフォルダへアップロード
   * 入力情報確認画面（check.php）へ遷移する

### **フローマップ**

## **ユーザー情報入力画面**

要件が確認できたところで、さっそくsignup.phpファイルのプログラミングに入っていきましょう。

### **1. ユーザー情報入力フォームの確認**

ユーザー登録画面で使うフォームは下記になります。

送信方法(method)や送信先(action)、各入力項目のtypeとnameを確認しておきましょう。

##### **signup.php**

<form method**=**"POST" action**=**"signup.php" enctype**=**"multipart/form-data">  
 <div class**=**"form-group">  
 <label for**=**"name">ユーザー名</label>  
 <input type**=**"text" name**=**"input\_name" class**=**"form-control" id**=**"name" placeholder**=**"山田 太郎">  
 </div>  
 <div class**=**"form-group">  
 <label for**=**"email">メールアドレス</label>  
 <input type**=**"email" name**=**"input\_email" class**=**"form-control" id**=**"email" placeholder**=**"example@gmail.com">  
 </div>  
 <div class**=**"form-group">  
 <label for**=**"password">パスワード</label>  
 <input type**=**"password" name**=**"input\_password" class**=**"form-control" id**=**"password" placeholder**=**"4 ~ 16文字のパスワード">  
 </div>  
 <div class**=**"form-group">  
 <label for**=**"img\_name">プロフィール画像</label>  
 <input type**=**"file" name**=**"input\_img\_name" id**=**"img\_name">  
 </div>  
 <input type**=**"submit" class**=**"btn btn-default" value**=**"確認">  
</form>

ここで、formタグの各値に注目しましょう。

methodにはPOSTが利用されているので、$\_POSTで値を受け取ることが出来ます。actionにはcheck.phpが指定されていますが、signup.php内で入力項目に不備がないか確認するためにsignup.phpに書き換えましょう。

inputタグは下記の4つがあります。

　　1. ユーザー名

　　2. メールアドレス

　　3. パスワード

　　4. プロフィール画像

## **2.バリデーション実装**

HTMLの確認が出来たので、ここからはPHPを使った機能実装に入っていきます。

まず、バリデーションとは、入力フォームに対して正しく情報が入力されたかどうかを確認し、ミスがあった場合はそのページ上にエラーメッセージを表示しユーザーへの再入力を求める機能のことを指します。

例えば、ユーザー情報を入力する際に**プロフィール画像を選択し忘れて**いたり、**入力したパスワードが短すぎる**まま確認ボタンを押してしまった場合など、そのままでは次の画面に移ってほしくない時にエラーメッセージを表示し、ユーザーに情報の入力を求めるために利用されます。

こういった場合に、ユーザーへどの入力項目にミスがあり、どのように修正する必要があるのか、を的確に示すエラーメッセージを出力する必要があります。そうでなければ、せっかく入力までしてくれたユーザーが登録途中で面倒になり、登録をやめてしまうこともあり得ます。入力フォームでの離脱率はWebサービスにおいて重要なポイントのひとつなので、しっかりと要件にあったバリデーションを実装していきましょう。ここでは、要件で確認したとおりに4つの項目を満たすバリデーションを実装していきます。

### **2-1.空チェック**

まずはバリデーションの仕組みについて理解するため、簡単な**ユーザー名の空チェック**から入っていきます。

**空チェック**とは、入力フォームが空（blank）のまま送信されていないかを検証するためのバリデーションです。

実装の流れをまとめると下記。

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nameキーをセットし、blankという値を保存
4. HTMLのユーザー名入力フォーム下にもし$errors配列のname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 ”ユーザー名を入力してください”を出力

#### **1の処理**

「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」

signup.phpに$errors配列の定義文を記述します。このとき、まだ何もエラーはでないため[]（空の配列）として定義します。

##### **signup.php**

**<?php**  
 $errors **=** **[]**;  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
<head>  
・・・

#### **2, 3の処理**

「送信されたデータと''（空文字）とを比較」

「一致する場合は$errors配列にnameキーをセットし、blankという値を保存」

条件により比較し、結果で処理を分岐するため条件分岐文の**if文**を使用します。比較するのは送信されたユーザー名のデータと空文字なので、ひとつひとつ見ていきましょう。

フォームからの送信データを受け取るにはまずformタグの内容を確認しGET送信なのかPOST送信なのかを知る必要があります。signup.phpのformタグのmethod属性部分でPOST送信指定がされていました。従って、このフォームからの送信データはスーパーグローバル変数の$\_POSTを使用することで受け取ることができます。

また、ユーザー名はinputタグのname属性値がinput\_nameとなっているので、$\_POST['input\_name']とすることでユーザー名入力フォームの情報を扱うことが可能です。

さらに、これから記述するプログラムはユーザーが確認ボタンを押した時のみ処理したいプログラムです。 そのため、if(!empty($\_POST)){}を使ってフォーム送信時（確認ボタンを押したとき）のみ実行されるよう条件分岐する必要があります。

この処理を書かなかった場合、ユーザーが初めてsignup.phpを閲覧した際にはもちろん$\_POST['input\_name']というキーが存在しないため、画面にはNotice: Undefined index: input\_name ~というエラーがでてしまいます。そのため、empty()関数を使用して配列の中身が空かどうかをチェックし、!を付けて空でなければtrueを返すように条件をセットしました。（emtpty()関数はセットした変数もしくは配列が空だった場合falseを返します。）

以上のことから、下記のようなプログラムを記述しユーザー名の空チェックのPHPプログラムを完成させます。

##### **signup.php**

**<?php**  
 $errors **=** [];  
 **if** (**!empty**($\_POST)) {  
 $name **=** $\_POST['input\_name'];  
 *// ユーザー名の空チェック*  
 **if** ($name **==** '') {  
 $errors['name'] **=** 'blank';  
 }  
 }  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
・・・

$\_POST['input\_name']は今後頻繁に使用するため、$nameという簡単な変数へ代入しておきました。

#### **4の処理**

「HTMLのユーザー名入力フォーム下にもし$errors配列のname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 “ユーザー名を入力してください” を出力」

このプログラムにより、ユーザーがユーザー名を入力せずに確認ボタンを押した際に空だった$errors配列にはnameキーとblankのペア情報が代入されます。

この配列を利用し、画面上にエラーメッセージを表示するプログラムを記述します。

$errors['name']にblankが代入されているとき = ユーザーが空のまま確認ボタンを押したということになるので、$errors['name']が存在し且つblankが代入されていればエラーメッセージがでるようにユーザー名のinputタグ下にプログラムを追加します。

##### **signup.php**

<!DOCTYPE html>  
<html lang="ja">  
・・・  
 <form method="POST" action="signup.php" enctype="multipart/form-data">  
 <div class="form-group">  
 <label for="name">ユーザー名</label>  
 <input type="text" name="input\_name" class="form-control" id="name" placeholder="山田 太郎">  
 **<?php** **if**(isset($errors['name']) **&&** $errors['name'] **==** 'blank'): **?>**  
 <p class="text-danger">ユーザー名を入力してください</p>  
 **<?php** endif; **?>**  
 </div>  
・・・  
</html>

ここまでプログラムが完成したら、一度signup.phpをブラウザで開きユーザー名の入力フォームを空のまま確認ボタンを押してみましょう。

入力フォームの下に赤色でエラーメッセージが出力されれば**成功**です。



#### **▼練習問題1**

ユーザー名と同じように**メールアドレス**、**パスワード**についても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、まずは上記を参考に自分の力で実装しましょう。自力で考え実装する力が実践では必要です。**

|  |  |
| --- | --- |
| **項目** | **使用する値** |
| メールアドレスの変数名 | $email |
| メールアドレスのエラーキー名 | email |
| パスワードの変数名 | $password |
| パスワードのエラーキー名 | password |
| 表示するエラーメッセージ（email） | メールアドレスを入力してください |
| 表示するエラーメッセージ（password） | パスワードを入力してください |

#### 

#### **▼練習問題1の解答プログラム**

##### **signup.php**

**<?php**  
 $errors **=** **[]**;  
  
 **if** (**!empty**($\_POST)) {  
 $name **=** $\_POST['input\_name'];  
 $email **=** $\_POST['input\_email'];  
 $password **=** $\_POST['input\_password'];  
  
 *// ユーザー名の空チェック*  
 **if** ($name **==** '') {  
 $errors['name'] **=** 'blank';  
 }  
  
 *// メールアドレスの空チェック*  
 **if** ($email **==** '') {  
 $errors['email'] **=** 'blank';  
 }  
  
 *// パスワードの空チェック*  
 **if** ($password **==** '') {  
 $errors['password'] **=** 'blank';  
 }  
 }  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
 <form method="POST" action="signup.php" enctype="multipart/form-data">  
 <div class="form-group">  
 <label for="name">ユーザー名</label>  
 <input type="text" name="input\_name" class="form-control" id="name" placeholder="山田 太郎">  
 **<?php** **if**(isset($errors['name']) **&&** $errors['name'] **==** 'blank') : **?>**  
 <p class="text-danger">ユーザー名を入力してください</p>  
 **<?php** endif; **?>**  
 </div>  
 <div class="form-group">  
 <label for="email">メールアドレス</label>  
 <input type="email" name="input\_email" class="form-control" id="email" placeholder="example@gmail.com">  
 **<?php** **if**(isset($errors['email']) **&&** $errors['email'] **==** 'blank') : **?>**  
 <p class="text-danger">メールアドレスを入力してください</p>  
 **<?php** endif; **?>**  
 </div>  
 <div class="form-group">  
 <label for="password">パスワード</label>  
 <input type="password" name="input\_password" class="form-control" id="password" placeholder="4 ~ 16文字のパスワード">  
 **<?php** **if**(isset($errors['password']) **&&** $errors['password'] **==** 'blank') : **?>**  
 <p class="text-danger">パスワードを入力してください</p>  
 **<?php** endif; **?>**  
 </div>  
・・・  
</html>



### **2-2.文字数チェック**

パスワードには空チェックの他に、セキュリティの観点から文字数チェックも実装します。基本は空チェックのロジックをそのまま流用します。

まずは入力されたパスワードが何文字なのかを知る必要があるため、strlen()関数を使って$passwordの文字数を取得し、$count変数へ代入します。

続いて、パスワードの空チェックif文に、elseif文を繋げて$countが4以下または16以上だった場合は文字数制限（length）エラーが起きたことを$errors配列に保存します。

##### **signup.php**

**<?php**  
 $errors **=** [];  
  
 **if** (**!empty**($\_POST)) {  
 ・・・  
 *// パスワードの空チェック*  
 $count **=** strlen($password); *// hogehogeとパスワードを入力した場合、8が$countに代入される*  
 **if** ($password **==** '') {  
 $errors['password'] **=** 'blank';  
 } **elseif** ($count **<** 4 **||** 16 **<** $count) { *// ||演算子を使って4文字未満または16文字より多き場合にエラー配列にlengthを代入*  
 $errors['password'] **=** 'length';  
 }  
 }  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
</html>

続いて、空チェックのときと同じくもし$errorsにpasswordキーが存在し且つlengthという値が入っていた場合にはまた別のエラーメッセージが表示されるように下記プログラムを記述します。

##### **signup.php**

・・・  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
 <div class="form-group">  
 <label for="password">パスワード</label>  
 <input type="password" name="input\_password" class="form-control" id="password" placeholder="4 ~ 16文字のパスワード">  
 **<?php** **if**(isset($errors['password']) **&&** $errors['password'] **==** 'blank') : **?>**  
 <p class="text-danger">パスワードを入力してください</p>  
 **<?php** endif; **?>**  
 **<?php** **if**(isset($errors['password']) **&&** $errors['password'] **==** 'length') : **?>**  
 <p class="text-danger">パスワードは4 ~ 16文字で入力してください</p>  
 **<?php** endif; **?>**  
 </div>  
・・・  
</html>

すべてのプログラムを記述し終わったら、ブラウザで再度正しく実装できたか確認します。

このとき、パスワードの入力フォームには4文字未満もしくは16文字より多い文字数のパスワードを入力し確認ボタンを押し、パスワード入力フォーム下に表示されるエラーメッセージの内容が変わることを確認しましょう。



### **2-3.画像の空チェック**

各項目のバリデーションが実装できたので、次はinputタグのtype="file"で作成した画像選択フォームのバリデーションに移ります。

こちら上記とは別で取り上げている理由として、他のタイプのinputタグと違い$\_POSTで送信データを受け取ることができないからです。

変わりにファイルアップロード専用のスーパーグローバル変数$\_FILESを使用します。

この変数は、inputタグのtype="file"で選択され送信されたデータを取得し、下記のルールで使用が可能です。

1. formタグにenctype属性と値にmultipart/form-dataが設定されている
2. formタグでPOST送信されている

また、基本的にファイルアップロード時に使用する構文は下記の2つ。

**<?php**  
 $\_FILES['キー']['name']; *// 画像名を取得*  
 $\_FILES['キー']['tmp\_name']; *// 送信された画像データそのものを取得*  
**?>**

inputタグtype fileで選択されたファイルの情報は一時的にブラウザに保存されます。保存するのは画像のパス（ローカルPC上の場所）のみで、送信時に読み込んで実際に画像データを送信します。

$\_FILES['キー']['name']は単に画像名のみを取得し、$\_FILES['キー']['tmp\_name']は送信された画像データそのものを取得することができます。

上記から、送信時に画像の空チェックを行うには$\_FILES['キー']['name']を取得し、空だった場合は$errorsに値を代入するようにプログラムします。

##### **signup.php**

**<?php**  
 $errors **=** **[]**;  
  
 **if** (**!empty**($\_POST)) {  
 ・・・  
 } **elseif** ($count **<** 4 **||** 16 **<** $count) { *// ||演算子を使って4文字未満または16文字より多き場合にエラー配列にlengthを代入*  
 $errors['password'] **=** 'length';  
 }  
  
 *// 画像名を取得*  
 $file\_name **=** $\_FILES['input\_img\_name']['name'];  
 **if** (**!empty**($file\_name)) {  
 *// 拡張子チェックの処理*  
 } **else** {  
 $errors['img\_name'] **=** 'blank';  
 }  
 }  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
</html>

これで画像未選択のままデータを送信すると、$errors['img\_name']にblankが代入されるため、HTML側にも前回同様エラーメッセージを出力しましょう。

##### **signup.php**

・・・  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
 **<?php** **if**(isset($errors['password']) **&&** $errors['password'] **==** 'length') : **?>**  
 <p class="text-danger">パスワードは4 ~ 16文字で入力してください</p>  
 **<?php** endif; **?>**  
 </div>  
 <div class="form-group">  
 <label for="img\_name">プロフィール画像</label>  
 <input type="file" name="input\_img\_name" id="img\_name">  
 **<?php** **if**(isset($errors['img\_name']) **&&** $errors['img\_name'] **==** 'blank') : **?>**  
 <p class="text-danger">画像を選択してください</p>  
 **<?php** endif; **?>**  
 </div>  
・・・  
</html>

ここまでのプログラムが実装できたら、画像未選択状態で確認ボタンを押してみましょう。

下記のようにエラーメッセージが出れば、画像に対する空チェックは実装完了です。 

## **2-4.画像の拡張子チェック**

続いて、特定の画像のみ選択、アップロードできるようにバリデーションをかけていきます。

### **画像のみ選択可にする**

現在のinputタグでは、画像データ以外のファイル、例えばhtmlやjavascriptファイルなども選択し送信することができてしまいます。悪意あるユーザーが不正プログラムファイルをアップロードする。といったことも考えられるため、画像ファイルのみ選択可になるようプログラムに修正を加えます。

処理自体はシンプルで、inputタグにaccept="image/\*"を追加するだけです。

##### **signup.php**

・・・  
**<!DOCTYPE html>**  
<html lang**=**"ja">  
・・・  
 <div class**=**"form-group">  
 <label for**=**"img\_name">プロフィール画像</label>  
 <input type**=**"file" name**=**"input\_img\_name" id**=**"img\_name" accept**=**"image/\*"> *<!-- accept属性を追加 -->*  
 **<?php if(isset($errors['img\_name']) && $errors['img\_name'] == 'blank') : ?>**  
 <p class**=**"text-danger">画像を選択してください</p>  
 **<?php endif; ?>**  
 </div>  
・・・  
</html>

これだけでファイル選択時、画像データ以外は灰色になり選択できなくなります。

### **指定する拡張子画像のみアップロード可にする**

画像データのみ選択可になりましたが、画像データにも様々な種類があります。

Learn SNSでは、アップロードできる画像の種類を「jpg」「png」「gif」の3種類に制限する形で拡張子バリデーションの学習を行います。

画像拡張子チェックのフローは以下の通り。

1. 画像ファイル名の（拡張子）を取得し変数化
2. 大文字が含まれていた場合はすべて小文字化（PNG→png など）
3. jpg,png,gifと比較し、どれにも当てはまらない場合は$errors['img\_name']にtypeを代入する

#### **1. 画像ファイル名の（拡張子）を取得し変数化**

画像ファイル名は先ほど作成した$file\_name変数は持っているので、こちらの末尾3文字をsubstr()関数を使用して取得します。

substr()関数は、指定した文字列に「何文字目から取得」という情報を与えてそのレンジの文字列を取得する関数です。 何文字目かの指定は0から始め、マイナスの値を使用した場合は末尾から遡って取得します。

**<?php**  
 $str **=** 'abcdefghi';  
 **echo** substr($str,0); *// 0番目から取得 → abcdefghi*  
 **echo** substr($str,1); *// 1番目から取得 → bcdefghi*  
 **echo** substr($str,2); *// 2番目から取得 → cdefghi*  
 **echo** substr($str,3); *// 3番目から取得 → defghi*  
 **echo** substr($str,**-**1); *// 末尾1番目から取得 → i*  
 **echo** substr($str,**-**2); *// 末尾2番目から取得 → hi*  
 **echo** substr($str,**-**3); *// 末尾3番目から取得 → ghi*  
**?>**

この関数を使い、$file\_nameの末尾3文字を取得することで拡張子の文字列を変数化します。

**<?php**  
 *// 例*  
 $file\_type **=** substr($file\_name, **-**3);  
**?>**

#### **2. 大文字が含まれていた場合はすべて小文字化（PNG→png など）**

if文は大文字か小文字かも判別するため、大文字拡張子の画像ファイルが選択された場合は大文字→小文字と変換するstrtolower()関数を使用します。

**<?php**  
 *// 例*  
 $file\_type **=** strtolower($file\_type);  
**?>**

#### **3. jpg,png,gifと比較し、どれにも当てはまらない場合は$errors['img\_name']にtypeを代入する**

最後にif文を使ってjpg,png,gifの3パターンと比較し、どれにも当てはまらない場合は$errors['img\_name']にtypeを代入する処理とHTML側にエラーメッセージを出力する処理を記述しバリデーションを完成させます。

##### **signup.php**

**<?php**  
 $errors **=** [];  
  
 **if** (**!empty**($\_POST)) {  
 ・・・  
 *// 画像名を取得*  
 $file\_name **=** $\_FILES['input\_img\_name']['name'];  
 **if** (**!empty**($file\_name)) {  
 $file\_type **=** substr($file\_name, **-**3); *// 画像名の後ろから3文字を取得*  
 $file\_type **=** strtolower($file\_type); *// 大文字が含まれていた場合すべて小文字化*  
 **if** ($file\_type **!=** 'jpg' **&&** $file\_type **!=** 'png' **&&** $file\_type **!=** 'gif') {  
 $errors['img\_name'] **=** 'type';  
 }  
 } **else** {  
 $errors['img\_name'] **=** 'blank';  
 }  
 }  
**?>**  
<!DOCTYPE html>  
・・・

##### **signup.php**

・・・  
**<!DOCTYPE html>**  
<html lang**=**"ja">  
・・・  
 <div class**=**"form-group">  
 <label for**=**"img\_name">プロフィール画像</label>  
 <input type**=**"file" name**=**"input\_img\_name" id**=**"img\_name" accept**=**"image/\*"> *<!-- accept属性を追加 -->*  
 **<?php if(isset($errors['img\_name']) && $errors['img\_name'] == 'type') : ?>**  
 <p class**=**"text-danger">拡張子が「jpg」「png」「gif」の画像を選択してください</p>  
 **<?php endif; ?>**  
 </div>  
・・・  
</html>



## 

## 

## **バリデーション成功時の処理**

最後にバリデーションすべてを通過した（成功）場合の処理について。

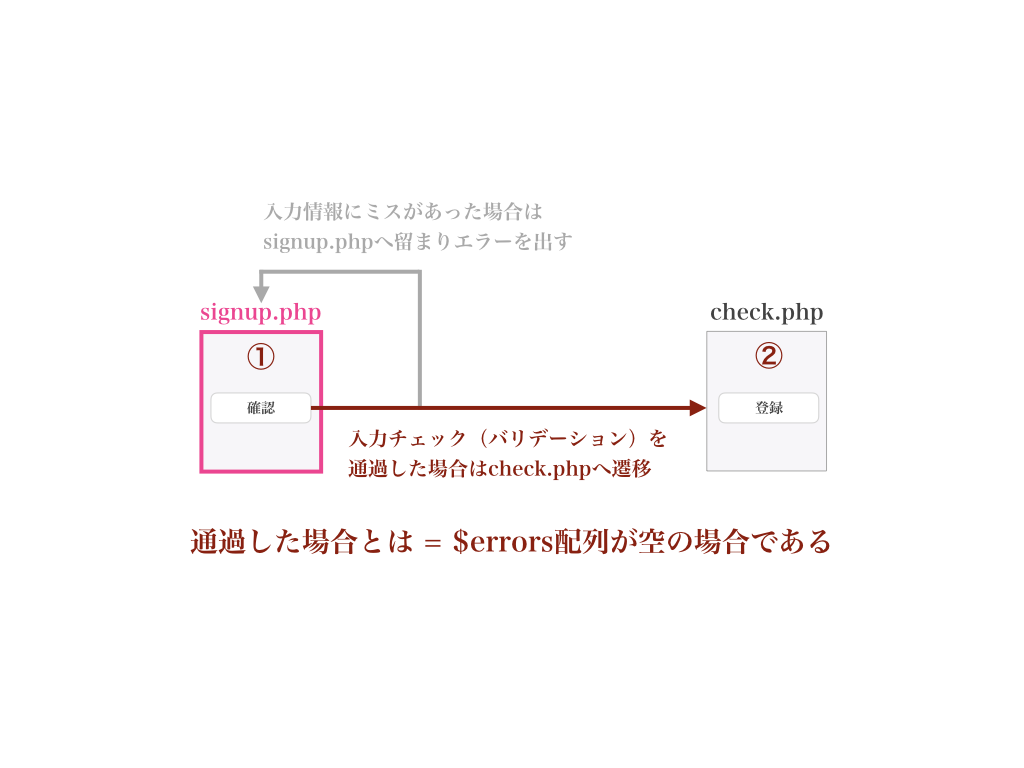
バリデーションをすべて通過したかどうか、プログラムでどう表現すれば良いでしょうか？答えは$errorsにあります。この配列は、何かしらのエラーがあれば値を代入することのなっています。

もし、すべてのバリデーションでエラーがなかった場合は、$errorsは最初に定義した空の状態のままなので、$errorsが空かどうかがバリデーション成功かどうかを判定するための条件になります。

**<?php**  
 *// 例*  
 **if** (**empty**($errors)) {  
 *// $errorsが空だった場合はバリデーション成功*  
 *// 成功時の処理を記述する*  
 }  
**?>**

上記のif文を通ると、必要な処理を行った後に入力情報確認ページ(check.php)へ遷移します。

##### **フローマップの詳細（$errorsが空かどうかが重要）**

****

では次に、バリデーション成功時の処理としては何が必要になるかを解説していきます。

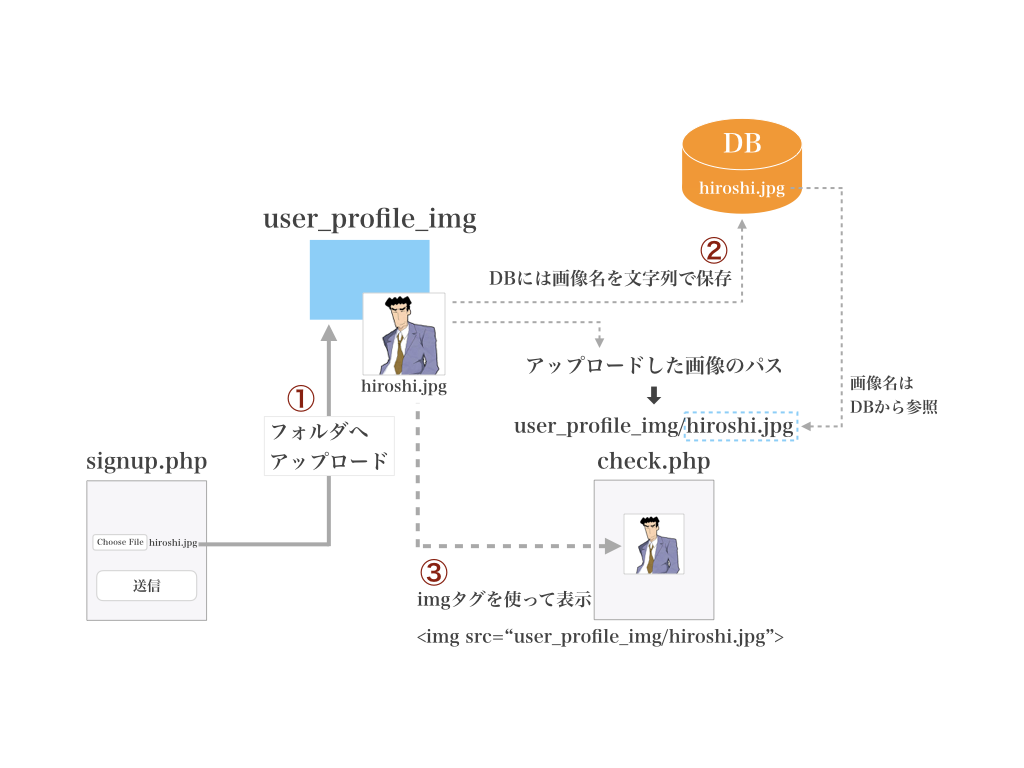
#### **バリデーション成功時の処理**

1. プロフィール画像のアップロード
2. セッションヘ送信データを保存する

ではひとつずつ実装していきましょう。

### **1. 画像のアップロード**

まず、実装機能の全体像を説明し、詳細に入っていきます。

ユーザーが選択し送信した画像データの保存 / 使用のロジックは下記の図。 

#### **【前提知識】画像ファイルの扱い - DBとの関係**

DBは基本的に画像データをそのまま直接保存することはできません。あくまで文字や数字データを管理します。

そのため、ユーザーが送信した画像データは指定したフォルダへアップロードし、DBにはアップロードされた画像の名前を文字データで保存します。

保存した画像を取得して表示したい場合は、画像名をDBから取得しimgタグのパスへセットする形でフォルダへアップロードされた画像を表示します。

処理の流れは上記表の通り。

1. フォルダへアップロード
2. DBには画像名を文字列で保存
3. imgタグを使って表示

2番の処理はcheck.phpで情報確認後、登録ボタンを押した際に行うのでここでは1番と3番を実装しましょう。

また、DBの変わりに画像名の保存はセッションという機能を使用します。（セッションについては下記2の項目で）

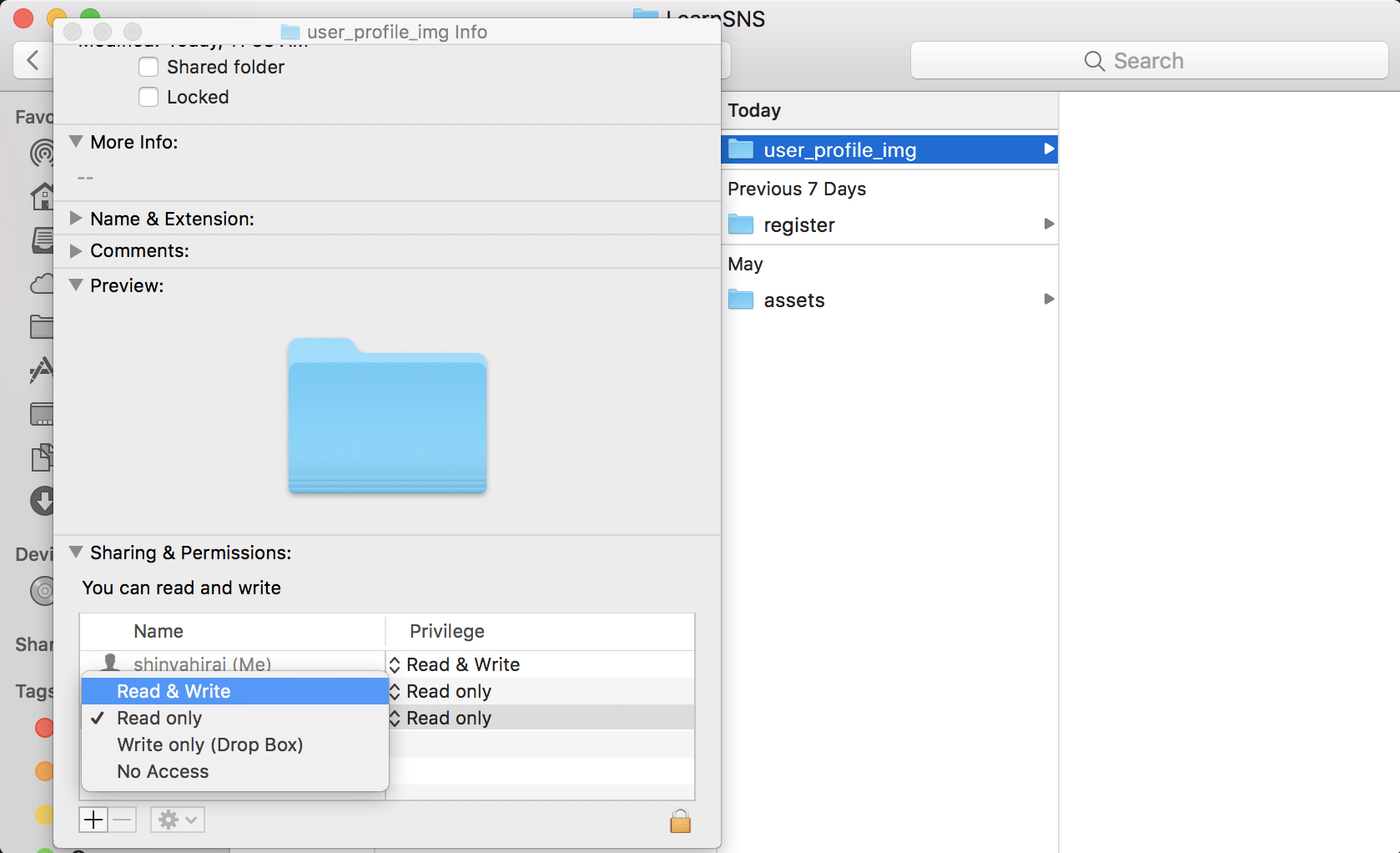
それでは、画像アップロード機能を実装していきます。

#### **アップロード先フォルダの準備**

まずはじめに、アップロード先のフォルダをLearnSNSフォルダ内にuser\_profile\_imgという名前で作成します。

その後、user\_profile\_imgを選択した状態で副クリック（右クリック）を押してメニューを表示し、「情報を見る」項目を選択してフォルダの詳細情報を表示します。

詳細情報画面の一番下までスクロールすると「共有とアクセス数」という項目があるので、その中にリスト表示されている一番下「everyone」の権限を「読み出しのみ」から「読み / 書き」に変更しましょう。

[フォルダ / ファイルの権限変更方法](https://support.apple.com/kb/PH25287?locale=en_US&viewlocale=ja_JP) 

#### **一意のファイル名を生成**

アップロード先フォルダの準備ができたので、次はアップロードする画像ファイル本体の準備。

WEBシステム、特に複数のユーザーが使用するSNSはさまざまなユーザーが自由に画像をアップロードすることができます。しかし、もし仮にアップロードする画像名が重複していた場合前の画像ファイルを後から入ってきた画像ファイルが上書きしてしまうため、画像名を毎回一意（重複しない）になるように変更してからアップロードする必要があります。

今回は、アップロードした瞬間の時間を取得し、ファイル名の最初につけることである程度重複を防ぐ処理を実装します。

時間のフォーマットはyyyyMMddhhmmss、仮に2017年08月25日07時18分20秒にhiroshi.jpg画像をアップロードすると下記例のような名前になるようにプログラムを組みます。

##### **ファイル名の例**

'20170825071820hiroshi.jpg'

この時間を文字列で取得するために、date()関数を使用します。date()関数を()の中に使用したいフォーマットをセットすることでその瞬間の時間を文字列で取得することができる関数です。

[PHPで日付関数を使いこなす（date, strtotime）](http://qiita.com/shuntaro_tamura/items/b7908e6db527e1543837)

前回画像ファイル名を取得し変数化した$file\_nameと合わせて、送信画像ファイル名を作成するプログラムは下記になります。

**<?php**  
 date\_default\_timezone\_set('Asia/Manila'); // フィリピン時間に設定

$date\_str **=** date('YmdHis'); *// YmdHisを指定することで取得フォーマットを指定*  
 $submit\_file\_name **=** $date\_str **.** $file\_name;  
**?>**

#### **画像アップロード**

アップロードする際の一意な画像ファイル名が作成できたので、いよいよ画像をアップロードします。

PHPでは画像のアップロードにmove\_uploaded\_file()関数を使用します。この関数はデータ送信時、一時的にブラウザに保存された画像データ（テンポラリーファイル）と、アップロード先のパスを指定することでデータをアップロードする機能を持った関数です。

##### **move\_uploaded\_file()関数の書き方**

move\_uploaded\_file(テンポラリーファイル, アップロード先パス)

テンポラリーファイルは$\_FILES['キー']['tmp\_name']で取得でき、キーはinput type="file"のname属性値input\_img\_nameです。

アップロード先パスはregisterフォルダ上にあるsignup.phpから見てひとつ階層を戻ったところにあるuser\_profile\_imgフォルダの中なので、../user\_profile\_img/がアップロード先のパスになります。

今回はアップロードする際、画像名を先ほど作成した$submit\_file\_nameの文字列に変更したいので、上記パスに更に繋げて指定します。

**<?php**  
 *// '../user\_profile\_img/' . $submit\_file\_name と文字連結をすることで*  
 *// ../user\_profile\_img/20170825071820hiroshi.jpg のような保存先を指定しています。*  
 move\_uploaded\_file($\_FILES['input\_img\_name']['tmp\_name'], '../user\_profile\_img/' **.** $submit\_file\_name);  
**?>**

上記までのプログラムをすべてエラーがなかったときのif文の中にまとめ、最終的に出来上がるプログラムを完成させます。

##### **signup.php**

**<?php**  
 $errors **=** [];  
  
 **if** (**!empty**($\_POST)) {  
 ・・・  
 *// 画像名を取得*  
 $file\_name **=** $\_FILES['input\_img\_name']['name'];  
 **if** (**!empty**($file\_name)) {  
 $file\_type **=** substr($file\_name, **-**3); *// 画像名の後ろから3文字を取得*  
 $file\_type **=** strtolower($file\_type); *// 大文字が含まれていた場合すべて小文字化*  
 **if** ($file\_type **!=** 'jpg' **&&** $file\_type **!=** 'png' **&&** $file\_type **!=** 'gif') {  
 $errors['img\_name'] **=** 'type';  
 }  
 } **else** {  
 $errors['img\_name'] **=** 'blank';  
 }  
 }  
  
 *// エラーがなかったときの処理*  
 **if** (**empty**($errors)) {  
 $date\_str **=** date('YmdHis');  
 $submit\_file\_name **=** $date\_str **.** $file\_name;  
  
 move\_uploaded\_file($\_FILES['input\_img\_name']['tmp\_name'], '../user\_profile\_img/' **.** $submit\_file\_name);  
 }  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
</html>

これでデータ送信時、すべてのバリデーションを通過し$errorsが空であればアップロード先に画像データが送信されます。

### **2. セッションヘ送信データを保存する**

最後に、送信データを次の入力情報確認ページへ受け渡すためのプログラムを完成させます。

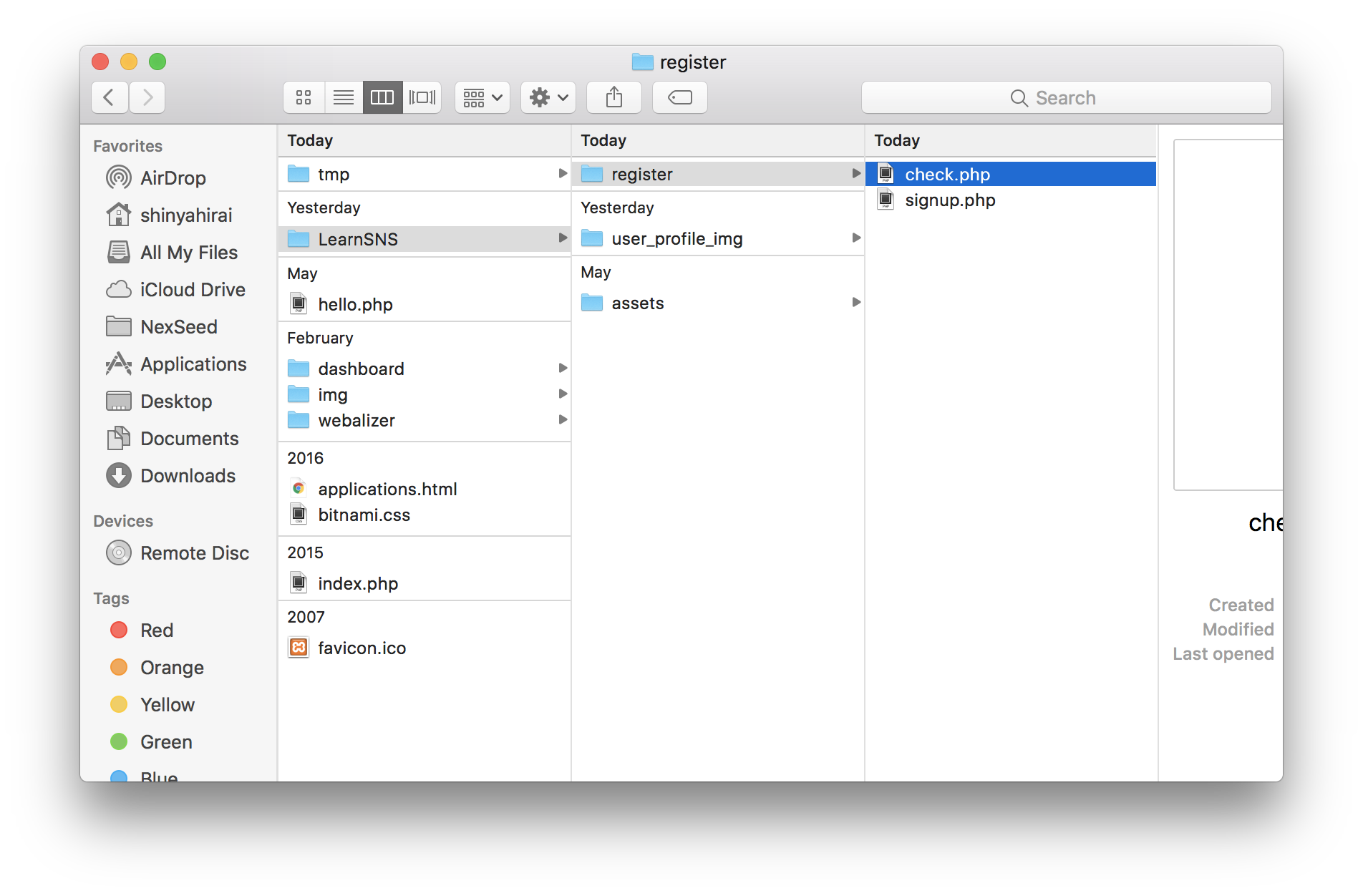
まず、画像のアップロードが終了したら次のcheck.phpへ遷移するようheader()関数を使って遷移処理をし、check.phpを作成します。

##### **signup.php**

**<?php**  
 $errors **=** [];  
 ・・・  
 *// エラーがなかったときの処理*  
 **if** (**empty**($errors)) {  
 $date\_str **=** date('YmdHis');  
 $submit\_file\_name **=** $date\_str **.** $file\_name;  
  
 move\_uploaded\_file($\_FILES['input\_img\_name']['tmp\_name'], '../user\_profile\_img/' **.** $submit\_file\_name);  
  
 header('Location: check.php');  
 **exit**();  
 }  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
</html>

registerフォルダ内にcheck.phpを作成します。

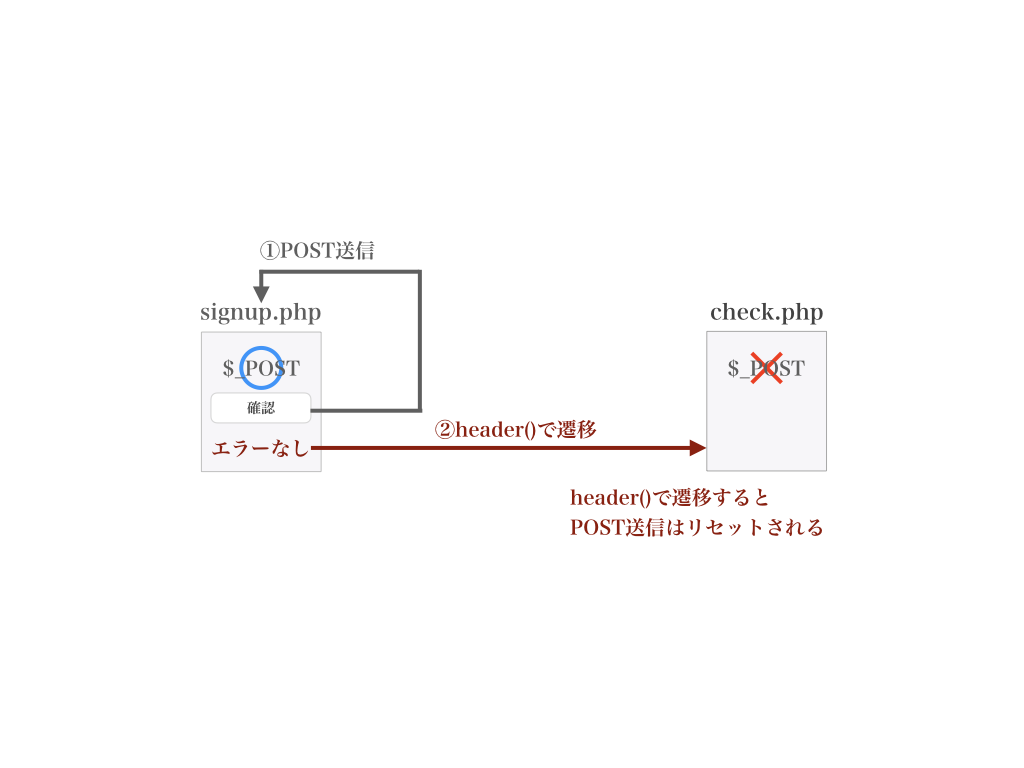
register/check.php



作成できたら、さっそく送信データをcheck.phpで受け取り、画面に出力してみます。

ここでよくある勘違いが、ユーザーが入力したデータは**POST送信**したので、check.phpで$\_POSTを使って取得できる。と思ってしまうことです。

ここまでのプログラムの流れをよく振り返ってみましょう。

signup.phpのformタグがPOST送信するときの送信先は自分自身、つまりsignup.phpです。 

上記遷移の詳細を表した図の①がformを送信した際の矢印、このときはPOST送信されているので$\_POSTを使用することができます。

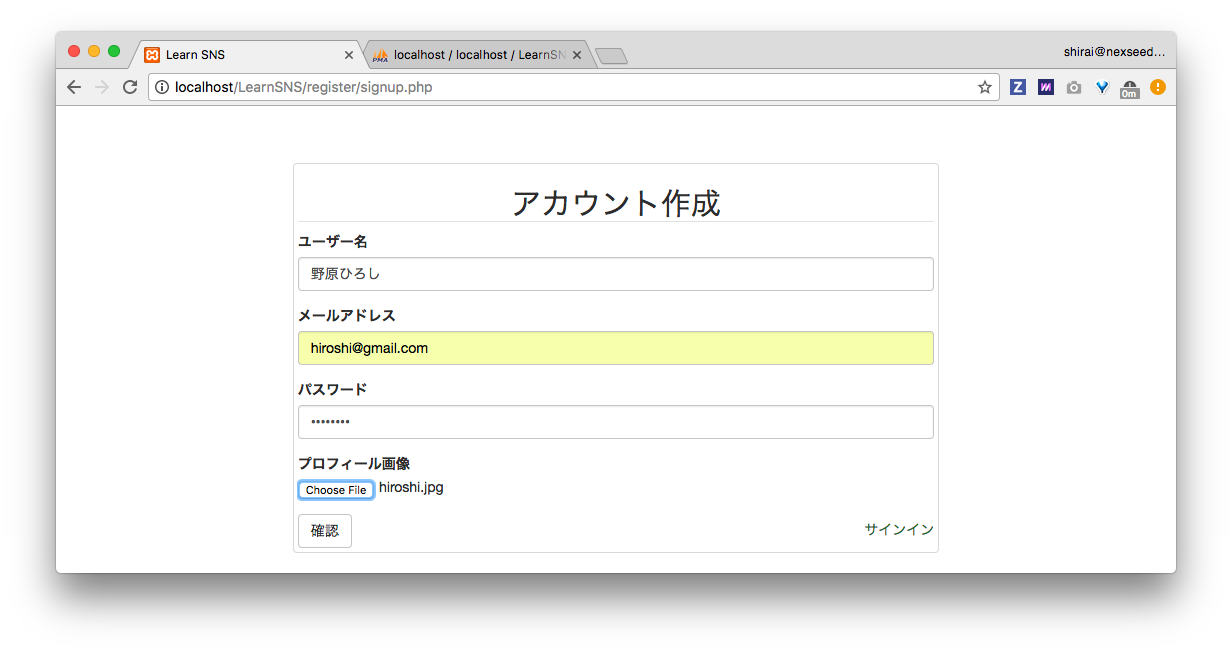
一方、実際に画像のアップロード処理などを終えてcheck.phpに遷移する際はheader()関数を使用しているので、この時点でPOST送信はリセットされ、check.phpでは使用不可となります。

試しにcheck.phpに下記$\_POSTをvar\_dump()で出力するプログラムを記述し、ブラウザで確認してみましょう。signup.phpから情報を入力し、送信した上でcheck.phpを確認することで画像が正しくアップロードされているかも確認できます。

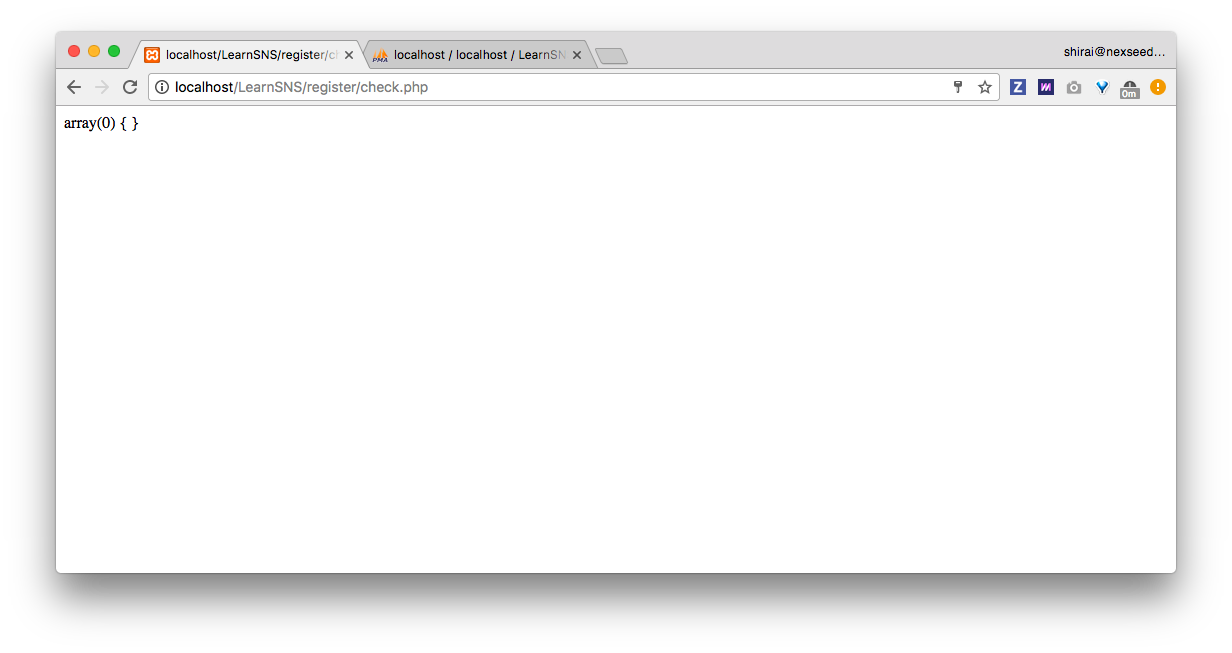
##### **check.php**

**<?php** *// check.phpへの初回書き込みなので、<?phpと?>のphpタグの記述をお忘れなく*  
 var\_dump($\_POST);   
**?>**

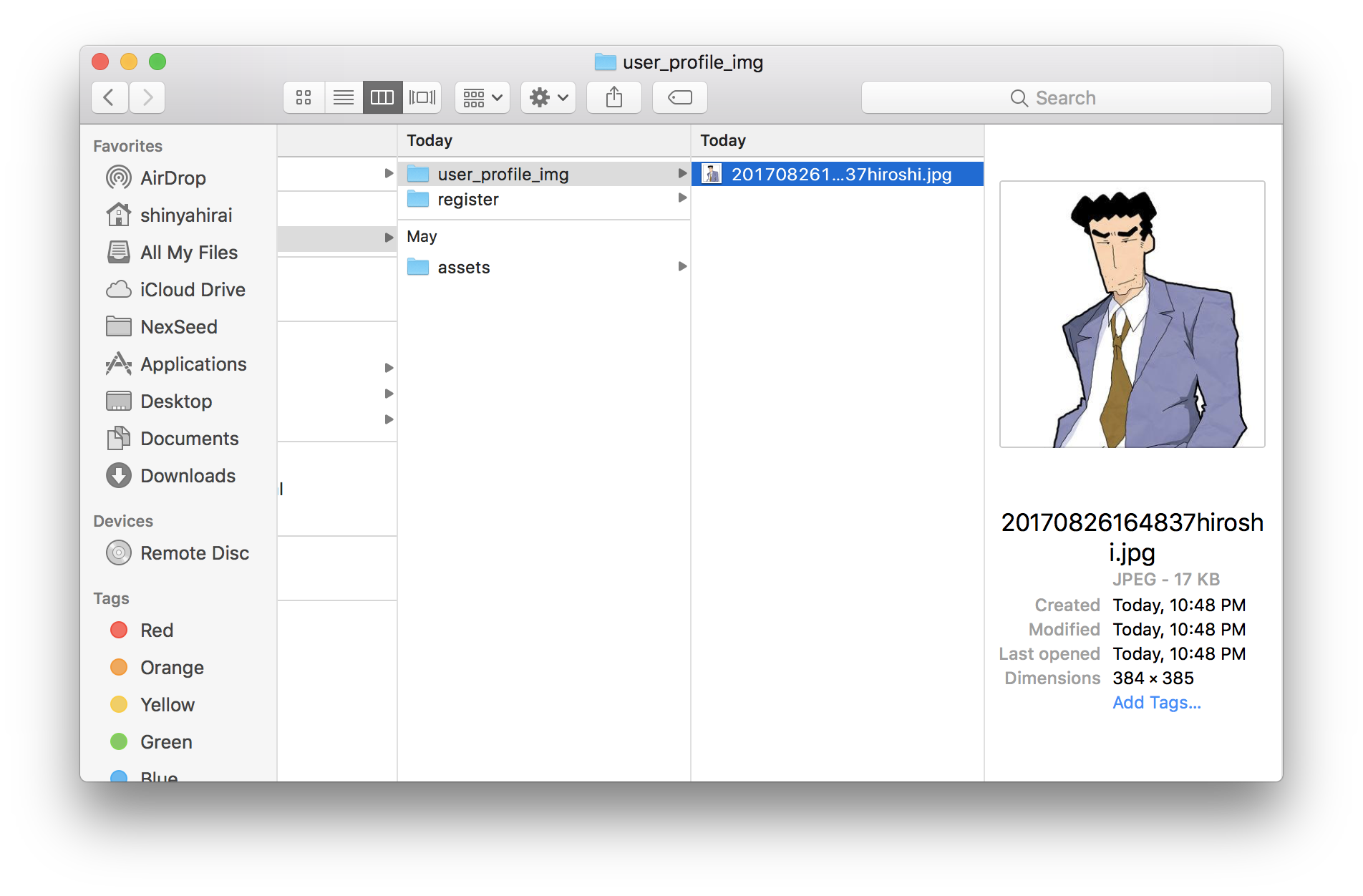
##### **signup.phpで情報を入力し送信**

****

##### **check.phpが表示されるが、var\_dump()で出力した$\_POSTは空**

****

##### **user\_profile\_imgフォルダ内に指定したフォーマット画像名で画像がアップロードされている**

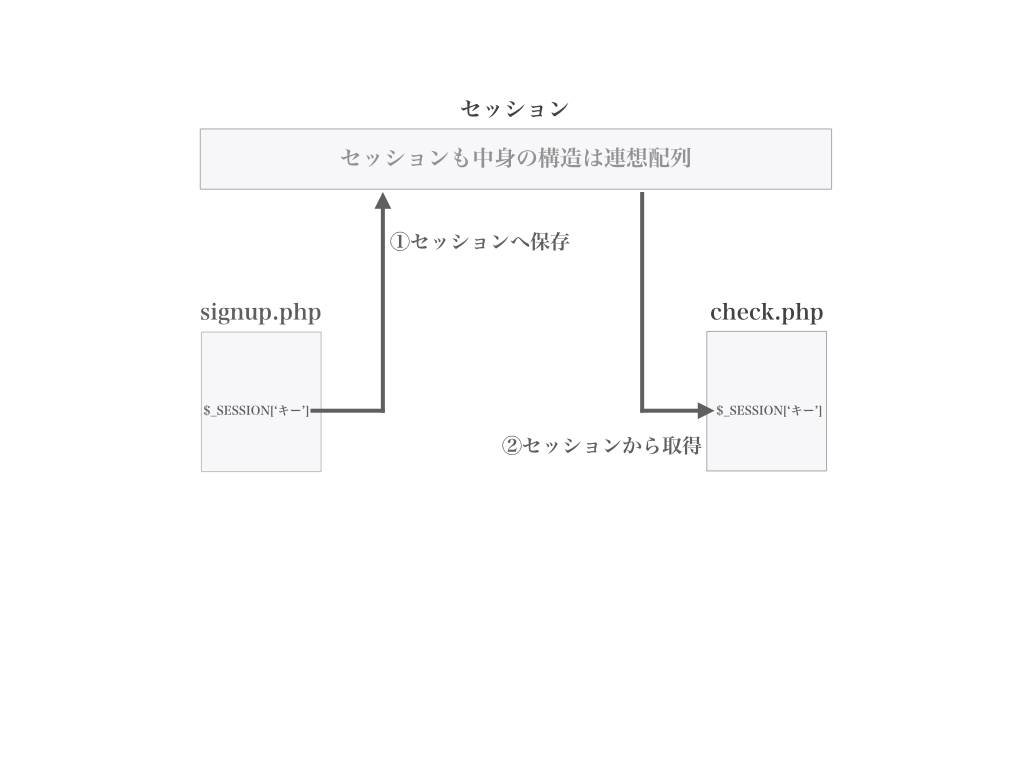
****

このように、header()関数で遷移した場合check.phpでは$\_POSTが空になり使用不可となるため、別の仕組みを利用して送信データを受け渡す必要があります。

そこで利用するのが**セッション**という機能。

セッションとは、各サーバーに用意された一時的にデータを保存することができる機能（サーバーに備え付けられた簡易データベースのようなもの）。

セッションに情報を保存することで、サーバー内であればどのファイルからでも同じデータを受け渡すことができます。



上記図のようにセッションを使ってsignup.phpで入力されたユーザー情報をcheck.phpへ受け渡すために、PHPではスーパーグローバル変数の$\_SESSIONを使用します。

$\_SESSIONは連想配列形式で値をセッションに保存し、また取得することができるスーパーグローバル変数です。

使用条件として、ファイルの最初にsession\_start()を宣言する必要があるので、注意が必要です。

それではまずsession\_start()を2枚のPHPファイルの最初に宣言し、$\_SESSION使用の準備をします。

##### **signup.php**

**<?php**  
 session\_start();  
 $errors **=** [];  
 ・・・  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
</html>

##### **check.php**

**<?php**  
 session\_start();  
 *// var\_dump()は削除してしまいましょう。*  
**?>**

条件が整ったので、画像アップロード後にセッションへ送信データをそれぞれ保存するプログラムを記述します。

##### **signup.php**

**<?php**  
 $errors **=** [];  
 ・・・  
 *// エラーがなかったときの処理*  
 **if** (**empty**($errors)) {  
 $date\_str **=** date('YmdHis');  
 $submit\_file\_name **=** $date\_str **.** $file\_name;  
  
 move\_uploaded\_file($\_FILES['input\_img\_name']['tmp\_name'], '../user\_profile\_img/' **.** $submit\_file\_name);  
  
 $\_SESSION['LearnSNS']['name'] **=** $\_POST['input\_name'];  
 $\_SESSION['LearnSNS']['email'] **=** $\_POST['input\_email'];  
 $\_SESSION['LearnSNS']['password'] **=** $\_POST['input\_password'];  
 *// 上記3つは$\_SESSION['register'] = $\_POST;という書き方で1文にまとめることもできます*  
 $\_SESSION['LearnSNS']['img\_name'] **=** $submit\_file\_name;  
  
 header('Location: check.php');  
 **exit**();  
 }  
**?>**  
<!DOCTYPE html>  
<html lang="ja">  
・・・  
</html>

セッションはサーバー内すべてのファイルで共通しているということもあり、LearnSNSというキーを設け多次元配列化し他のシステムとの重複を防いだ上で各情報を保存しています。

## **check.phpで送信データ表示の確認**

このページの最後に、入力情報をcheck.php側でセッションから取得し出力してみましょう。

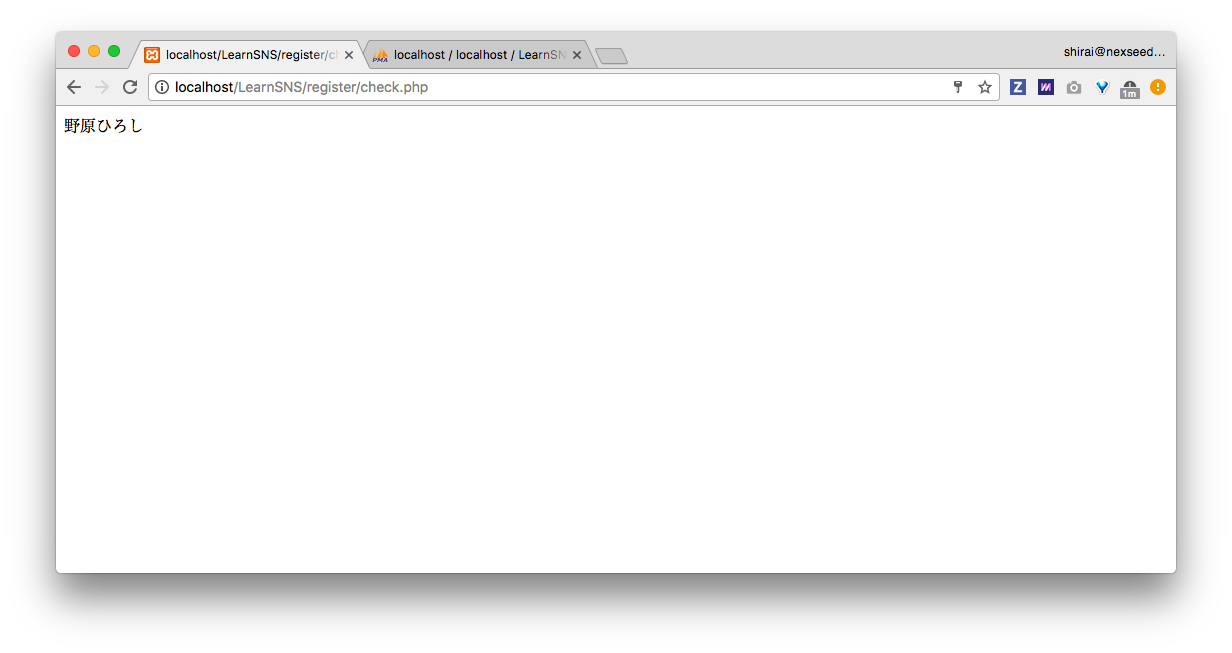
まずcheck.phpにユーザー名をセッションから取得し出力するテストプログラムを記述します。

##### **check.php**

**<?php**  
 session\_start();  
  
 *// 出力テスト*  
 **echo** $\_SESSION['LearnSNS']['name'] **.** '<br>';  
**?>**

その後、一度signup.phpから入力処理をやり直しcheck.phpでユーザー名が出力されることを確認します。

##### **一度signup.phpからやり直さず前回送信したままcheck.phpを更新してもsignup.phpに先ほど記述した$\_SESSIONへの保存プログラムは未実行のため、check.php側では$\_SESSIONにnameキーが登録されていないというエラーがでてしまうので注意が必要です。**

****

#### **▼練習問題2**

ユーザー名以外の情報もすべて同じようにecho文でcheck.phpに出力してみる。

#### **▼練習問題2の解答プログラム**

##### **check.php**

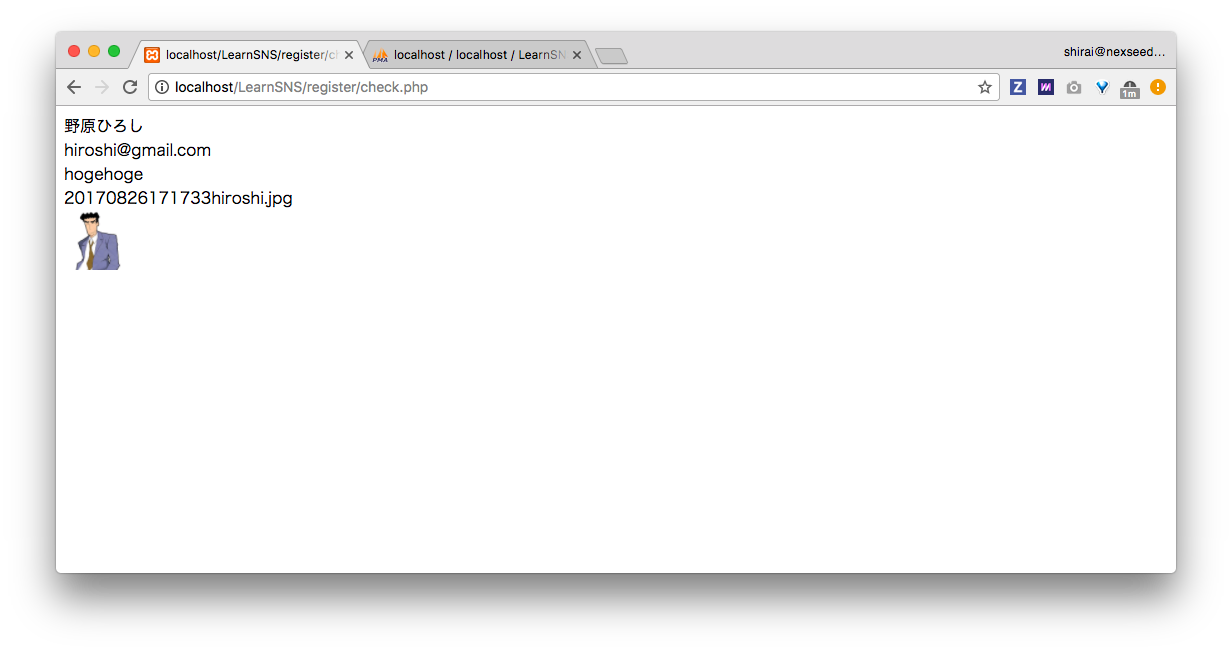
**<?php**  
 session\_start();  
  
 **echo** $\_SESSION['LearnSNS']['name'] **.** '<br>';  
 **echo** $\_SESSION['LearnSNS']['email'] **.** '<br>';  
 **echo** $\_SESSION['LearnSNS']['password'] **.** '<br>';  
 **echo** $\_SESSION['LearnSNS']['img\_name'] **.** '<br>';  
**?>**

#### **▼練習問題3**

上記画像名を文字列で出力しているプログラムを元に、imgタグを使用して画像として画面に出力してみる。（画像の出力サイズは横幅が60pxになるよう調整）

##### **check.php**

**<?php**  
 session\_start();  
  
 **echo** $\_SESSION['LearnSNS']['name'] **.** '<br>';  
 **echo** $\_SESSION['LearnSNS']['email'] **.** '<br>';  
 **echo** $\_SESSION['LearnSNS']['password'] **.** '<br>';  
 **echo** $\_SESSION['LearnSNS']['img\_name'] **.** '<br>';  
**?>**  
  
<!DOCTYPE html>  
<html lang="ja">  
<head>  
 <meta charset="utf-8">  
 <title></title>  
</head>  
<body>  
 <img src="../user\_profile\_img/**<?php** **echo** $\_SESSION['LearnSNS']['img\_name']; **?>**" width="60">  
</body>  
</html>



# **まとめ**

以上でsignup.phpの実装が完了しました。

### **実装した機能まとめ**

* バリデーション
* 画像アップロード
* セッション