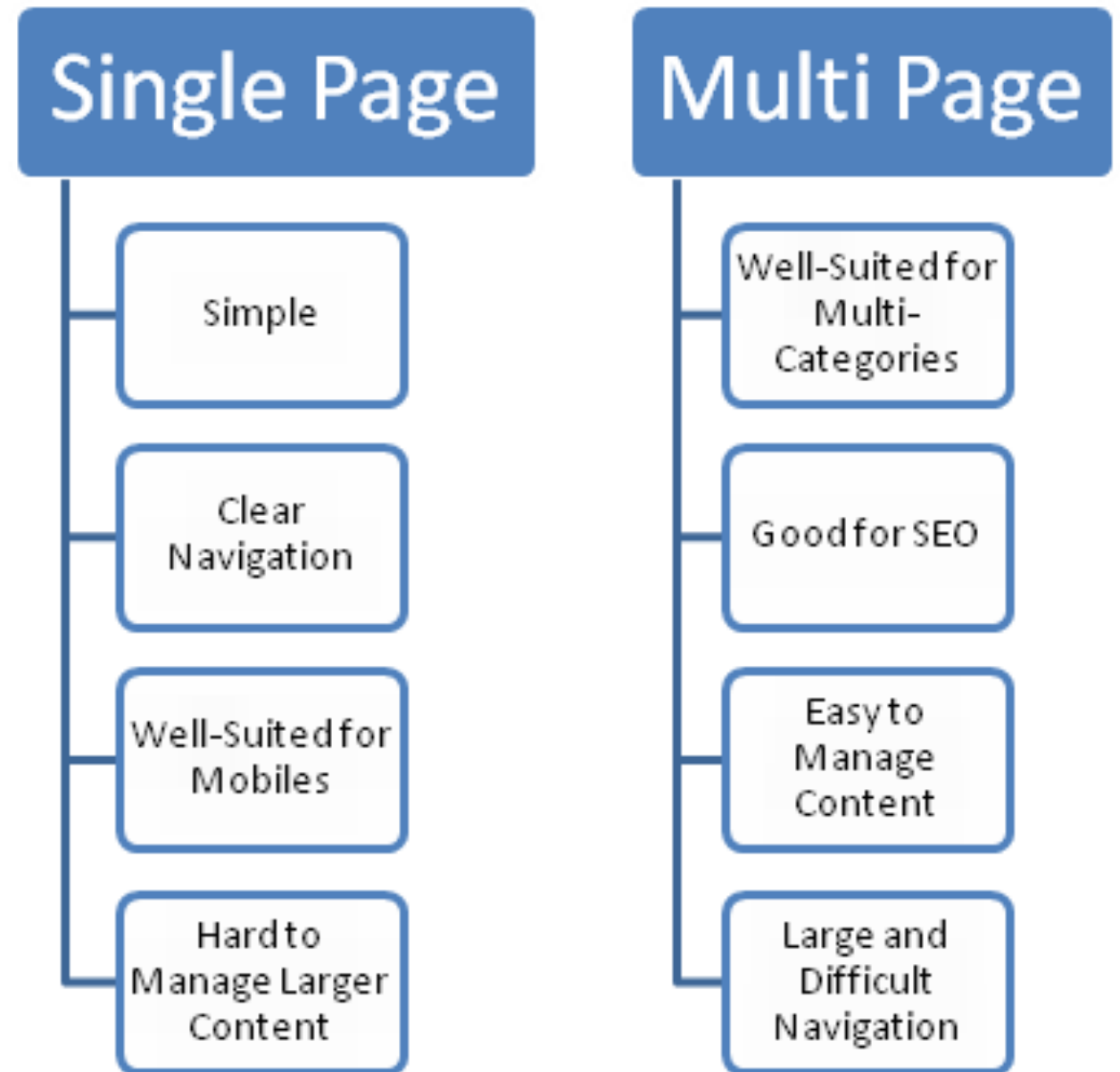


MULTI PAGE SITES

Rick Mercer



Single Page Applications

- So far, all websites have been Single Page Applications
 - Single-Page Applications (SPAs) are Web apps that load a single HTML page and dynamically update that page as the user interacts with the app
- The Quotations Service is Multi page
 - You still use HTML, AJAX, JS, and PHP
 - However you will link to a different page for Registration, Login, and Add Quote (you will `$_SESSION`)
 - You are also required to have different functionality when is logged in (you will `$_SESSION`)


Quotations Service

- An approach to building your multi-page site
 - 1) Design your data base, with two tables
 - At first, you only need the quotations, users later
 - 2) Populate the tables with a small number of records
 - 3) Test your DB functions in the context of model.php

Using code like we've seen in DataBaseAdapter

Run your tests as CLI applications

- 4) Build index.php page that shows all Quotations in the database
- 5) Add 3 pages and href links to each



"Walking on water and developing software from a specification are easy if both are frozen."	
--Edward Berard	
<input type="button" value="7"/>	<input type="button" value="flag"/>

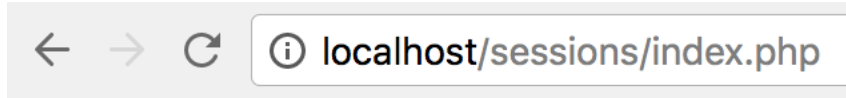
" Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it in the same way twice."	
--C. Alexander	
<input type="button" value="5"/>	<input type="button" value="flag"/>

"Today is always a nice day"	
--Rick	
<input type="button" value="5"/>	<input type="button" value="flag"/>

Multi Page Case Study

- Let's build a multi-page static (no DB) website
 - Has a php home page with links to
 - an html login page
 - an html page to allow withdrawals and deposits
 - When logged in the user can make deposits and withdraws to a fake BankAccount

1) At start up

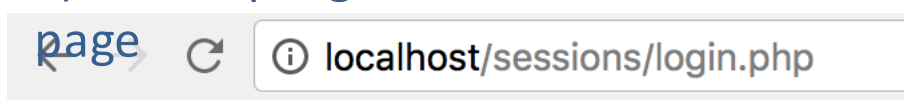


Bank Teller

[Login](#)
[Transaction](#)

Logout

2) Load up login

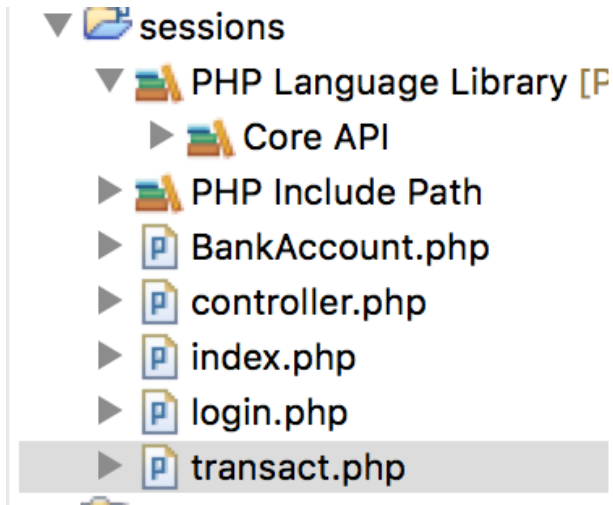


Login

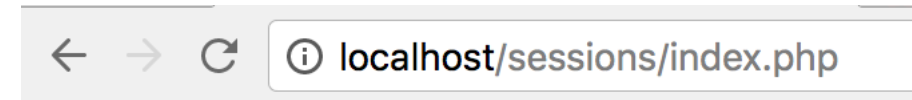
User ID:

Password:

Code Demo



3) After loading transact.php to deposit 22.22, index.php is loaded with header



Bank Teller

[Login](#)
[Transaction](#)

Logout

Current Balance: 22.22

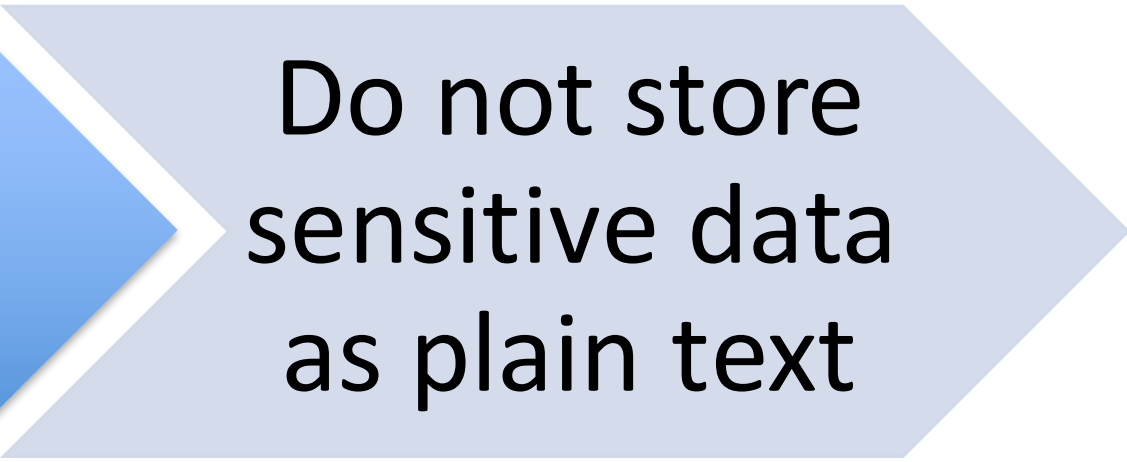
See Screencast of Code Demo

- Source code available as `session.zip`

See Screencast of Code Demo



Our first
security
issue



Do not store
sensitive data
as plain text

Registering new account

Salt and hash the passwords before storing the long string in the data base.

Register

Username

Password

Register

Passwords should not be plain text

- Spec shows we need to add a new table

```
CREATE TABLE users (  
    id int(6) unsigned AUTO_INCREMENT,  
    username varchar(64) ,  
    hash varchar(255) ,  
    PRIMARY KEY (id)  
);
```

Passwords should not be plain text

- Should convert the plain text password in the database column '**hash**' **varchar(255)**
 - Must have 60 characters at least
- Many passwords are stored in plain text
 - Is this a good idea?
- Do not store the password in plain text format
- Hash *and* salt the password
 - Read <http://blog.codinghorror.com/youre-probably-storing-passwords-incorrectly/>

Hash and Salt

- A hash is cryptographic algorithm that converts data into a fixed length string
 - Looks nothing like the actual password
 - A hash is also a one-way function which means that there isn't a function to reverse or undo a hash
- A salt is a bit of additional random data that makes a hash significantly more difficult to crack

\$2y\$10\$6z7GKa9kpDN7KC3ICW1Hi.f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K

The diagram shows a bcrypt hash string: `$2y$10$6z7GKa9kpDN7KC3ICW1Hi.f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K`. The string is color-coded to show its components: `$2y$` is red, `10$` is blue, `6z7GKa9kpDN7KC3ICW1Hi.` is green, and `f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K` is orange. Brackets below the string point to labels: a red bracket for `$2y$` is labeled "Algorithm", a blue bracket for `10$` is labeled "Algorithm options (eg cost)", a green bracket for `6z7GKa9kpDN7KC3ICW1Hi.` is labeled "Salt", and an orange bracket for `f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K` is labeled "Hashed password".

Algorithm

Algorithm options (eg cost)

Salt

Hashed password

PHP

`$2y$10$6z7GKa9kpDN7KC3ICW1Hi.f0/to7Y/x36WUKNP0IndHdkdR9Ae3K`

- Algorithm
- Algorithm options (eg cost)
- Salt
- Hashed password

- PHP's `password_hash` to hash and salt data

```
<?php
```

```
$pwd = '1234'; // ← What the user typed as plain text
```

```
$hashed_pwd = password_hash($pwd, PASSWORD_DEFAULT);
```

```
echo $hashed_pwd; // Store this value (see below), not '1234'
```

```
?>
```

- Run this code 4 times to see what a hacker would have to decrypt, the same hashed password is not 1234

```
$2y$10$xDyg2vxlMnGaUNPGcbHQIeljUs4Jqdofoz3F4P1/1ElF.vR5EV5.K
```

```
$2y$10$Xlt0p2UGaMvj1/S7rFe7c048wgChpQ3k4cdfTRGKM3AoJAzODKa20
```

```
$2y$10$9tL9WKXhy2Ejtv6f4YAXGOPTbft5QcR1dR0zaZkyWlv1ec/UxwLg2
```

```
$2y$10$.s9QJxb4pzhSafK9tfu8sOuOcOIcPeiYiCZ2qCS/wwDS3Vtx.mW1W
```

Store the salted hash, not '1234'

- Use varchar(255) in column hash

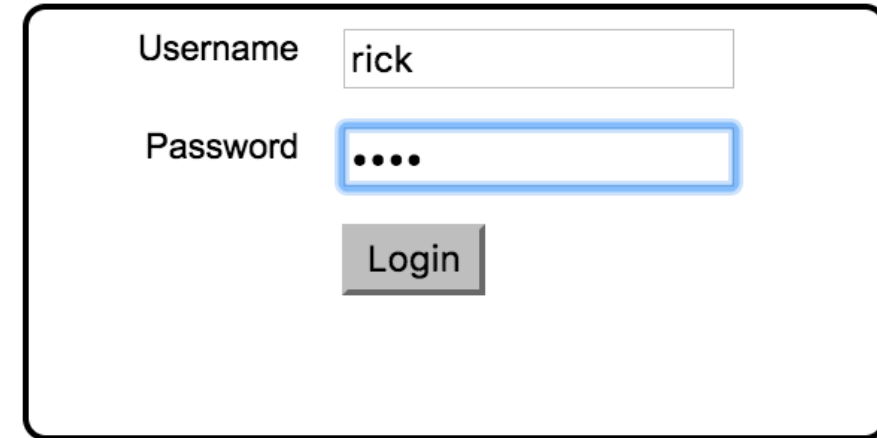
```
MariaDB [quotes]> select * from users;
```

```
+-----+-----+-----+
| id | username | hash |
+-----+-----+-----+
| 8 | abc | $2y$10$j08h21hxrXH5fyls5cnx3eGD9eU4BBDNCPeJF7ApiAkY/pfKvBbOa |
| 9 | def | $2y$10$ogcRkd7b0lVQR8kbjLWcV.HjiQh2ws1IqAsYvjt6tavEGWIZFTTOO |
| 10 | ghi | $2y$10$uCKqOOZap1mNn/iVQ0uIp.GCW9ulhRNWpuYOHsxh8N5mUaxqzseqK |
| 14 | fourth | $2y$10$Pm82DMymVUYASmSkCKMiu.fO5fF8EVWgQj0TFxA1GiU51Bc6uejnG |
| 15 | rick | $2y$10$aAPlbj/.rQ4jFQw2LkIws./IWzUUXCgoGvLre9CKy4dhwVtg7jCF6 |
| 16 | ankprw | $2y$10$stD06Sxm8/FdPE/6r4RVruMdugzxox/j2ivMGzd.Est7xYCjezcMu |
+-----+-----+-----+
6 rows in set (0.00 sec)
```

Reminder: Logging in

- You should have a form for the user to input information
- Verify the user's credentials
- Notify if the password doesn't match
- Use input type "password" in case someone is looking over a shoulder
 - Or the professor forgets his screen is in front of 100 students and its recorded on Panopto

Login



A login form with a rounded rectangular border. It contains two input fields and a button. The first field is labeled 'Username' and contains the text 'rick'. The second field is labeled 'Password' and contains four dots, indicating a password input type. Below the password field is a button labeled 'Login'.

Username	<input type="text" value="rick"/>
Password	<input type="password" value="...."/>
<input type="button" value="Login"/>	

Confirming passwords

- `password_verify` is the other useful needed PHP function

`boolean password_verify(string $pwd, string $hash)`

- Get the plain text password—`$pwd`—from the form and the salted hashed password—`$hash`—from the data base

```
<?php
```

```
$pwd = '1234'; // Assume this is entered in the login form
```

```
$hash = password_hash($pwd, PASSWORD_DEFAULT);
```

```
echo password_verify('A' . $pwd, $hash) . PHP_EOL; // false, no output
```

```
echo password_verify('1234', $hash) . PHP_EOL; // true, output 1
```

```
?>
```

New things needed in Quotations

- `session_start()`
 - at the top of every page
- `$_SESSION[]`
 - A global array accessible to all pages
 - Use this in a PHP block to
 - know if someone is logged in
 - set an error: Allow the login page know login failed to report problem
- `header ('Location: quotes.php');`
 - This statement will load the page quotes.php

New things needed in Quotations

- `isset($_GET['ID'])`
 - So the controller can tell what page sent the form
- `unset` or `session.destroy()`
 - to log out user
- Remember to use `session_start` at top of each file

```
session_start (); // Do this in every file before accessing $_SESSION
$_SESSION ['user'] = $username;
```