Dokumentation zur Prüfungsvorbereitung

Fach: Informatik (mündliche Prüfung)

Prüfer: Stephan Seidel

Abgabetermin Dokumentation: 26.06.2017

Inhalt

*

Ausblick auf den geplanten Ablauf der Präsentation

Meine Präsentation wird sich mit folgenden Gesichtspunkten bezogen auf die Aufgabenstellung befassen:

- Erläuterung der Ausgangssituation (aktueller Zustand der Blinkfair-Website)
- Analyse und Erläuterung von technischen Problemen
- Begründung der Entscheidung der kompletten Neuentwicklung eines Bestellformulars
- Überblick über genutzte Entwicklungsmodelle und Techniken
- Konkrete Erläuterungen der gegenüber der Original-Website vorgenommenen Verbesserungen
- Erläuterung von Sicherheitsaspekten, die bei der Erstellung eines Online-Bestellformulars beachtet werden müssen

Probleme der bisherigen Website/des bisherigen Bestellformulars

Nachfolgend beschäftige ich mich analytisch mit den Problemen der bisherigen Blinkfair-Website:

Quellcode

Der Quellcode der Website liegt in großen monolithischen Dateien vor, die teils auch bisher zu vier Auszeichnungs-/Programmiersprachen (HTML, CSS, PHP und JavaScript) parallel enthalten. Dies bringt diverse Nachteile mit sich:

1. Würde die Website ein Projekt darstellen, welches regelmäßig unterschiedlichen Entwicklergruppen wie Designer und Programmierer verändert wird, könnten mehrere Teile nicht parallel entwickelt werden, da der Programmierer unter Umständen für Code-Änderungen an denselben Dateien gleichzeitig arbeiten muss wie der Designer.

2. Zum anderen stiftet dieses Vorgehen teils sehr starke Verwirrung, da gerade JavaScript und PHP sich in vielen syntaktischen Eigenschaften sehr ähnlich sind. Dies ist dadurch bedingt, dass sich diese beiden auf das Web spezialisierten Programmiersprachen über verschiedene Pfade (PHP: Perl, JavaScript: Java) letztendlich an der hohen Programmiersprache C++ orientieren und dadurch wie im folgenden Beispiel zu sehen, Ähnlichkeiten aufweisen:

```javascript
var $i = 0;
while ($i < 10) {
    $i++;
}

document.write($i.toString());
```

```php
$i = 0;
while ($i < 10) {
    $i++;
}

print $i;
```

Beide Codebeispiele (links JavaScript-Code, rechts PHP-Code) tun das Gleiche: Sie geben die Zahl 10 aus. Durch die Verwendung des Dollarzeichens im JavaScript-Variablennamen (nicht unbedingt zwingend) sehen sich beide zum Verwechseln ähnlich, nur die Variablen-Deklaration und Ausgabe unterscheiden sich. Daher sollte man zur Übersicht die gleichzeitige Verwendung von JavaScript und PHP in einer Datei vermeiden.

3. Zum anderen ist mittlerweile im HTTP-Standard die sogenannte **Content Security Policy (CSP)** vorgesehen, diese sorgt dafür, sofern vom Webserver gesetzt, dass ein Webbrowser wie Firefox Inline (Eingebetteten)-Code in HTML-Dateien bezogen auf JavaScript und CSS ignoriert, um zum Beispiel über XSS (Cross Site Scripting) die Einschleusung von böswilligen Code zu verhindern. Um die eigene Website gegen derartige Angriffe zu schützen, ist es sehr empfehlenswert, die CSP auch einzusetzen, dies wäre mit der aktuellen Blinkfair-Website jedoch nicht möglich, da dadurch auch der legitime JavaScript- und CSS-Code außer Kraft gesetzt wird.

Mögliche Lösungsansätze

Als erster Schritt sollte natürlich in Erwägung gezogen werden, **Strukturierung**, **Programmierung** und **Darstellung** der Webseite komplett zu trennen. Dies bedeutet konkret:

- Die an den Webbrowser ausgelieferten HTML-Dateien enthalten wirklich nur HTML und kein CSS oder JavaScript
- JavaScript- und CSS-Code wird in separate CSS- und JS-Dateien ausgegliedert und entsprechend über `<link rel="stylesheet" type="text/css" href="pfad/zu/datei.css" />` beziehungsweise `<link type="application/javascript" href="pfad/zu/datei.js" />` in das HTML-Dokument eingebunden.

Diese Änderung macht den Code übersichtlicher und somit einfacher zu warten, außerdem wird er wie schon eben erwähnt, damit kompatibel mit der CSP.

Das nächste, schwieriger zu lösende Problem ist, die starke Vermischung von PHP- und HTML-Code. Dazu sollte man in Bezug auf das Beispiel mit einem Projekt, an dem ein Programmierer und Designer arbeiten, beachten, dass HTML-Code meistens eher von dem Designer bearbeitet werden würde und der PHP-Code vom Programmierer, daher ist auch hier auch eine Trennung sinnvoll. Die einfachste Möglichkeit wäre den PHP-Code, in einen Backend- und Frontend-Teil zu teilen, dies bedeutet nur der PHP-Code, der direkt die Ausgabe der Website beeinflusst kommt zusammen mit dem HTML in eine Datei, der Rest (wie zum Beispiel Datenbank-Abfragen) wird in separate Dateien ausgelagert.

Eine noch weitergehende Möglichkeit wäre die Verwendung von Layout-Engines wie **Smarty** oder **Twig**. Diese sind zwar selber grundsätzlich in PHP programmiert - stellen also an den Webserver meistens keine besonderen Zusatzanforderungen, bieten aber einen von PHP komplett unabhängigen Syntax, der „unauffälliger“ ist als Inline-PHP-Code:

```
<ul>
<?php
$users = ['Alice', 'Bob', 'Chris',
'Dolores', 'Ernie', 'Franz'];

foreach ($users as $user) {
    print '<li>'.$user.'</li>';
}
?>
</ul>
```

```
{% set users = ['Alice', 'Bob',
'Chris', 'Dolores', 'Ernie',
'Franz'] %}

<ul>
{% for users in user %}
    <li>{{ user }}</li>
{%  endfor %}
</ul>
```

*Beide Codebeispiele (links PHP-Code, rechts Markup der Twig Layout Engine) geben eine Liste der angegebenen User aus. Der Twig-Code ist allerdings übersichtlicher, da der HTML-Code ohne **print**-Anweisung, anders als in PHP eingebunden werden kann und der Twig-Code nicht wie bei PHP mit Anfangs- und Endmarkern (**<?php … ?>**) umschlossen werden muss. Vorteile sind dadurch, dass Projektmitarbeiter wie Designer keine Erfahrungen mit PHP haben müssen und nur die einfache Twig-Syntax lernen müssen.*

Twig ist hauptsächlich durch das Web-Framework **Symfony** als dessen Hauslösung für das Template-Rendering bekannt geworden. Es lässt sich jedoch auch problemlos außerhalb von Symfony-Projekten verwenden, was die Integration zum Beispiel bei der Überarbeitung einer bestehenden Website ermöglicht, ohne sofort auf Symfony umsteigen zu müssen.

Encoding

Sämtliche PHP-Dateien sind leider bisher leider als **ISO-8859** kodiert, dies bedeutet, dass die Website so nur in der Lage ist, Zeichen des sogenannten ASCII-Alphabetes sowie einige westeuropäische Zeichen auszugeben. Bei PHP bestimmt das Encoding der Datei auch gleichzeitig, in welchem Encoding die Ausgabe an den Browser gesendet wird. Andere Zeichen (wie zum Beispiel, wenn ein Kunde seinen Namen mit einem osteuropäischen Sonderzeichen eingibt) würden fehlerhaft oder auch gar nicht ausgegeben werden. Daher sollte man sämtliche Dateien des Websiten-Projektes in **UTF-8-Kodierung** speichern.

Datenbank

Bei der Datenbank der alten Blinkfair-Website entstehen leider ebenfalls diverse Probleme. Es existiert keine zentrale Kopie des SQL-Datenbankschemas. Dies erschwert es für jemanden, der die Website später vielleicht pflegen möchte, zu verstehen, wie die Datenbank aufgebaut ist. Stattdessen sind die einzelnen Tabellen-Definition in mehreren PHP-Dateien verstreut, diese erstellen daraus die Datenbank-Tabellen bei Bedarf. Dies macht es schwer, das sogenannte Datenbank-Schema später einwandfrei rekonstruieren zu können, falls man zum Beispiel auf die originale Datenbank keinen Zugriff hat.

Leider hat die Datenbank außerdem essentielle Layout-Fehler, so wird das Datum als Plain Text gespeichert. Dies stört zwar prinzipiell das einfache Abfragen der Datenbank nicht, bringt aber diverse Nachteile mit sich: So können zum Beispiel mit einer Datenbank nicht nur Datensätze abfragt werden, die im Datumsfeld einen bestimmten Zeitraum haben, da der Datenbank-Server dafür ein bestimmtes Format benötigt und den einfachen Klartext nicht als Datum verarbeiten kann. MySQL, PostgreSQL und auch andere Datenbank-Systeme bieten dafür einen speziellen Feldtyp an, **Timestamp** genannt. Dieser speichert den gewünschten Zeitpunkt meistens in den Sekunden, die seit dem 01. Januar 1970 vergangen sind (sogenannte UNIX-Zeitstempel). Das hat den Vorteil, dass sich mit mathematischen Bedingungen recht einfach Filterkriterien bezogen auf den Zeitpunkt festlegen lassen:

```
SELECT * FROM orders WHERE date > now() - 60 * 60 * 24 * 365;
```

*Folgende SQL-Abfrage fragt alle Datensätze aus Tabelle **orders** ab, die im Feld **date*** maximal einen Zeitstempel haben, der ein Jahr zurückliegt.

Diese Schreibweise allerdings teils sehr unübersichtlich, beim Datenbankserver PostgreSQL kann man auch folgende leserfreundlichere Variante verwenden:

```
SELECT * FROM orders WHERE date > now() - '1 year'::INTERVAL;
```

*Die folgende PostgreSQL-Abfrage tut das Gleiche wie das generische SQL aus der vorherigen Code-Auflistung, hat aber den Vorteil, dass der gewünschte Zeitraum im Klartext gelesen werden kann. Das an die Zeichenkette angehängte **::INTERVAL** weist den Datenbankserver an, den String in einen entsprechenden Interval zu verwandeln (Zeitspanne eines gewissen Zeitraums – wie ein Jahr – in Sekunden).*

Zum PHP-seitigen Zugriff auf die Datenbank verwendet die alte Blinkfair-Website die alte PHP-MySQL-Schnittstelle (die Funktionen mit dem Namen, der mit ***mysql_**** beginnt), diese Funktionsaufrufe sollten ersetzt werden, da diese Schnittstelle zum einen als unsicher und veraltet gilt. Zum anderen ist die Schnittstelle mit PHP 7 komplett entfernt worden, das bedeutet, dass die alte Blinkfair-Website nicht mehr auf Webservern mit modernen PHP lauffähig ist. Zur Alternative stehen mehrere Funktionen zur Verfügung, zum einen wäre dies die neuere MySQLi-Schnittstelle von PHP, die sich in der Benutzung vom Vorgänger kaum unterscheidet. Sie ist daher sehr einfach als Ersatz zu benutzen, aber dafür nicht sehr zukunftsorientiert.

Eine bessere Alternative sind daher die **PHP Data Objects (PDO)**. Diese unterstützen ein Layer, was Vieles kapselt, das bedeutet, unterschiedliche Datenbank-Systeme können mit denselben Funktionen angesprochen werden, was die Portierung einer Website auf ein anderes Datenbanksystem erleichtert.
Zum anderen wird über **Prepared Statements** sogenanntes **Parameter Binding** unterstützt, das bedeutet, das zum Beispiel Parameter in Where-Klauseln zuerst mit Platzhaltern gefüllt werden und erst bei der Ausführung der Abfrage mit den eigentlichen Werten gefüllt werden. Vorteil ist hier, dass die Werte dabei so escaped (potenziell böswillige Zeichen, wie zum Beispiel ein Anführungszeichen (') oder Semikolon (;) werden maskiert) werden, dass sie keinen Einfluss auf den eigentlichen SQL-Code nehmen können – was ein gutes Mittel ist, um SQL-Injections abzuwehren. Eine SQL-Injection bedeutet das Einschleusen von unerwünschten SQL in eine Abfrage über Parameter, die der Benutzer beeinflussen kann:

Das Beispiel-Skript erwartet einen URL-Parameter ID, über den mitgeteilt wird, welcher Datensatz angezeigt werden soll, diese wird an eine SQL-Abfrage übergeben:

Erwarteter Aufruf:

Aufgerufene URL: `http://webserver/find.php?ID=42`
Erzeugtes SQL: `SELECT author, subject, text FROM artikel WHERE ID=42;`

SQL-Injection:

Aufgerufene URL: `http://webserver/find.php?ID=42;UPDATE+USER+SET+TYPE="admin"+WHERE+name="hacker"`
Erzeugtes SQL: `SELECT author, subject, text FROM artikel WHERE ID=42;UPDATE USER SET TYPE="admin" WHERE name="hacker";`

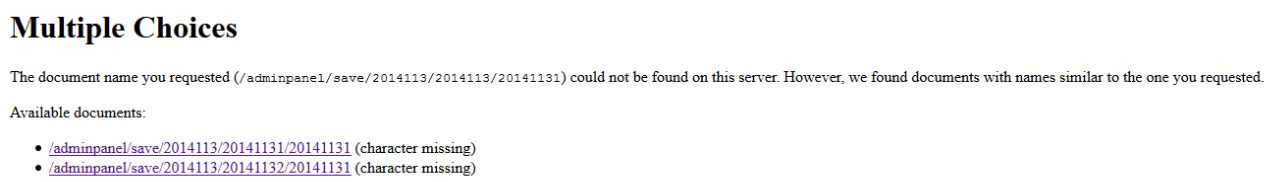
Da der URL-Parameter nicht ausreichend gefiltert wird, kann man über ihn an die eigentliche SQL-Abfrage eine weitere anhängen, die in dem Beispiel den angreifenden Benutzer zum Admin macht. Dies hätte zum Beispiel mittels Prepared Statements und Parameter-Binding bei PDO vermieden werden können. (Quelle: Wikipedia, leicht abgewandelt)

Datenschutz

In der bisherigen Blinkfair-Website steckt außerdem ein großes Datenschutz-Problem: Die eingegebenen Kundendaten werden als Textdatei auf dem Webspace im Unterordner ***adminpanel/save*** gespeichert und sind bei Kenntnis der entsprechenden URL auch direkt im Browser aufzurufen (z.B. http://blinkfair.stadtteilschule-blankenese.de/adminpanel/save/2014113/20141131), ohne jegliche Authentifizierung. Der einzige Schutz, der vor jeder Bestellung steht, ist eine index.html im gleichen Verzeichnis, die einen Redirect erzeugt:

```
<!DOCTYPE html><html><head><title>Blink Fair Handelsgenossen-
schaft</title><meta http-equiv="refresh" content="0; URL=http://
blinkfair.stadtteilschule-blankenese.de/index.html"></head><body></
body></html>
```

Dies ist aber kein zuverlässiger Schutz, bei jedem modernen Browser kann man diese Redirects, die in einer HTML-Datei definiert sind, ausschalten. Außerdem muss man zur Anzeige der eigentlichen Bestelldaten gar nicht über die index.html gehen. Es wurden auch keine Versuche unternommen, zum Beispiel mittels entsprechender Konfiguration des Webservers die Einsicht in diese Verzeichnisse aus dem Internet heraus zu verhindern. Außerdem ist zusätzlich der Webserver des Providers so konfiguriert, dass er, wenn man diese URL bewusst mit Tippfehlern eingibt, einem automatisch die richtige URL vorschlägt:



Multiple Choices

The document name you requested (`/adminpanel/save/2014113/2014113/20141131`) could not be found on this server. However, we found documents with names similar to the one you requested.

Available documents:

- /adminpanel/save/2014113/20141131/20141131 (character missing)
- /adminpanel/save/2014113/20141132/20141131 (character missing)

Dieser Umstand ist nicht sehr schön und außerdem datenschutzrechtlich sehr problematisch.

Geplante Verbesserungen

Ich plane aufgrund des schwer verständlichen bisherigen Quellcodes eine komplette Neuimplementierung zumindest des Bestellformulars. Dabei werde ich auf die im vorherigen Kapitel erläuterten Verbesserungsvorschläge zurückgreifen, dies bedeutet unter anderem die Verwendung von PDO für den Zugriff auf die Datenbank. Unter anderem plane ich ebenfalls die Verwendung von Open-Source-Bibliotheken, die unter anderem bestehende PHP-Funktionen ersetzen, die im Auslieferungszustand von PHP teils unzureichend sind (wie zum Beispiel das Versenden von E-Mails).

Diese Lösungen arbeiten jedoch nur in bestimmten Bereichen, man muss meistens auch nicht weniger Code als bei der Verwendung von reinem PHP schreiben. Sie ermöglichen aber die Umsetzung von Vorhaben, die mit reinem PHP entweder schwierig oder gar nicht möglich sind. Daher kann man hier noch nicht von einem Baukastensystem (vgl. Aufgabenstellung) sprechen. Ein

Baukastensystem wäre ein Webframework, welches Lösungen für alle Bereiche mitbringt (wie zum Beispiel Symfony und Silex).

Verwendete Open-Source-Bibliotheken

Composer

Der **Composer** ist ein Paketmanager für PHP. Er ermöglicht es andere PHP-Pakete automatisch zu installieren und einzurichten. Er bietet unter anderem den Vorteil, dass man die Abhängigkeiten von PHP-Bibliotheken nicht mehr manuell suchen muss, sondern diese automatisch installiert. Zum anderen generiert der Composer eine sogenannte **Autoload-Datei**, die einmal eingebunden alle über den Composer installierten Bibliotheken lädt, so dass man nicht mehr jeweils einzeln über **require_once** einbinden muss. Zudem werden zusätzlich benötigte PHP-Dateien der eingebundenen Bibliotheken zur Laufzeit automatisch geladen.

Swiftmailer

Der **Swiftmailer** ist eine Open-Source-Bibliothek zum Versenden von E-Mails. Sie zeichnet sich dadurch aus, dass sie sehr einfach zu benutzen ist und viele Probleme umschifft, die mit den Standardfunktionen zum E-Mail-Versand von PHP auftreten. Hier sei unter anderem die komplizierte Behandlung von Umlauten zu nennen. Swiftmailer ist allerdings nur sinnvoll verwendbar, wenn der verwendete Webserver aktuelle PHP-Versionen anbieten kann. In der Entwicklung ist das Projekt sehr wenig konservativ und macht früh von Funktionen in neuen PHP-Versionen gebrauch, so erfordert die aktuelle Switftmailer-Version 6 bereits ein installiertes PHP 7, was aber beispielsweise noch nicht alle Webhoster anbieten.

Twig

Twig ist eine Template-Engine, die eine sehr einfach zu erlernende Syntax besitzt. Dadurch lassen sich entsprechende HTML-Seiten besser lesen, als wenn zum Beispiel innerhalb der HTML-Seite PHP eingefügt wird, um zum Beispiel Variablen auszugeben. Weitere Stärken ist die Erweiterbarkeit, so lassen sich unproblematisch eigene Filter, Funktionen und Operatoren definieren. Außerdem wird bei Inhalt von außerhalb, zum Beispiel Daten aus einer Datenbank automatisch HTML-Escaping vorgenommen, so das zum Beispiel böswillig eingeschleuster HTML-Code automatisch unschädlich gemacht wird. Bei PHP passiert dies im Gegensatz dazu nicht automatisch, sondern es muss immer explizit über die Funktion **htmlspecialchars** bewerkstelligt werden, was häufig vergessen wird und so Sicherheitslücken in Websiten aufreißt.

Objektorientierung

Meine Neuentwicklung des Bestellformulars wird in überwiegender Form die objektorientierte Syntax von PHP verwenden. Die Gründe hierfür sind unter anderem, dass seit PHP 5 bei der Weiterentwicklung bei der Programmiersprache auf Objektorientierung gelegt ist, so sind neue Funktionen (wie PDO) nur mit einer objektorientierten Schnittstelle verfügbar. Zum anderen erlaubt

objektorientierte Programmierung eine gute Wiederverwendung von immer mal wieder benötigten Code-Bestandteilen.

Quellen

Artikel

- Diverse Autoren: Content Security Policy, Wikipedia, abrufbar unter https://de.wikipedia.org/wiki/Content_Security_Policy, abgerufen am 21.06.2017
- Diverse Autoren: SQL-Injection, Wikipedia, abrufbar unter https://de.wikipedia.org/wiki/SQL-Injection, abgerufen am 21.06.2017

Bildquellen

- Composer Logo: https://getcomposer.org/img/logo-composer-transparent5.png
- Swiftmailer Logo: http://swiftmailer.org/images/logo.png
- Twig Logo: https://cdn.css-tricks.com/wp-content/uploads/2015/08/twig1.png

Codebeispiele

- Sämtliche Codebeispiele sind soweit nicht anders angegeben ein eigenes Werk.
- Das Codebeispiel im Unterkapitel „Datenschutz“ entstammt aus der alten Blinkfair-Website.
- Das Beispiel-Listening für die SQL Injection wurde mit geringer Änderung aus dem Wikipedia-Artikel „SQL-Injection“ entnommen (Diverse Autoren: SQL-Injection, Wikipedia, abrufbar unter https://de.wikipedia.org/wiki/SQL-Injection, abgerufen am 21.06.2017).

Dokumentationen

- Composer Project: Composer Reference Book (in englischer Sprache), abrufbar unter https://getcomposer.org/doc/, abgerufen am 20.06.2017
- Foundeo Inc.: Content Security Policy Reference (in englischer Sprache), abrufbar unter https://content-security-policy.com/, abgerufen am 25.06.2017
- PHP Project: Funktionsreferenz PHP Data Objects (teilw. in englischer Sprache), PHP-Dokumentation, abrufbar unter http://php.net/manual/de/book.pdo.php, abgerufen am 23.06.2017
- PHP Project: Funktionsreferenz Ursprüngliche MySQL API (teilw. in englischer Sprache), PHP-Dokumentation, abrufbar unter http://php.net/manual/de/book.mysql.php, abgerufen am 23.06.2017
- PHP Project: Funktionsreferenz MySQL Improved Extension (teilw. in englischer Sprache), PHP-Dokumentation, abrufbar unter http://php.net/manual/de/book.mysqli.php, abgerufen am 24.06.2017
- Sensio Labs: Twig 2.0 Documentation (in englischer Sprache), abrufbar unter https://twig.sensiolabs.org/doc/2.x/, abgerufen am 26.06.2017
- Switftmailer Project: The Swiftmailer Book (in englischer Sprache), abrufbar unter http://swiftmailer.org/pdf/Swiftmailer.pdf, abgerufen am 26.06.2017