Homepage-Projekt e3fi2 2020/21

Ivanne Thijssen

Im Rahmen des Homepage-Projektes wurde Ende 2020 die Webseite note.ivanne.de erstellt. Diese Projekt-Dokumentation ist Voraussetzung für eine gute Note im Berufsschulfach Anwendungsentwicklung und beschreibt, inwiefern Gestaltungskriterien eingehalten sowie bestimmte Technologien eingesetzt wurden.

Das Wissen über MVC (Model-View-Controller) habe ich im Wesentlichen aus einer Video-Reihe vom Youtube-Kanal *Code With Dary* über OOP (Objektorientierte Programmierung), MVC und CRUD (Create, Read, Update, Delete) in PHP.

Meine Homepage ist im Internet öffentlich zugänglich und verlinkt direkt auf ihren offenen Quellcode bei GitHub. Hauptzweck der Webapplikation ist eine persönliche Termin-Verwaltung mit optionalen Erinnerungen per E-Mail bei Fälligkeit. Jedem Termin können bis zu 50 Bilder angehängt werden.

1. Datenhandling

Bis auf das Paket PHPMailer, das aus einem externen Repository unverändert übernommen wurde, ist die gesamte Struktur vom Hauptverzeichnis *clean* hier abgebildet: https://note.ivanne.de/clean/public/doc/tree.html

Wer im Browser note.ivanne.de eintippt, wird durch .htaccess auf den Plad *clean/public/* weitergeleitet. An der Wurzel dieses Ordners gibt es wiederum eine .htaccess-Datei, die mittels Rewrite-Engine für sog. saubere URLs (Clean Uniform Resource Locator) sorgt. So muss sich kein Website-Besucher die unschöne Zeichenkette *index.php?url=* ansehen. Stattdessen wird jede GET-Anfrage von einer Instanz der Klasse app/mains/Core.php nach dem Format */<controller>/<methode>/<parameter>* behandelt. Beispielsweise wird mit */pages/impressum* die Methode impressum() der Klasse app/controllers/Pages.php aufgerufen, wo die View app/views/pages/impressum.php – mit assoziativem Array als Argument zur Datenübergabe ans Frontend (z.B. der Seitentitel) – gerendert wird. Noch ein Beispiel: Mit */notes/delete/42* wäre die Notiz Nummer 42 gelöscht. Die Methode delete() ruft übrigens die Lösch-Methode aus dem Model app/models/Note.php auf. Model-Klassen sind immer die Singularform der zugehörigen Controller-Kindsklasse und besitzen als statische Eigenschaft ein Objekt der Klasse app/mains/Database.php.

In public/index.php wird lediglich require_once '../app/require.php' befohlen. In app/require.php werden zunächst alle Konstanten Variablen aus der config-Datei geholt. Letztere ist wegen geheimen Daten in .gitignore eingetragen und somit nicht bei GitHub zu finden. In der ebenfalls *required* Datei app/helpers/start.php wird durch Umleitung auf HTTPS sichergestellt, dass die Verbindung zum Apache-Server in Dresden nicht ohne Verschlüsselung stattfinden kann. Dann wird schließlich der Core-Konstruktor aufgerufen. Die o.g. URL-Regel mit Schrägstrichtrennungen gilt nur dann nicht, wenn an der Stelle *<controller>* der Verzeichnisname *scripts* steht. Diese Ausnahme erlaubt mir insbesondere die Ausführung von XHR (XMLHttpRequest) ohne MVC. Ich habe versucht, einen AjaxController in das MVC-Konzept zu integrieren und bis jetzt nicht ganz verstanden,

warum ich statt der gewünschten View nur die ganze Standard-Index-Seite in das HTML-Zielelement ausgeben konnte.

2. JavaScript

In meinem Projekt wird das JavaScript Objekt XHR genutzt, um PHP-Skripte auszuführen und das Ergebnis davon in einem DIV-Element auszugeben, ohne dass das komplette HTML-Dokument neu laden muss. So kann der Applikationsnutzer seine durch Textsuche gefilterten Notizen schon während der Eingabe (HTML-Attribut *oninput*) angezeigt bekommen. Er kann das Suchergebnis auch auf einen Fälligkeitsdatumsbereich sofort einschränken und beliebig umsortieren – alles ohne Seitenaktualisierung.

Da die Notizen in der Listenansicht direkt editierbar sind, ist z.B. die Funktion transformToArea(obj) dafür da, angeklickte Notiz-Texte in TEXTAREA-Elemente umzuwandeln, deren Größe veränderbar ist, um die Textbearbeitung benutzerfreundlicher zu gestalten.
Wenn der Benutzer mit der Maus über das Kreuz-Icon fährt (HTML-Attribut *onmouseover*), wird die entsprechende Notiz-Zeile mit einem niedrigeren Opazität-Anteil versehen. D.h. Die Notiz, die im Klick-Fall ganz gelöscht wäre, wird erst mit durchsichtigeren Farben kenntlich gemacht, sobald der Cursor die Schaltfläche auch nur berührt – wobei der Effekt durch die CSS-Eigenschaft *transition* angenehm verzögert wird.

Auf jQuery verzichte ich bewusst, nicht zuletzt weil wir im Unterricht Pure JavaScript angewendet haben. Die Funktion vanillaLoad(php, data, callback) dient als Ersatz für die grob äquivalente *load*-Methode aus einer jQuery-Bibliothek. Dabei steht php für den Skript-Dateinamen, data für ein FormData-Objekt (die POST-Variablen) und callback für die Funktion, die dann mit dem Ajax-Rückgabewert als Argument ausgeführt werden soll. Während der Wartezeit, die vor allem beim Hochladen von Bildern einige Sekunden lang sein kann, blende ich mit showRunning() ein animiertes GIF mit einem joggenden Menschen ein (passend zum Hintergrund-Bild).
Mit inputPicture(id) wird ein Bild-Datei-Inputfeld hinzugefügt, wenn der Nutzer auf das Plus-Icon in der Bild-Tabellenzelle in app/views/notes/create.php klickt. Jedes neue Bild-INPUT-Element trägt den selben Namen *pictures[]*, sodass später bei der POST-Anfrage ein Bilddatei-Array übergeben wird.

Hier sind ca. 30 weiteren Funktionen gesammelt, auf die an der Stelle nicht näher eingegangen wird:
https://note.ivanne.de/clean/public/javascript/functions.js

3. PHP

Das Projekt, hier *Clean Code* genannt, umfasst folgende Anwendungsfälle:
https://note.ivanne.de/clean/public/doc/UseCaseDiagram.png

Der Navigationsreiter Notizen ist nur für angemeldete Nutzer sichtbar. Ob ein Besucher eingeloggt ist oder nicht, wird durch Prüfung des globalen PHP-Session-Arrays mit dem Schlüssel *userid* festgestellt. Denn bei jedem erfolgreichem Login wird u.a. genau diese

Variable gesetzt. Bei Logout wird sie logischerweise zerstört. Welcher Menüpunkt gerade ausgewählt ist, finde ich anhand des Wertes von $view im Backend heraus. Die dafür programmierte Vergleichsfunktion currentActive($view, $menu) wird in app/view/includes/navheader.php aufgerufen, um den aktuellen Menüpunkt über die CSS-Klasse *.active* mit Textdekoration (einem Strich obendrüber) zu kennzeichnen.

In der Mitte des Anwendungsfall-Diagramms geht es um die Auflistung der Notizen. Je Notiz werden neben Text, Fälligkeit, Wiederholungsintervall und Alarm-Checkbox auch ggf. hochgeladene Bilder in Miniaturgröße angezeigt. Damit selbst mobile Geräte sämtliche Bilddateien darstellen können, greift die Funktion resizePicture($pixdata, $pixtype, $divisor) auf die beim Laden des BODY-Elementes (HTML-Attribut *onload*) in JavaScript ermittelte und via Ajax-Call in der PHP-Variable $_SESSION['screenwidth'] gespeicherte Bildschirmbreite zu, um die Bilder optimal zu skalieren.

Mit den Anwendungsfällen auf der Rechten Seite kommen wir zum Thema Fälligkeitserinnerung. Jede Notiz hat einen Zeitpunkt für die erste Fälligkeit. Eine Notiz kann deshalb auch als Aufgabe, Termin, *ToDo* oder *Reminder* bezeichnet werden. Ist bei einer Notiz die Alarm-Checkbox angewählt, wird bei Eintritt der Fälligkeit eine Erinnerungsmail an die bei Registrierung angegebene E-Mail-Adresse versendet – und zwar 10-minutengenau, da der Cronjob alle 10 Minuten den Mailing-Algorithmus in app/crontrollers/Alarms.php startet. Wenn bei *Intervall* nicht *Einmalig* steht, dann werden E-Mails solange automatisch gesendet, bis das Häkchen bei Alarm entfernt wird.

4. XML

Falls der Nutzer all seine Notizen ungefiltert in Textform sehen möchte, kann er auf den Link *Notizliste ohne Bilder im XML-Format öffnen* klicken. Andersherum hat er auch die Möglichkeit, Notizen in einer XML-Datei hochzuladen. Dafür wird die PHP-Klasse XMLWriter bzw. die PHP-Funktion simplexml_load_file eingesetzt.

5. Datenbank

Mit der Software *MySQL Workbench* habe ich 3 Tabellen in der 3. Normalform – und folgendes ERM (Entity-Relationship-Modell) erstellt:
https://note.ivanne.de/clean/public/doc/MySQLdesign.PNG

In der Relation *users* wird das Attribut *tempcode* mit einer zufälligen Zeichenkette befüllt, sobald der Benutzer einen Reset-Link anfordert, um sein Passwort zurückzusetzen. Dieser Code hat eine Gültigkeitsdauer von 15 Minuten. Bei Versand des Reset-Links wird in der Spalte *lastreset* der aktuelle Zeitstempel geschrieben, um den temporären Reset-Code-Ablauf gewährleisten zu können.

In der Relation *notes* sind in der Spalte *repeat* nur folgende Intervall-Werte zu finden: Einmalig|Täglich|Wöchentlich|Monatlich|Jährlich|[1-9][0-9].

In der Relation *pictures* sei zu bemerken, dass der Typ von *pixdata* nicht BLOB (Binary Large OBject), sondern MEDIUMBLOB ist. Grund dafür ist die Größeneinschränkung von BLOB auf

nur 64 KB. Mit MEDIUMBLOB kann die Dateigröße theoretisch bis zu 16 MB betragen, jedoch wird sie in app/models/Note.php auf 5 MB begrenzt.

Wie bereits im ersten Kapitel erwähnt hat jede Model-Klasse ein Database-Objekt. In PHP gibt es zwei bekannte APIs (Anwendungsschnittstellen) für Datenbankverbindungen: mysqli Extension und PDO (PHP Data Objects). Ich hatte zuvor nur von mysqli gehört aber in den MVC-Framework-Beispielen, die ich bei GitHub gefunden habe, wird mit PDO gearbeitet. Der DBH (Database Handler) ist also auch in meinem neuen Projekt eine PDO-Instanz.

6. Canvas

Spielerei mit Mandelbrotmenge, durch Überlappung 3D-Look
Bild herunterladbar

7. Style

Responsive Tabellenansicht mit HTML-Attribut *data-label* und CSS-Eingenschaft *content*
Pseudoklassen, display flex, font-faces