|  |  |  |
| --- | --- | --- |
| Thema | Zeigen | wer |
| Demo der Anwendung | * Kalender.php, index.php * Einmal durchklicken (auf individuelle Anforderungen eingehen)   ACHTUNG: Funktionierende Beispiele wählen -> am besten vorher schon mal ausprobiert).   * Fälle überlegen! -> Alles zeigen, dauert zu lange | Feli |
| RegEx-Einsatz | * Code (editEvent.php, editCategory.php) * Beispiel | Feli |
| Struktur/Architektur | * Objekte/Klassen * Datenbank * PDO (mit FetchClass) | Chris |
| Entwurfsmustereinsatz | * Factory (EventFactory.php) * Observer (editCategory.php) | Chris |

**Was sollten wir eventuell noch tun sollten:**

* Kommentare nachziehen (PHPDoc) -> „guten Willen“ zeigen
* Code aufräumen

ACHTUNG: muss funktionsfähig bleiben. Eventuell zwei unterschiedliche Umgebungen:

1. Jetzige Anwendung in htdocs-Ordner als Demo zeigen.
2. Lokale Kopie erzeugen und Programm dokumentieren und aufräumen -> dieses in IDE zeigen

**Fehler:**

* Ende-Zeit eintragen
* Termin ohne Kategorie

**Nochmal „lernen“:**

1. Magische Methoden

* Werden immer dann aufgerufen, wenn man auf eine private oder eine nicht existierende Variable eines Objekts zugreift

Es können dadurch „virtuelle“ Parameter erschaffen werden

Wir benötigen diese eigentlich nicht mehr, weil siehe 2.

1. PDO allgemein / PDO FetchClass

* PHP Data Objects sind eine Schnittstelle, mit welcher auf Datenbanken zugegriffen werden kann. PDO bietet eine Abstraktionsschicht für den *Datenzugriff*, das bedeutet, egal welche Datenbank benutzt wird, dieselben Funktionen verwenden können, um Abfragen zu erstellen und Daten zu lesen.
* Mit der Option FetchClass werden die eingelesen Datensätze direkt in Objekte der übergebenen Klasse umgewandelt.

Dazu wird der Wert jeder Column der Variable mit dem entsprechenden Columnnamen zugeordnet. Wenn diese nicht existieren, wird die „magische“ \_\_set-Methode aufgerufen. Ist diese nicht definiert, wird ein public member mit dem Namen der Column erschaffen.

1. Entwurfsmuster und wie wir diese genau umgesetzt haben (zugehörige Klassen)

* FactoryPattern:
* ObserverPattern:

1. RegEx-Prüfung von Eingaben

* F

1. Unterschied der Strings („ und ‚)
2. Unte

Eigene Verbesserungsvorschläge/ erkannte Fehler:

* Statt mit globaler Datenbank-Variable zu arbeiten, hätten wir die Klasse Datenbank mit statischen Funktionen implementieren können
* Fehlerabfragen -> eigener Meldungsbereich dafür auf der Seite („Statuszeile“).
* Tooltips auf Icons
* Strukturierung der Oberflächen (HTML-Codes) verbessern. Eventuell eigene Datei.
* Javascripte in eigene Datei auslagern.
* Klassen in eigene Dateien auslagern bzw. keine direkten Aufrufe ebendieser Seite
  + Durch die Includes kam es zu Fehler bei abfragen z.B. des Parameters $\_GET[]

# OOP

Die Objektorientierung haben wir folgendermaßen umgesetzt:

Die Anwendung wurde in einzelne logische Einheiten unterteilt, welche unterschiedliche Anforderungen erfüllen sollen. Diese Einheiten werden mittels Klassen in einer objektorientierten Form implementiert.

Daraus ergaben sich erstmal grundsätzlich vier Klassen.

Die Klasse „Database“ regelt den reinen Datenbankzugriff. Im Konstruktor wird die Verbindung hergestellt und eine Instanz der Klasse wird in einer globalen Variable gespeichert, welche von anderen Klassen genutzt werden kann. Die Klasse stellt grundlegende Operationen bereit, um mit der Datenbank physisch zu interagieren.

Des weiteren gibt es die Klasse „Kalender“, welches die einzelnen Ansichten, wie Monats- und Wochenansicht, erstellt und als HTML-Code zurück gibt.

In den einzelnen Ansichten sollen auch die Termine in der gewählten Zeitspanne angezeigt werden. Dazu wird auf Terminobjekte der Klasse „Event“ zurückgegriffen.

Diese Event-Objekte stellt sämtliche Funktionen bereit, die sowohl der Kalender zur Anzeige benötigt (z.B. Ausgabe des HTML-Codes zur Darstellung) als auch diejenigen, die zur Kommunikation mit der Datenbank benötigt werden (Löschen genau dieses Objekts aus der Datenbank, Ändern…, usw.).

Da eine der funktionalen Anforderungen das Hinzufügen einer Kategorie zu einem Termin war und eine Kategorie potenziell zu mehreren Terminen gehören kann, gibt es auch hierfür eine eigene Klasse „Category“. In der Datenbank hat jede Kategorie eine eindeutige ID. Die Zuordnung einer Kategorie zum Termin erfolgt über diese ID. Die ID der Kategorie wird in einem Feld der Tabelle Termin gespeichert.

Die Klasse „Category“ enthält Funktionen zur Kommunikation mit der Datenbank.

Da das Erstellen der Termin-Objekte auf unterschiedliche Arten erfolgen kann, z.B. über unterschiedliche Suchprofile auf der Datenbank oder auch als neues Objekt, noch ohne Datenbankbezug, haben wir das Entwurfsmuster Factory eingesetzt und eine Klasse „EventFactory“ erstellt. Diese Klasse implementiert mehrere Methoden zur Erzeugung von Event-Objekten, abhängig von der jeweiligen Anforderung an diese Objekte.