|  |  |  |
| --- | --- | --- |
| 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

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[]