Praxisarbeit

Modul 133

kxi – Jetski Services

Dokumentation Technik: IPERKA

Autor: Özgür Isbert

E-Mail: [oezguer.isbert@student.ibz.ch](mailto:oezguer.isbert@student.ibz.ch)

Dozent: Lukas Müller

Table of Contents

[Vorwort 3](#_Toc59103446)

[GitHub 3](#_Toc59103447)

[Voraussetzungen 3](#_Toc59103448)

[IPERKA 4](#_Toc59103449)

[Information 4](#_Toc59103450)

[Planung 4](#_Toc59103451)

[Entscheidung 5](#_Toc59103452)

[Realisierung 6](#_Toc59103453)

[Model 6](#_Toc59103454)

[Repositories 7](#_Toc59103455)

[SQL 8](#_Toc59103456)

[Hilfs-Funktionen 9](#_Toc59103457)

[Controller 9](#_Toc59103458)

[Views 10](#_Toc59103459)

[Kontrolle 10](#_Toc59103460)

[Auswertung 12](#_Toc59103461)

[Verbesserungen 12](#_Toc59103462)

# Vorwort

Dies ist eine Fortführung des Moduls 307 und beendet das Modul 133. In diesem Modul geht es um „Web-Applikationen mit Session-Handling realisieren“. Ich habe mir die Aufgabe angeschaut und habe dies via mein MVC-Pattern eigens erstellt und implementiert.

## GitHub

Das Projekt ist auf GitHub unter

<https://github.com/oezguerisbert/modul-133/tree/main/Praxisarbeit>

[](https://github.com/oezguerisbert/modul-133/tree/main/Praxisarbeit)

aufzufinden.   
Nach dem Runterladen der Dateien kann man direkt die Seite starten.

Das Projekt ist automatisiert, das heißt sämtliche Skripte werden im Vorfeld geladen und in der Datenbank abgelegt. Sie müssen beim Start des Skriptes die Datenbankkonfiguration eingeben, welche den Rest dann abklärt.

## Voraussetzungen

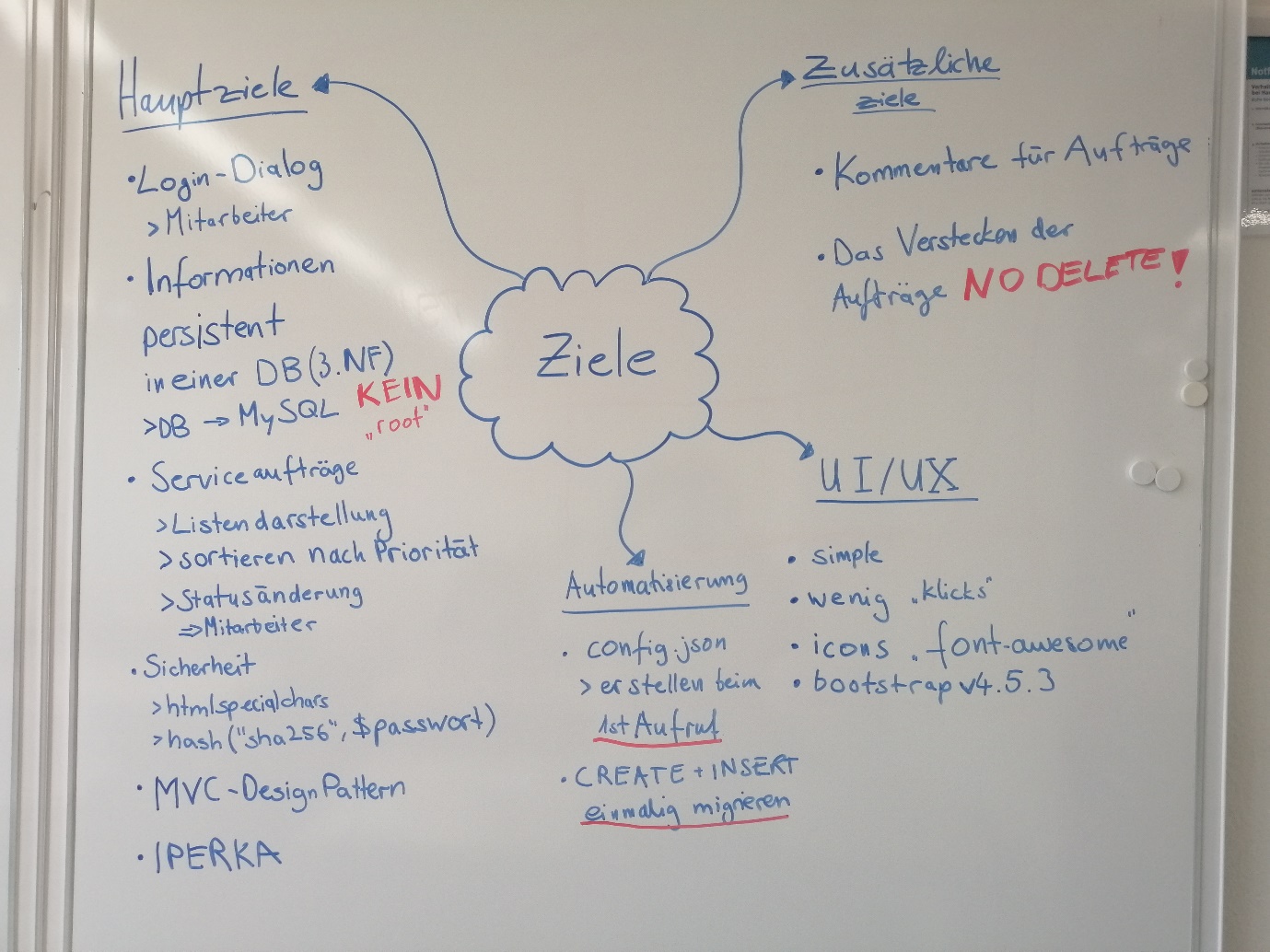
Um dieses Projekt laufen zu lassen müssen bestimme Vorkehrungen vorgenommen werden.

* PHP-Versionen > 7
* MySQL Version >= 5
* MySQL Datenbank mit dem Namen „modul133“.

# IPERKA

## Information

Informationen wurden aus der Aufgabe entnommen. Hier ist ein Mindmap zu den Aufgaben, die erfüllt werden müssen.



## Planung

Für die Planung habe ich das CI-Prinzip (Continuous Integration) übernommen. Hierbei werden die Jeweiligen Funktionen und Implementationen weiterbauend hinzugefügt. Es wird nicht auf eine Version festgelegt, aber dennoch versioniert gearbeitet.

Als Datenstruktur wurde das MVC Pattern übernommen und implementiert. Die Models wurden in dem Unterordner “classes” abgespeichert.

## Entscheidung

Aus den 10 Pflichtoptionen mussten wir 2 auswählen. Ich habe mich für “Kommentare für Aufträge” und “Das Verstecken der Aufträge !NO\_DELETE” entschieden. Begründet mit simplen Änderungen in der Datenbank und Zweckvolles Nutzen der Oberfläche, wurde dieser Entscheid abgehalten.

Zusätzlich habe ich mir die Mühe gemacht und als weiteren Aufwand die Automatisierung der Datenbanksysteme und Konfiguration vorgenommen. Sobald das Projekt gestartet wird auf einem Server und die Seite zum ersten Mal aufgerufen wurde, muss man den Server einstellen. Dies kann entweder direkt über die Oberfläche oder per POST-Methode mit den diversen Einstellungen mitgeliefert werden und das System wird vorkonfiguriert. Die Datenbank soll im Vorfeld migriert werden. Sämtliche SQL-Datenbanktabellen und Fremdschlüssel sollen eingerichtet werden und nicht mehr aktualisiert werden. Bei einem *wipe* d.h ein Reset der Datenbank, muss man nur die „migration“-Option auf true stellen und die Datenbank wird wieder neu aufgesetzt.

Als weitere Entscheidung habe ich die Icon Sets von der Schriftart „FontAwesome“ übernommen. Diese sind einfache Grafiken, welche die Oberfläche einfacher gestalten. Somit wird dann der Punkt „einfach halten“ eingehalten.

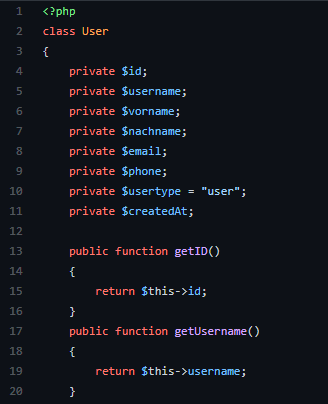
## Realisierung

Ich habe zuvor das Projekt aus dem Modul 307 übernommen, so wie es aufzufinden ist. Dieses kann man unter [GitHub (Modul 307)](https://github.com/oezguerisbert/modul-307/tree/main/Praxisarbeit) finden.

Ich habe für die Umsetzung von MVC (Model-View-Controller) die Ordnerstruktur erneuert und die Ordner „classes“, „controllers“, „repositories“, „sql“ und „views“ erstellt. In diesen Ordnern sind die jeweiligen korrespondierenden Code-Snippets vorhanden.

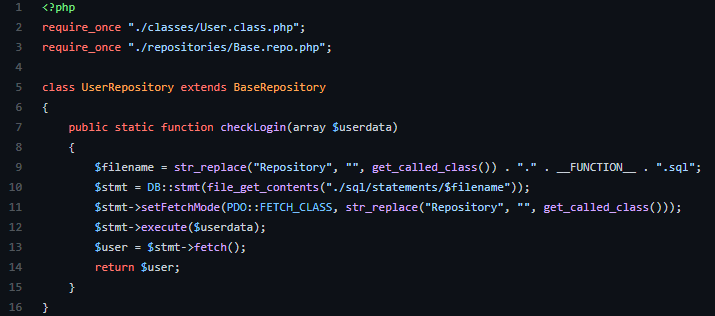
### Model

Beispiel: Die Klasse User



### Repositories

Die Werte werden später über die Repositories beladen. Repositories sind die Schnittstellen von den Klassen zu der Datenbank. Beispiel: Die Repository UserRepository

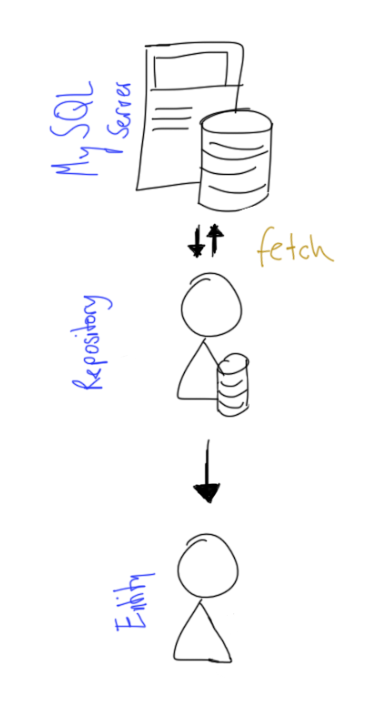


Die BaseRepository:



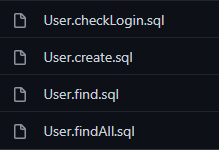
### SQL

Der Sinn hinter diesem Code ist das ich die SQL-Queries in separate Dateien schreiben kann und die Logik vom Command trennen kann. Die Methode findAll() ist im Grunde automatisiert. Die Klasse und die SQL-Dateien, die diese Methode aufrufen tut, werden dazu benötigt.



Dabei habe ich die Namenskonvention: **<Klasse>.<Methode>.sql** genommen.

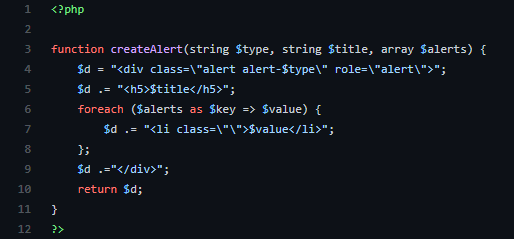
Hier ein Screenshot-Ausschnitt der Dateien im „sql“-Ordner:



Somit wurden die Klassen **Auftrag, Kommentar, Modus, Service** und **User** in diesem Format mit SQL-Queries bestückt und können vom Programm eingelesen werden. Falls ich Veränderungen in der MySQL-Seite ändern möchte kann ich dies per Datei machen und muss nicht in der Logik des Skriptes arbeiten. Für individuelle Einstellungen - sei es Logik basierend oder SQL-Statements - kann ich dies per Overwrites übernehmen.

### Hilfs-Funktionen

Die Hilfs-Funktionen sind auffindbar im Ordner „incs“. Diese werden nur für Hilfen verwendet, wie der Name schon sagt. Beispiel hier wäre **createAlert.func.inc.php**:



### Controller

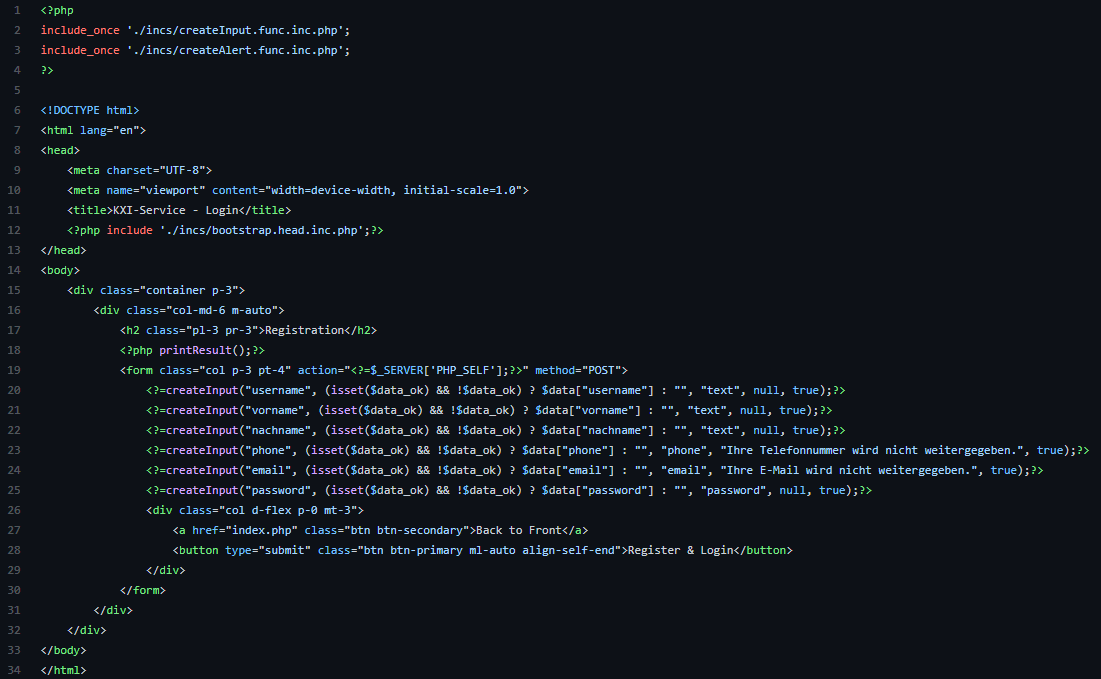
Ich habe die Controller (MV**C**) einfach aufgebaut. Diese haben die Logik der Einzelnen Komponenten für die View. Beispiel hier: **dashboard.controller.php**



### Views

Ich habe meine Views in den Ordner „views“ geschrieben.

Die Views enthalten jediglich nur View-Komponenten. Hier ein Beispiel: **register.view.php**



Die Views sollten so wenig wie möglich an Logik enthalten, außer es ist notwendig. Die eigentlichen Logikstrukturen sollten in den Hilfs-Funktionen enthalten sein. Siehe[**createInput.func.inc.php**](https://github.com/oezguerisbert/Arbeitsprobe/blob/main/incs/createInput.func.inc.php)

## Kontrolle

Zur Kontrolle habe ich die Extension **XDebug** und php-Funktion **var\_dump()** verwendet.

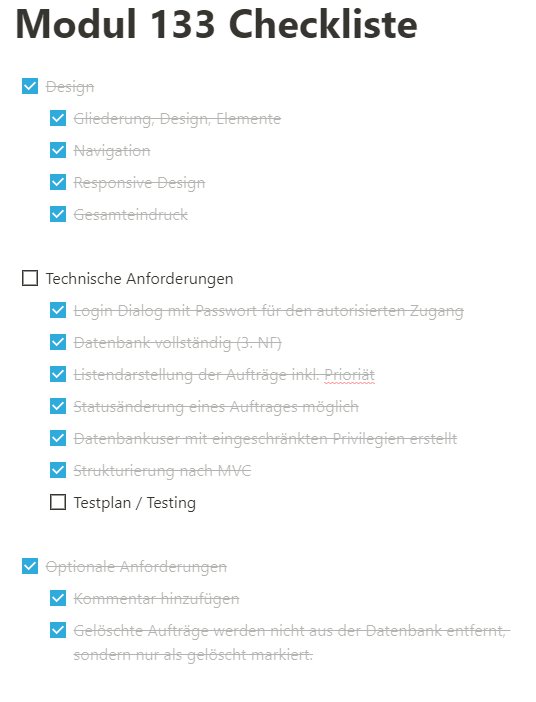


Beispiel hier: Ich wollte wissen welche Variablen verwendet werden und wie sie befüllt sind. Damit ich dementsprechend die View anpassen kann.

Die Kontrolle verlief recht zügig, da wenige Bugs/Logikfehler entstanden. Ich müsste zusätzlich die Inline-Kommentare hinzufügen.

Ich habe das Testing mithilfe von User-Eingaben getestet. In diesem Projekt fehlen die PHP-Unit-Tests, welche in Zukunft nachträglich eingereicht werden.

Die Checkliste wurde wie beim Modul 307 von den Bewertungskriterien übernommen. Hier ist meine Checkliste:



Wie zu sehen ist muss ich noch das Testing einbauen, wie vorhin erwähnt.

## Auswertung

Für die Auswertung wurde die Checkliste verwendet und zusätzlich wurde der Aufwand der jeweiligen *Meilensteine* berücksichtigt. Diese waren einwandfrei eingehalten worden.

### Verbesserungen

Ich hätte meine Kommentare zum Code - während ich den schreibe - hinzufügen sollen, das hätte mir die Arbeit erleichtert. Ich habe mich auf ein nachträgliches hinzufügen der Kommentare entschieden, aber mir ist aufgefallen, dass diese im Aufwand wenig sind und mir viel weiter helfen im Prozess.

Ich habe das Projekt in 3-4 Tagen aufgebaut und bin mit dem Endergebnis zufrieden. Es gab wenige Probleme, die ich mit schneller Research beheben konnte. Die meisten Seiten, die ich verwendet habe, sind über *duckduckgo* auffindbar. Die PHP-Dokumentationsseite war sehr hilfreich, da diese ebenfalls Beispiele enthält.