Datenbank Projekt - Philip Hannemann

Einleitung

Es gibt viele Webseiten, welche die Inhalte aus mehreren Seiten extrahieren und in einer eigenen Webseite zusammengefasst darstellen wie beispielsweise lieferheld.de oder opodo.de. Diesem Projekt liegt diese Idee zugrunde. Hierbei sind aus 3 unterschiedlichen Restaurants die Speisekarten extrahiert worden und in einer eigenen Webseite visualisiert worden.
Wenn Gerichte mit einem ähnlichen Namen existieren, werden diese für den Vergleich nebeneinander dargestellt und es können somit Preise und Zutaten der einzelnen Gerichte verglichen werden.
Bei dieser Aufgabe wurden Chemnitzer Webseiten ausgesucht, welche alle Lieferungen durchführen und alle Pizza, Döner, Nudeln und etliches mehr verkaufen.
Alle diese Kategorien wurden extrahiert und wurden auf der entstandenen Webseite visualisiert.
Diese Webseite ist unter der Domäne http://philip.cplusplustutorials.de/FrontEnd/ erreichbar.

Das Projekt

Allgemeiner Aufbau

Das Projekt wurde Großteils mit php, html und css umgesetzt und besteht aus 4 Bestandteilen. Ein Teil, welcher die Daten aus den Webseiten extrahiert und in eine Datenbank überführt, die Datenbank, eine REST Schnittstelle, welche die Daten aus der Datenbank als JSON für Webservices zur verfügung stellt und die Webseite, welche die Daten der REST-Schnittstelle nutzt, um die Speisekarten zu visualisieren und zu vergleichen.

Der Crowler

Ein wichtiger Bestandteil dieses Projektes ist die Daten der einzelnen Seiten zu extrahieren. Die Webseiten welche genutzt wurden waren http://www.himalaya-pizza.de, http://www.bologna-pizza.de/ und http://www.pamir-pizza.de. Die einzelnen Webseiten hatten alle einen unterschiedlichen Aufbau, weswegen, alle getrennt voneinander extrahiert wurden. Auch wurden meist Pizzen anders implementiert, wie der rest der Speisekarte, da nur bei Pizzen unterschiedliche Preise pro große existierten. Zusätzlich zu der Speisekarte, wurden noch die Adressen und Namen der einzelnen Lieferservices extrahiert.

Allgemeiner Ablauf die Inhalte zu extrahieren

1. Überschrift extrahieren und als Kategorie absprechen
2. Die Webseite ihren Content zu unterteilen
   (Jedes Menü soll hier einem Content entsprechen)
3. Innerhalb dieses Contents sollen danach die einzelnen für den Content gefunden werden
   (Beispielsweise Preis)
4. Der Content wird als Array zurück gegeben und an die anderen angehangen, sodass am Ende ein Array mit allen Gerichten einer Kategorie entsteht.
5. Alle Arrays einer Kategorie werden danach verbunden und bekommen jeweils den Eintrag welcher Kategorie diese angehören.
6. In einer weiteren Datei werden danach alle Daten eines Restaurants zusammengeführt, sortiert und in ein einheitliches Format für das Eintragen in die Datenbank vorbereitet

Pamir Pizza

Pamir Pizza ist eine Webseite mit 9 verschiedenen Kategorien: Vorspeisen, Pizza, Nudeln, Aufläufe, Schnitzel Gerichte, Indische Gerichte, Döner-Spezialitäten, Chicken & Pommes und Getränke.
Vorspeisen und Nudeln haben hierbei ein anderes Seitenlayout als der Rest der Webseite, weswegen diese separat betrachtet werden.

Vorspeisen und Nudeln

Die Extraktion der Seitenressourcen wurde hierbei immer mit CURL gelöst. Diese extrahiert alle relevanten Daten welche danach als String in einer Variablen zur Verfügung stehen. Es wurden viele Versuche durchgeführt mit xPath danach die den String in XML zu parsen. Da dies jedoch nach vielen Versuchen nicht funktioniert hatte, habe ich mich dazu entschlossen selber ein Parsingverfahren zu entwerfen.
Die Idee hierbei war, dass auch die anderen zu extrahierenden Seiten von diesem parsingverfahren profitieren und es leichter macht dessen Inhalt zu extrahieren.

Kategorien Vorspeisen und Nudeln (pamirNudeln.php)

Für das extrahieren der Daten wurde eine Funktion geschrieben, welche HTML Tags extrahieren kann *tagSearch($line)*. Diese Funktion benötigt einen String und gibt danach den ersten Tag aus der in dem String gefunden wurde. Diese Funktion war bei den Kategorie Vorspeisen und Nudeln ausreichend um die Tag-Struktur der Datei zu analysieren, doch bei anderen Quellen erwies sich diese doch als sehr eingeschränkt, da sie immer nur den ersten Tag betrachtet und immer an der richtigen Stelle zwischen den Tags der String geteilt werden muss. Daher wurden im späteren Verlauf noch andere Funktion geschrieben die einen ähnlichen Zweck erfüllen sollten und nicht nur den Anfang des Strings betrachten.

Da bei Pamir Pizza die Tags bei Vorspeisen und Nudeln alle auf eine neue Zeile geschrieben wurde, war es hier möglich an dem Zeilenumbruch den String zu unterteilen und mit *tagSearch($line)* die Zeile nach ihrem Tag zu untersuchen.

Die Überschrift wurde mit dem HTML Tag „h1“ und der Funktion parseHomePageGroup($content) extrahiert. Die Kategorien waren so aufgebaut, dass der Content in eine Tabelle hinterlegt wurde und in jeder neuen Zeile ein neuer Content eingebunden war. Durch das Suchen nach dem „table“-Tag und dem „tr“-Tag konnte somit ein Content extrahiert werden und weiter analysiert werden. Die Extraktion wurde immer durch eine Funktion mit dem Namen *parseHomePageContent($content, $category)* durchgeführt. Dabei steht $content für die Seite als String und *$category* als die Kategorie, welche durch Extraktion der Überschrift bekannt ist.
Die Information wurde an die Funktion *mealParser($content, $category)* weitergereicht, wobei *$content* hierbei nur noch einen Content beinhalten sollte.
Mithilfe einer Funktion *contentForTag($line, $tag)* konnten in diesem Beispiel die Zutaten extrahiert werden. Diese Funktion extrahiert aus einem String ($line) den Inhalt für einen Tag (*$tag*). Und gibt diesen wieder als String zurück.
Für die anderen Speisen-Parameter wurde eine Funktion *tagParseValues($line, $tag, $value)* entworfen. Diese Funktion durchsucht einen String (*$line)* nach einem Tag (*$tag*) und gibt den Inhalt eines Parameters (*$value*) zurück.
Beispiel:
$line = "<input type="hidden" name="Beschreibung" value="Napoli">";
$result = *tagParseValues($line, "input", „name");*

$result würde hierbei den String "Napoli" erhalten.

Durch diese Methode wurde beispielsweise mit Hilfe des Namens „Beschreibung“ herausgefunden welcher Tag den Namen der Speise enthält und von diesem dann mithilfe der selben Funktion der Parameter *value* extrahiert.

Diese Funktionen genügten um alle Informationen der beiden Kategorien zu extrahieren. Da die Zutaten jedoch vieles beinhaltete, was keine wirklichen Zutaten waren oder Wörter wie „mit“, „oder“, „überbacken“, … wurden die Zutaten vor dem einfügen in das Array noch auf reale Zutaten geprüft. Auch wurden Vermerke auf Zusatzstoffe entfernt. Zwischen alle gefundenen Zutaten wurde ein Komma gesetzt, was es leichter machen sollte die Zutaten mit den der anderen Webseiten zu vergleichen. Die Funktion hierfür heißt *devideToppings($toppings).* Sie empfängt einen String und gibt den veränderten String zurück.

Nachdem alle Parameter einer Speise extrahiert wurden werden diese an parseHomePageContent($content, $category) als Array zurück gegeben. Diese Funktion setzt alle Arrays der einzelnen Contents zusammen und gibt anschließend diese zurück.

Kategorie Pizza (pamirPizza.php)
Wie vorher schon erwähnt ist Pizza eine Kategorie die etwas anders ist, da sie 4 unterschiedliche Preise pro Essen beinhaltet.
Das Grundgerüst ist jedoch identisch zu dem Vorherigen Kapitel.

geändert hat sich hier somit vordergründig die Funktion *mealParser($content, $category) und parseHomePageContent($content, $category).*
Der Content war in diesem Fall durch den Tag „form“ gut zu extrahieren.
In der Funktion *mealParser($content, $category)* werden statt des einen Preises alle 4 Preise extrahiert und die Zutaten werden in diesem Fall ebenfalls durch ein Tag-Value-Pair extrahiert. Durch diese Extraktion, sind auch nur die relevanten Zutaten enthalten und eine nachträgliche Untersuchung auf richtige Zutaten ist nicht nötig.

Restliche Essenskategorien (pamirMenu.php)
Auch bei diesen Kategorien hat sich an dem Gerüst nichts geändert. Der Content wurde Innerhalb eines „div“-Tags extrahiert mit der Klasse „WarenBox“.
Und der Content selber wurde auch wieder anhand von anderen Tags extrahiert.
Die Zutaten hier benötigten eine neue Funktion *multiLineTagContent($content, $tag, $class, $value)*. Diese Funktion ist vergleichbar mit *tagParseValues($line, $tag, $class).* Bei dieser wird jedoch der Inhalt eines bestimmten Tags mit einer bestimmten Spezifikation zurück gegeben.
Auch musste wieder auf die *devideToppings($toppings)* Funktion zurück gegriffen werden. Bei dieser Extraktion wurden alle 6 unterschiedlichen Kategorien der Inhalt extrahiert und immer mal wieder waren unerwünschte andere Inhalte in den Zutaten zu finden wie beispielsweise „span“-Tags. All diese neuen unerwünschten Inhalte, kamen auf eine „Blacklist“ ($replacements) und wurden ebenfalls entfernt.

Adresse (adressPamir.php)
Die Adresse ist kein wiederkehrendender Inhalt in der Datei, daher unterscheidet sich hierbei die Struktur. Hierfür gibt es nur eine Funktion, welche die benötigten Daten aus der Datei entnehmen soll *parseRestaurantDetails($content).*
Der Name und die Webseite können mithilfe der bereis öfter verwendeten Funktion *tagParseValues($line, $tag, $value*) gelöst werden. Die Adresse hingegen ist als JavaScript code hinterlegt welche in jedes Zeichen in Prozentschreibweise nach Weblink konform darstellt. Mithilfe der in php enthaltenen rawurldecode($string) Funktion war es möglich diese Zeichenkette in lesbaren html code umzuwandeln und auch die Adresse zu extrahieren.

Aufbereiten der extrahierten Informationen

In der Datei getPamir.php werden alle erzeugten Arrays zusammengeführt.
Die Arrays aus pamirNudeln.php und pamirMenu.php werden zusammengeführt und repräsentieren die Speisen des Restaurants Pamir Pizza. Diese werden dann mithilfe der Funktion *sortArray($array*) sortiert. Die Funktion nutzt eine php Funktion namens uasort($array, ‚func'). Diese ist so aufgebaut, dass definiert werden kann nach welchen Bedingungen das array sortiert werden soll. Diese Bedingung wird mit einer selbstgeschriebenen Funktion definiert (hier: *cmp($a, $b)*). Dabei ist der Input zwei Elemente des Arrays und es soll 1 oder -1 als Rückgabewert existieren. -1 wenn $a < $b und 1 wenn $a > $b. Die Speisen wurden somit mit absteigender Relevanz sortiert: Kategorie, Name, Preis, Inhalt.
Die Daten werden als Array($pizza, $meals, $adress) zur Verfügung gestellt.
Hiermit wird auch die Art der Arrays als Standart für die anderen Restaurants angenommen und diese wurde darauf angepasst.

Himalaya Pizza

Die Struktur des Crowlers der Pamir Pizza wurde genutzt, um auch die Speisekarte der Himalaya Pizza zu extrahieren. Hierbei wurde jedoch nicht unterschieden zwischen Pizza und den anderen Gerichten. Die Datei himalayaMenu.php ist so geschrieben, dass sie für alle Einträge der Webseite gilt.

Speisekarte extrahieren (himalayaMenu.php)

Bei der Himalaya Pizza war der erste Eindruck der Webseite trügerisch. Sie sah optisch ähnlich der Pamir Pizza aus, war im Hintergrund aber doch deutlich anders aufgebaut.
Hierbei, waren die Zutaten schwerer dem Content zuzuordnen, da zwar ebenfalls ein „form" existierte, aber losgelöst von dem tatsächlichen html Codes. Da durch die Inhalte in dem „form" Tag aber deutlich besser zu extrahieren und ausfallsicherer sind, wurde trotzdem auf den „form"-Tag zurück gegriffen und den vorhergehenden extrahierten Zutaten zugeordnet. Diese werden im Anschluss an die bereits bekannte Funktion *mealParser($content, $category, $topping)* weitergereicht jedoch mit einem zusätzlichen Parameter, den Zutaten (*$topping*).
Um hierbei alle unterschiedlichen Speisen und Pizzen extrahieren zu können, wurde in jedem Fall nur ein Preis aufgeschrieben und die dazu passende Größe im Falle einer Pizza.
Außerdem wurde diese Struktur gewählt, da selbst bei Pizzen für jede einzelne Größe ein eigener „form"-Tag existierte.
Eine Funktion die hierbei sehr viel anstelle der *tagSearch($line)* Funktion genutzt wurde ist isTagInLine($line, $tag). Diese bieten die Möglichkeit statt immer den ersten Tag eines Strings zu erhalten die Möglichkeit einen String nach einem Tag zu durchsuchen und damit zu überprüfen ob dieser in dem String enthalten ist. Die Rückgabe ist true wenn der Tag gefunden wurde und false, wenn dieser nicht enthalten war.

Auch wurde die Funktion *devideToppings($toppings)* auf einen deutlich größeren Umfang an Fehlerhaften Zutaten eingebunden, sodass immer nur die richtigen Zutaten auch weitergereicht wurden.
Pizza und der Rest der Speisekarte wird im Nachhinein jedoch trotzdem wieder getrennt voneinander abgespeichert, da Pamir Pizza als Vorbild genutzt wird und auch die Datenbank Speicherung zwei Tabellen vorsieht (Speisen und Pizzen).

Adresse extrahieren (adressHimalaya.php)

Auch bei dem Restaurant Himalaya gilt, dass die Adresse ein nicht wiederkehrender Eintrag ist und daher eine leicht abgewandelte Struktur erfordert.
Die Struktur der Datei adressPamir.php wurde hierfür wieder als Vorbild verwendet.
Auch hier werden die Details des Restaurants unter Nutzung der Funktionen zu Extraktion von Tag-Values in der Funktion *parseRestaurantDetails($content)* extrahiert. Die Funktion *getHimalayaAdress( )* stellt hierfür die Schnittstelle zur Verfügung, um diese Daten abzurufen.

Aufbereiten der extrahierten Informationen

Mit der Datei *adressHimalaya.php* werden hierbei die Daten zusammengeführt und in das Format, dass durch Pamir Pizza vorgegeben wurde umgewandelt und sortiert.
Als erstes werden die empfangenen Daten sortiert, mit einer auf dieses Array angepassten *cmp($a, $b)* Funktion, da in diesem Fall immer nur ein Preis vorliegt und der Parameter „Größe“ dazugekommen ist.
Nachdem die Daten in eine sinnvolle Reihenfolge gebracht wurden wurden diese an die Datenkonform angepasst unter Nutzung der Funktionen *makeArrayKonfirmationMenu($array)* und *makeArrayKonfirmationPizza($array)*. Diese beiden Funktionen wandeln die eingehenden Arrays für die Speisekarte und die Pizzen in deren jeweilige Vorgabe um. Ganz besonders wichtig war hierbei die Pizzen umzuwandeln, da hier aus 4 Einträgen des gleichen Namens aber unterschiedliche Größen beinhalten, zu einer Pizza mit 4 unterschiedlichen Preisen zu erstellen.
Als Rückgabe wurde auch hier die vorher erstellte Konform eingehalten und ein Array mit den Einträgen $pizza, $meals, $adress übergeben.
Auf diese kann durch die Funktion *getHimalaya( )* zugegriffen werden.

Bologna Pizza

Auch Bologna Pizza ist ein Chemnitzer Lieferservice und beinhaltet viele unterschiedliche Kategorien welche sich mit denen, der vorhergehend beschriebenen, in einigen Punkten gleichen.
Da diese Seite einige Fehler mit Umlauten hat wurde hierbei zuallererst die fehlerhaft entstandenen Zeichen durch die eigentlich darzustellenden Zeichen ersetzt (zum Beispiel: „Ã¶" wurde durch ein „ö" ersetzt).

Speisekarte extrahieren (bolognaMenu.php)

Genau wie bei dem Himalaya Pizzaservice wurde auch hier ein Crowler geschrieben, welcher die gesamte Speisekarte extrahiert und nicht zwischen Pizza und dem Rest unterscheidet.
Die Struktur ist wie in den beiden anderen Restaurants, erst die Contentgruppe zu identifizieren und sie im Anschluss auf deren Inhalt zu überprüfen.
Um eine Contentgruppe zu extrahieren wurde hierbei das wechselnde Muster des Hintergrundes genutzt. Jede neu gefundene Contentgruppe wird danach wie gewohnt an *mealParser($content, $category)* übergeben. In dieser werden danach ähnlich wie bei Pamir die einzelnen 4 Preise extrahiert und in einem gemeinsamen Array zur verfügung gestellt.
Hierbei ist zu beachten, dass nicht jeder Eintrag immer alle Werte zur Verfügung stellt. Selbst die Pizzen haben nicht in jedem Fall alle 4 Größen. In diesen Fällen sind die jeweiligen Attribute leer geblieben.
Um der Konform zu genügen Pizzen von den restlichen Speisen zu trennen, wurde hierbei wie auch bei Himalaya Pizza in der *parseHomePageContent($content, $category)* der geparkte Content in zwei unterschiedliche Arrays gespeichert. Die Kategorien „Pizza" und „Amerikanische Pizzen“ wurden zu der Kategorie „Pizza“ zusammengefasst damit eine höhere Übereinstimmung der drei Restaurants und ihren Kategorien existiert.
Diese zwei Arrays (Speisen und Pizzen) werden mit der Funktion *getBolognaMenu( )* zur Verfügung gestellt.

Adresse extrahieren (adressBologna.php)

Um die Adresse des Bologna Lieferservice zu extrahieren wurde die selbe Struktur wie bei den beiden vorhergehenden Restaurants zur Adressextraktion verwendet.
Da im HTML Code nie die Webseite selbst als Link hinterlegt war wie bei den beiden vorhergehenden Restaurants wurde diese hier manuell eingetragen zurück gegeben.
Die Informationen welche nach außen zur Verfügung gestellt wurden waren hierbei, wie auch bei den beiden anderen, in der Konform: Name des Restaurants, Adresse des Restaurants, Webseite des Restaurants. Diese können mit der Funktion *getAdressBologna( )* erhalten werden.

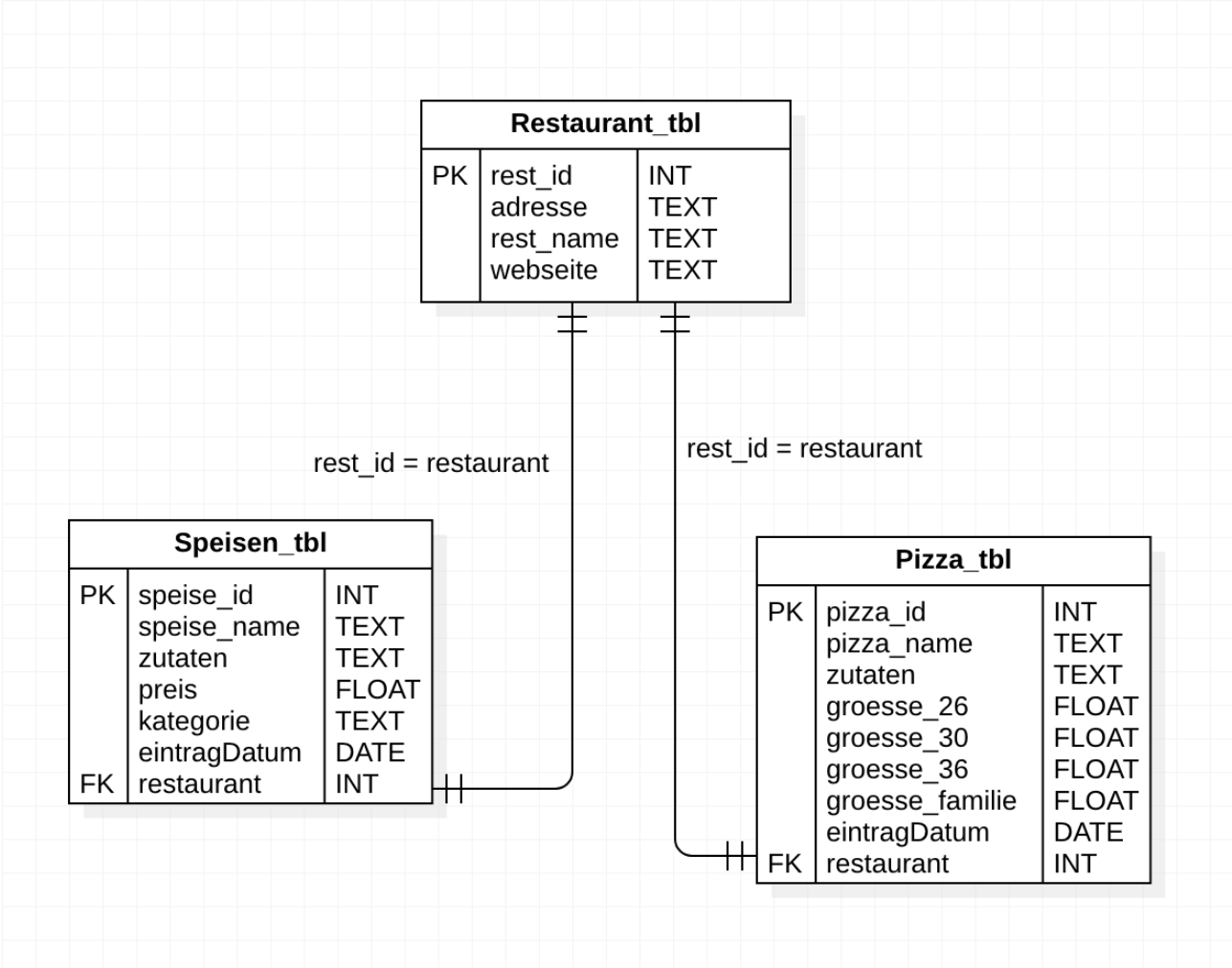
Aufbereiten der extrahierten Informationen

Die Daten müssen auch in diesem Fall zusammengeführt werden und an die Konform angepasst werden. Das wird in der Datei *getBologna.php* umgesetzt. Die Daten werden mit einer leicht veränderten *cmp($a, $b)* Funktion und dessen Aufruf *sortArray($array)* sortiert und anschließend an die Konform angepasst, was in diesem Fall nur bedeutet den Key des Arrays der Speisen von „p1" zu „preis“ zu ändern und die restlichen Preiseinträge welche leer sind zu ignorieren.
Die Funktion getBologna( ) stellt diese Arrays in folgender Reihenfolge in einem Array zur Verfügung: Array(Pizza, Speisen, Restaurant Informationen).

Datenbank

Aufbau der Datenbank

Die Datenbank, welche für dieses Projekt erstellt wurde trägt den Namen „philipsprojekt“ und besteht aus drei Tabellen „Speisen_tbl“, „Pizza_tbl“ und „Restaurant_tbl“.



Datenbank füllen

All die empfangen Daten der Restaurants werden durch die Datei *Datenbank.php* in die Datenbank eingetragen. Die Datei beinhaltet 3 unterschiedliche Aktionen der jeweiligen eingehenden Arrays. Sind Einträge hinzugekommen? Sind Einträge weggefallen? Oder hat sich ein Eintrag geändert?

Diese drei Fragen werden mit der jeweiligen Update Funktion ermittelt (*updatePizza($menu, $conn, $restID), updateMenu($menu, $conn, $restID), updateAdress($in, $conn, $restID)*).

Hierbei stehen die Parameter *$menu/$in, $conn, $restID* für das Eingabearray, die Verbindung zur Datenbank und die ID des Restaurants, zu der die Einträge gehören.

In den jeweiligen Funktionen sorgt eine Funktion mit dem Namen *changed($arrayIN, $arrayDB)* dafür herauszufinden ob sich etwas geändert hat in den Einträgen und gibt zurück, welche Aktion für welchen Eintrag durchzuführen ist.

Es wird ein Array zurück geben welches erst alle Einträge listet, welche hinzu gekommen sind, dann, welche gelöscht wurden und zum Schluss welche sich verändert haben.
Um diese Informationen zu erhalten müssen die beiden Arrays, welche übergeben werden die selbe Struktur besitzen. Um diese Bedingung zu erreichen werden die Einträge aus der Datenbank mit der Funktion *changeTableForEquationMenu($dbTable)* in die Struktur der gecrowlten Daten gebracht. Mit der in php existierenden Funktion array_search( ) wurde dann für jeden Eintrag dessen Key zurückgeben. Falls der Eintrag nicht gefunden wurde wird false zurückgegeben. Wenn „false“ die Ausgabe ist, kann somit dieser Eintrag entweder gelöscht oder zu der Datenbank hinzugefügt werden (je nachdem, ob ein Eintrag aus der Eingabetabelle in der Datenbanktabelle nicht gefunden werden kann, oder andersrum).
Wenn jedoch der Eintrag in beiden Tabellen vorhanden ist, muss dieser auf Gleichheit geprüft werden. Dies wird mit der Funktion *compare($arrayIN, $arrayDB)* gelöst, welche die php Funktion *array_diff( )* nutzt um zu überprüfen ob unterschiede in den beiden Arrays existieren. Die *array_diff( )* gibt hierbei den Key zurück, wo sich die beiden Arrays unterscheiden. Die Rückgabe der Funktion *compare($arrayIN, $arrayDB)* ist *true* wenn beide Arrays gleich sind und *false* wenn sie sich unterscheiden.
Mit den Funktionen:

- insertIntoDBPizza($menu, $conn, $id)
- insertIntoDBMenu($menu, $conn, $id)
- removeFromDBPizza($menu, $conn, $id)
- removeFromDBMenu($menu, $conn, $id)
- changeDBcolomsPizza($menu, $conn, $id)
- changeDBcolomsMenu($menu, $conn, $id)

können die erfassten Daten welche sich ändern sollen auf die Datenbank angewendet werden. Alle dieser Funktionen beinhalten SQL code, welcher die Eingabe einschließt.

Es ist im CRON des Servers eingetragen, dass die Datei *Datenbank.php* wöchentlich 3:00 Uhr jeden Montag gestartet wird.

REST

Um die Daten aus der Datenbank allgemein für Webservices zu Verfügung zu stellen wurde die REST Schnittstelle entworfen. Hierbei ist es möglich die Daten als JSON zu empfangen.
Es existiert eine Datei *database.php* welche eine Klasse beinhaltet, um sich auf die Datenbank zu verbinden und die Verbindungsinformation zu erhalten.
Des Weiteren existieren im Ordner „objects“ die Dateien speisen.php, pizza.php und restaurant.php. Diese Dateien beinhalten jeweils eine Klasse der zugehörigen Tabelle(„Speisen_tbl“, „Pizza_tbl“ und „Restaurant_tbl“). Die Klassen haben jeweils eine Methode *search($keywords),* um in der jeweiligen Tabelle Einträge welche ein bestimmten String enthalten, zurück zu geben.

In den Ordnern „speisen“, „pizza“ und „restaurant“ ist jeweils eine Datei *read.php* mit der die Einträge der Tabelle in JSON ausgegeben werden können.

Frontend

Um die in den vorherigen beschriebenen Kapiteln beschriebenen Daten darzustellen wurde eine Webseite entworfen, welche die REST API nutzt um die Daten der Datenbank richtig darzustellen und in ihr nach bestimmten Eingaben zu filtern. Um ein gutes Aussehen der Seite zu erreichen wurde auf die CSS Datei von www.w3schools.com/ zurückgegriffen. Diese liegt in dem Ordner „FrontEnd/css/w3.css“
Die Webseite ist in 3 Seiten unterteilt: index.php, Speisekarte.php und restaurants.php.
Die index Seite beinhaltet hierbei die Erklärung, wofür die Webseite gemacht ist.
Die *restaurants.php* Datei beinhaltet die extrahierten Informationen zu den einzelnen Restaurants. Und extrahiert die Daten mit der REST API 'http://philip.cplusplustutorials.de/REST/restaurant/read.php'.
Die erhaltenen Informationen der API wurden mit der php Funktion *json_decode($json, true)* in ein php Array umgewandelt und mit html dargestellt.

Die Datei speisekarte.php beinhaltet eine Seite welche die Speisen aller Restaurant darstellt und per input Feld zu durchsuchen.
Hierfür wird per POST Befehl die Eingabe des Feldes an die REST API als Suchwert übergeben und daraus die jeweils benötigte Tabelle erhalten.
Die Schnittstellen hierfür sind 'http://philip.cplusplustutorials.de/REST/pizza/read.php' und 'http://philip.cplusplustutorials.de/REST/speisen/read.php'.
Die erhaltenen Arrays werden nach Kategorie unterteilt und an eine Funktion *tableForCategory($category, $array)* der Datei php/table.php übergeben.
Die Funktion erstellt die Tabelle stellt die Speisen nebeneinander dar, welche von unterschiedlichen Restaurants sind und einen ähnlichen Namen haben.
Da durch die REST API die Daten nach Namen sortiert sind, ist es hier möglich aufeinanderfolgende Einträge auf Gleichheit zu überprüfen.
Ob zwei Einträge von zwei Restaurants nebeneinander dargestellt werden dürfen soll die Funktion *checkForEqualaty($elem1, $elem2)* überprüfen. Diese ruft je nach bedarf checkForEqualatyPizza($elem1, $elem2) oder checkForEqualatyMeals($elem1, $elem2) auf.
Innerhalb dieser Funktionen wird mit der php Funktion *similar_text( )* auf Übereinstimmung geprüft. Bei einer Übereinstimmung von mehr als 80%, dürfen die beiden Gerichte als gleichwertig gelten. Somit gilt beispielsweise „Döner-Pizza“ ist gleich „Dönerpizza“.

Mit der *getTableCell($elem)* Funktion wird für einen bestimmten Eintrag der HTML Code für einen Zelleneintrag zurückgegeben. Und je nachdem, ob das Essen, was mit $elem übergab wird eine Pizza oder eine andere Speise ist, wird die Funktion *getTableCellPizza($elem)* oder *getTableCellMeal($elem)* aufgerufen.

Die Funktion *getRow($rest1, $rest2, $rest3)* fügt dann die einzelnen Einträge zusammen und gibt eine Tabellenzeile zurück.

Nachdem alle Tabelleneinträge in HTML umgewandelt wurden, wird dieser dargestellt.

Abschließende Reflexion des Projektes

Es kamen viele Herausforderungen auf mit denen ich ursprünglich gar nicht gerechnet habe. Rückblickend war es auch nicht die sinnvollste Idee dem Crowler komplett selber zu schreiben und keine Bibliothek wie xPath zu nutzen. Ursprünglich schien es sehr einfach ohne xPath den Inhalt zu extrahieren. Jedoch hab ich die unterschiede der einzelnen Webseiten doch etwas verkannt.
Auch war ursprünglich mein Plan mit AJAX ein dynamisches Suchfeld zu implementieren, jedoch gab es unerwartete Fehler, wodurch ich doch auf den POST Befehl zurück greifen musste.

REST API nutzen

Der Link für die Extraktion der Speisen ist 'http://philip.cplusplustutorials.de/REST/speisen/read.php'

Der Link für die Extraktion der Restaurants ist 'http://philip.cplusplustutorials.de/REST/restaurant/read.php'

Der Link für die Extraktion der Pizzen ist 'http://philip.cplusplustutorials.de/REST/pizza/read.php'

Diese können jeweils mit „?s=wert“ durchsuchtwerden nach einem String

Als Ausgabe wird immer JSON zurück gegeben, welches zuerst den Namen des Inhaltes beinhaltet und im Anschluss die einzelnen Tabellenwerte.