2022

2. semester – projekt 2

Dennis Nielsen, Freja Hyldgaard og René Seebach

Webudvikling - IBA Erhvervsakademi Kolding

25-03-2022

Indholdsfortegnelse

Indledning

I dette projekt skal vi arbejde med dataintegration, hvor vi har fået til opgave at arbejde med fortidsminder. Her skal vi gøre det muligt for brugere at indtaste en lokation, hvor de skal modtage en liste over de fortidsminder, som er indenfor den afstand, som vores brugere angiver. Hertil skal vi angive relevant data såsom adresse m.m.. Derudover skal vi opgøre de nærmeste restauranter.

Problemformulering

Hvordan kan vi via API'er angive hvilke fortidsminder der er i en given afstand til en lokation, som angives af en bruger.

Metodeovervejelser

PHP

Vi har valgt at arbejde i PHP, da vi har arbejdet med PHP gennem vores undervisning med Dataintegration. Her kan vi bedst relatere til det vi har lavet i undervisningen, og blive inspireret af, hvordan vi kan løse opgaverne. Derudover ønskede vi også at blive udfordret og her var løsningen med PHP passende.

MySQL - relationsdatabaser

Da vi arbejder i PHP, har vi valgt at arbejde med MySQL og relationsdatabaser. Databasen bruger vi som et Data Warehouse, hvor vi kan hente data fra databasen hver gang, i stedet for at vi skal hente dataene ude gennem API'erne.

Research

Det første vi gjorde, var at lave noget research på de api'er, som vi får brug for. Først fandt vi Kulturministeriets hjemmeside, hvor vi kunne downloade en CSV-fil, som indeholdt de data over fortidsminder, som ligger i en bestemt eller alle kommuner (Slots- og Kulturstyrelsen n.d.)
Hertil prøvede vi at konvertere CSV-filen til en XML-fil, som vi kunne arbejde ud fra. En af problematikkerne var et stort datasæt på tværs af landets kommuner. Vi forsøgte at reducere disse til Vejle kommune, dog fik vi stadig et uoverskueligt skema med spørgsmålstegn selvom vi brugte utf-8.
Vi fandt dog efterfølgende frem til, at vi kunne bruge api'er indefra DAWA (*Adgangsadresser* n.d.).
Her kunne vi både få api'er til adresser og fortidsminder, som er det vi skal bruge for at løse opgaven.

Fortidsminder

Vi fandt frem til følgende api, som giver os alle de fortidsminder der er:
https://api.dataforsyningen.dk/steder?hovedtype=Fortidsminde

Find adresse indenfor given afstand:

For at finde elementer som er indenfor en given afstand, fandt vi et andet api, som finder de adresser, som er indenfor den givne afstand og koordinater:
https://api.dataforsyningen.dk/adresser?cirkel=x,y,r
Her er x = længdegrad, y = breddegrad og r = radius.

Fortidsminder indenfor given afstand:

Men da vi ikke skal have fat i adresser men fortidsminder, tænkte vi, om vi kunne tage "cirkel"-delen af URL'en og tilføje den til vores fortidsminde-api. Dette virkede, og derfor bliver vores andet api vi skal arbejde med:
https://api.dataforsyningen.dk/steder?hovedtype=Fortidsminde&cirkel=x,y,r
Her er x = længdegrad, y = breddegrad og r = radius.

cURL

For at hente data fra et api, kan vi bruge cURL. cURL er en form for bibliotek hvor man kan lave HTTP request i PHP (*PHP: Introduction - Manual* n.d.)
Vi kan blandt andet bruge curl_init, som initialiser en cURL-session og curl_exec, som udfører en cURL-session. Til cURL kan man også curl_setopt, hvor man har mulighed for at indstille en "option" for en cURL-overførsel (*PHP: CURL - Manual* n.d.) (Toft n.d.).

Analyse

Databasen

Da vi skal arbejde med mange data fra API'er, har vi valgt at anvende en database, hvor vi vil putte alle vores data i. På den måde kan vi lave en form for Data Warehouse, hvor vi henter data derfra hver gang. Ud fra de API'er, som vi fandt i vores research, har vi lavet et overblik over, hvordan vores database skal ud:

Database For projekt 2			
Database	**table**	**colum**	**Value**
fortidsminderdb	steder	id	varchar
fortidsminderdb	steder	hovedtype	varchar
fortidsminderdb	steder	undertype	varchar
fortidsminderdb	steder	primærnavn	varchar
fortidsminderdb	steder	primærnavnesta	varchar
fortidsminderdb	steder	længde	float
fortidsminderdb	steder	bredde	float
fortidsminderdb	steder	kommunenavn	varchar
fortidsminderdb	steder	kommunekode	int
fortidsminderdb	restaurant	navn	varchar
fortidsminderdb	restaurant	adresse	varchar
fortidsminderdb	restaurant	postnummer	int
fortidsminderdb	restaurant	bynavn	varchar
fortidsminderdb	restaurant	kontrolrapport	varchar
fortidsminderdb	restaurant	geo_længde	float
fortidsminderdb	restaurant	geo_bredde	float

Databasen skal indeholde to tabeller. En tabel til fortidsminder og en tabel til restauranter. Hver af tabeller skal indeholde de informationer om, som vi synes er mest relevante, og som er dem vi skal bruge senere hen.

Views

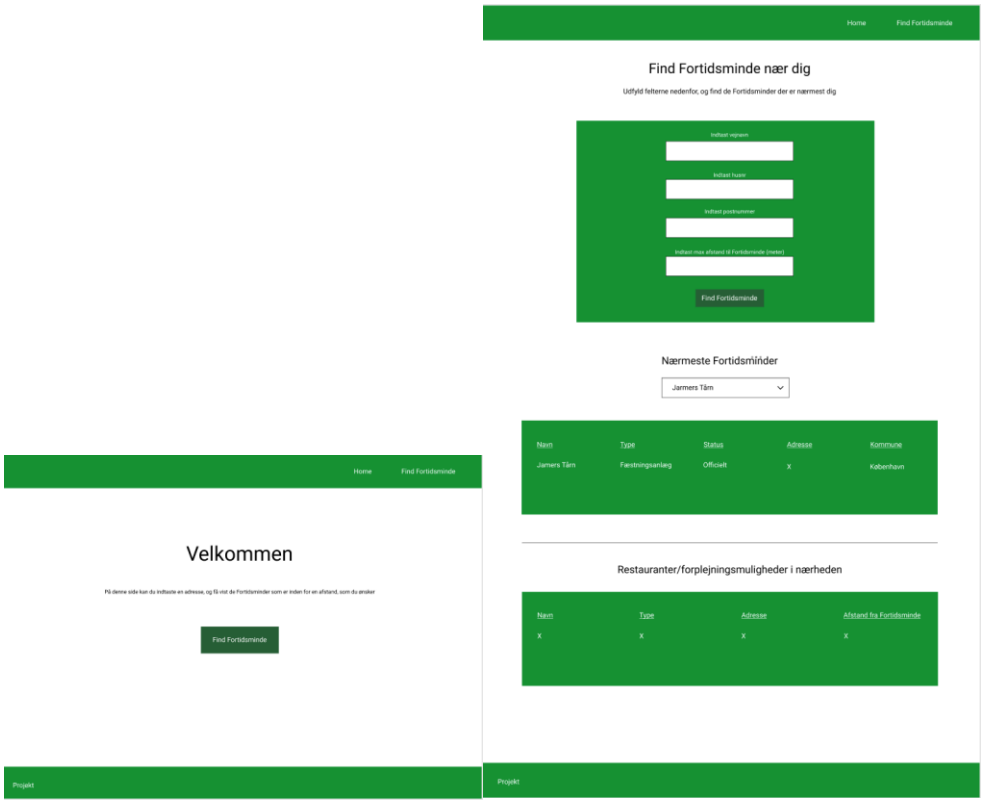
Vi har valgt at lave to sider til projektet. En forside/velkommen-side og en underside, hvor en bruger har mulighed for at angive en adresse. Når en bruger har angivet en adresse, vil der komme en dropdown frem, med en liste over de fortidsminder, som passer til de kriterier, som brugeren angav i vores formular. Her er det meningen at brugeren skal vælge et fortidsminde fra vores dropdown, hvorefter informationer omkring dette fortidsminde vises frem.

Lagring af data

I projektet har vi valgt at lægge vægt på, hvordan data behandles til visning af informationer til brugeren når denne søger på et fortidsminde samt restauranter. Vi forbinder API-kald sammen med data warehouse for at imødekomme en robust og hurtig forbindelse til alle brugere. Uden disse ville det tage brugeren længere tid at hente de nødvendige informationer.

Beslutning om brugergrænseflade

I projektet var det vigtigt at skabe en god brugergrænseflade til vores brugere. Til dette skabte vi ei Figma-dokument, hvor vi lavede en high fidelity løsning til illustration af alle sider. Dette har været med til at sikre god forståelse for alle gruppemedlemmer, og gav grundlaget for valg af nedenstående:



Konstruktion

Data Warehouse

Til vores projekt var det nødvendigt at bruge API´er til at tiltrække data fra til at gemme i vores MySQL database. Vores database fortidsminderdb, som illustreret i analysen, består af to tabeller henholdsvis 'steder' samt 'restaurant'.

```
CREATE TABLE IF NOT EXISTS `steder` (
  `id` varchar(100) NOT NULL primary key,
  `hovedtype` varchar(200) NOT NULL,
  `undertype` varchar(200) NOT NULL,
  `primærtnavn` varchar(200) NOT NULL,
  `primærnavnestatus` varchar(200) NOT NULL,
  `kommunenavn` varchar(200) NOT NULL,
  `kommunekode` varchar(100) NOT NULL,
  `længde` float NOT NULL,
  `bredde` float NOT NULL,
  `adresse` varchar(100) NOT NULL
);
```

Illustration af steder tabellen.

Den første del af arbejdet bestod i at oprette en database ved hjælp af sql-sætninger som vist foroven. Vi skabte en fil kaldet fortidsminder.sql hvor det var muligt at kopiere samt sætte ind i selve MySQL.

Derefter var det muligt for os at bygge en dbfortidminder.php fil til at skabe alle de nødvendige informationer.

```
// Get data from API
$url = "https://api.dataforsyningen.dk/steder?hovedtype=Fortidsminde";

$json = file_get_contents($url);

// Convert JSON string into an array // When true, JSON objects will be returned as associative arrays
$json = json_decode($json, true);

// Extracting data
foreach ($json as $data) {


  $adress = getAdresse($data['visueltcenter'][0], $data['visueltcenter'][1]);
  // Database query to insert data
  $sql =
    "INSERT INTO steder(id, hovedtype, undertype, primærtnavn, primærnavnestatus,
    kommunenavn, kommunekode, længde, bredde, adresse) VALUES
  ('" . $data["id"] . "', '" . $data["hovedtype"] . "', '" . $data["undertype"] . "', '" . $data["primærtnavn"] . "',
  '" . $data["primærnavnestatus"] . "', '" . $data['kommuner'][0]["navn"] . "', '" . $data['kommuner'][0]["kode"] .
    "', '" . $data['visueltcenter'][0] . "', '" . $data['visueltcenter'][1] . "', '" . $adress[0]['adressebetegnelse'] . "'); ";

  $connect->query($sql);
}
$connect->close();
```

Illustration af dbfortidminder.php fil

Det første vi gør, er at hente alle vores fortidminder ved hjælp af en API-url fra DAWA. Derefter går vi alle data igennem og bruger en funktion som er med til at hente en specifik adresse til vores fortidsminde.

```
function getAdresse($længde, $bredde)
{
  $url = "https://api.dataforsyningen.dk/adresser?cirkel=$længde,$bredde,10";

  $json = file_get_contents($url);

  $json = json_decode($json, true);
```

Illustration af getAdresse funktion.

Vores funktion består af at kunne finde en adresse der ligger nærmest de valgte koordinater fra vores første API (længde samt bredde). Funktionen tjekker den nærmeste radius på koordinatet og returnere derefter vores JSON-data. Vores funktion har flere if-statements der tjekker den nærmeste radius. Når denne er tom, forsøger funktionen at tage næste nærmeste radius adresse osv. Dette sikrer det bedst mulige datasæt til brugeren.

Efter funktionskald sætter vi vores værdier ind i databasen og lukker dernæst for forbindelse for ikke at lade den stå åbent og dermed forårsage fejl i systemet.

Problemstillinger i forhold til databehandling

Til selve opgaven i projektet bestod vores datasæt udelukkende af store datakilder. Det har været med til at udfordre både os og selve den måde at gribe tingene an på.

En af de første udfordringer var, at vi fik browseren til at eksekvere vores fil til databasen. Dette kunne ikke lade sig gøre da der ligger en timeout, og den dermed stoppede selve processen. Til denne problemstilling fandt vi ud af at det var nødvendigt at eksekvere filen ved hjælp af command prompt som følgende:

```
reneseebach@Renes-MBP php_lib % php dbfortidminder.php
```

Men det viste sig ikke at være nok. Da MySQL kun tillader at køre et bestemt antal rækker var vi nødt til at oprette en global variable som tillod et bestemt antal rækker som vist nedenfor:

```
SET GLOBAL max_allowed_packet=524288000;
```

Derudover skulle vi til enhver fil (dbfortidminder.php & fødevare.php) beskrive hvor lang tid den fik lov til at køre (default er 30 sekunder), da det ellers ville resultere i at stoppe før den var færdig. Vi blev nødt til at tilføje følgende som angiver hvor lang tid i sekunder filen køre:

```
// set execution time.
ini_set('max_execution_time', 36000);
```

Når man arbejder med store filer finder man også hurtig ud af, at der er en begrænsning på hvor meget memory der max må anvendes. Til dette formål har vi valgt at sætte den til uendelig men i princippet er det muligt at definere den præcis som man ønsker ved at tilføje følgende:

```
// set memory limit ulimited.
ini_set('memory_limit', '-1');
```

Har man ikke ovenstående på plads er det simpelthen ikke muligt at arbejde med så store datakilder uden at løbe ind i problemer. Med ovenstående var det muligt for os at køre vores data ind i data warehouse. Vores data warehouse kan ses som et lager af data, som man løbende kan opdatere en gang om måneden. Det er bedst at opdatere om natten og systemet kan selv opdatere dette efter egen indstilling. Et fortidsminde er ikke noget man til daglig har brug for at opdatere, og derfor er en gang om måneden passende.

Få vist Fortidsminder indenfor angivne adresse

Den første opgave går ud på, at få vist de fortidsminder, som er indenfor en given afstand af en given adresse. Som det første lavede vi en form, hvor det er muligt for en bruger at indtaste et vejnavn, husnr, postnummer og afstand. Disse informationer skal vi bruge til at finde de fortidsminder, som ligger indenfor de angivne oplysninger.
Vores form er opbygget på følgende måde:

```
<form class="inputFrom" action="fortidsminder.php" method="POST">
  <div class="searchInputLabel">
    <label class="labelstructure">Indtast vejnavn</label>
    <input name="vejnavn" type="text" placeholder="Vejnavn">
  </div>
  <div class="searchInputLabel">
    <label class="labelstructure">Indtast husnr</label>
    <input name="husnr" type="text" placeholder="Husnr">
  </div>
  <div class="searchInputLabel">
    <label class="labelstructure">Indtast postnummer</label>
    <input name="postnummer" type="text" placeholder="Postnummer">
  </div>
  <div class="searchInputLabel">
    <label class="labelstructure">Indtast afstand (meter)</label>
    <input name="afstand" type="text" placeholder="Max afstand" min="0">
  </div>
  <br>
  <div class="searchInputLabel">
    <input class="inputFront" name="submit" type="submit" value="Find fortidsminde">
  </div>
</form>
```

Informationerne som bliver postet i denne form, skal vi bruge til vores API'er, for at få de nødvendige informationer vi skal bruge.

Det første vi gør, er at angive variabler i PHP, som vi sætter lig med hver af værdierne som bliver postet af vores bruger. Her bruger vi Superglobalen $_POST, hvor vi kan hente de data som bliver postet.

```php
<?php

include "pretty_dump.php";

$vejnavn = $_POST['vejnavn'];
$husnr = $_POST['husnr'];
$postnummer = $_POST['postnummer'];
$afstand = $_POST['afstand'];
```

Disse værdier bruger vi i vores URL til vores api, for at få de data, som matcher præcis den adresse, som vores bruger indtaster, og dermed får den rigtige URL.

```php
$url_adress = "https://api.dataforsyningen.dk/adgangsadresser?vejnavn=" . rawurlencode("$vejnavn") . "&husnr=" . $husnr . "&postnr=" . $postnummer . "&struktur=mini";
```

Ud fra vores api bruger vi cURL til at hente de data, som vi skal bruge, og smider det ind i en variabel, som vi skal bruge senere:

```php
// create cURL resource
$curl = curl_init($url_adress);

// set cURL options
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($curl, CURLOPT_USERAGENT, 'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.2) Gecko/20090729 Firefox/3.5.2 GTB5');
curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, 0);

// Execute http request
$page = curl_exec($curl);
$data_adress = json_decode($page);
```

De data vi får tilbage, er angivet som JSON, men det er pakket ind i et array, hvilket gør at vi ikke direkte kan hente bestemte værdier, ved at henvende sig til key'en i objektet. Men da vi via vores form får brugeren til at angive både et vejnavn, husnr og postnummer, vil det sige, at vi kun vil få et sæt data ud. Derfor kan vi bruge index [0], for at komme ind til vores objekt, hvorefter vi kan få fat i de data vi ønsker.
Fra det første API, skal vi bruge længde- og breddegraderne fra den adresse, som brugeren har indtastet. Ud fra vores API-data kan vi se, at det er værdierne, som har key "x" og "y". Disse vil vi referere til, og gemmer værdierne i hver sin variabel, som vi skal bruge senere:

```php
$longitude_adress = $data_adress[0]->x;
$latitude_adress = $data_adress[0]->y;
```

Få fortidsminde indenfor given afstand

Vi har nu længde- og breddegraderne på adressen samt en afstand. Dette er alle de værdier, som vi skal bruge til det andet API, som vi sammensatte i vores research:

https://api.dataforsyningen.dk/steder?hovedtype=Fortidsminde&cirkel=x,y,r
Her er x = længdegrad, y = breddegrad og r = radius.

```
$longitude_adress = $data_adress[0]->x;
$latitude_adress = $data_adress[0]->y;

/* ------------------------------------------------------------------------------------------------------------*/

$afstand = $_POST['afstand'];

$url_circle = "https://api.dataforsyningen.dk/steder?hovedtype=Fortidsminde&cirkel=" . $longitude_adress . "," . $latitude_adress . "," . $afstand;
```

Igen bruger vi cURL til at hente data fra API'en, som vi putter i en variabel ($data_circle);

```
// create cURL resource
$curl2 = curl_init($url_circle);

// set cURL options
curl_setopt($curl2, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($curl2, CURLOPT_USERAGENT, 'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.2) Gecko/20090729 Firefox/3.5.2 GTB5');
curl_setopt($curl2, CURLOPT_SSL_VERIFYPEER, 0);

// Execute http request
$page2 = curl_exec($curl2);
$data_circle = json_decode($page2);
```

Vis informationer om valgte fortidsminde

Af de indhentede data fra ovenstående, indsætter vi navnet på fortidsmindet i en dropdown, som også får ID'et fra samme object med som en value. Det gør at vi med en query kan udvælge præcis det vi vil vise, ud fra det givne ID i databasen.
Hele processen foregår med AJAX (XMLHttpRequest), det gør det meget nemmere at håndtere dropdowns, da data bliver vist med det samme, i stedet for at skulle indlæse siden igen, eller trykke på en knap, og det er specielt praktisk i dette projekt, da vi har 2 dropdowns på samme side.

```
function showData(str) {
  if (str == "") {
    document.getElementById("dataTable").innerHTML = "";
    return;
  } else {
    let xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("dataTable").innerHTML = this.responseText;
      }
    };
    xmlhttp.open("GET", "fortidsminder_handler.php?q=" + str, true);
    xmlhttp.send();
```

Eksemplet her viser hvordan vi bruger AJAX i funktionen showData(). Funktionen har 1 parameter som er en value (ID) på en option fra en dropdown. Som det første kigger funktionen om en option er selected, hvis ikke noget er selected (str == ""), fjerner den indholdet af "dataTable" og stopper funktionen. Er der derimod valgt en option, udføres følgende:

1. Laver et ny XMLHttpRequest object
2. Laver en funktion som bliver executed når server response er klar (readyState == 4)
3. Sender requesten videre (xmlhttp.open(), xmlhttp.send())

Som det kan ses i xmlhttp.open() bygges der en query string i URLen, den sidste parameter "true" betyder at xmlhttp.open() gøres asynkron, hvor false er synkron.

```
<form>
  <select name="data" onchange="showData(this.value)">
    <option>-- Vælg Fortidsminde --</option>
    <?php
    include "form_handler.php";
    foreach ($data_circle as $data) {
      $navn = $data->primærtnavn;
      $id = $data->id;
      echo "<option value='" . $id . "'>" . $navn . "</option>";
    }
    ?>
  </select>
</form>

<br>
<div id="dataTable"></div>
```

På billedet herover ser vi en dropdown menu, som med "onchange" altså når der selectes forskellige options, starter showData() funktionen beskrevet ovenover, med den valgte options ID som parameter.
I bunden af billedet ses div'en med id=dataTable, som er her vores data bliver vist.

```php
$q = strval($_GET['q']);
mysqli_select_db($con, "fortidsminderdb");
$sql = "SELECT * FROM steder WHERE id = '" . $q . "'";
$result = mysqli_query($con, $sql);

echo "<table>
<tr>
<th>Navn</th>
<th>Type</th>
<th>Kommune</th>
</tr>";
while ($row = mysqli_fetch_array($result)) {
  echo "<tr>";
  echo "<td>" . $row['primærtnavn'] . "</td>";
  echo "<td>" . $row['undertype'] . "</td>";
  echo "<td>" . $row['kommunenavn'] . "</td>";
  echo "</tr>";
}
echo "</table>";
mysqli_close($con);
```

På billedet her bliver requesten behandlet. Variablen $q sørger for at supervariablen $_GET['q'] er en string. "q" som er inde i arrayet, er den querystring fra URL'en som beskrevet tidligere, altså ID'et på den valgte option i formen. Derefter skabes der forbindelse til databasen og med en sql sætning returneres der data hvor id= querystring (ID).
Herefter laves så et HTML table som indeholder de valgte data, hvorefter det bliver sendt tilbage til div'en med id=dataTable.

Det var en kort beskrivelse af hvordan vi benytter os af AJAX samt en dropdown til at få vist data omkring fortidsminder. Anden del omfatter visning af restauranter indenfor given afstand fra et bestemt fortidsminde, og her benytter vi os af samme princip som netop beskrevet.

Evaluering af proces

Vi har gennem vores arbejde med projektet arbejdet flittigt på at nå i mål, hvilket har indebåret både individuelle opgaver, men også problemstillinger vi i fællesskab har løst.
Vi brugte noget tid på at finde de rigtige API'er, som vi skulle arbejde med. Men da vi fandt frem til det rigtige sted, fandt vi hurtigt ud af, hvilke API'er og data vi skulle bruge.
Vi kom godt fra start og fik hurtigt løst nogle funktioner, men vi har også stødt på udfordringer undervejs, hvor koden ikke har virket.
Generelt har gruppearbejdet og projektet fungeret fint. Opdelingen af opgaver har gjort, at vi alle har fået arbejdet på noget backend. Hvis vi har haft problemer, har vi været gode til at hjælpe hinanden og fået løst problemet, så godt vi kunne.

Konklusion

Vi kan konkludere, at vi har fået løst den første opgave stort set 100%. Vi har gjort det muligt for en bruger at indtaste en adresse og en afstand, hvorefter brugeren vil få en dropdown med de fortidsminder, som ligger indenfor den angivne afstand. Hertil har vi sørget for, at når en bruger vælger et fortidsminde, så vil der dukke informationer frem om dette fortidsminde.
I forhold til at vise restauranter frem, som er i nærheden af det valgte fortidsminde, kan vi konkludere, at vi har fået det ordnet på den måde, at alle restauranter indenfor samme postnummer bliver vist i en dropdown. Dette er selvfølgelig ikke helt optimalt, at der kan være langt hen til en restaurant, men vi kunne ikke nå, at lave det på den måde, at det kun er restauranter indenfor en given afstand, som kommer frem.

Referencer

Adgangsadresser (n.d.) available from <https://dawadocs.dataforsyningen.dk/dok/api/adgangsadresse#s%C3%B8gning> [24 March 2022]

PHP: CURL - Manual (n.d.) available from <https://www.php.net/manual/en/book.curl.php> [24 March 2022]

PHP: Introduction - Manual (n.d.) available from <https://www.php.net/manual/en/intro.curl.php> [24 March 2022]

Slots- og Kulturstyrelsen (n.d.) *Fund Og Fortidsminder* [online] available from <https://www.kulturarv.dk/fundogfortidsminder/Download/> [24 March 2022]

Toft, P. (n.d.) *REST/Assignment1_accessing_api_through_curl_1.Php · Main · Dataintegration / Dataintegration* [online] available from <https://gitlab.com/dataintegration2/dataintegration/-/blob/main/REST/assignment1_accessing_api_through_curl_1.php> [25 March 2022]