



# PERSOONLIJK DOSSIER

Oman Biodiversity

PROJECT WERK  
2014-2015

Tom Adriaens  
CVO ANTWERPEN – CAMPUS HOBOKEN  
Informatica – specialisatie programmeren

## Inhoud

Inleiding	2
Taken	3
Databanken	3
Use-casses	6
Data Access Layer	6
Switch met verschillende databanken	7
Bootbox	9
Modal	10
Besluit	13

## Inleiding

Zoals u in het algemeen dossier hebt kunnen lezen, was de opdracht een begeleidende website te maken voor een tentoonstelling over de biodiversiteit van Oman. De tentoonstelling werd vooral bezocht door scholen en had een doelpubliek van 8 tot 12 jarigen.

De informatie, die handelt over tentoongestelde dieren en die ingevuld werd door wetenschappers en stagiaires, moest op een dynamische wijze op de website komen. We dienden er ook rekening mee houden dat deze informatie gevalideerd moest kunnen worden voor publicatie.

De informatie over de dieren moest een logische samenhang hebben. De tentoonstelling bestond uit vier delen met name : planten ( de leefwerelden waar de dieren in voorkomen ), landdieren, microben en tenslotte zeedieren. De dieren behoren tot een familie en een variëteit. Deze komen dan weer voor in een bepaald gebied in Oman en mogelijk in een bepaald seizoen.

De website moest ook interactief zijn. Gebruikers moesten de mogelijkheid hebben om waar een bepaald dier gespot werd, te posten op de website . In een latere fase heeft de opdrachtgever hier vanaf gezien en gekozen om interactie te hebben met de sociale media zoals Facebook en Instagram. Bezoekers aan de website moeten de mogelijkheid hebben zich in te schrijven voor een nieuwsbrief.

De keuze van de backend taal is JAVA geworden met een databank in MySQL. Onze groep had net de cursus programmeren 4, JAVA achter de rug en de host van de opdrachtgever ondersteunde enkel PHP waar niemand van onze groep ervaring mee had. We zijn toen in opdracht van onze opdrachtgever op zoek gegaan naar een andere host en kwamen tot de ontdekking dat hosting met C# en een Sql databank zeer duur zou zijn. Tijdens het verder verloop van het project hebben we daar goedkopere oplossingen voor gevonden maar omwille van tijdsdruk was de keuze gemaakt.

In de frontend hebben we gekozen om Bootstrap te gebruiken. 100% responsive design was een must omdat er op de tentoonstelling gebruik werd gemaakt van QR codes om meer informatie op te vragen over de tentoongestelde objecten.

Voor mij was het hele project een heel steile leercurve. De meeste van de gebruikte technieken zijn wel aanbod gekomen tijdens de studies maar zeker niet in deze mate. Ik heb degelijk moeten bij schaven in AJAX , JAVASCRIPT, JQUERY en Bootstrap maar het was zeker de moeite waard.

## Taken

- ❖ Use cases schrijven
- ❖ Databank helpen uitwerken
- ❖ Databank diagrammen genereren
- ❖ Dal methodes schrijven
- ❖ Service methodes schrijven
- ❖ Servlets methodes schrijven
- ❖ Forms uitwerken
- ❖ Ajax toevoegen
- ❖ Layout elementen toevoegen
- ❖ Downloads opmaak en implicatie
- ❖ Debugging
- ❖ Correspondentie met de opdrachtgever

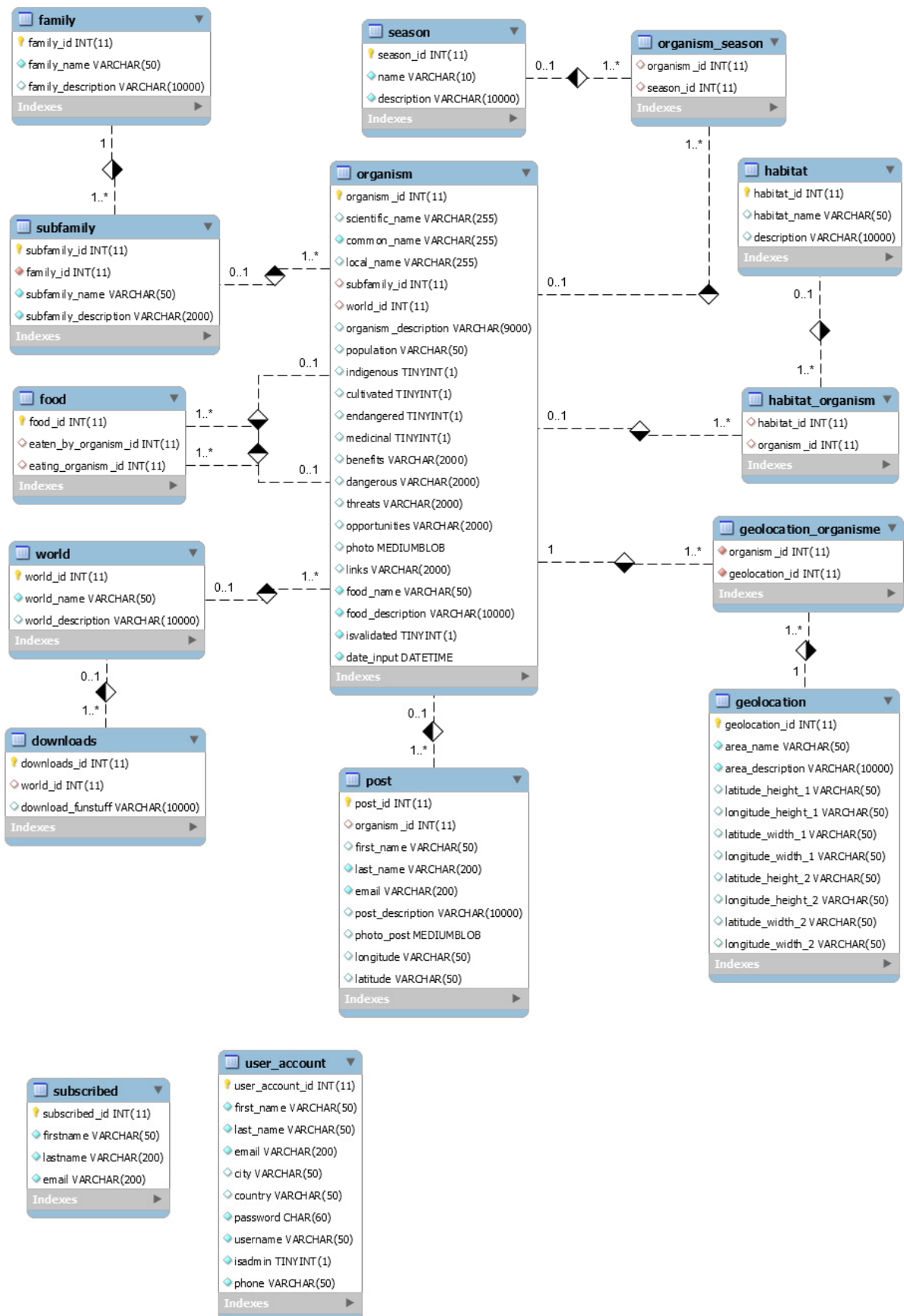
## Databanken

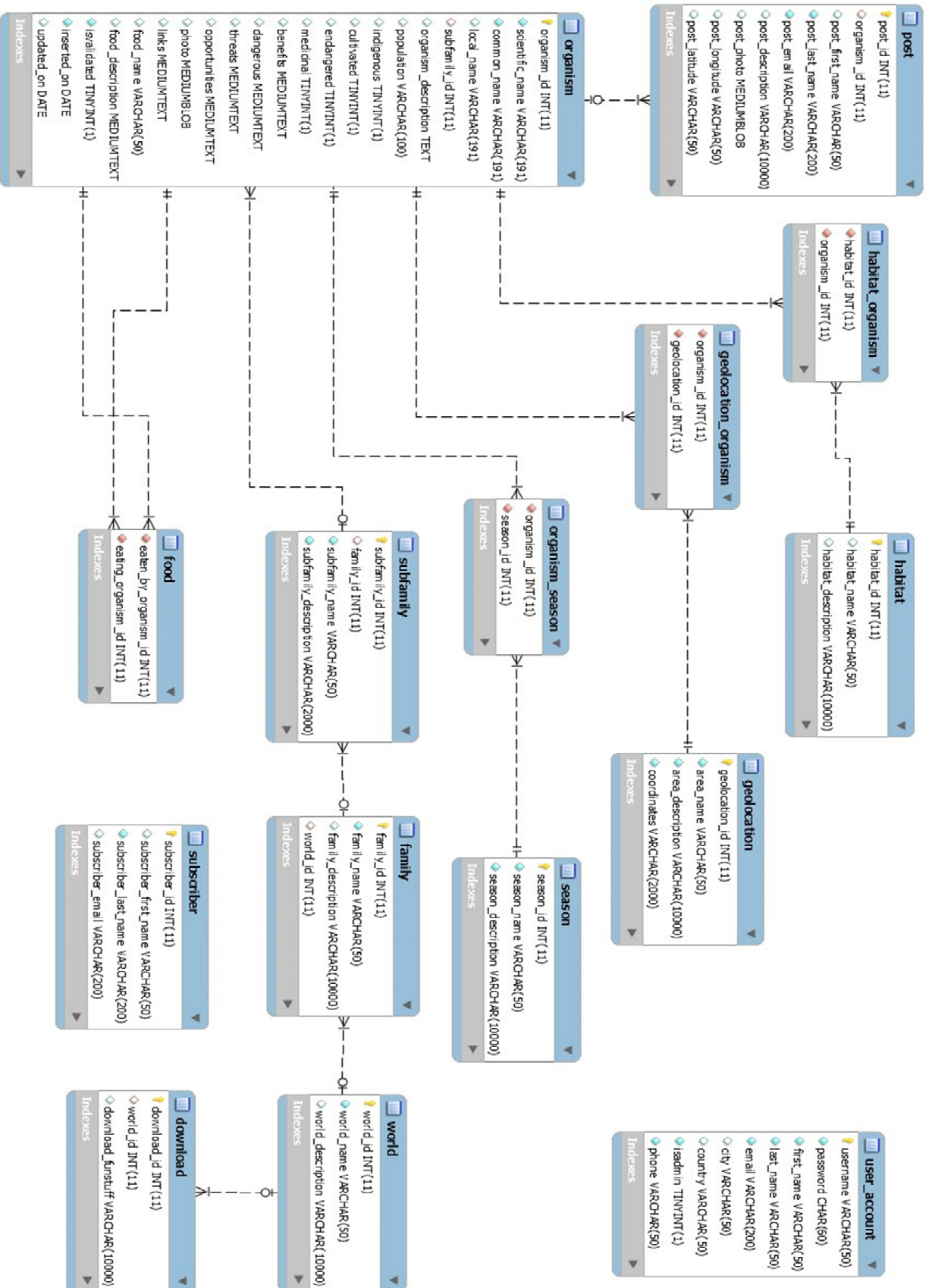
De opstelling van de databank was niet gemakkelijk. Dat we voor de web applicatie een login systeem moesten hebben met een gewone gebruiker en administratieve gebruiker met alle rechten, lag voor de hand. Dit kon door enkel een boolean toe te voegen. De tabel voor de nieuwsbrief was ook gemakkelijk, maar de samenhang van de dieren was net iets moeilijker.

Lange en heftige discussies waren het gevolg. Hier heb ik heel goed leren omgaan met phpMyAdmin en Workbench. Om de discussies te verkorten, heb ik telkens de voorgestelde verandering met phpMyAdmin doorgevoerd. Zonder omslachtig invoeren van code kon ik snel veranderingen aanbrengen en dan via Workbench een databankdiagram aanmaken. Aan deze diagrammen werd snel herkend waar nog velden moesten toegevoegd worden, welke overbodig waren, welke relaties vereenvoudigd konden worden of eerder een many to many relatie moesten hebben. Zo zijn er zeker een dertigtal voorstellen de revue gepasseerd voor we tot ons uiteindelijke resultaat zijn gekomen. Natuurlijk was phpMyAdmin niet voldoende om alle problemen op te lossen, vandaar zijn er aan het Sql script nog manuele veranderingen aangebracht, voornamelijk waar relaties moesten bewerkt worden zoals bv. cascade on delete.

Het resultaat is een genormaliseerde databank, misschien wel tot op het punt waar u zal denken dat het overdreven is, maar wij moesten er rekening mee houden dat deze website nog in het Arabisch word vertaald. Zo hebben we er bv. ook de werelden of seizoenen, terwijl dit ook hard coded gekund had.

Hieronder vind u het eerste en het uiteindelijke databank design.





## Use-cases

Op één na heb ik use-cases voor mijn rekening genomen, deze vindt u terug in het algemeen dossier. Als oudste van de groep heb ik de meeste jaren als technisch analfabeet achter de rug en was ik de aangewezen persoon. De meeste CRUD use-cases zijn zoals verwacht enkel niet de Delete functies. Ik heb erop aangedrongen dat bij World, Family en Breed dubbele bevestigingen werden toegevoegd met heel duidelijke boodschappen wat er gebeurt als deze verwijderd worden. Ook vond ik het belangrijk om te vermijden dat de laatste administratieve gebruiker kon worden verwijderd. Terwijl voor ons het gevolg duidelijk was, vond ik dat voor gewone gebruikers deze samenhang en het gevolg niet voor de hand liggend zijn maar wel nefast voor de databank.

## Data Access Layer

Hier hebben we nagedacht over de keuzes Hibernate, JPA of JDBC. Na een lang overleg hebben we gekozen voor JDBC met als belangrijkste reden dat dit door alle teamleden gekend was. Ik heb hier DaPost, DaWorld, DaSubscriber, DaFamily, DaHabitat en DaSeason voor mijn rekening genomen.

Elke DAL Klasse bevat minstens:

- ❖ insert
- ❖ delete
- ❖ selectAll
- ❖ selectOneById
- ❖ update
- ❖ checkIfExist

```
public static void insert(World world) throws SQLException {  
    try {  
        conn = DataSource.getConnection();  
        conn.setAutoCommit(false);  
        stmt = conn.prepareStatement("INSERT INTO world (world_name, world_description) VALUES (?,?)");  
        stmt.setString(1, world.getWorldName());  
        stmt.setString(2, world.getDescription());  
        stmt.executeUpdate();  
        conn.commit();  
    } catch (SQLException ex) {  
        conn.rollback();  
        System.out.println(ex.getMessage());  
    } finally {  
        conn.setAutoCommit(true);  
        conn.close();  
    }  
}
```

Om verbinding te maken met de databank hebben we gebruik gemaakt van de klasse DataSource met de statische methode getConnection(). Deze hebben we telkens opnieuw gebruikt in al onze DAL klasse. Ook hebben we AutoCommit pas aangezet als de hele Query doorlopen is. In de bovenstaande insert methode is het misschien overbodig maar zeker niet in een selectAll() Query.

Rollback zorgt ervoor dat alle ingevoegde data terug worden gezet, indien er iets zou foutlopen in de Query, en dat alle gebruikte resources weer vrij gemaakt worden.

Om Sql-injection op dit niveau te vermijden, hebben we er opgelet dat dat alle parameters gebonden zijn doormiddel van een vraagteken.

Boodschappen doorgeven naar de frontend gebeurt in de Service. Zo wordt daar een fout met de databank opgevangen met een Sql Exception, vertaald in een leesbare boodschap voor de gebruiker. Deze zit in elke methode ingebakken maar om te kijken of een object reeds bestaat moest een extra methode geschreven worden namelijk `checkIfExists`.

```
public static boolean checkIfExists(String worldName) throws SQLException {  
    boolean match;  
    conn = DataSource.getConnection();  
    stmt = conn.prepareStatement("SELECT COUNT(*) world_name FROM world WHERE world_name = ?");  
    stmt.setString(1, worldName);  
    ResultSet rs = stmt.executeQuery();  
    rs.next();  
    match = rs.getInt(1) == 1;  
    conn.close();  
    return match;  
}
```

Door gebruik te maken van `SELECT COUNT(*)` met de te zoeken parameter en deze Query 1 maal te doorlopen moet het resultaat 1 zijn. Indien dit niet het geval is, zal de boolean match 0 zijn en weten we dat de parameter niet bestaat.

## Switch met verschillende databanken

Als eerste moesten we een verbinding maken met de databank en een mogelijkheid voorzien dat de applicatie in twee talen ging werken en dus ook een tweede databank voorzien. Een aparte DAL klasse voor de verbinding met een statische methode `getConnection()` leek ons het beste idee en een simpele if/else leek ons de makkelijkste oplossing.

Eerst moest de Engelstalige databank klaar zijn en hadden we niet verder nagedacht over het toevoegen van een andere databank. Nu we echter nog een tweede moesten toevoegen en de andere Dal functies reeds geschreven waren, waar we gebruik maakten van de statische `getConnection()` methode, konden we daar niet telkens een extra parameter meegeven. Overal een boolean English/Arabic aan toevoegen, leek ons geen goed idee.

Lenny en ik zijn op het idee gekomen deze te integreren via de login pagina. Dit maakt het ook voor de gebruiker gemakkelijk omdat deze niet op elke pagina moet kiezen op welke taal de databank moet staan. Op deze manier konden we het ook gemakkelijk toevoegen aan het reeds gebruikte Sessie Object.



```

int language = Integer.parseInt(request.getParameter("language"));
String username = request.getParameter("username");
HttpSession session = request.getSession();
try {
    if (language == 1) {
        ServDataSource.switchDataSource(true);
    } else {
        ServDataSource.switchDataSource(false);
    }
}

```

Nu moesten we enkel nog een Service aanmaken, zodat het ook in de DAL klasse Datasource kon worden opgevangen.

```

public class ServDataSource {
    public static void switchDataSource(boolean language) throws SQLException{
        DataSource.setLanguage(language);
    }
}

```

Daar waren twee extra methodes voor nodig:

```

public static boolean isLanguage() {
    return language;
}
public static void setLanguage(boolean aLanguage) {
    language = aLanguage;
}

```

Om uiteindelijk te komen tot:

```

public static java.sql.Connection getConnection() throws SQLException {
    if (!isLanguage()) {
        return (DriverManager.getConnection("jdbc:mysql://localhost:3306/omandb?useUn
    } else {
        return (DriverManager.getConnection("jdbc:mysql://localhost:3306/omandbarabic?
    }
}

```

Achteraf gezien is dit een eenvoudige oplossing maar het heeft ons toch wat tijd gekost om er achter te komen.

## Bootbox

Omdat ik erop aangedrongen had dat voor de delete functies zekere boodschappen moesten weergegeven worden met dubbele confirmaties, heb ik een andere plugin van Bootstrap geïmplementeerd namelijk Bootbox. Deze creëert, managet en verwijdert dialogboxen of JS event- handlers zonder in te grijpen op de DOM elementen.

Om hier mee van start te gaan moesten we een kleine javascript library importeren. Alle events worden gestuurd vanuit onze AJAX.js in de \$(document).ready(function (), dus ook dit moest van hieruit werken.

Door het click event op de knop met de id “delete-family-btn” te verbinden, word er een confirm box geactiveerd met een boodschap. Indien de gebruiker OK aangeeft wordt de callback, result gelijk aan true. In onderstaand voorbeeld triggert dit echter een tweede confirm box. Indien deze keer result ook op true gezet wordt, zal de AJAX functie starten. De family met de id die gelijk gesteld werd aan \$(this).attr("value") wordt doorgegeven aan de controller DeleteFamily, die een id verwacht om deze vervolgens door te geven aan de service ServFamily om daarna via de DaFamily verwijderd te worden uit de databank. Vervolgens worden alle relevante load functies weer geladen vanuit Loaders.js.

```
// functie voor delete family btn in dashboard.jsp
$(document).on('click', '.table #delete-family-btn', function () {
    var id = $(this).attr("value");
    bootbox.confirm("<center>This will delete all references to this family \n\
        in breeds & linked organisms.<br><br> <b>Are you sure?</b></center>\n\
        ", function (result) {
        if (result === true) {
            bootbox.confirm("<center><b>ARE YOU VERY SURE?</b><br><br>This \n\
                will delete all references to this family in breeds\n\
                & linked organisms.</center> ", function (result) {
                if (result === true) {
                    $.ajax({
                        url: 'DeleteFamily?id=' + id,
                        type: 'POST',
                        dataType: 'text',
                        cache: false,
                        async: true
                    }).done(function () {
                        loadFamilies();
                        loadSubFamilies();
                    });
                }
            });
        }
    });
});
});
// functie voor delete habitat btn in dashboard.jsp
```

Omdat de standaard kleuren niet pasten in ons design, moesten we enkel nog een kleine aanpassing doen in Bootbox.js en in de admin.css

```

if (!button.className) {
  if (total <= 2 && index === total-1) {
    // always add a primary to the main option in a two-button dialog
    button.className = "button-grey";
  } else {
    button.className = "button-grey";
  }
}

```

```

.button-grey {
  font-weight: bold;
  height: 2em;
  color: white;
  font-size: 1.1em;
  border: 0px solid #000000;
  border-radius: 0.3em 0.3em 0.3em 0.3em;
  background-color: #666666;
  height: 2.3em;
  min-width: 8em;
}

.button-grey:hover{
  color: white;
  background-color: #444444;
}

```

## Modal

Nadat de admin pagina een tijdje in gebruik was hebben we de feedback gekregen dat de gebruikers het moeilijk vonden om een familie of een variëteit toe te voegen tijdens het ingeven van een organisme. Men moest het tabblad verlaten om één of twee anderen te openen en daar deze eerst toe te voegen.

Om dit op te lossen hebben we gebruik gemaakt van een dialog functie van Bootstrap namelijk Modal. Omdat deze reeds in onze bootstrap.js zat, moest niets extra toegevoegd worden. Modal werkt op hetzelfde principe als BootBox maar is iets uitgebreider. Dus als de gebruiker in het create-organism form naast Breed op de Add knop drukt, verschijnt er een nieuw venster.

```

<button type="button" class="button-grey" data-toggle="modal" data-target="#add-family-breed">Add</button>

```

Na het drukken op deze knop wordt de initiële staat van de classe aria-hidden = true op false gezet. De opmaak van een Modal werkt net hetzelfde als de rest van bootstrap met rows en columns. Ik heb er wel de meeste elementen van de standaard opmaak moeten uithalen, want deze was te uitgebreid.

```
<div class="modal fade" id="add-family-breed" tabindex="-1" role="dialog" aria-labelledby="add" aria-hidden="true">
  <div class="modal-dialog" id="modal-dialog">
    <div class="modal-content">
      <div class="modal-body">
        <div class="row">
          <button type="button" class="close" data-dismiss="modal" id="close-modal" aria-hidden="true"><i>times</i></button>
        </div>
        <div class="row">
          <div class="col-md-6">
            <form class="form form-horizontal" id="create-family-modal" data-csyle="validator">
              <fieldset>
                <legend class="text-center">Add Family</legend>
                <div class="form-group">
                  <label class="col-sm-3 control-label" for="family-name">Name</label>
                  <div class="col-sm-9">
                    <input class="form-control" type="text" name="family-name" maxlength="50" data-delay="1200" pattern="^[a-zA-Z0-9- ]+$" required/>
                    <span class="help-block with-errors">Up to 50 characters upper/lower case(no digits)</span>
                  </div>
                </div>
                <div class="form-group">
                  <label class="col-sm-3 control-label" for="family-description">Description</label>
                  <div class="col-sm-9">
                    <textarea rows="3" class="form-control" name="family-description" data-error="error" data-pattern="^[a-zA-Z0-9- ]+$" required/>
                    <span class="help-block with-errors"></span>
                  </div>
                </div>
              </fieldset>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>
```

De velden zijn dezelfde als van de forms create-family en create-subfamily, evenals de verwerking via AJAX. Hier moesten de respectievelijke functies herhaald worden met de id add-family-breed.

```
// functie inserten van family. in modal dashboard.jsp
$('#create-family-modal').submit(function (e) {

    var $message = $('#create-family-message-modal');
    $message.show();
    $.ajax({
        url: 'InsertFamily',
        type: 'POST',
        dataType: 'text',
        data: $('#create-family-modal').serialize()
    }).done(function (data) {
        if (data === 'succes') {
            $message.append('<div class="alert alert-success" role="alert">\n\
            <a href="#" class="close" data-dismiss="alert">&times;</a>\n\
            Family successfully created.</div>');
            loadFamilies();
            $('#create-family-modal')[0].reset();
            setTimeout(function () {
                $('#create-family-modal').find('label').parent().attr('class', 'form-group');
            }, 2800);
        } else if (data === 'exists') {
            $message.append('<div class="alert alert-danger" role="alert">\n\
            <a href="#" class="close" data-dismiss="alert">&times;</a>\n\
            Family name already exists!</div>');
        } else if (data === 'sql') {
            $message.append('<div class="alert alert-danger" role="alert">\n\
            <a href="#" class="close" data-dismiss="alert">&times;</a>\n\
            Service unavailable!</div>');
        } else if (data === 'required') {
            $message.append('<div class="alert alert-danger" role="alert">\n\
            <a href="#" class="close" data-dismiss="alert">&times;</a>\n\
            Fill in the required fields!</div>');
        }
        setTimeout(function () {
            $message.fadeOut('slow');
            $message.empty();
        }, 2800);
    });
    e.preventDefault();
});
```

De dropdown listen werden gevuld via de AJAX functies loadWorlds en LoadFamilies. Hier moest er alleen voor gezorgd worden dat deze de gelijke id kregen als in het originele form. Dit was dan ook een voordeel om met AJAX te werken. Zodra er één werd geüpdatet , volgde ook de andere.

```
// functie vult tabel in World tab in dashboard.jsp
function loadWorlds() {

    var $stable = $('#worlds-table');
    var $ddl = $('#world-ddl-insert-organism, #world-ddl-insert-family, \n\
                #world-ddl-update-family, #world-ddl-pending, #world-ddl-published,\n\
                #world-ddl-queue');

    $.ajax({
        url: 'SelectAllWorlds',
        type: 'GET',
        dataType: 'json',
        cache: false,
        async: true
    }).done(function (data) {
        $stable.html('');
        $ddl.html('');
        $ddl.append('<option value="" disabled selected>Select World</option>');
        $stable.append('<tr>\n\
                        <th>Name</th>\n\
                        <th>Description</th>\n\
                        <th></th>\n\
                        </tr>');

        data.forEach(function (world) {
            $ddl.append('<option value="' + world.worldId + '">'
                        + world.worldName + '</option>');
            $stable.append('<tr>\n\
                            <td>' + world.worldName + '</td>\n\
                            <td>' + world.description + '</td>\n\
                            <td>\n\
                            <button class="no-button" id="update-world-btn" \n\
                                type="submit" value="' + world.worldId + '">\n\
                                <span class="icon-pencil2"></span></button>\n\
                            <button class="no-button" id="delete-world-btn" \n\
                                type="submit" value="' + world.worldId + '">\n\
                                <span class="icon-cross"></span></button>\n\
                            </td>\n\
                            </tr>');
        });
        adminCheck();
    });
};
```

## Besluit

Het was een zeer fijne en leerrijke opdracht met heel veel werk. Ik had het geluk deel uit te maken van een team dat goed kon samenwerken en ook de tijd nam me uitleg te geven bij de delen die ik niet begreep. Ik nam deel aan alle onderdelen van het project met uitzondering van de foto's, regex en de geolocatie. Zo heb ik heel veel geleerd over Bootstrap, AJAX, JAVASCRIPT en JQUERY, items die in de lessen niet of bijna niet aanbod zijn gekomen maar naar mijn bescheiden mening een must zouden moeten zijn in de leerstof en na kort onderzoek ook op de arbeidsmarkt zeer gevraagd zijn.

We zijn allemaal trots op het resultaat, hebben tevreden opdrachtgevers en een eindwerk dat niet misstaat in een Curriculum Vitae.