



ITPE3200-1 22H Web Application Group Exam - Part 1

Candidate numbers: 100, 112, 252, 265 & 285

Group number on inspera: 9597

Course code: ITPE3200

Course name: Web Application

Number of words: 2767

Number of pages: 35

Deadline: 31.10.2022, 12:00

OSLO METROPOLITAN UNIVERSITY
STORBYUNIVERSITETET

Table of contents

1.0 Introduction	3
2.0 Prototyping	4
2.1 The Process	4
3.0 Requirements specification and UML diagrams	7
3.1. Requirements specification	7
3.2 Use case	7
3.3 Sequence diagram	9
3.4 Class diagram	10
4.0 Frontend	11
4.1 Header - Index	11
4.2 UFO observations	11
4.3 Footer	12
4.4 About us explanation	12
4.4.1 Header - About us	13
4.4.2 Events/ history	13
4.4.3 Explore page	14
4.4.4 Our team	14
4.4.5 Responsiveness	15
4.5 Color combinations	15
5.0 Backend	17
5.1 Model	17
5.2 Database	18
5.3 Repository	19
5.4 Controller	24
5.5 View	25
5.5.1 Create sighting	26
5.5.2 Update sighting	27
5.5.3 Delete sighting	29
5.5.4 Feed	29
5.5.5 Extra functionality on the homepage	31
References	33

1.0 Introduction

For this assignment we chose the second task which was to create a web solution that can maintain a database of UFO observations. We developed our website where an admin can create, read, update and delete UFO observations. Our web page consists of four pages, these are Home, Feed, Submit a report and About Us. In this report we will explain about our process, how the frontend and backend works.

2.0 Prototyping

2.1 The Process

The process for the front end started with creating a prototype for the website using Miro's whiteboard. We start by brainstorming and making a low-fidelity prototype. In the brainstorming process we agreed to have 4 html pages, which is index/Home, submit a report, gallery and about us.



Figure 1 - Low-fidelity prototype for website

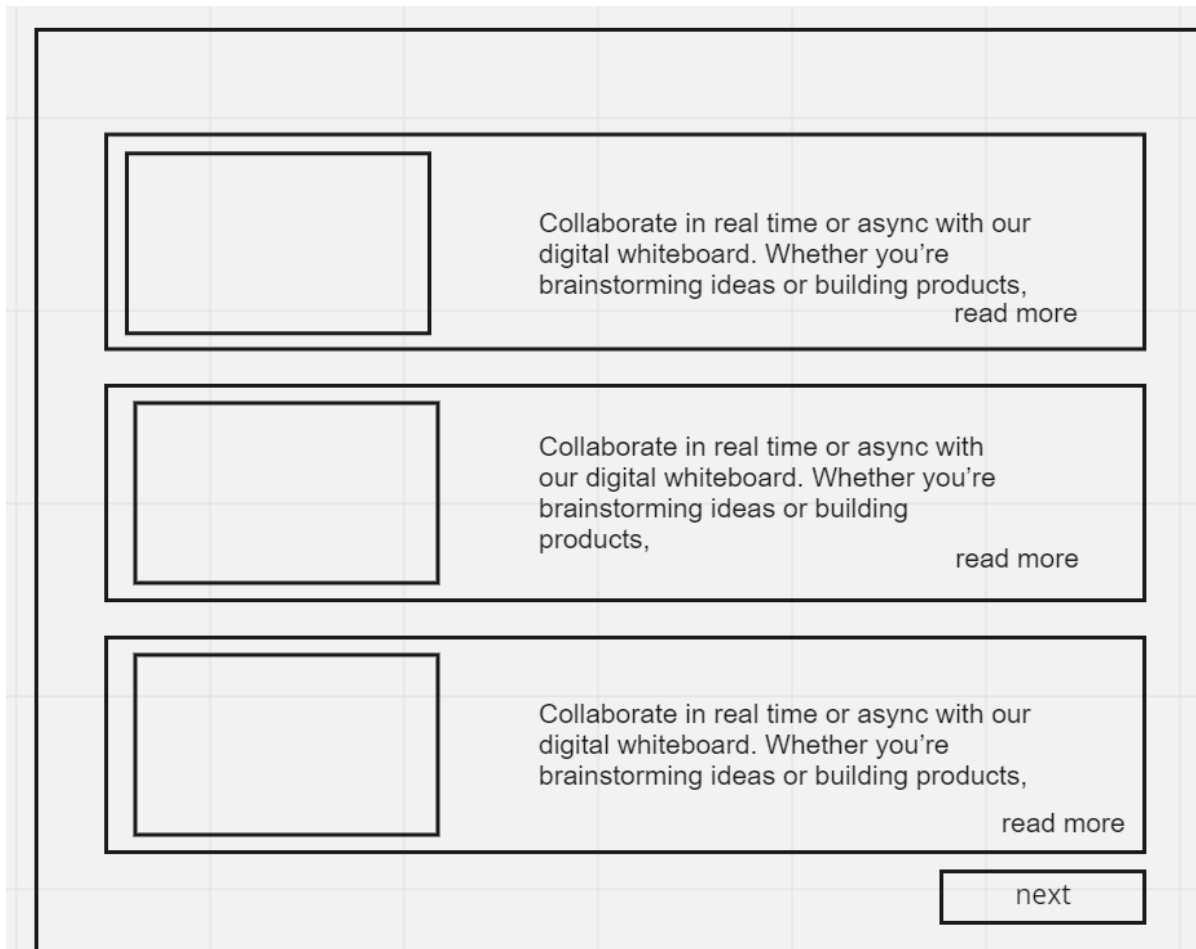


Figure 2 - Low-fidelity prototype for website

The “finished” website does contain several changes from the original ideas and prototypes. Little by little, we saw the need to adjust the design to be able to contain all the info we wanted to present. This can e.g. especially be seen in the “feed” section with the reported sightings. Instead of pictures on the left side, we ended up having the date of publishing, date and time of the sighting etc. there instead, as shown in the picture below. We did drop having the name of the publisher shown, as these are not connected to several users yet due to the web page being an application for admin use for now.

Date Published	Sighting in City, Country	Lastname, Firstname
Date / Time	Comments	
Duration		

Sighting in City, Country

By Lastname, Firstname

Thursday, 13 October 2022

Duration: 23 minutes

Comments: I found a spaceship behind my house!

Figure 3 - Low-fidelity prototype for website, Sighting

The prototype shows a web page layout for a 'File report' section. At the top, there is a header bar containing a 'File report' title and a 'logo' placeholder. Below the header, the main content area is divided into two columns. Each column contains four text input fields, each preceded by a placeholder text 'write something' or 'write'. A 'submit' button is located at the bottom right of the main content area.

Figure 4 - Low-fidelity prototype for website, File report

3.0 Requirements specification and UML diagrams

3.1. Requirements specification

Function	Description
Menubar	It will sort the right products for the user
sort	The user can select the database by selecting typing, id, country or city.
read	The user can read the content of our website
create	The user can create a UFO observation
update	The user can update
delete	The user can delete

Figure 5 - requirements specification

3.2 Use case

A use case is a written description or model of how your website's users will complete tasks. in the image below we see that there are users of this website. One is an admin and the other one is a user or reporter. A reporter can create a report by filling in the form and can read other sightings that are created by others. but update and delete can only be by admin. The admin can do everything on this website.

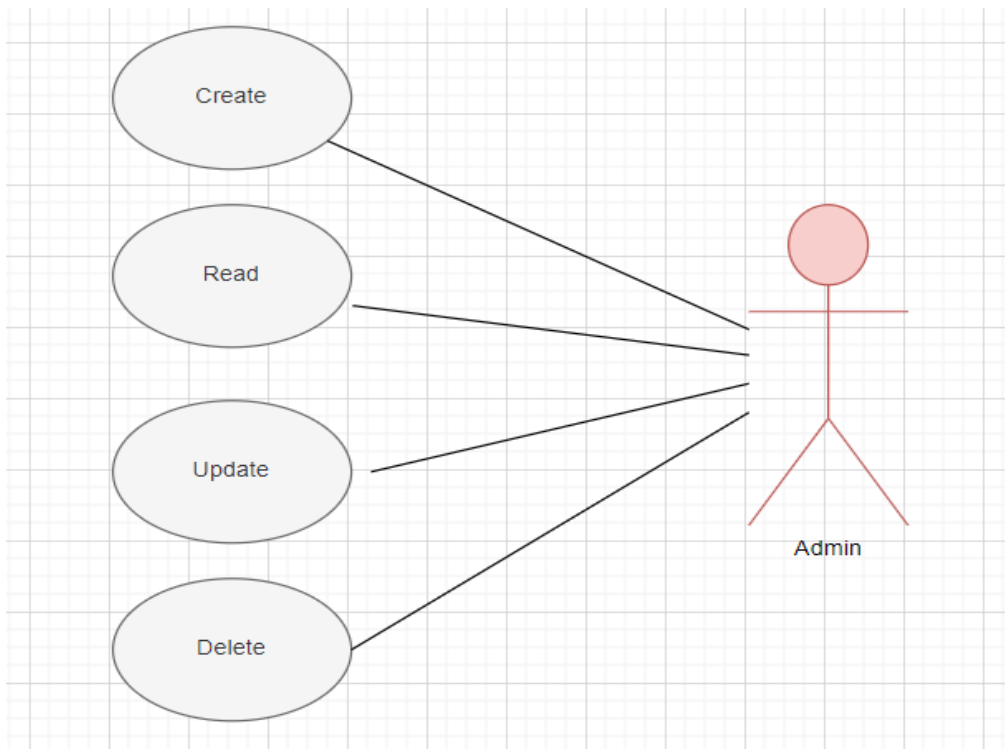


Figure 5 - user case for our website

3.3 Sequence diagram

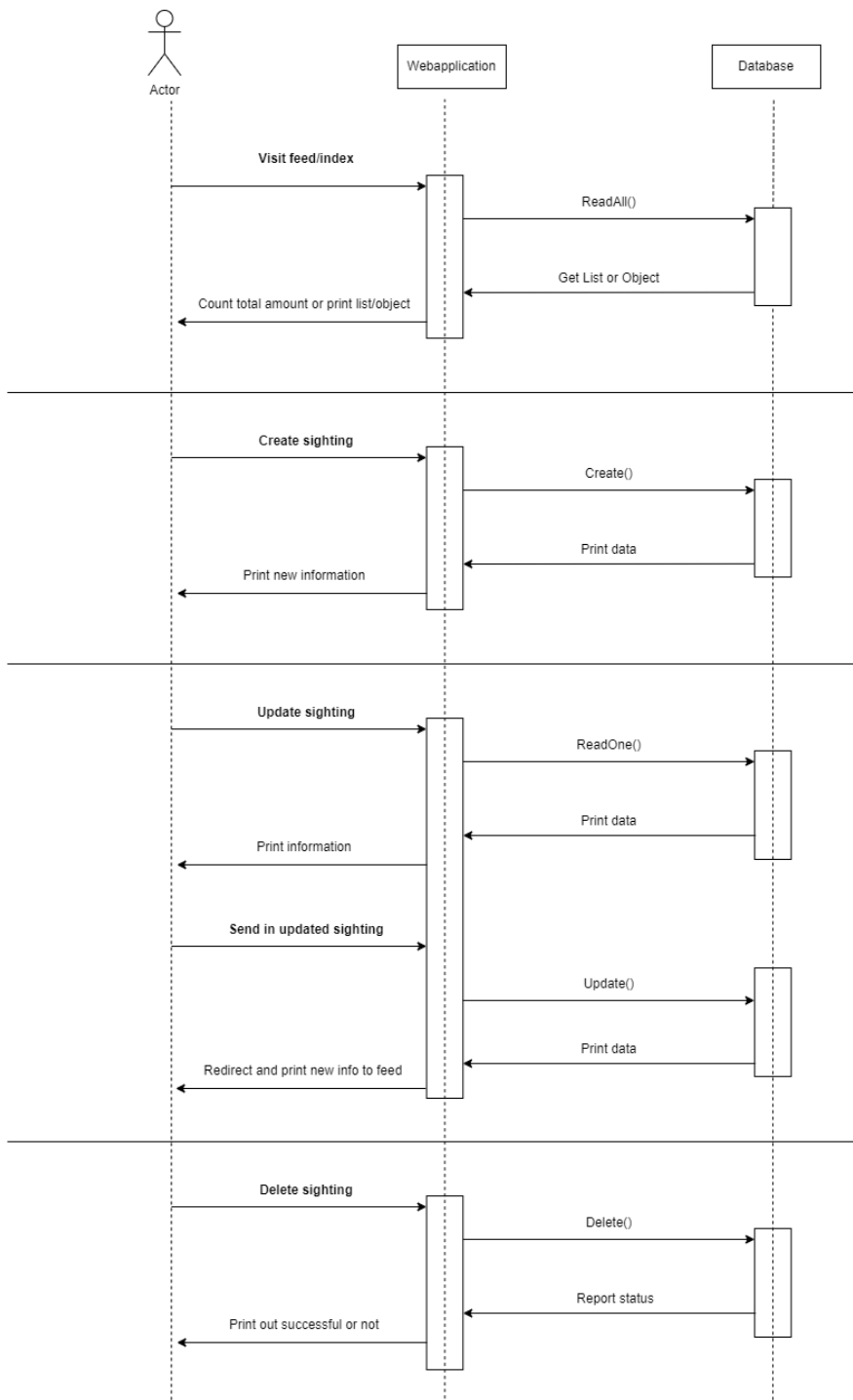


Figure 6 - sequence diagram for our website

3.4 Class diagram

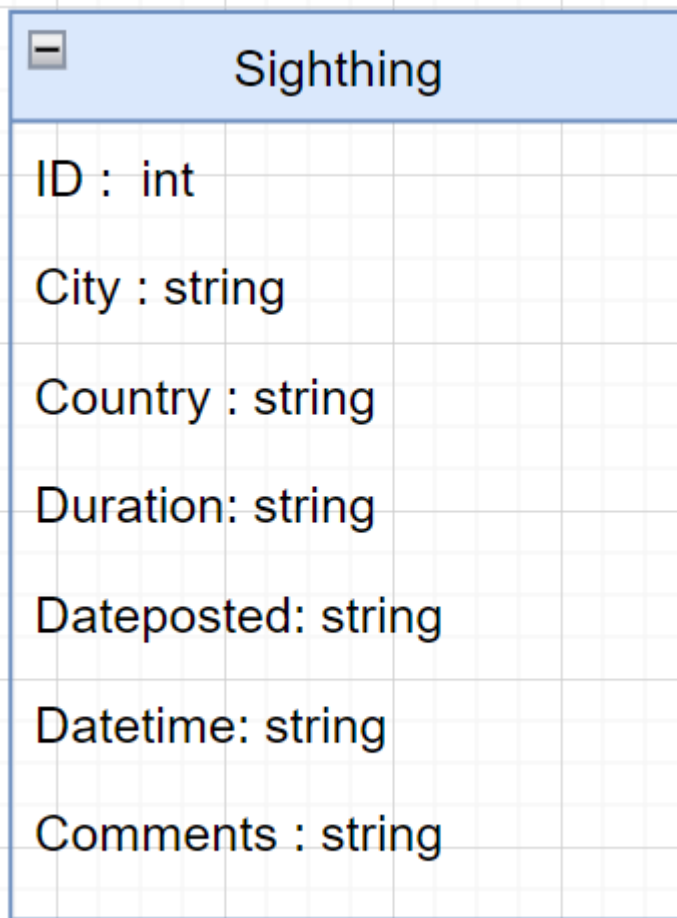


Figure 7 - class diagram for sighting

We only have one table in this project for now, which we created a class diagram for. The class diagram will be extended for the second part of the exam, as there will be more tables to include. In the picture above the attributes are listed under the class Sighting; ID, City, Country, Duration, Dateposted, Datetime and Comments. They are all of type strings, except for ID which is an integer.

4.0 Frontend

4.1 Header - Index



Figure 8- menu bar for our website

For the header, we tried to make a responsive and rather simple one. We took inspiration from [w3schools](#) with some alterations to the code and the design aspect in terms of paddings, font sizes and more. Instead of a picture logo as seen in the prototype (figure 4), we ended up just using text as a logo.

4.2 UFO observations

Ufo observation page is where you see the reported articles and where the create, update, read, share buttons are found. We took inspiration from [Bootstrap](#) to make this page.

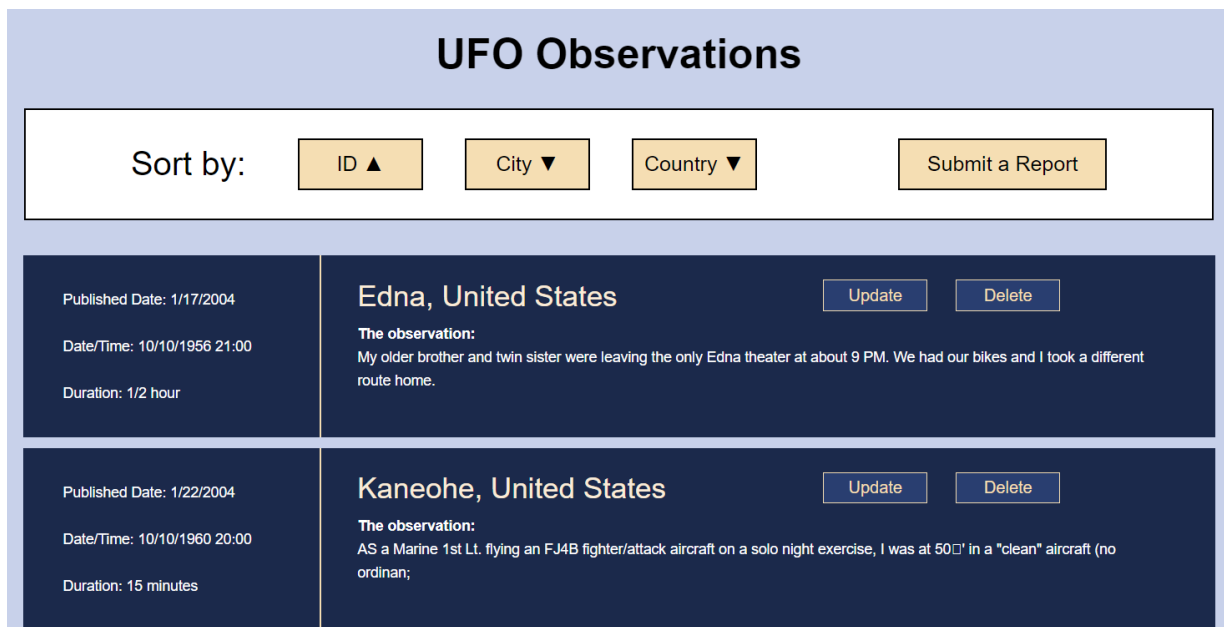


Figure 9 - Submit a report

4.3 Footer

The footer is also created with inspiration from [Bootstrap](#). As we see in the picture below it has 3 sections where the first one talks about the UFO observations website, the middle one contact information and at the last we have the submit a report section. To make these 3 sections we used bootstrap row and column.



Figure 10 - Footer

4.4 About us explanation

Our About us page consists of 4 sections. These are header, events/history, explore page and our team. We took inspiration from [this video](#) to code the header, events/history and explore section. For our team section we took inspiration from [this video](#). The header section has a background image and two html headings. We chose a picture that focuses on UFOs spaceship. We have used comments to make the code easier to understand.

4.4.1 Header - About us

```
<!--Headlines?-->
<header>
  <div class="top-header">
    <div class="header-content">
      <h2>Explore About Us</h2>
      <div class="line"></div>
      <h1>A Wonderful UFO World</h1>
    </div>
  </div>
</header>
```

Figure 11 - Code of header

To code the header, we have header tags and div tags and html headings. We can use our class name to style this part.

4.4.2 Events/ history

```
<!--Events/History-->
<section class="events">
  <h1>History</h1>
  <div class="line"></div>
  <div class="title">
    <div class="row">
      <div class="col">
        
        <h4>Apollo</h4>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad mini
      </div>
      <div class="col">
        
        <h4>UFO Sightings</h4>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad mini
      </div>
    </div>
  </div>
</section>
```

Figure 12 - Code of events

For the events/history section we wanted to give the user a brief description of UFO's history. To code this section, we have used the section tags, div tags, html headings and img tag. Furthermore we have used rows and col attributes. To style this section we can simply use the class names.

4.4.3 Explore page

For the explore page we chose to add a background image and added text which describe a fictive description of the idea behind our website.

```
<!--Explore-->
<section class="explore">
  <div class="explore-content">
    <h1>EXPLORE THE UFO PAGE</h1>
    <div class="line"></div>
    <p>
      The UFO website has been updated with the posting of 657 new sighting reports,
      We are a UFO detector, which shows "evidence" in the form of posts of UFOs through images/videos and descriptive texts. What happened and what was seen etc.
    </p>
  </div>
</section>
```

Figure 13 - Code of explore

4.4.4 Our team

For our team section, we have chosen to add fictive team members with animated pictures.

```
<!--Team-->
<div class="overskrift-team">
  <h1>Our team</h1>
</div>
<section class="ourTeam">
  <div class="teams">
    
    <div class="name">Diana Price</div>
    <div class="desig">Developer</div>
    <div class="about"> Hei alle sammen. How are you?</div>
  </div>

  <div class="teams">
    
    <div class="name">Alexandra Carrillo</div>
    <div class="desig">Designer</div>
    <div class="about"> Hei alle sammen. How are you?</div>
  </div>
</section>
```

Figure 14 - Code of team

To code this section. We coded section tag, div tags and img tag. For this section we chose to make fictive team members with animated pictures. For this particular section For this section we used the same code format for the rest of the fictive characters. We just needed to change the images and texts according to each fictive character.

4.4.5 Responsiveness

To make the About us page responsive, we added this section of code:

```
/*---*/
@media only screen and (max-width: 850px) {
  /*Events*/
  .row {
    flex-direction: column;
  }

  .row .col {
    margin: 20px auto;
  }

  .col img {
    max-width: 90%;
  }

  .explore-content {
    width: 100%;
  }
}
```

Figure 15 - Code of responsive

4.5 Color combinations

We wanted a dark blue color throughout the site. Using white in the background could be a bit too sharp of a contrast, so we ended up using a light color that is a bit darker. The color code for the dark blue is #141f38, and the color code for the lighter color used is #c8d1ea. The text color varies between white, black and a yellowish color called “wheat”.



Figure 16 - our color choice, for our background #c8d1ea



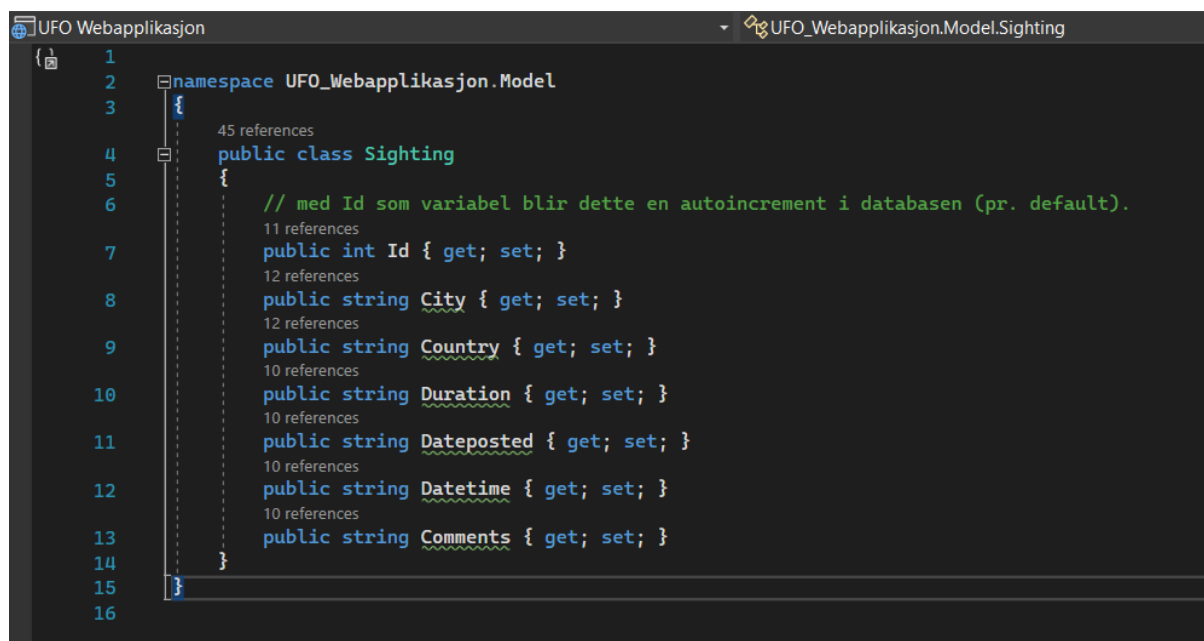
Figure 17 - our color choice, theme color, #141f38

5.0 Backend

This CRUD-application is developed to enable users to view or create new observations, update or delete observations. A dataset has been used as placeholders which contains already registered observations data. We have used the videos and other learning material from the course ITPE3200 WebApplication itself and both modified and further developed the code to fit our project. We have used the materials provided in the first three modules/lectures on Canvas.

5.1 Model

In order to create a CRUD web application, we would first need a class named “Sighting” where we can define the attributes of the class. For this purpose, we chose to implement seven attributes which will later all belong to a single table named “Sightings” in the database that will also be created. The “Sightings” table will be defined later in a file called ‘SightingContext.cs’.



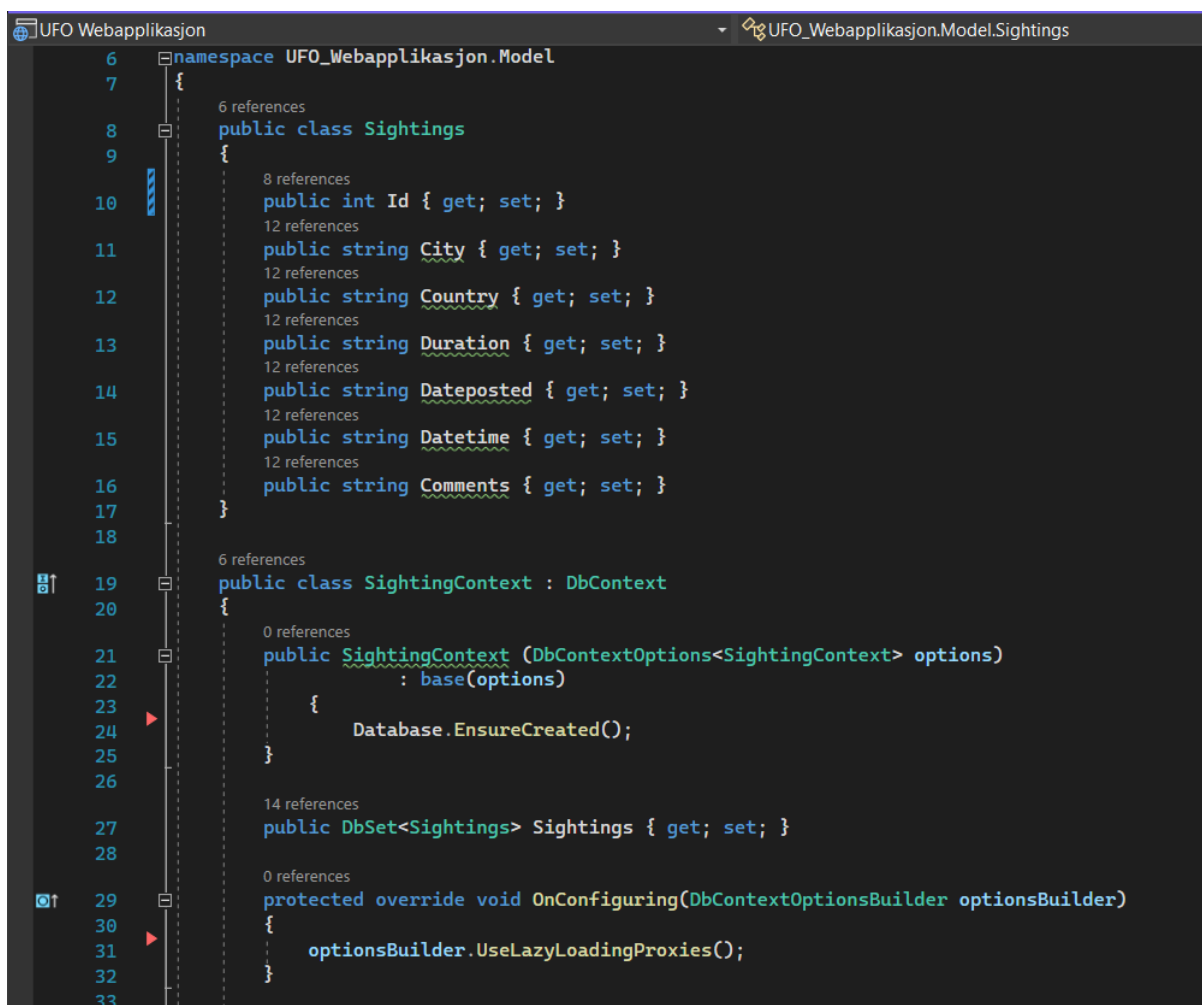
```
1 namespace UFO_Webapplikasjon.Model
2 {
3     45 references
4     public class Sighting
5     {
6         // med Id som variabel blir dette en autoincrement i databasen (pr. default).
7         11 references
8         public int Id { get; set; }
9         12 references
10        public string City { get; set; }
11        12 references
12        public string Country { get; set; }
13        10 references
14        public string Duration { get; set; }
15        10 references
16        public string Dateposted { get; set; }
17        10 references
18        public string Datetime { get; set; }
19        10 references
20        public string Comments { get; set; }
21    }
22 }
```

Figure 18 - Sighting.cs (Model)

The attributes that belong to the class ‘Sightings’ are respectively: Country, City, Duration, Date-posted, Datetime, Comment and an ID for the auto-incremented primary key assigned in the database that will be created for this project. This class will act as the blueprint for the communication between the client and server applications.

5.2 Database

We also defined attributes in the file “SightingContext.cs”, but the purpose of these attributes were set into the table named ‘Sightings’ which would act as the blueprint for how the database would be created and how it would store the data it receives from the web application. Additionally, a service had to be added to the ‘Startup.cs’ file and a package named ‘Microsoft.EntityFrameworkCore’ had to be installed as well for the ‘SightingContext’ file to function as desired.



```
6 namespace UFO_Webapplikasjon.Model
7 {
8     6 references
9     public class Sightings
10    {
11        8 references
12        public int Id { get; set; }
13        12 references
14        public string City { get; set; }
15        12 references
16        public string Country { get; set; }
17        12 references
18        public string Duration { get; set; }
19        12 references
20        public string Dataposted { get; set; }
21        12 references
22        public string Datetime { get; set; }
23        12 references
24        public string Comments { get; set; }
25    }
26
27    6 references
28    public class SightingContext : DbContext
29    {
30        0 references
31        public SightingContext(DbContextOptions<SightingContext> options)
32            : base(options)
33        {
34            Database.EnsureCreated();
35        }
36
37        14 references
38        public DbSet<Sightings> Sightings { get; set; }
39
40        0 references
41        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
42        {
43            optionsBuilder.UseLazyLoadingProxies();
44        }
45    }
46 }
```

Figure 19 - SightingContext.cs

```

1 using UFO.Webapplikasjon.DAL;
2 using UFO.Webapplikasjon.Model;
3 using Microsoft.EntityFrameworkCore;
4
5 namespace UFO.Webapplikasjon
6 {
7     1 reference
8     public class Startup
9     {
10
11         0 references
12         public void ConfigureServices(IServiceCollection services)
13         {
14             services.AddControllers();
15             services.AddDbContext<SightingContext>(options =>
16                 options.UseSqlite("Data Source=Sighting.db"));
17         }
18     }
19 }

```

Figure 20 - Startup.cs

The data stored in our database is from a dataset we got from [Kaggle](https://www.kaggle.com/datasets) that had similar attributes to ours. The dataset was a csv-file which we adjusted to the our model in Python with the Pandas library. We later imported it manually by using the SQLite DB Browser. This dataset is only meant to be used as placeholders for this project instead of manually making the data on our own. There was a bug with the importation where most of the comments would end with 3 semicolons (;;;), which wasn't in the dataset. We were also testing out the data table beforehand with the "DBInit.cs"-file which initiated and created custom data for our database.

```

0 references
public static class DBInit
{
    0 references
    public static void Initialize(IApplicationBuilder app)
    {
        using (var serviceScope = app.ApplicationServices.CreateScope())
        {
            var context = serviceScope.ServiceProvider.GetService<SightingContext>();

            // må slette og opprette databasen hver gang når den skal initieres (seed'es)
            context.Database.EnsureDeleted();
            context.Database.EnsureCreated();

            var kunde1 = new Sightings { City = "Oslo", Country = "Norway", Duration = "2 hours", Dateposted = "03/12/2022", Datetime = "02/12/2022", Comments = "Just testing this application" };
            var kunde2 = new Sightings { City = "Kobenhagen", Country = "Denmark", Duration = "3 min", Dateposted = "11/11/2011", Datetime = "14/02/2007", Comments = "Nothing really happened" };

            context.Sightings.Add(kunde1);
            context.Sightings.Add(kunde2);

            context.SaveChanges();
        }
    }
}

```

Figure 21 - DBInit.cs

5.3 Repository

Our repository class named "SightingRepository.cs" had methods which provided access to the data in the database. We had methods such as Create, Read, ReadAll, Update and Delete were all implemented in order to enable the CRUD functionalities.

We have used asynchronous programming towards the database as this has become the standard in ASP.Net Core even if large amounts of traffic are not expected, which is the case with our project. The learning material in the module "Kunde CRUD - asynkron kode mot DB" is used. We modified our code in order to handle asynchronous method calls to the database, for example by modifying the line of code "public bool Lagre(Sighting innSighting)" to "public async Task<bool> Lagre(Sighting innSighting)" etc.

A number of methods were implemented to support the CRUD-functionalities. These methods are Create, ReadAll, Update, Read and Delete. Almost all of the implemented methods are built on the concept of try & catch that can handle two or more different cases.

The method 'Create' takes in an object with attributes that corresponds to a new sighting that has been newly registered and adds it to the database. The method 'ReadAll' enables the user/admin(client) to display/view all registered observations in the database, this method will return a list of observations(objects). The method 'Delete' takes in an 'ID' based on which observation that is meant to be deleted and removes it from the database. The method 'Read' takes in an 'ID' based on which observation that the user would want access/view/display. Additionally, the method 'Update' takes in an object of type Sighting (observations class) and updates an observation in the database.

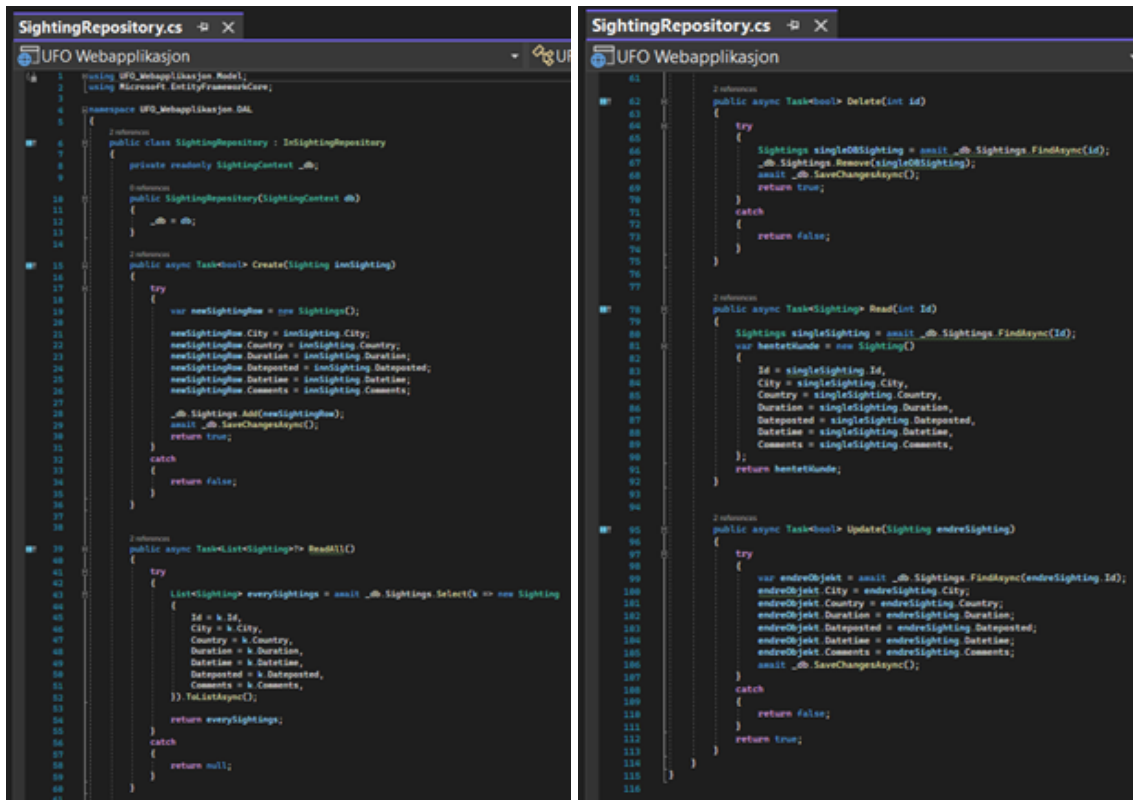


Figure 22 - Repository.cs

Custom methods based on some of the CRUD methods were also made to add another functionality to the website like sorting. We were able to sort the lists that were fetched with the method “ReadAll” by using OrderBy() and OrderByDescending(). These functions used the attributes as a parameter to sort the lists alphabetically in an ascending or descending order. That way we would be able to sort the lists through ids, countries, cities. We also tried to sort the lists by date and time, but we somehow couldn’t implement it because the functions couldn’t sort the date format (mm/dd/yyyy) in the correct way. There was also a bug with this sorting method because the function sorts all the words starting with the capital letters before adding in sorted lowercase words in a new line.

```

2 references
public async Task<List<Sighting>?> ReadCountryAsc()
{
    try
    {
        List<Sighting> everySightings = await _db.Sightings.Select(k => new Sighting
        {
            Id = k.Id,
            City = k.City,
            Country = k.Country,
            Duration = k.Duration,
            Datetime = k.Datetime,
            Dateposted = k.Dateposted,
            Comments = k.Comments,
        }).OrderBy(x => x.Country).ToListAsync();

        return everySightings;
    }
    catch
    {
        return null;
    }
}

```

Figure 23 - OrderBy()

```

2 references
public async Task<List<Sighting>?> ReadCountryDesc()
{
    try
    {
        List<Sighting> everySightings = await _db.Sightings.Select(k => new Sighting
        {
            Id = k.Id,
            City = k.City,
            Country = k.Country,
            Duration = k.Duration,
            Datetime = k.Datetime,
            Dateposted = k.Dateposted,
            Comments = k.Comments,
        }).OrderByDescending(x => x.Country).ToListAsync();

        return everySightings;
    }
    catch
    {
        return null;
    }
}

```

Figure 24 - OrderByDescending()

```

2 references
public async Task<List<Sighting>??> ReadIdDesc()...
2 references
public async Task<List<Sighting>??> ReadCountryAsc()...
2 references
public async Task<List<Sighting>??> ReadCountryDesc()...
2 references
public async Task<List<Sighting>??> ReadCityAsc()...
2 references
public async Task<List<Sighting>??> ReadCityDesc()...

```

Figure 25 - Sorting List Functions

We also made a method “ReadLatest()” which got the last record in the database. We started off by using OrderByDescending() to reverse the list. Then we used the function FirstOrDefault() which got us the first record of the list.

```

2 references
public async Task<Sighting> ReadLatest()
{
    try
    {
        List<Sighting> everySightings = await _db.Sightings.Select(k => new Sighting
        {
            Id = k.Id,
            City = k.City,
            Country = k.Country,
            Duration = k.Duration,
            Datetime = k.Datetime,
            Dateposted = k.Dateposted,
            Comments = k.Comments,
        }).OrderByDescending(x => x.Id).ToListAsync();

        var latestRecord = everySightings.FirstOrDefault();

        return latestRecord;
    }
    catch
    {
        return null;
    }
}

```

Figure 26 - FirstOrDefault()

Afterwards, we created an interface named ‘InSightingRepository.cs’ which would implement all methods in the ‘SightingRepository.cs’ file and act as a connection to the Controller class.

```

using UFO_Webapplikasjon.Model;

namespace UFO_Webapplikasjon.DAL
{
    4 references
    public interface InSightingRepository
    {
        2 references
        Task<bool> Create(Sighting innSighting);
        2 references
        Task<List<Sighting>> ReadAll();
        2 references
        Task<List<Sighting>> ReadIdDesc();
        2 references
        Task<List<Sighting>> ReadCountryAsc();
        2 references
        Task<List<Sighting>> ReadCountryDesc();
        2 references
        Task<List<Sighting>> ReadCityAsc();
        2 references
        Task<List<Sighting>> ReadCityDesc();
        2 references
        Task<Sighting> ReadLatest();
        2 references
        Task<bool> Delete(int id);
        2 references
        Task<Sighting> ReadOne(int id);
        2 references
        Task<bool> Update(Sighting updateSighting);
    }
}

```

Figure 27 - InSightingRepository.cs

5.4 Controller

Then, a service was added to the 'Startup.cs' file in order to enable the implementations already made. This service will register the dependency injection for the interface in the Controller class.

```

services.AddScoped<InSightingRepository, SightingRepository>();

```

Figure 28 - Added service in Startup.cs


```

namespace UFO_Webapplikasjon.Controllers
{
    [Route("[controller]/[action]")]
    1 reference
    public class SightingController : ControllerBase
    {
        private readonly InSightingRepository _db;

        0 references
        public SightingController(InSightingRepository db)
        {
            _db = db;
        }

        0 references
        public async Task<bool> Create(Sighting innSighting)
        {
            return await _db.Create(innSighting);
        }

        0 references
        public async Task<List<Sighting>> ReadAll()
        {
            return await _db.ReadAll();
        }

        0 references
        public async Task<List<Sighting>> ReadIdDesc()...
        0 references
        public async Task<List<Sighting>> ReadCountryAsc()...
        0 references
        public async Task<List<Sighting>> ReadCountryDesc()...
        0 references
        public async Task<List<Sighting>> ReadCityAsc()...
        0 references
        public async Task<List<Sighting>> ReadCityDesc()...

        0 references
        public async Task<bool> Delete(int id)
        {
            return await _db.Delete(id);
        }
    }
}

```

Figure 29 - Controller.cs (Part 1)

```

0 references
public async Task<Sighting> ReadLatest()
{
    return await _db.ReadLatest();
}

0 references
public async Task<Sighting> ReadOne(int id)
{
    return await _db.ReadOne(id);
}

0 references
public async Task<bool> Update(Sighting updateSighting)
{
    return await _db.Update(updateSighting);
}
}

```

Figure 30 - Controller.cs (Part 2)

5.5 View

It would now be possible for us to access the database from the website with the Controller class in place. The website url like for example “sighting/create” could be used to send our data to the functions in the repository through our controller.

```
const url = "Sighting/create";
$.post(url, report, function (OK) {
  if (OK) {
    window.location.href = 'index.html';
  }
  else {
    $("#feil").html("Feil i db - prøv igjen senere");
  }
});
```

Figure 31 - Controller and Website Communication

5.5.1 Create sighting

The page will allow the user to create a post and add in their sighting experience with Create(). We took into consideration that the duration could have different time units so it was implemented as a select box which later on combined the string with the number from the input box. The published date for the post would be created by getting the date from built-in javascript function "Date()" while also formatting the date to mm/dd/yyyy. The date from input will then be sent to the database with Create().

Figure 32 - Form to register an observation

```
// Henter nåværende dato for når lagringsknappen trykkes
let date = new Date();

// Formaterer datoen til mm/dd/yyyy (https://stackoverflow.com/questions/11591854/format-date-to-mm-dd-yyyy-in-javascript)
let formatDate = ((date.getMonth() > 8) ? (date.getMonth() + 1) : ('0' + (date.getMonth() + 1))) + '/' + ((date.getDate() > 9) ? date.getDate() : ('0' + date.getDate())) + '/' + date.getFullYear();
```

Figure 33 - Getting creation date and formatting to mm/dd/yyyy

```
// Henter og kombinerer Streng og Select-verdi
let durationNumber = $("#durationNumber").val();
let durationUnit = $("#durationUnit").val();
let duration = durationNumber + durationUnit;
```

Figure 34 - Combining duration number and time unit

```
const report = {
  city: $("#city").val(),
  country : $("#country").val(),
  duration : duration,
  dateposted : formatDate,
  datetime : $("#datetime").val(),
  comments : $("#comments").val(),
}
const url = "Sighting/create";
$.post(url, report, function (OK) {
  if (OK) {
    window.location.href = 'index.html';
  }
  else {
    $("#feil").html("Feil i db - prøv igjen senere");
  }
});
};
```

Figure 35 - createSighting

5.5.2 Update sighting

The page will show all the data based on the ID in the case of updating a specific sighting. The data was fetched from `ReadOne()` and the changes then were sent through `Update()`. The input box of “Date posted” is disabled for any kind of changes because we wouldn’t want to edit when the post was created.

Update an observation

City

Country

Duration

Date posted

Date time

Comments

My older brother and twin sister were leaving the only Edna theater at about 9 PM. We had our bikes and I took a different route home.

Figure 36 - Form to update an observation

```

1  $(function () {
2      // hent kunden med kunde-id fra url og vis denne i skjemaet
3      const id = window.location.search.substring(1);
4
5      const url = "Sighting/readOne?" + id;
6      $.get(url, function (report) {
7          $("#id").val(report.id); // må ha med id inn skjemaet, hidden i html
8          $("#city").val(report.city);
9          $("#country").val(report.country);
10         $("#duration").val(report.duration);
11         $("#dateposted").val(report.dateposted);
12         $("#datetime").val(report.datetime);
13         $("#comments").val(report.comments);
14     });
15 }
16
17
18 function updateSighting() {
19     const report = {
20         id: $("#id").val(), // må ha med denne som ikke har blitt endret for å vite hvilken kunde som skal endres
21         city: $("#city").val(),
22         country: $("#country").val(),
23         duration: $("#duration").val(),
24         dateposted: $("#dateposted").val(), // skal det være mulig å endre på utgivelsesdatoen (admin)
25         datetime: $("#datetime").val(),
26         comments: $("#comments").val(),
27     };
28     $.post("Sighting/update", report, function (OK) {
29         if (OK) {
30             window.location.href = './article.html';
31         }
32         else {
33             $("#feil").html("Feil i db - prøv igjen senere");
34         }
35     });
36 }

```

Figure 37 - Update.js

5.5.3 Delete sighting

The function “deleteSighting()” takes in an “Id” for the object that is meant to be deleted. The function will fetch the “Id” from the application and remove the sighting from the database through Delete(). It will also display message if the deletion failed.

```
function deleteSighting(id) {  
  const url = "Sighting/delete?id="+id;  
  $.get(url, function (OK) {  
    if (OK) {  
      window.location.href = 'article.html';  
    }  
    else {  
      $("#feil").html("Feil i db - prøv igjen senere");  
    }  
  });  
}
```

Figure 38 - deleteSighting, JQuery

5.5.4 Feed

The feed gets all the different records in the database with the method ReadAll() and displays them as a post. The posts allows the user to view, update or delete and we also added in a way to sort the table in an alphabetical order which allowed the user to choose what order they would prefer. The “Submit a Report”-button is only a shortcut if anyone wants to create a post.

UFO Observations	
Sort by: ID ▲ City ▼ Country ▼ Submit a Report	
Published Date: 1/17/2004 Date/Time: 10/10/1956 21:00 Duration: 1/2 hour	Edna, United States Update Delete The observation: My older brother and twin sister were leaving the only Edna theater at about 9 PM. We had our bikes and I took a different route home.
Published Date: 1/22/2004 Date/Time: 10/10/1960 20:00 Duration: 15 minutes	Kaneohe, United States Update Delete The observation: AS a Marine 1st Lt. flying an FJ4B fighter/attack aircraft on a solo night exercise, I was at 500' in a "clean" aircraft (no ordinar;

Figure 39 - Feed

The list from ReadAll() in the Repository was fetched through the function readAllSightings() set to display right away when the user visits the website. The function formaterSightings() then loops through all the records in the list and displays the content in the form of a post. The sort function does the same as ReadAll() as well as changing the buttons to show that the user can reverse order again. The sort functions will only change the order based on a single condition. The user will not be able to sort both city and country at the same time.

```
1  $(function(){
2      readAllSightings();
3  });
4
5  function readAllSightings() {
6      $.get("sighting/readAll", function (reports) {
7          formaterSightings(reports);
8      });
9      $("#idAsc").hide();
10     $("#idDesc").show();
11 }
12
13 // Akkurat lik readAllSightings men med litt andre justeringer
14 function sortByIdDesc()...
21 function sortByCountryASC()...
28 function sortByCountryDESC()...
35 function sortByCityASC()...
42 function sortByCityDESC()...
49
```

Figure 40 - readAll() and sort functions in JQuery

```
function formaterSightings(reports) {
  let ut = "<div class='container'>";

  for (let report of reports) {
    ut += "<div class='row observation'>" +
      "<div class='col-3 dateF visible'>" +
        "<div>Published Date: " + report.dateposted + "</div>" +
        "<div>Date/Time: " + report.datetime + "</div>" +
        "<div>Duration: " + report.duration + "</div>" +
      "</div>" +

      "<div class='col-3 dateF notVisible'>" +
      "<div class='row'>" +
      "<div class='col-7 dategroup'>" +
        "<div>Published Date: " + report.dateposted + "</div>" +
        "<div>Date/Time: " + report.datetime + "</div>" +
        "<div>Duration: " + report.duration + "</div>" +
      "</div><div class='col-5 btnGroup'>" +
        "<a class='btnCrud' id='update1' href='update.html?id=" + report.id + "'>Update</a>" +
        "<div></div>" +
        "<button class='btnCrud' onclick='deleteSighting(" + report.id + ")'>Delete</button>" +
      "</div></div></div>" +

      "<div class='col-9 commentF'><div class='row'>" +
      "<div class='col-7 titleP'><h3>" +
        "<span class='fl'>" + report.city + "</span>, <span class='fl'>" + report.country + "</span>" +
      "</h3></div>" +
      "<div class='col-5 visible'><div class='row'>" +
        "<a class='col-5 btnCrud' id='update1' href='update.html?id=" + report.id + "'>Update</a>" +
        "<div class='col-1'></div>" +
        "<button class='col-5 btnCrud' onclick='deleteSighting(" + report.id + ")'>Delete</button>" +
      "</div></div></div>" +
      "<div class='textF'><b>The observation:</b><br />" + report.comments + "</div>" +
    "</div></div>";
  }
  ut += "</div>";
  $("#reports").html(ut);
}
```

Figure 41 - `formaterSightings()`, JQuery

5.5.5 Extra functionality on the homepage

We added a way to see a total amount of records in the database and a way to see the newest record made in the database. The last record in the database would hypothetically have the newest date if it were to be created from the methods we made, but it doesn't really sort based on the date it was made.

Articles

839

LATEST OBSERVATION

Published Date: 10/24/2022

Date/Time: 10/10/1994 18:30

Duration: 3 seconds

Bergen, Norway

The observation:

Nothing really happened.

Update

Delete

See more

Figure 42 - Display of the total amount of articles and the latest sighting.

We had set the functions set on startup to display it right away. The function `readLatestSightings()` receives only a single object from the Controller. The function `formaterSightings()` will later on take the object as a parameter to display the content

without a need for a for-loop because there's no list. As previously mentioned, the object is the first record of a list with a descending order based on the Id.

```
$(function(){
    totalSightings();
    readLatestReport();
});

function totalSightings() {
    $.get("sighting/readAll", function (reports) {
        countTotal(reports)
    });
}

function readLatestReport() {
    $.get("sighting/readLatest", function (reports) {
        formaterSightings(reports);
    });
}
```

Figure 43 - *totalSightings()* and *readLatestReport()*

The amount of articles was counted in the function *totalSightings()* by using a variable which incremented through a for-loop for every record in the list we got from the method *ReadAll()*. It was later printed out to the ids in the html file with JQuery.

```
function countTotal(reports) {
    // Telleren
    let count = 0;

    // Looper gjennom alle kolonnene i tabellen
    for (let report of reports) {
        count++;
    }

    // Summen printes ut
    let ut = count;
    $("#count").html(ut);
    $("#count2").html(ut);
}
```

Figure 44 - *countTotal()*

References

Learning materials for the course on Canvas:

First three modules(Entity Framework, Entity Framework 2 & DAL)

Bootstrap

<https://getbootstrap.com/docs/4.0/layout/grid/>

For the code:

Header:

https://www.w3schools.com/howto/howto_css_responsive_header.asp

Youtube tutorials:

https://www.youtube.com/watch?v=ZdJSHEczi_0&t=1894s

<https://www.youtube.com/watch?v=4q8AvrLrLAM>

Dataset

<https://www.kaggle.com/datasets/NUFORC/ufo-sightings>

OrderBy(), OrderByDescending() and FirstOrDefault()

<https://www.tutorialsteacher.com/linq/linq-sorting-operators-orderby-orderbydescending>

<https://stackoverflow.com/questions/3925258/c-sharp-list-orderby-descending>

<https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.firstordefault?view=net-7.0>

Date Format (mm/dd/yyyy)

<https://stackoverflow.com/questions/11591854/format-date-to-mm-dd-yyyy-in-javascript>

Image references:

Header

https://www.theparisreview.org/blog/wp-content/uploads/2019/08/adobestock_143826172.jpeg

background image for explore the page

<https://www.pixelstalk.net/wp-content/uploads/images1/Backgrounds-ufo-sky-desert.jpg>

<https://www.pixelstalk.net/wp-content/uploads/images1/Backgrounds-ufo-sky-desert-768x480.jpg>

About us, our team section

Person 1:

link adresse:

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwallpapercave.com%2Fw%2Fwp1395321&psig=AOvVaw2KQ-Aul6VWkEM2wqBIR9TT&ust=1666996700413000&source=images&cd=vfe&ved=0CA0QjRxqFwoTCKDr9eS8gfsCFQAAAAAdAAAAABAD>

bilde adresse:

<https://wallpapercave.com/uwp/uwp1395321.jpeg>

Person 2:

link adresse:

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.pinterest.com%2Fpin%2F778841329295264548%2F&psig=AOvVaw2KQ-Aul6VWkEM2wqBIR9TT&ust=1666996700413000&source=images&cd=vfe&ved=0CA0QjRxqFwoTCKDr9eS8gfsCFQAAAAAdAAAAABAJ>

bilde adresse:

<https://i.pinimg.com/originals/a8/de/fc/a8defca1a0ea8ecb46644595c8c1d6c8.jpg>

Person 3:

link adresse:

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwallpapercave.com%2Fw%2Fwp2078068&psig=AOvVaw2KQ-Aul6VWkEM2wqBIR9TT&ust=1666996700413000&source=images&cd=vfe&ved=0CA0QjRxqFwoTCKDr9eS8gfsCFQAAAAAdAAAAABAP>

bilde adresse:

<https://wallpapercave.com/uwp/uwp2078068.jpeg>

Person 4:

link adresse:

<https://no.pinterest.com/pin/816840451147707810/>

bilde adresse:

<https://i.pinimg.com/originals/1c/85/a1/1c85a1793a9a245c2bdca5d6ca05f206.jpg>

Person 5:

link adresse:

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.pinterest.com%2Fpin%2F558939003754013994%2F&psig=AOvVaw2CatYHH-qIWQZiMIk7aBhG&ust=16669960>

[17974000&source=images&cd=vfe&ved=0CA0QjRxqFwoTCICxt9i6gfsCFQAAAAAdAA
AAABAD](https://www.google.com/search?q=17974000&source=images&cd=vfe&ved=0CA0QjRxqFwoTCICxt9i6gfsCFQAAAAAdAA
AAABAD)

Bildeadresser:

<https://i.pinimg.com/564x/27/cf/1a/27cf1a6a64a3dd823e88e993da4692b8.jpg>

Front page image:

[https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.livescience.com%2Fearth-directions-for-aliens&psig=AOvVaw1qtVvICMQ-MmgoMvGHRTBP&ust=1664885062441000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCIDMya6CxPoCFQAAAAAdAAA
AABAc](https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.livescience.com%2Fearth-directions-for-aliens&psig=AOvVaw1qtVvICMQ-MmgoMvGHRTBP&ust=1664885062441000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCIDMya6CxPoCFQAAAAAdAAA
AABAc)

UFO history image 1 and text:

https://no.wikipedia.org/wiki/Uidentifisert_flygende_objekt

UFO history image 2 and text:

<https://www.smithsonianmag.com/history/how-ufo-reports-change-with-technology-times-180968011/>