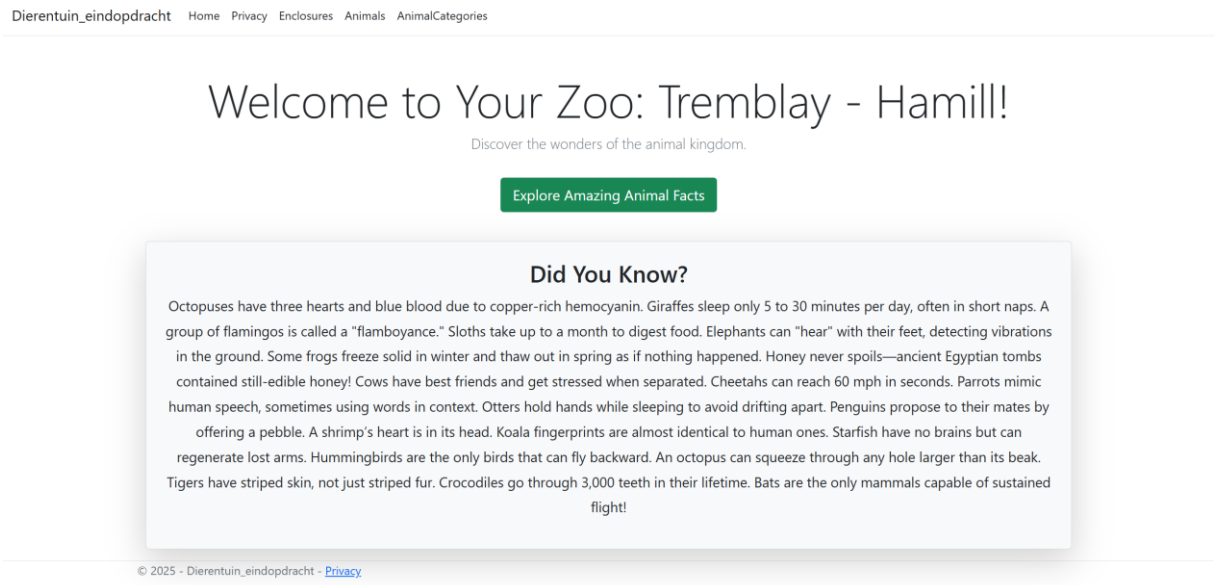
Dierentuin Documentatie



Contents

Inleiding

Om csharp af te ronden moesten wij in een tweetal een webapp creëren met ef core die een dierentuin onderhoudt. Aangezien dit framework nog al lastig kan zijn om te begrijpen is dit rapport gemaakt om begrip te creëren over hoe de applicatie werkt.

Belangrijk om te weten

In de meeste ef core apps gebruiken mensen de controllers om gelijk data uit de context te halen en queries te schrijven. Wij heb besloten om dit niet te doen aangezien het schijden van de functionaliteit (queries) en de data die teruggestuurd wordt te scheiden. Hiervoor is een service directory aangemaakt, deze klasses bevatten de context en queryen er data uit in de vorm van functies.

Ook zullen er bij sommige controller acties specifieke routes gedefiniëerd zijn. Dit komt omdat de code anders niet werkte omdat de pagina niet gevonden kon worden. Als er een route bij een actie staat betekent dit dat de code niet werkt zonder de route.

Api Documentatie

In de dierentuin wordt gebruik gemaakt van meerdere controller acties die technisch gezien als api bieden. Net zoals eerder verteld in belangrijk om te weten worden er in ons project functies uit de service classes gebruikt om data te queryen en daarna toegepast in controllers waarin er daadwerkelijk een request wordt gestuurd. Zo hebben we logica gescheden om alles overzichtelijk te houden. In dit stuk zal elke controller functie weergegeven worden, gezegd worden welke controllers deze bevatten en tot slot in het kort wat het doet. Om de code te zien staan er afbeeldingen weergegeven in de bijlage: Api Code.

1. Index - Enclosures, Categories, Animals: Deze actie retourneerd alles uit de database van een bepaalde object. Bij Animals staan er nog veel

2. Filters – Animals: Aangezien de animal tabel nog al groot is leek het ons een goed idee om speciale filters toe te passen, als je op een table header klikt wordt er gefilterd doormiddel van orderby().

3. Create – Enclosures, Categories, Animals: Deze actie doet een post request aan de hand van een Dto. Door eerst de data in een dto te zetten kan de controller de ingevulde data bereiken. Deze controller bestaat uit twee acties omdat een de pagina weergeeft en de ander het creëren uitvoert.

4. Edit – Enclosures, Categories, Animals: Deze actie doet ook een post request aan de hand van een dto maar bevat ook de optie om dieren toe te voegen bij enclosures en categories. Doormiddel van checboxes worden de dieren naar een lijst in de edit functie gestuurd en als het ID aangevinkt is toegevoegd.

5. Delete – Enclosures, Categories, Animals: Deze actie verwijdert een object uit de database aan de hand van een het Id. Dit Id wordt gepakt doormiddel van een knop.

6. Sunset en Sunrise – Enclosures, Animals: Deze actie geeft op basis van een dier zijn Arise en BedTime de dieren weer die op basis van de tijd in het echt aan het slapen zijn .

7. FeedingTime – Enclosures, Animals : Deze actie geeft weer wat een bepaald dier eet of wat de dieren binnen een verblijf eten.

8. AutoAssign – Animals: De autoassign actie zorgt ervoor dat dieren in een verblijf worden gestopt waarin de size van een enclosure groter is dan de spaceRequirement van een dier. Dit is onze manier van het toevoegen van CheckConstraints. Ook worden er nieuwe verblijven aangemaakt als het aantal verblijven in de database 0 is.

9. RemoveEnclosure – Enclosures: Deze actie zorgt ervoor dat alle verblijven en indelingen verwijderd worden. De Enclosures moeten allemaal verwijderd kunnen worden om indelingen weg te halen, deze functie valt goed samen als je dieren wilt autoassignen.

10. Get – Enclosures, Categories, Animals: Deze laat alle objecten van een entiteit zien aan de hand van een nieuwe endpoint, deze functie returned daarbij ook een Ok in plaats van een view.

11. EnclosureAnimals en CategoryAnimals – Enclosures, Categories: Deze functie laat alle dieren van een verblijf of catgorie zien. Door deze actie uit te voeren kan er gekeken worden via enclosures en categories welke dieren er in een verblijf zitten of welke dieren er tot een bepaald categorie horen.

Database

Animal

Attributen van de Animal-klasse

- **AnimalId** (int) – Unieke identificatie van het dier. (Primaire sleutel)
- **Name** (string) – Naam van het dier (verplicht).
- **Species** (string) – De soort waartoe het dier behoort.
- **Prey** (string?) – Optioneel: Het prooidier van het dier, indien van toepassing.
- **SpaceRequirement** (double) – De benodigde ruimte in de dierentuin voor dit dier.
- **FeedingTime** (string) – Tijdstip waarop het dier gevoed wordt.
- **Arise** (TimeSpan) – Tijdstip waarop het dier actief wordt.
- **BedTime** (TimeSpan) – Tijdstip waarop het dier gaat slapen.

Enums (categorieën waarin het dier valt):

- **Size** (ZooEnums.Size) – Grootte van het dier.
- **DietaryClass** (ZooEnums.DietaryClass) – Voedingsklasse (bijv. herbivoor, carnivoor).
- **ActivityPattern** (ZooEnums.ActivityPattern) – Activiteitspatroon (bijv. dagactief, nachtdier).
- **SecurityRequirement** (ZooEnums.SecurityLevel) – Beveiligingsniveau dat nodig is voor dit dier.

Relaties (Navigation Properties)

- **EnclosureId** (int?) – Buitenlandse sleutel (FK) naar Enclosure.
- **AnimalCategoryId** (int?) – Buitenlandse sleutel (FK) naar AnimalCategory.
- **ZooId** (int) – Buitenlandse sleutel (FK) naar Zoo.
- **Enclosure** (Enclosure?) – Navigatie-eigenschap naar de **verblijfsruimte** van het dier.
- **Category** (AnimalCategory?) – Navigatie-eigenschap naar de **categorie** van het dier.
- **Zoo** (Zoo) – Navigatie-eigenschap naar de **dierentuin** waar het dier zich bevindt.

Relatieschema

- Een Animal behoort tot **één Zoo** (*1-Zoo, Veel-Animals*).
- Een Animal kan tot **één Enclosure** behoren (*1-Enclosure, Veel-Animals*).
- Een Animal kan tot **één AnimalCategory** behoren (*1-Category, Veel-Animals*).

AnimalCategory

Attributen van de AnimalCategory-klasse

- **AnimalCategoryId** (int) – Unieke identificatie van de diercategorie (Primaire sleutel).
- **Name** (string) – Naam van de diercategorie.

Relaties (Navigation Properties)

- **Animals** (ICollection<Animal>?) – Een collectie van **dieren** die tot deze categorie behoren.

Relatieschema

- Een AnimalCategory kan **meerdere Animals** bevatten (*1-Category, Veel-Animals*).
- Een Animal behoort tot **één AnimalCategory** (*1-Category, Veel-Animals*).

Enclosure

Attributen van de Enclosure-klasse

- **EnclosureId** (int) – Unieke identificatie van het verblijf (Primaire sleutel).
- **Name** (string) – Naam van het verblijf.
- **Size** (double) – Grootte van de dieren die erin mogen(**verplicht veld**).

Enums

- **Climate** – Het klimaat van het verblijf, gedefinieerd in ZooEnums.Climate.
- **Habitat** – Het type habitat van het verblijf, gedefinieerd in ZooEnums.HabitatType.
- **SecurityLevel** – Het beveiligingsniveau van het verblijf, gedefinieerd in ZooEnums.SecurityLevel.

Relaties (Navigation Properties)

- **Animals** (ICollection<Animal>?) – Een collectie van **dieren** die in dit verblijf leven.
- **ZooId** (int) – De unieke identificatie van de **zoo** waartoe dit verblijf behoort (**foreign key**).
- **Zoo** (Zoo) – De dierentuin waartoe dit verblijf behoort.

Relatieschema

- Een Enclosure kan **meerdere Animals** bevatten (*1-Enclosure, Veel-Animals*).
- Een Enclosure behoort tot **één Zoo** (*1-Zoo, Veel-Enclosures*).

Zoo

Attributen van de Zoo-klasse

- **ZooId** (int) – Unieke identificatie van de dierentuin (Primaire sleutel).
- **Name** (string) – Naam van de dierentuin.

Relaties (Navigation Properties)

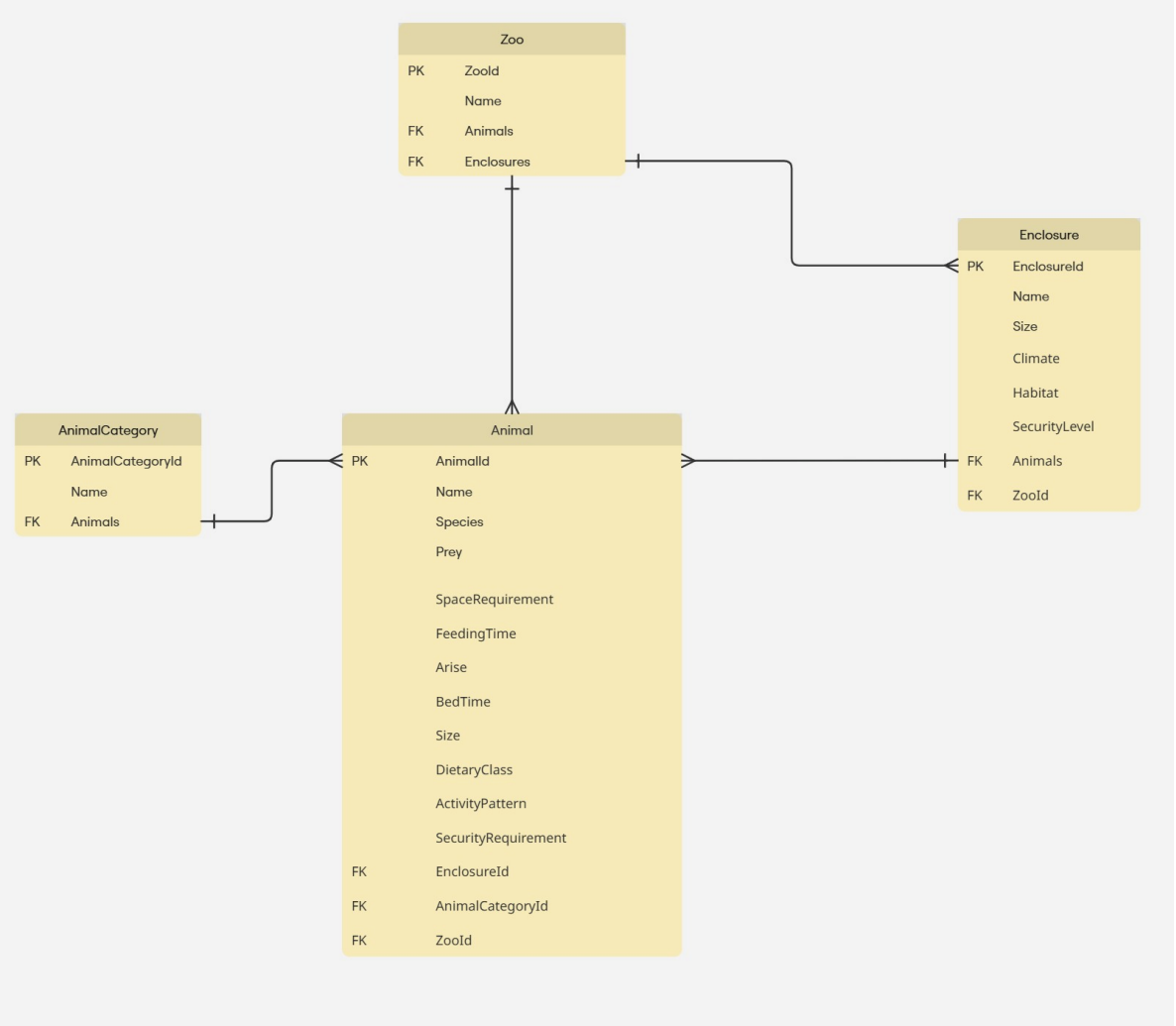
- **Animals** (ICollection<Animal>) – Een collectie van **dieren** die in de dierentuin leven.
- **Enclosures** (ICollection<Enclosure>) – Een collectie van **verblijven** in de dierentuin.

Relatieschema

- Een Zoo kan **meerdere Animals** hebben (*1-Zoo, Veel-Animals*).
- Een Zoo kan **meerdere Enclosures** hebben (*1-Zoo, Veel-Enclosures*).

Erd

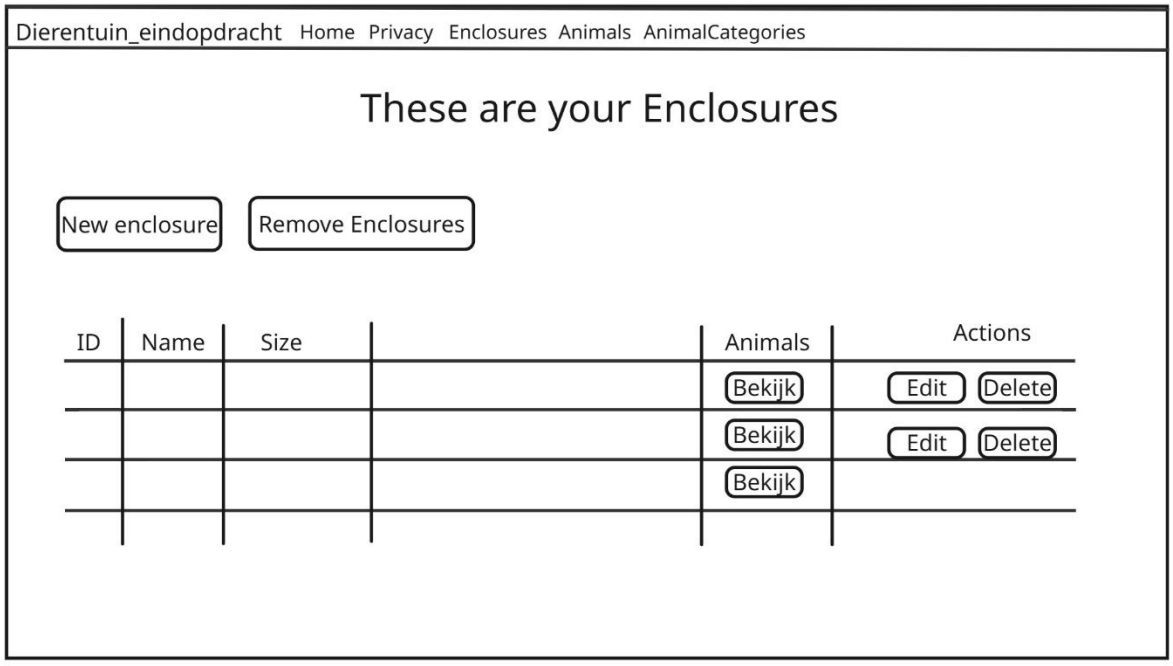
Om de database in het kort vast te leggen hebben we een erd gecreëerd die in het kort de database schetst. Hierin wordt weergegeven hoe de entiteiten binnen de app aan elkaar relateren en welke attributen ze bevatten.



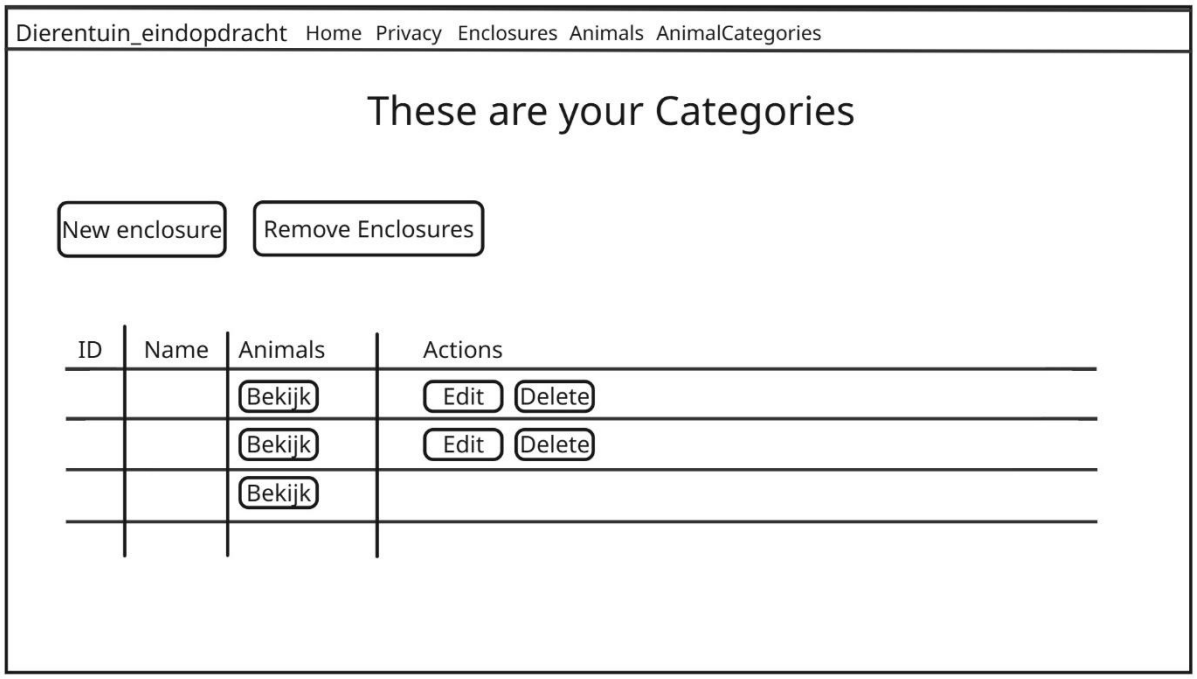
WireFrames

Om in te schatten hoe onze web app er ongeveer uit zou zien was het nodig om wireframes te creëren. Om de applicatie een professionele en niet “extra” uiterlijk te geven hebben we besloten om ons grotendeels aan de basis layout van een ef core website te houden.

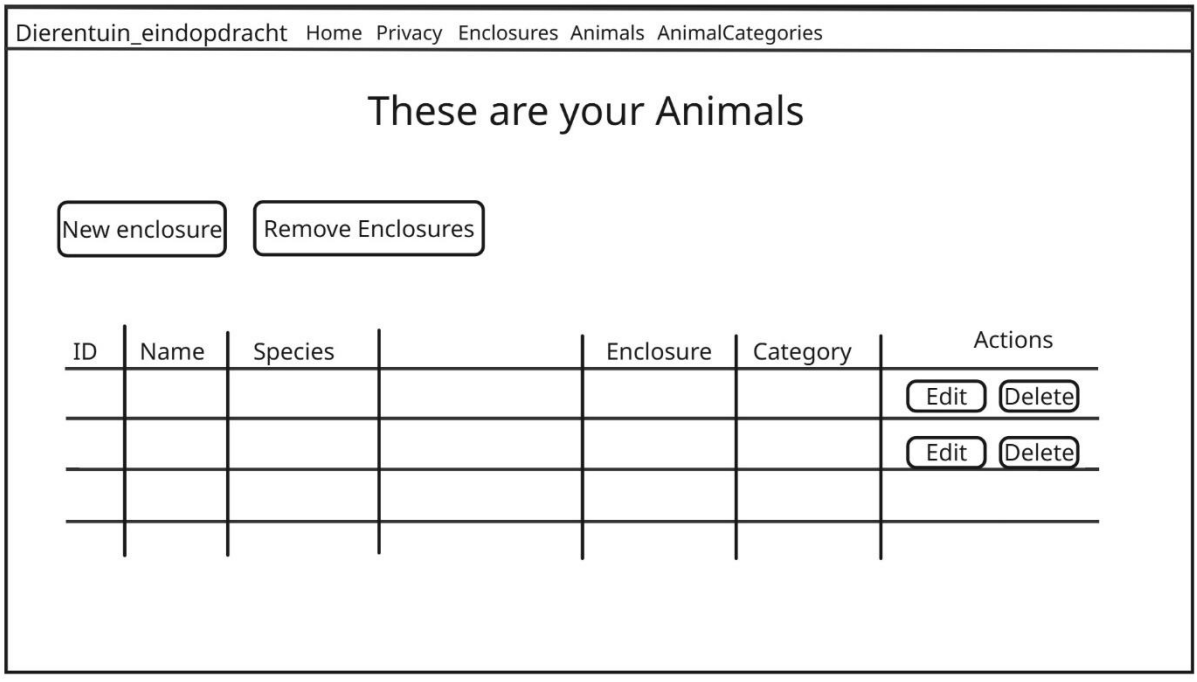
Dit is de home pagina, vanuit hier kan je naar andere paginas gaan of op de knop klikken die je verstuurd naar een pagina met veel grappige dierenfeitjes. Het doel van deze pagina is om de gebruiker te verwelkomen met een goede indruk van de website.



Dit is de Enclosure/verblijven pagina



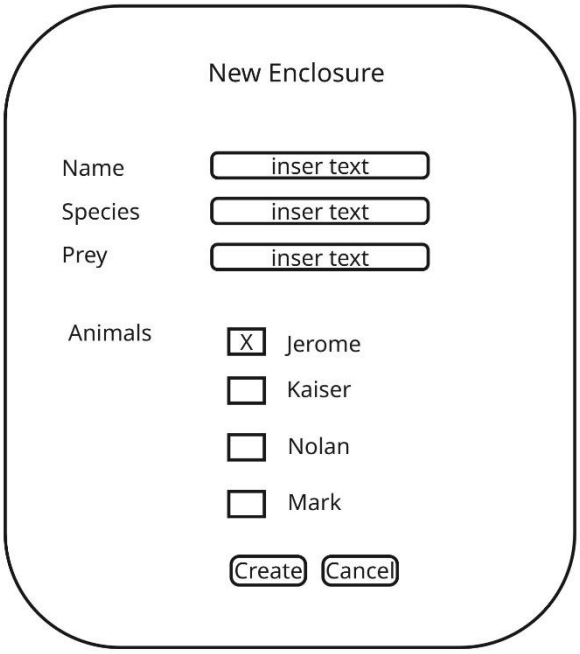
Dit is de AnimalCategory/ categorieën pagina



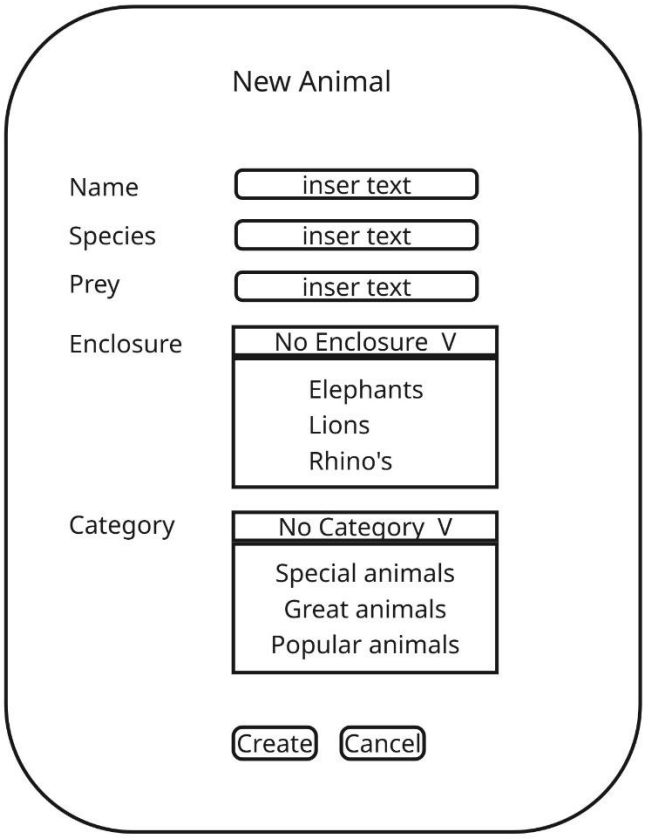
En dit is de Animal/ dieren pagina

deze paginas moeten een simpele maar overzichtelijke layout hebben waarin de knoppen makkelijk te vinden zijn. Daarom hebben we ervoor gekozen om een apparte sectie voor acties op specifieke en alle objecten van een entiteit te maken. Voor specifieke objecten staan er knoppen onder Actions en voor alle objecten van een entiteit staan de knoppen boven aan. Dit principe voor knoppen zal vaker terug komen

Tot slot hebben we de create/edit pagina, Deze verschillen nog al voor enclosures/categories en animals aangezien de één, een many kant is en de ander een one kant van de relatie. Daarom hebben wij hier twee wireframes voor gemaakt om de manier van het definiëren van de relaties weer te geven op de front end. We zijn er uiteindelijk op gekomen dat checkboxes gebruikt worden voor het toevoegen en verwijderen van dieren net zoals hieronder:



En voor het toevoegen van een verblijf aan een dier hebben wij besloten om een select field te maken waarin alle mogelijke verblijven voorkomen.



Aan de hand van deze wireframes was het mogelijk om een inschatting te maken over de uiteindelijke Gui van de dierentuin app.

Reflectie

Goede punten

In de project heb ik ontzetten veel geleerd over ef core framework voor het bouwen van webapps. Voordat ik dit project startte wist ik niet dat ik een framework kon gebruiken om het creëren van een website 10x sneller te doen. Tijdens het project heb ik veel requirements uitgeschreven voordat ik ze probeerde te creëren, dit hielp mij ontzettend veel met het doelgericht werken aan de website.

Slechte punten

Hoewel het creëren van een website een leuk ding is, komen er ook stomme dingen bij van toepassing. Doordat er veel problemen waren waar wij niet doorheen kwamen zijn we veel vast blijven haken en gedemotiveerd geraakt. Het creëren van de website viel ergens onder een grote stapel papieren en werd achterwegen gelaten vanwege andere dingen die we liever deden. Hierdoor verloren we tijd om sommige functionaliteiten af te maken.

Verbeterpunten

In het volgende project zullen wij moeten werken aan het opgeven, hiervoor zullen wij meerdere dagen een kortere tijd aan het project werken in plaats van één keer in de week veel tijd te besteden. Door kleinere tijdsintervallen in te plannen zullen wij minder last hebben van de impact van het niet afkrijgen van dingen en geen motivatie verliezen om verder te werken.

API Code

Index:

```
public IActionResult Index(string searchstring)
{
    var animals = animalService.GetAnimals();

    if (!String.IsNullOrEmpty(searchstring))
    {
        animals = animals
                    .Where(a =>
                        a.Name.ToLower().Contains(searchstring.ToLower())||
                        a.Species.ToLower().Contains(searchstring.ToLower())||
                        (a.Prey != null && a.Prey.ToLower().Contains(searchstring.ToLower()))||
                        a.SpaceRequirement.ToString().Contains(searchstring)||
                        a.FeedingTime.ToLower().Contains(searchstring.ToLower())||
                        a.Arise.ToString().ToLower().Contains(searchstring.ToLower())||
                        a.BedTime.ToString().ToLower().Contains(searchstring.ToLower())||
                        a.Size.ToString().ToLower().Contains(searchstring.ToLower())||
                        a.DietaryClass.ToString().ToLower().Contains(searchstring.ToLower())||
                        a.ActivityPattern.ToString().ToLower().Contains(searchstring.ToLower())||
                        a.SecurityRequirement.ToString().ToLower().Contains(searchstring.ToLower())||
                        (a.Enclosure != null && a.Enclosure.Name.ToLower().Contains(searchstring.ToLower()))||
                        (a.Category != null && a.Category.Name.ToLower().Contains(searchstring.ToLower()))
                    ).ToList();
    }
    return View(animals);
}
```

Filters:

```
[HttpGet]
0 references
public IActionResult Filter(string filter)
{
    var animals = animalService.FilterAnimals(filter);

    return View("Index", animals);
}
```

Create:

```
[HttpGet("Create")]
0 references
public IActionResult Create()//Shows the creating view for creating animals, we need this to acces the create page
{
    var enclosures = enclosureService.GetEnclosures();
    ViewBag.Enclosures = enclosures;  //need this for easier display in form

    var categories = categoryService.GetCategories();
    ViewBag.Categories = categories;  //need this for easier display in form

    return View();
}

[HttpPost("Create")]
0 references
public IActionResult Create(AnimalDto animalDto)//Actually Creates the animal and returns to the index after valid creation
{
    if (!ModelState.IsValid)
    {
        var enclosures = enclosureService.GetEnclosures(); // we are required to refill the dropdowns everytime we get an invalid er
        ViewBag.Enclosures = enclosures;

        var categories = categoryService.GetCategories();
        ViewBag.Categories = categories;

        return View(animalDto); //we need this to help catch the savechanges error and return us to the creation page
    }

    animalService.CreateAnimal(animalDto);

    return RedirectToAction("Index");
}
```

Edit:

```
[HttpGet("Edit")]
0 references
public IActionResult Edit(int id)//shows us the animal data when you edit the animal
{
    var animalDto = animalService.ShowAnimal(id);

    var enclosures = enclosureService.GetEnclosures();
    ViewBag.Enclosures = enclosures;  //need this for easier display in form

    var categories = categoryService.GetCategories();
    ViewBag.Categories = categories;  //need this for easier display in form

    ViewData["AnimalId"] = id;

    return View(animalDto);
}
```

```
[HttpPost("Edit")]
0 references
public IActionResult Edit(int id, AnimalDto animalDto)//actually edits the animal
{
    var animal = animalService.FindAnimal(id);

    if (animal == null)
    {
        return RedirectToAction("Index", "Animal");
    }

    if (!ModelState.IsValid)
    {
        ViewData["AnimalId"] = id;

        var enclosures = enclosureService.GetEnclosures(); // we are required to refill the dropdowns everytime we get an invalid errors otherwi
        ViewBag.Enclosures = enclosures;

        var categories = categoryService.GetCategories();
        ViewBag.Categories = categories;

        return View(animalDto);
    }

    animalService.EditAnimal(id, animalDto);

    return RedirectToAction("Index", "Animal");
}
```

Delete:

```
[HttpGet] //deleting animals
0 references
public IActionResult Delete(int id)
{
    animalService.DeleteAnimal(id);

    return RedirectToAction("Index", "Animal");
}
```

Sunset Sunrise:

```
[HttpGet("Sunset/{enclosureId}")]
0 references
public IActionResult Sunset(int enclosureId) //sends the sleeping animals to the view
{
    var animals = enclosureService.GetSunsetAnimals(enclosureId);

    return PartialView("Sunset", animals);
}

[HttpGet("Sunrise/{enclosureId}")]
0 references
public IActionResult Sunrise(int enclosureId) //sends the animals that are awake to the view
{
    var animals = enclosureService.GetSunriseAnimals(enclosureId);

    return PartialView("Sunrise", animals);
}
```

FeedingTime:

```
[HttpGet("FeedingTime")]
0 references
public IActionResult FeedingTime(int enclosureId) // sends the enclosure's animals to the view
{
    var enclosure = enclosureService.GetEnclosureAnimals(enclosureId);

    return View(enclosure);
}
```

RemoveEnclosures:

```
[HttpPost]
0 references
public IActionResult RemoveEnclosures()
{
    enclosureService.RemoveAllEnclosures();

    return RedirectToAction("Index");
}
```

Get:

```
[HttpGet("api/enclosures")] //api display of enclosures
0 references
public IActionResult Get()
{
    var enclosures = enclosureService.GetEnclosures();
    return Ok(enclosures);
}
```

EnclosureAnimals:

```
[HttpGet("EnclosureAnimals")]
0 references
public IActionResult EnclosureAnimals(int enclosureId) //Returns an enclosure with it's animals to the view
{
    Console.WriteLine($"DEBUG: Ontvangen EnclosureId = {enclosureId}");

    var enclosures = enclosureService.GetEnclosureAnimals(enclosureId);

    return View("EnclosureAnimals", enclosures);
}
```