Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

DigiTiles: .NET Core tilleggsmodul for nopCommerce

Systemdokumentasjon

Versjon 1.4

Dokumentet er basert på Systemdokumentasjon utarbeidet ved NTNU. Revisjon og tilpasninger til bruk ved IDER, DATA-INF utført av Carsten Gunnar Helgesen, Svein-Ivar Lillehaug og Per Christian Engdal. Dokumentet finnes også i engelsk utgave.

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
25.04.2022	1.0	Utkast for kapittel 2, 3, 4, 5, 6 og 7.	**Emma**
29.04.2022	1.1	Utfylling av kapittel 2, 3 og 5.	**Emma**
01.05.2022	1.2	Utfylling av kapittel 8	**Vilde**
08.05.2022	1.3	Endret på kapittel 2, 3, 5, 6, 7 og 9.	**Emma**
14.05.2022	1.4	Ryddet oppsett i dokumentet	**Vilde**

INNHOLDSFORTEGNELSE

1 INNLEDNING

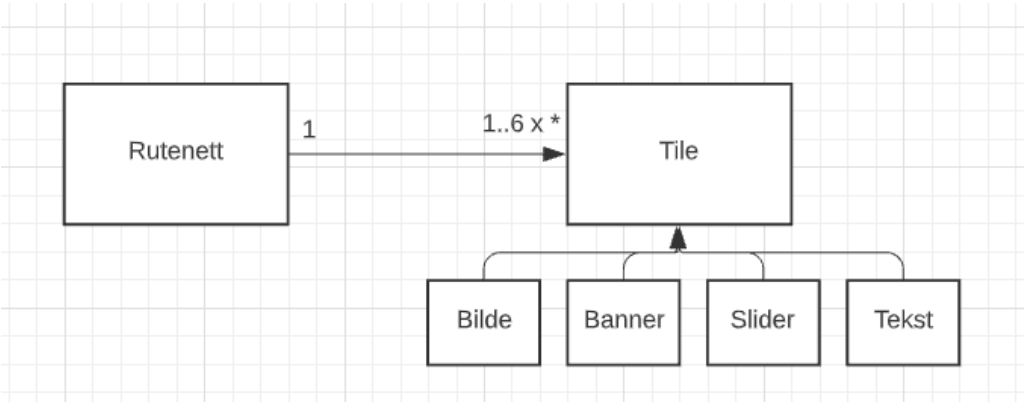
Systemdokumentet gir et innblikk i arkitekturen bak tilleggsmodulen prosjektgruppen har utviklet. Dokumentet er delt inn i åtte kapiteler; Arkitektur, Prosjektstruktur, Klassediagram, Databasemodell, Sikkerhet, Installasjon og Kjøring, Brukermanual, og Dokumentasjon av Kildekode.

Kapittelet *Arkitektur* beskriver verktøyets arkitektur ved hjelp av arkitekturskisser. Her vil de viktigste sub-systemene og komponentene bli skissert. Kapittelet *Prosjektstruktur* beskriver strukturen til verktøyet ved fremvisning av blant annet kildekode og biblioteker. Kapittelet *Klassediagram* beskriver verktøyets klassestruktur ved hjelp av et klassediagram. Kapittelet *Databasemodell* beskriver kort om databaseløsningen for verktøyet. Kapittelet *Sikkerhet* beskriver kort om verktøyets sikkerhetsløsning. Kapittelet *Installasjon og Kjøring* lister opp de viktigste eksterne komponentene som trengs for å ta i bruk verktøyet. Kapittelet *Brukermanual* er verktøyets brukermanual. Kapittelet *Dokumentasjon av Kildekode* beskriver og viser frem verktøyets dokumentasjon av kildekode.

2 ARKITEKTUR

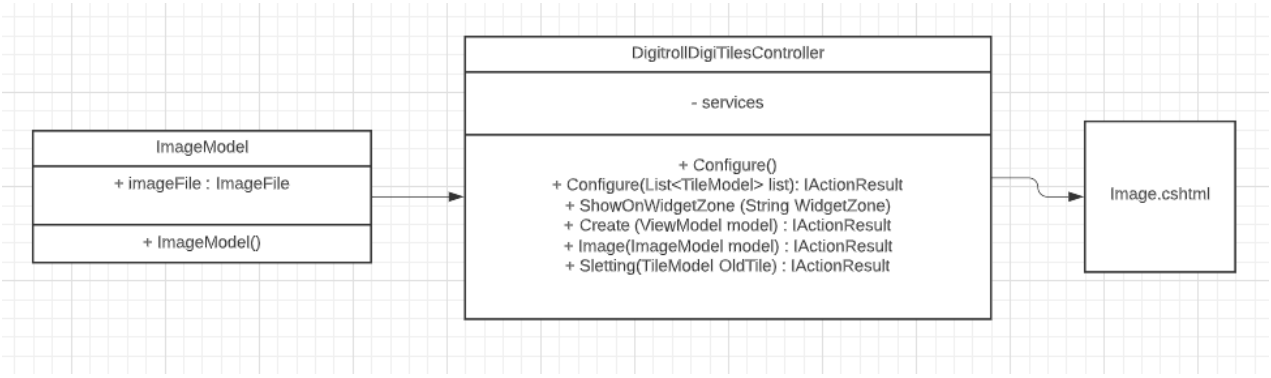
Domenemodell er et godt verktøy for å skape en bedre forståelse av produktets funksjonalitet. Figur 1 er en domenemodell av produktet, som viser forholdet mellom rutenettet og Tile-objektet. Et rutenett inneholder ingen eller flere Tiles, organisert i kolonner og rader. Hver Tile kan maksimalt være 6 lang i bredden, i det tilfelle fyller den ut hele raden.

En Tile deles inn i flere sub-Tiler, som “bilde” og “banner. Disse sub-Tilene arver fra “Tile”-objektet slik at uansett hvilken type det er snakk om så har de felles parametere.

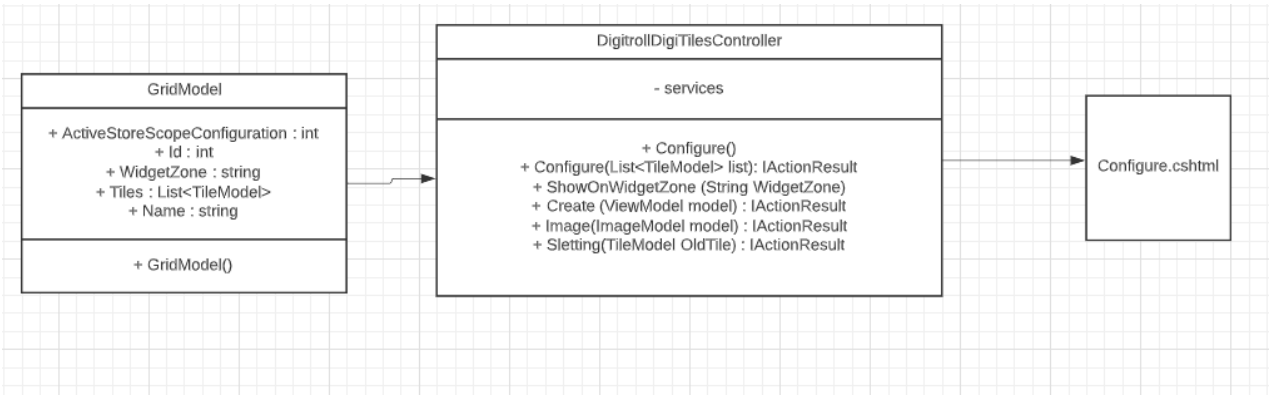


Figur 1: Domenemodell over produktet

Eksempler på kommunikasjonen mellom model–controller-view, kan sees i figur 2 og figur 3.



Figur 2: ImageModel blir sendt til Image.cshtml

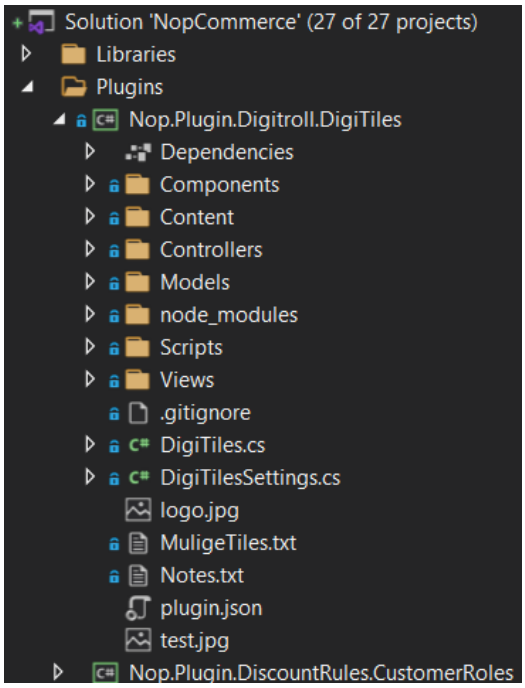


Figur 3: GridModel blir sendt til Configure.cshtml

3 PROSJEKTSTRUKTUR

3.1 Fil- og katalog-struktur

Prosjektet følger MVC-mønsteret, så mappestrukturen er blant annet delt inn i Models, Views og Controller-mapper. Mappestrukturen kan sees i figur 4.



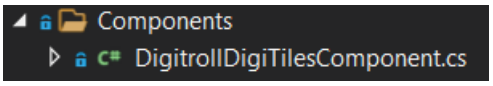
Figur 4: Mappestruktur, vist i Viusal Studio

3.2 Kildekode

Prosjektgruppen har ikke tillatelse fra oppdragsgiver til å dele tilkobling til kode-repoet, og benytter derfor en generell forklaring av prosjektets oppbygning. Kildekoden er delt inn etter mappestrukturen vist over.

3.2.1 Components

Components-mappen (figur 5) inneholder koden ansvarlig for å vise rutenettet brukeren har satt opp, på fremsiden av nettbutikken.



Figur 5: Components-mappe, innhold og mappestruktur

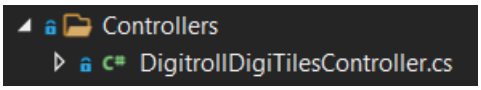
Koden er vist under i figur 6. Et spesifikt view («publicinfo.cshtml») blir sendt til riktig posisjon bestemt av parameteren widgetZone.

```
public class DigitrollDigiTilesComponent : NopViewComponent
{

    private readonly IStoreContext _storeContext;
    private readonly ISettingService _settingService;

    0 references
    public DigitrollDigiTilesComponent(
        IStoreContext storeContext,
        ISettingService settingService)
    {

        _storeContext = storeContext;
        _settingService = settingService;
    }

    /// <summary>
    /// This method is called by nopCommerce at startup and chooses where and what to show at the page.
    /// </summary>
    /// <param name="widgetZone"></param>
    /// <param name="additionalData"></param>
    /// <returns></returns>
    0 references
    public IViewComponentResult Invoke(string widgetZone, object additionalData)

        var digiTilesSettings = _settingService.LoadSetting<DigiTilesSettings>(_storeContext.CurrentStore.Id);
        GridModel model = digiTilesSettings.JsonGrid.ToGridModel();
        if (model.WidgetZone == widgetZone)
        {
            return View("~/Plugins/Digitroll.DigiTiles/Views/PublicInfo.cshtml", model);
        }
        return Content(string.Empty);
    }
}
```

Figur 6: DigitrollDigiTilesComponent.cs

3.2.2 Controller

Controllerklassen er ansvarlig for å overføre data mellom «models» og «views» og representerer «C»-delen av «MVC». Mappestrukturen vises i figur 7.



Figur 7: Controllers-mappe, innhold og struktur

Det er flere metoder i Controller-klassen, bare de viktigste metodene blir vist her.

Metoden som først blir kalt er Configure-metoden (figur 8), som er ansvarlig for å hente ut eksisterende «Grid»-objekt. Metoden sender dette objektet videre til configure.cshtml-viewet. som viser frem dataen til brukeren på configure-siden.

```
0 references
public IActionResult Configure()
{
    // Check access
    if (!_permissionService.Authorize(StandardPermissionProvider.ManageWidgets))
        return AccessDeniedView();

    // Load settings for chosen store
    var storeScope = _storeContext.ActiveStoreScopeConfiguration;
    var digiTilesSettings = _settingService.LoadSetting<DigiTilesSettings>(storeScope);
    GridModel grid = digiTilesSettings.JsonGrid.ToGridModel();

    ViewModel model;

    // Check if user has existing grid, if not:
    if ((!_settingService.SettingExists(digiTilesSettings, x => x.JsonGrid, storeScope)) || digiTilesSettings.JsonGrid == null || grid.Tiles == null)
    {
        model = new ViewModel
        {
            Grid = new GridModel("Navn", storeScope, "home_page_top")
        };

        //Serialize and save in digiTilesSettings
        digiTilesSettings.JsonGrid = model.Grid.ToGridModelSerialized();

        //Save in settings
        _settingService.SaveSetting(digiTilesSettings, x => x.JsonGrid, storeScope);
    }
    else
    {
        // Get model out of settingService
        model = new ViewModel
        {
            Grid = grid
        };

    }
    // Return view with model
    return View("~/Plugins/Digitroll.DigiTiles/Views/Configure.cshtml", model);
```

Figur 8: Configure-metode

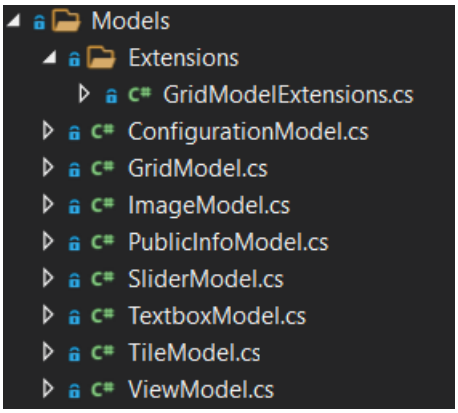
Når brukeren trykker på ny Tile-knappen, blir Create-metoden kalt (figur 9). Denne metoden tar inn et view-objekt som inneholder et Tile-objekt, som brukes til å sende brukeren videre til riktig view. Hvilket view avhenger av typen Tile.

```
[HttpPost]
0 references
public virtual IActionResult Create(ViewModel viewModel)
{

    if (!_permissionService.Authorize(StandardPermissionProvider.ManageWidgets))
        return AccessDeniedView();

    //load settings for a chosen store scope
    var storeScope = _storeContext.ActiveStoreScopeConfiguration;
    var digiTilesSettings = _settingService.LoadSetting<DigiTilesSettings>(storeScope);


    var grid = digiTilesSettings.JsonGrid.ToGridModel();

    var model = viewModel.Tile;

    if (grid.Tiles.Count == 0)
    {
        model.Id = 0;
    }
    else {
        model.Id = grid.Tiles.Max(tile => tile.Id) + 1;
    }

    switch (model.Type)
    {
        case TileTypeEnum.Bilde:
            ImageModel ImageModel = new ImageModel(model.Id, model.Name, model.Type);
            return View("~/Plugins/Digitroll.DigiTiles/Views/Image.cshtml", ImageModel);
        case TileTypeEnum.Banner:
            ImageModel BannerModel = new ImageModel(model.Id, model.Name, model.Type);
            return View("~/Plugins/Digitroll.DigiTiles/Views/Image.cshtml", BannerModel);
        default:
            return RedirectToAction("Configure");
    };
}
```

Figur 95: Create-metode

3.2.3 Models

Figur 10 viser mappestrukturen og innholdet til model-mappen. Model er data-delen av MVC-mønsteret. Som i controller-delen vises bare de viktigste klassene, da det ellers ville blitt mye repetisjon.



Figur 10: Models-mappe, innhold og struktur

ViewModel (figur 11) inneholder en GridModel og TileModel, da typiske MVC-mønstre ikke aksepterer flere enn en model sendt til et view. Dette har prosjsektgruppen løst ved å pakke inn to model-typer inn i en overmodel (viewmodel). Denne modellen blir sendt til configure-viewet for å både kunne vise et grid-objekt og å lage nytt Tile-objekt.

```
10 references
public class ViewModel
{
    1 reference
    public TileModel Tile { get; set; }
    9 references
    public GridModel Grid { get; set; }
}
```

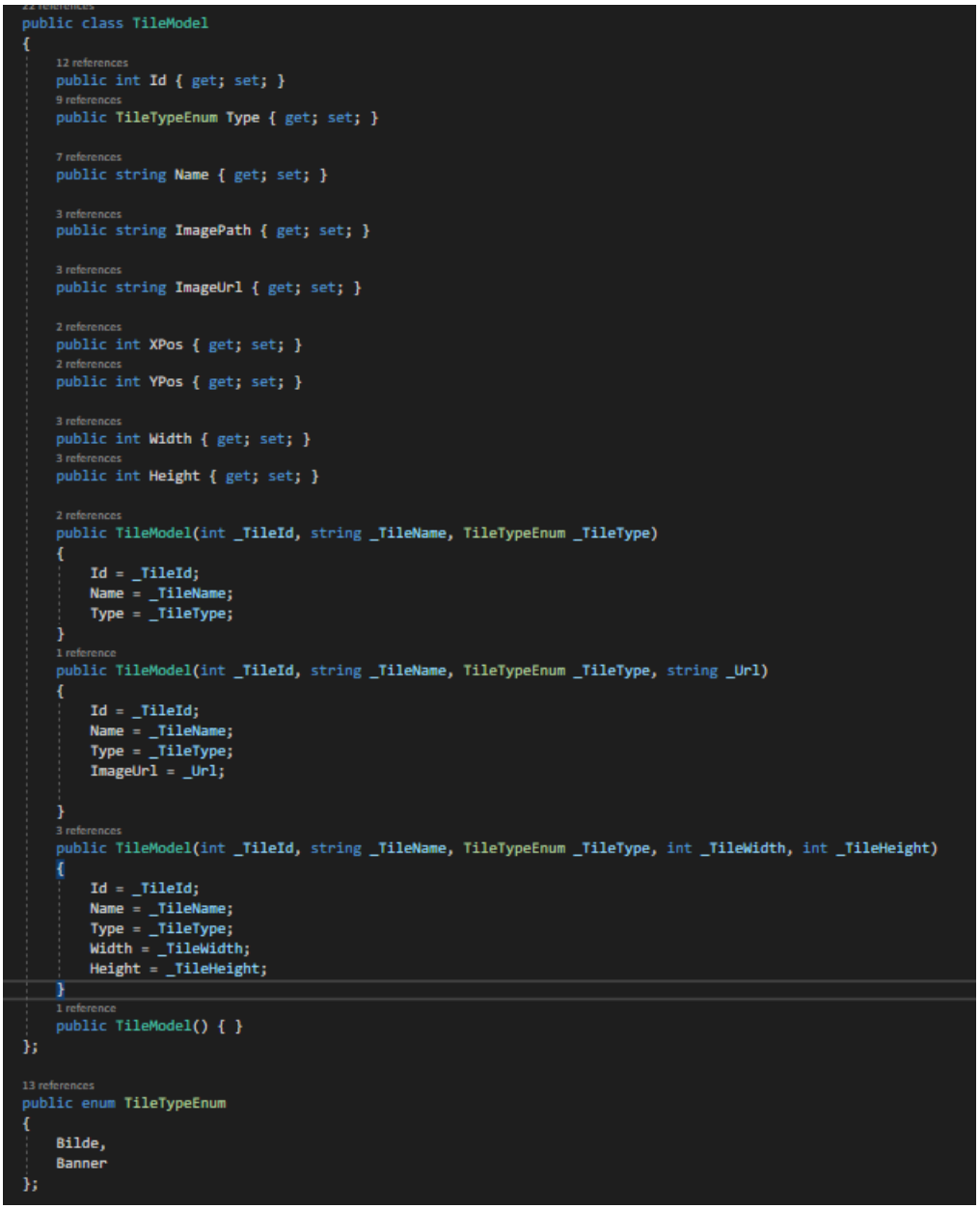
Figur 11: ViewModel

Gridobjektet (figur 12) inneholder blant annet en liste over alle Tiles som brukeren har lagt til, og hvor de skal plasseres på siden.

```
public class GridModel {
    1 reference
    public int ActiveStoreScopeConfiguration { get; set; }
    0 references
    public int Id { get; set; }
    3 references
    public string WidgetZone { get; set; }
    12 references
    public List<TileModel> Tiles { get; set; }
    2 references
    public string Name { get; set; }

    0 references
    public GridModel() {
        Tiles = new List<TileModel>();
    }
    4 references
    public GridModel(string _Name, int _ActiveStoreScopeConfiguration, string widgetZone)
    {
        Name = _Name;
        Tiles = new List<TileModel>();
        ActiveStoreScopeConfiguration = _ActiveStoreScopeConfiguration;
        WidgetZone = widgetZone;
    }
};
```

Figur 12: GridModel

Tilsene som bygger opp nettsiden er representert via TileModel (figur 13). Denne klassen blir arvet av alle andre Tiles. Nederst i klassen er det en liste over tilgjengelige Tile-typer.



```
public class TileModel
{
    12 references
    public int Id { get; set; }
    9 references
    public TileTypeEnum Type { get; set; }

    7 references
    public string Name { get; set; }

    3 references
    public string ImagePath { get; set; }

    3 references
    public string ImageUrl { get; set; }

    2 references
    public int XPos { get; set; }
    2 references
    public int YPos { get; set; }

    3 references
    public int Width { get; set; }
    3 references
    public int Height { get; set; }

    2 references
    public TileModel(int _TileId, string _TileName, TileTypeEnum _TileType)
    {
        Id = _TileId;
        Name = _TileName;
        Type = _TileType;
    }
    1 reference
    public TileModel(int _TileId, string _TileName, TileTypeEnum _TileType, string _Url)
    {
        Id = _TileId;
        Name = _TileName;
        Type = _TileType;
        ImageUrl = _Url;

    }
    3 references
    public TileModel(int _TileId, string _TileName, TileTypeEnum _TileType, int _TileWidth, int _TileHeight)
    {
        Id = _TileId;
        Name = _TileName;
        Type = _TileType;
        Width = _TileWidth;
        Height = _TileHeight;
    }
    1 reference
    public TileModel() { }
};

13 references
public enum TileTypeEnum
{
    Bilde,
    Banner
};
```

Figur 13: TileModel

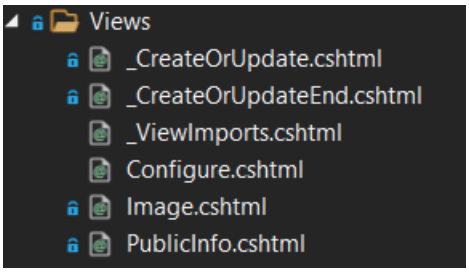
En av disse klassene er ImageModel-klassen (figur 14), som arver fra TileModel. Image typen representerer en Bilde-Tile.

```
3 references
public class ImageModel : TileModel
{
    3 references
    public IFormFile ImageFile { get; set; }

    0 references
    public ImageModel() { }

    2 references
    public ImageModel(int _tileId, string _tileName, TileTypeEnum _type) : base(_tileId, _tileName, _type)
    { }

    0 references
    public ImageModel(int _tileId, string _tileName, TileTypeEnum _type, string _url) : base(_tileId, _tileName, _type, _url)
    { }

    0 references
    public ImageModel(int _tileId, string _tileName, TileTypeEnum _type, int _tileWidth, int _tileHeight) : base(_tileId, _tileName, _type, _tileWidth, _tileHeight)
    { }
```

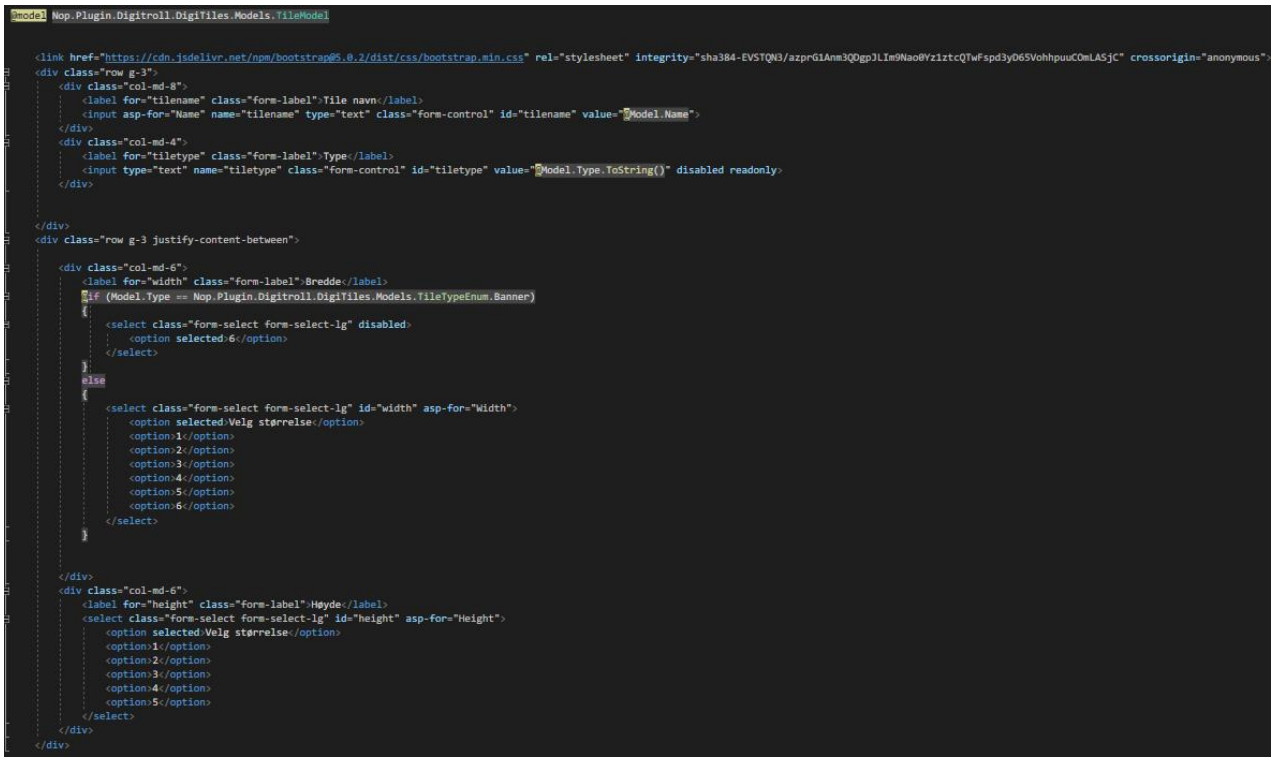
Figur 14: ImageModel

3.2.4 Views

Views representerer grensesnittet brukeren får se, og viser ofte dataen fra model-klassene sendt via controller-klassen. View-mappestrukturen vises i figur 15. Filer som starter med understrek (_) er «partial»-views, html som blir hentet av andre filer. Disse partial-filene blir hentet av ny-Tile siden som må endre seg avhengig av typen Tile. Ved å gjenbruke html-kode via partial-views er det mindre sjanse for feil, og det er lettere å endre. En av disse partial-viewsene kan sees i figur 16, *_CreateOrUpdate.cshtml*. Den andre cshtml filen vist her er *PublicInfo.cshtml* (figur 17), som er filen som blir vist på fremsiden av nettbutikken til brukeren.



Figur 15: View-mappe, innhold og struktur



```
@model Nop.Plugin.Digitroll.DigiTiles.Models.TileModel

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">
<div class="row g-3">
    <div class="col-md-8">
        <label for="tilename" class="form-label">Tile navn</label>
        <input asp-for="Name" name="tilename" type="text" class="form-control" id="tilename" value="@Model.Name">
    </div>
    <div class="col-md-4">
        <label for="tiletype" class="form-label">Type</label>
        <input type="text" name="tiletype" class="form-control" id="tiletype" value="@Model.Type.ToString()" disabled readonly>
    </div>


</div>
<div class="row g-3 justify-content-between">

    <div class="col-md-6">
        <label for="width" class="form-label">Bredde</label>
        @if (Model.Type == Nop.Plugin.Digitroll.DigiTiles.Models.TileTypeEnum.Banner)
        {
            <select class="form-select form-select-lg" disabled>
                <option selected>6</option>
            </select>
        }
        else
        {
            <select class="form-select form-select-lg" id="width" asp-for="Width">
                <option selected>Velg størrelse</option>
                <option>1</option>
                <option>2</option>
                <option>3</option>
                <option>4</option>
                <option>5</option>
                <option>6</option>
            </select>
        }


    </div>
    <div class="col-md-6">
        <label for="height" class="form-label">Høyde</label>
        <select class="form-select form-select-lg" id="height" asp-for="Height">
            <option selected>Velg størrelse</option>
            <option>1</option>
            <option>2</option>
            <option>3</option>
            <option>4</option>
            <option>5</option>
        </select>
    </div>
</div>
```

Figur 16: _CreateOrUpdate.cshtml

```
<head>
    <script>
        function changeMarginBanner() {

            banners = document.getElementsByClassName("tile-banner")

            var test = document.querySelector('.master-wrapper-content');
            var left_margin = window.getComputedStyle(test).getPropertyValue("margin-left"); // returns margin e.g. '655px'
            left_margin = left_margin.match(/\d+/); //returns bare number e.g. '655'
            left_margin = left_margin * (-1) // Returns a negative number e.g. -655

            for (let banner of banners) {
                banner.style.marginLeft = left_margin + "px"
            }
            console.log(" second: " + left_margin + "px")
        }
        window.addEventListener('resize', changeMarginBanner)
    </script>

    <style>
        .container {
            margin: 10px 0;
        }
        .tile-image {
            width: 100%;
            height: 100%;
        }
        .tile-image-content {
            width: 100%;
            height: 100%;
            padding: 10px;
        }
        .tile-banner-content {
            width: 100vw;
            height: 100%;
            padding: 10px 0px;
        }
        .tile-banner {
            width: 100%;
            height: 100%;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="grid-stack" gs-static="true">
            @foreach (TileModel t in Model.Tiles)
            {
                <div class="grid-stack-item ui-draggable-disabled ui-resizable-disabled" position="absolute" gs-no-move="true
                    @switch (t.Type)
                    {
                        case TileTypeEnum.Bilde:
                            <div class="grid-stac-item-content tile-image-content">
                                @if (t.ImageUrl != null)
                                {
                                    <a href="@t.ImageUrl">
                                        <img class="tile-image" src="@t.ImagePath" />
                                    </a>
                                }
                                else
                                {
                                    <img class="tile-image" src="@t.ImagePath" />
                                }
                            </div>
                            break;
                        case TileTypeEnum.Banner:
                            <div class="grid-stac-item-content tile-banner-content">
                                @if (t.ImageUrl != null)
                                {
                                    <a href="@t.ImageUrl">
                                        <img class="tile-banner" src="@t.ImagePath" />
                                    </a>
                                }
                                else
                                {
                                    <img class="tile-banner" src="@t.ImagePath" />
                                }
                            </div>
                            break;
                    }
                </div>
            }
        </div>

        <script>changeMarginBanner()</script>
    </div>
    <script type="text/javascript">
        GridStack.init({ column: 6, disableOneColumnMode: false, margin: 10 });
    </script>
</body>
```

Figur 17: Publicinfo.cshtml

3.2.5 Database

Koden under viser hvordan pluginen får tilgang til databasen for å lagre innstillingene til brukeren. Pluginen bruker nopCommerce sin innebygde «SettingService» som lagrer nettbutikk-innstillinger brukeren har satt.

Figur 18 viser hvordan SettingsObjektet «DigiTilesSettings» (vist i figur 20) blir lagret i databasen. Figur 19 viser hvordan dataen blir hentet fra databasen. For grundigere gjennomgang se kapittel 5, «databasemodell».

```
//Save in settings
_settingService.SaveSetting(digiTilesSettings, x => x.JsonGrid, storeScope);
```

Figur 18: Lagring i SettingService

```
//load settings for a chosen store scope
var storeScope = _storeContext.ActiveStoreScopeConfiguration;
var digiTilesSettings = _settingService.LoadSetting<DigiTilesSettings>(storeScope);

var temp = digiTilesSettings.JsonGrid.ToGridModel();

temp.Tiles = newTilesList;
```

Figur 19: Hente ut data fra SettingService

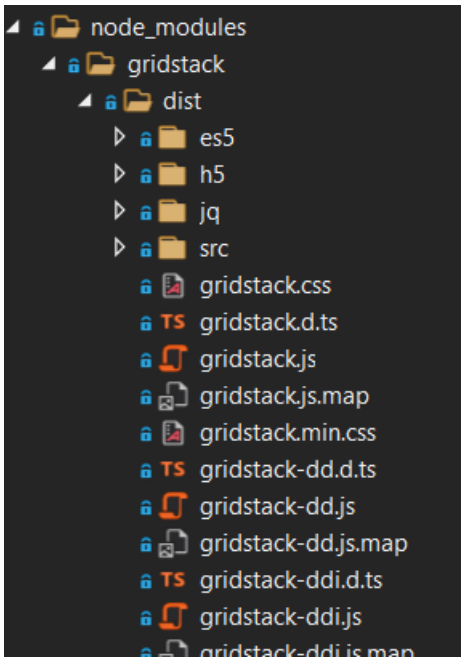
```
9 references
public class DigiTilesSettings : ISettings
{
    28 references
    public String JsonGrid { get; set; }

};
```

Figur 20: DigiTilesSettings, som har en parameter: JsonGrid

3.3 Biblioteker

DigiTiles bruker et bibliotek, gridstack.js (gridstack.js, 2020), et bibliotek som lager drag-and-drop rutenett. Biblioteket har MIT lisens og kan brukes fritt. Gridstack ble installert med programvare-

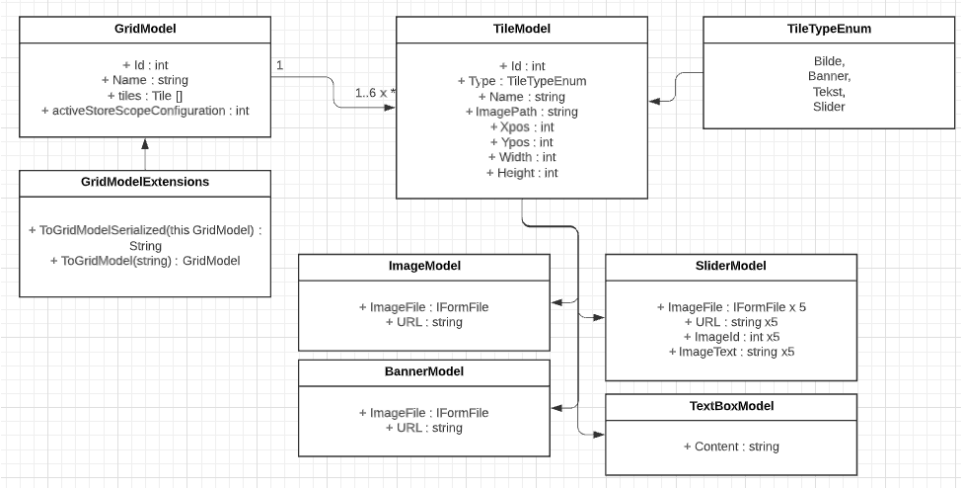
registeret npm (npm Docs, u.å). Biblioteket inneholder flere filer, hovedsakelig javascript (.js) og css-filer (figur 21).



Figur 21: gridstack.js mappestruktur og innhold

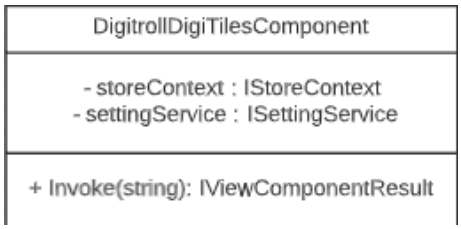
4 KLASSEDIAGRAM

Klassediagrammet under (figur 22) viser en oversikt over model-delen av serversiden med utgangspunkt i og klasser som samarbeider med GridModel.



Figur 22: Klassediagram for serversiden, oversikt over modelstrukturen

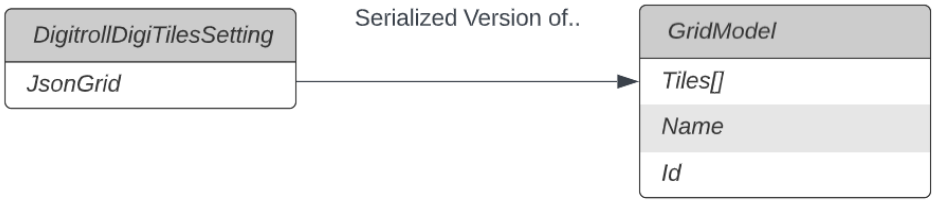
Klientsiden er relativt enkelt da ressurser fra nopCommerce sin side henter Invoke() metoden som videresender et view (en .cshtml-fil) som nopCommerce viser på valgt side. Klassen er representert i klassediagrammet vist i figur 23.



Figur 23: Klassediagram klientside

5 DATABASEMODELL

Prosjektgruppen bruker ikke egen løsning for databasetilgang, men bruker nopCommerce sin "settingsService" som kontrollerer tilgang til databasen. Denne servicen lar en lagre enkle, serialiserte, datasett uten problemer, sammenhengen er vist i figur 24.



Figur 24: DigitrollDigiTilesSettings og GridModel

En rask oppsummering av stegene er skrevet under. Skjermbilde av koden er vist i kapittel 3.2.5 («database»).

Lagring blir utført på denne metoden:

1. Dataen blir serialisert
2. Den serialiserte dataen blir lagret til et DigiTilesSettings-objekt.
3. Dette objektet blir sendt til SettingService, med en gitt storeContexten som representerer versjonen av nettbutikken brukeren er på.

For å hente ut data:

1. Henter storeContexten
2. Laster innstillinger fra SettingsService ved hjelp av storeContext
3. Deserialiserer dataen lagret i DigiTilesSettings

6 SIKKERHET

For å sikre at brukeren er logget inn og har tilgang til nettsiden og funksjonene som blir bedt om, bruker prosjektet nopCommerce sin egen service, permissionService. At brukeren har adgang, blir sjekket i hvert kall til metoder fra controller-klassen. Om brukeren ikke har tilgang vil de bli sendt videre til en side levert av samme service. Koden er vist i figur 25.

```
if (!_permissionService.Authorize(StandardPermissionProvider.ManageWidgets))
    return AccessDeniedView();
```

Figur 25: Skjermbilde av kall til sikkerhetsmetoden "Authorize"

7 INSTALLASJON OG KJØRING

Alle som vil ta i bruk tilleggsmodulen utviklet for bacheloren må ha plattformen nopCommerce versjon 4.30 tilgjengelig, i tillegg til en database koblet opp mot nopCommerce.

For arbeidsgiver (Digitroll) sine kunder vil tilleggsmodulen bli lastet opp på en server kundene har tilgang til, for lettere installasjon fra deres egen kopi av nopCommerce.

Brukere som ikke er kunder hos Digitroll må laste ned nopCommerce versjon 4.30 (med eller uten kildekode). nopCommerce 4.30 finner man her: https://github.com/nopSolutions/nopCommerce/releases/tag/release-4.30.

NopCommerce har sine egne systemkrav, disse kravene er beskrevet her: https://docs.nopcommerce.com/en/installation-and-upgrading/technology-and-system-requirements.html

Etter nedlastning må tilleggsmodulen lastes ned og plasseres i "plugin"-mappen i nopCommerce. Når brukeren kjører nopCommerce vil de bli bedt om å opprette en bruker og koble til database. Her kan man for eksempel bruke SQL Server Management Studio (SSMS), som lager en lokal database.

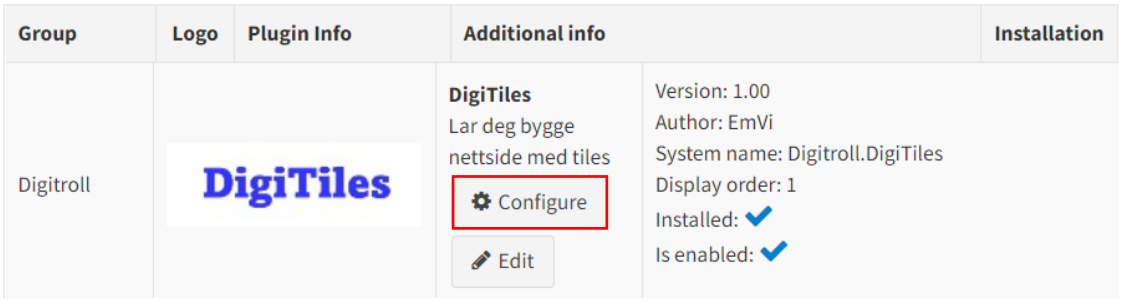
SSMS kan lastes ned herfra: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

Etter dette logger man inn på brukeren sin, og kan endre og ta i bruk plugins gjennom admin-siden.

8 BRUKERMANUAL

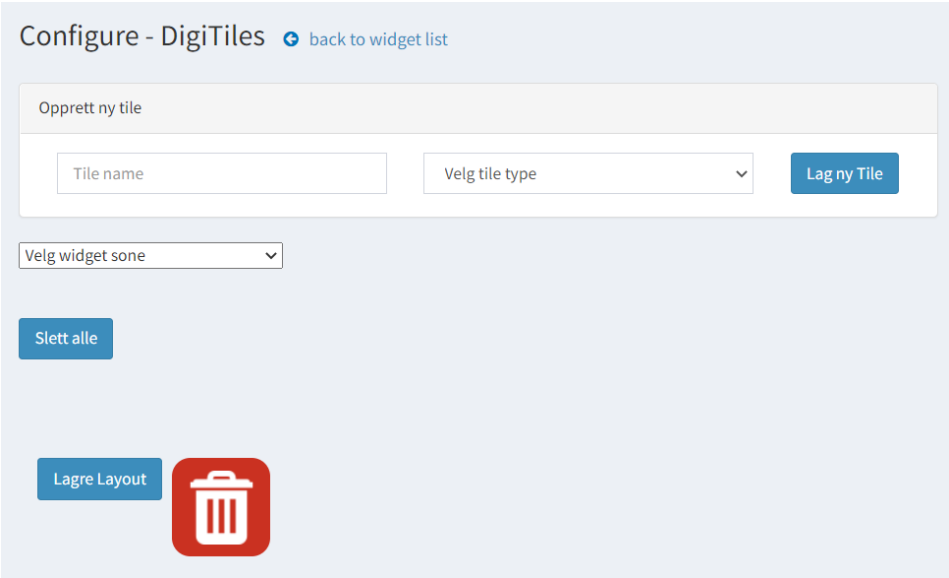
8.1 Konfigureringsside

Konfigureringssiden er tilleggsmodulens hovedmeny hvor brukeren kan konfigurere innholdet til Tilesene. Konfigureringssiden befinner seg i admin-siden, under *Configuration > Local plugins* som befinner seg i venstre panel. Brukeren skal for så trykke på Configure-knappen. Figur 26 er et skjermbilde av tilleggsmodulen under siden *Local plugins*.



Figur 26: DigiTiles blant en liste med andre tilleggsmoduler

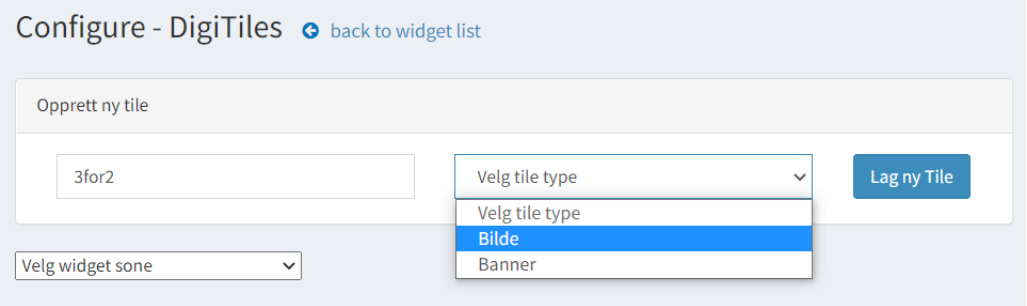
Brukeren er nå i konfigureringssiden til tilleggsmodulen DigiTiles. Figur 27 er tilleggsmodulens konfigureringsside ved første oppstart.



Figur 27: Konfigureringssiden til DigiTiles

8.2 Opprette en Tile

Ved oppretting av Tile, kan brukeren velge mellom to ulike typer Tile; *Bilde* eller *Banner*. Begge Tilene krever en opplasting av en bildefil som skal fremvises på Tilen. Et banner vil derimot bryte rutenettets bredde, og fylle til begge kanter av nettsiden. Brukeren kan derfor kun ha muligheten til å endre banneres høyde. Figur 28 er en illustrasjon av en bruker som lager et bilde-Tile.



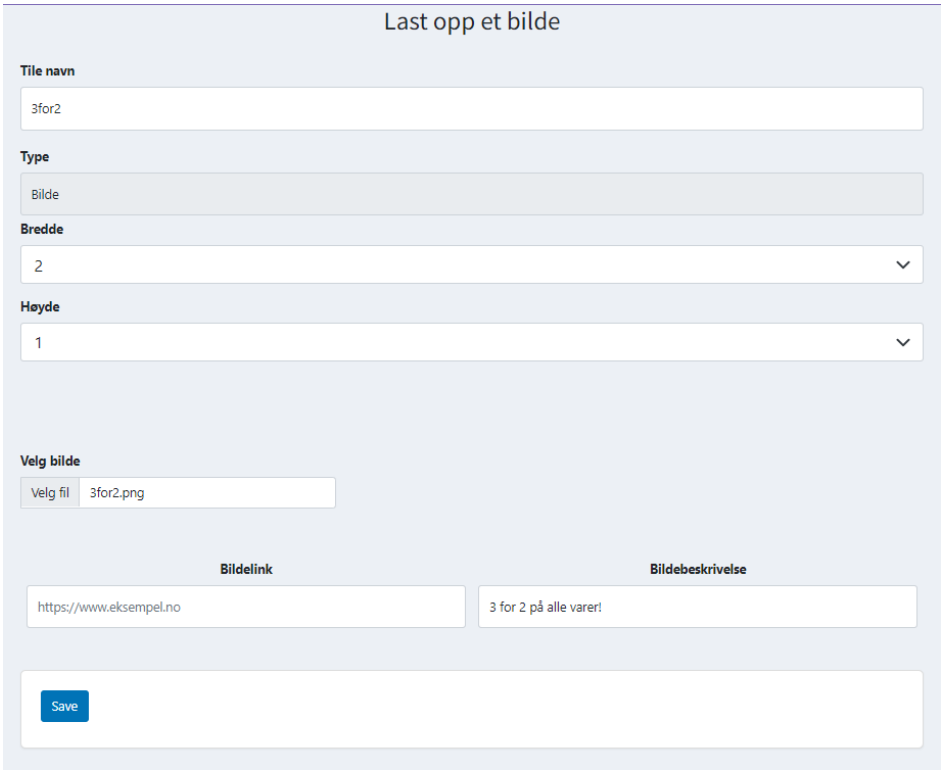
Figur 28: Legge til Tile

Ved oppretting av Tile, vil en «pop-up» dukke opp på skjermen. Her kan brukeren fylle inn informasjon om den ønskede Tilen. Banner- og bilde-Tile vil ha likt informasjon-skjema som skal fylles ut. En Tile krever navn og filopplastning. Høyde og bredden til en Tile er derimot ikke et krav, og om disse ikke er fylt ut, vil Tilen ved opprettelse få en standardverdi på 1x1.

Om brukeren ønsker, kan en link festes til Tilen. Da vil et museklikk på den aktuelle Tilen føres til linkens side. Det er ikke et krav å legge til en bildelink til Tilen.

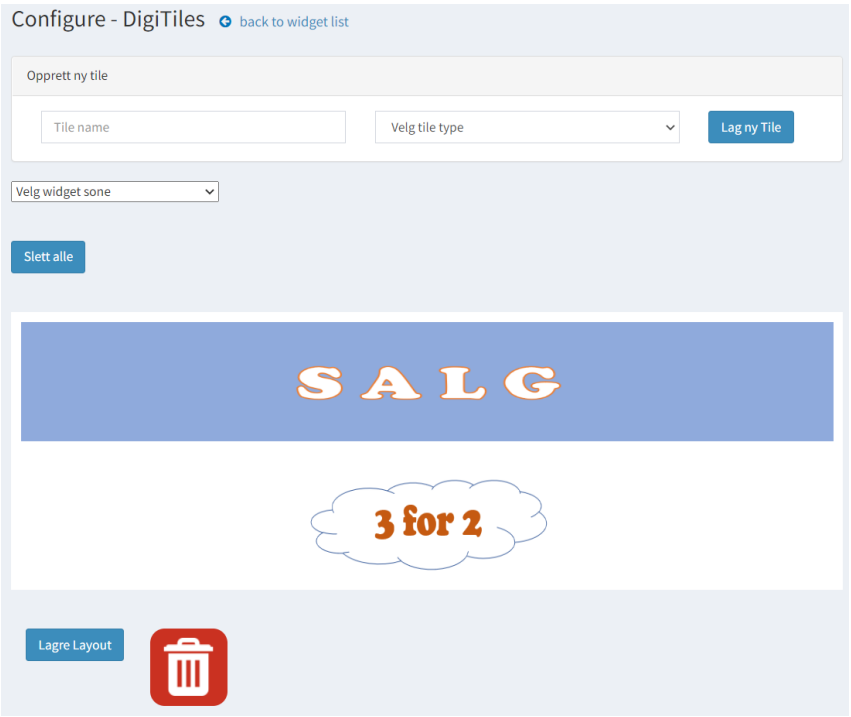
For personer med synshemninger, er det utviklet en bildebeskrivelse som kan festes på en Tile. Bildebeskrivelser er innhold som kan bli tatt opp av en skjermleser slik at innhold blir presentert ved hjelp av syntetisk tale. Da kan bilder med et beskrivende innhold bli presentert for de synshemmede. (Norges Blindeforbund, u. å.)

Figur 29 er en illustrasjon av en pop-up til en bilde-Tile.



Figur 29: Popup til bilde-Tile

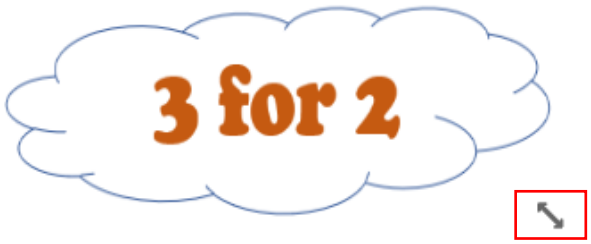
Når brukeren har opprettet en mengde Tiles, vil konfigureringssiden lage et rutenett med de aktuelle Tilesene. Figur 30 er en illustrasjon av konfigureringssiden med innhold.



Figur 30: Konfigureringssiden med innhold

8.3 Endre størrelse og plassering

En Tile kan endre størrelse ved hjelp av å dra på pilen som befinner seg høyre, nederst i hjørnet. Pilen vil dukke opp når musen svever over Tilen. For å endre bredden, skal pilen dras vertikalt. Høyden kan endres ved å dra pilen horisontalt. Ved å dra pilen diagonalt, kan man endre både bredden og høyden til en Tile samtidig. Figur 31 er en illustrasjon av en Tile med en synlig pil.



Figur 31: Endring av størrelse

Plasseringen til en Tile kan endres ved hjelp av *drag-and-drop*, vist i figur 32. Funksjonaliteten aktiveres ved hjelp av å klikke og holde på en Tile, for så å dra og slippe Tilen på ønsket plass. En Tile vil flytte seg om brukeren ønsker å plassere objektet over, under eller ved siden av den okkuperte plassen.

Figur 32: Drag-and-drop av en Tile

8.4 Slette Tile

En Tile kan slettes ved hjelp av drag-and-drop, ved å dra og slippe den ønskede Tilen over søppel-ikonet. Figur 33 er en illustrasjon av slette handlingen.

Figur 33: Drag-and-drop over søppel-ikon

Når en slipper Tilen, vil en varsling dukke på brukerens skjerm. Brukeren kan på denne måten angre handlingen. Figur 34 er et skjermbilde av varselet som aktives ved sletting.

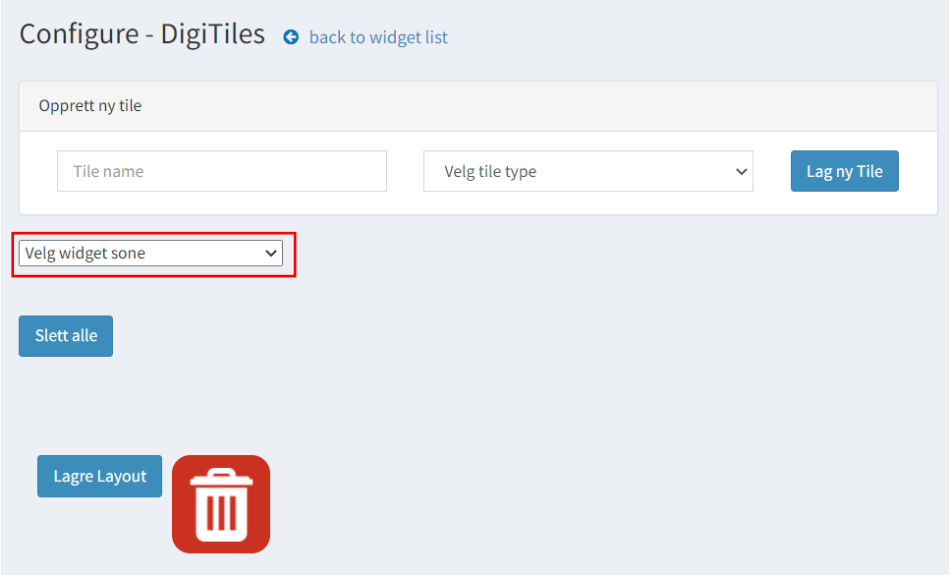
Figur 34: Varsel ved sletting

Alt innhold til et rutenett kan slettes ved å trykke på «Slett alle»-knappen. Figur 35 er et skjermbilde av knappen, som befinner seg over rutenettet.

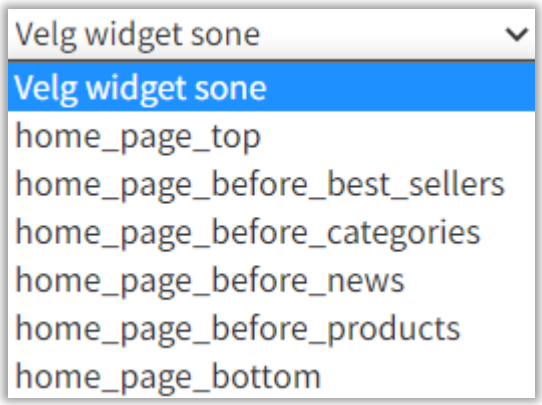
Figur 35: Knapp for å slette alt innhold i rutenett

8.5 Legge til widget sone

Rutenettets innhold kan presenteres i en ønsket *widget* sone. En widget sone er en innebygget funksjonalitet i nopCommerce med ulike soner i nettbutikkens sider. Sonene spesifiserer hvilken rekkefølge og posisjon innhold kommer i, og utvides til innholdets størrelse. Rutenettet med innhold vil plasseres i den valgte sonen. Figur 36 og figur 37 er et skjermbilde plasseringen, og de ulike widget sonene, av drop-down listen.



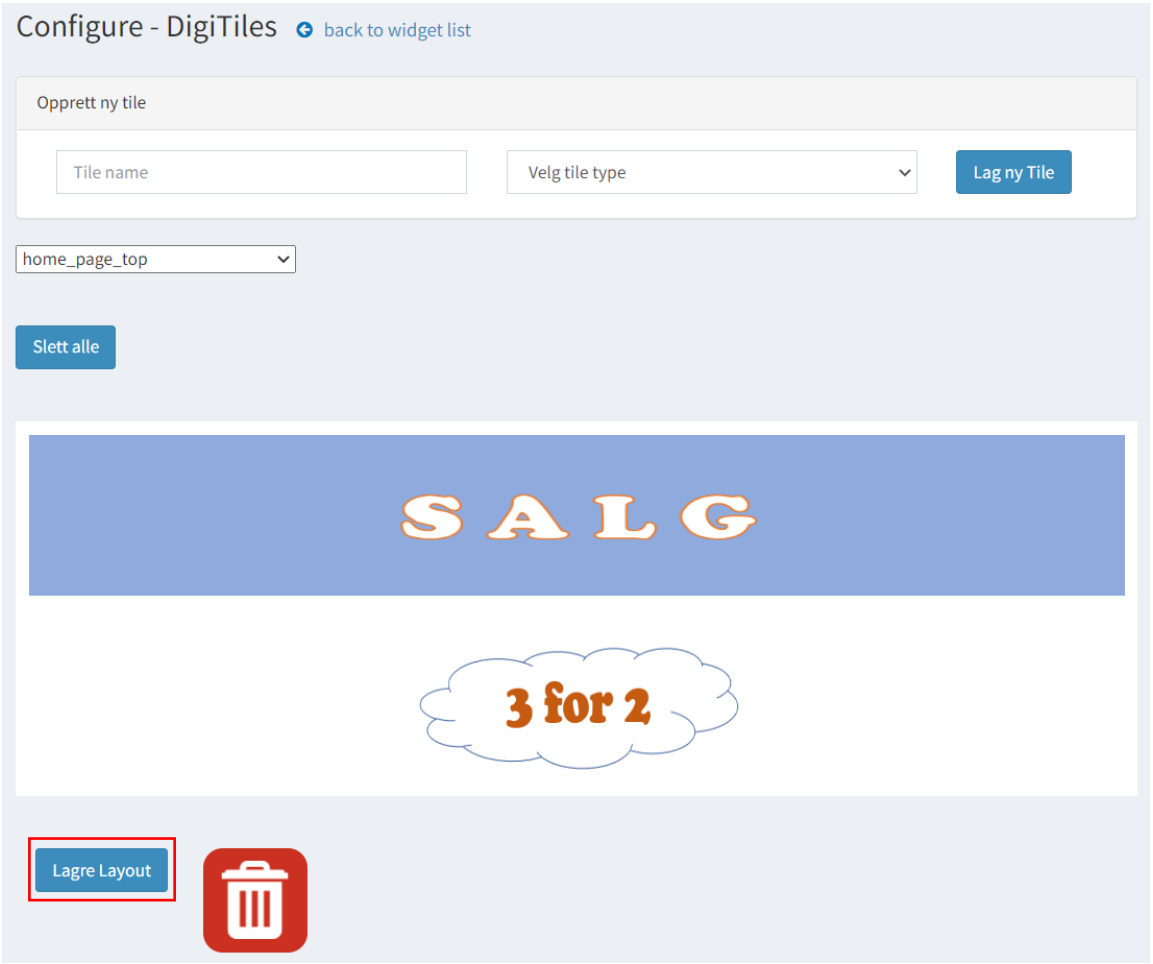
Figur 36: Plassering av drop-down liste i konfigueringssiden



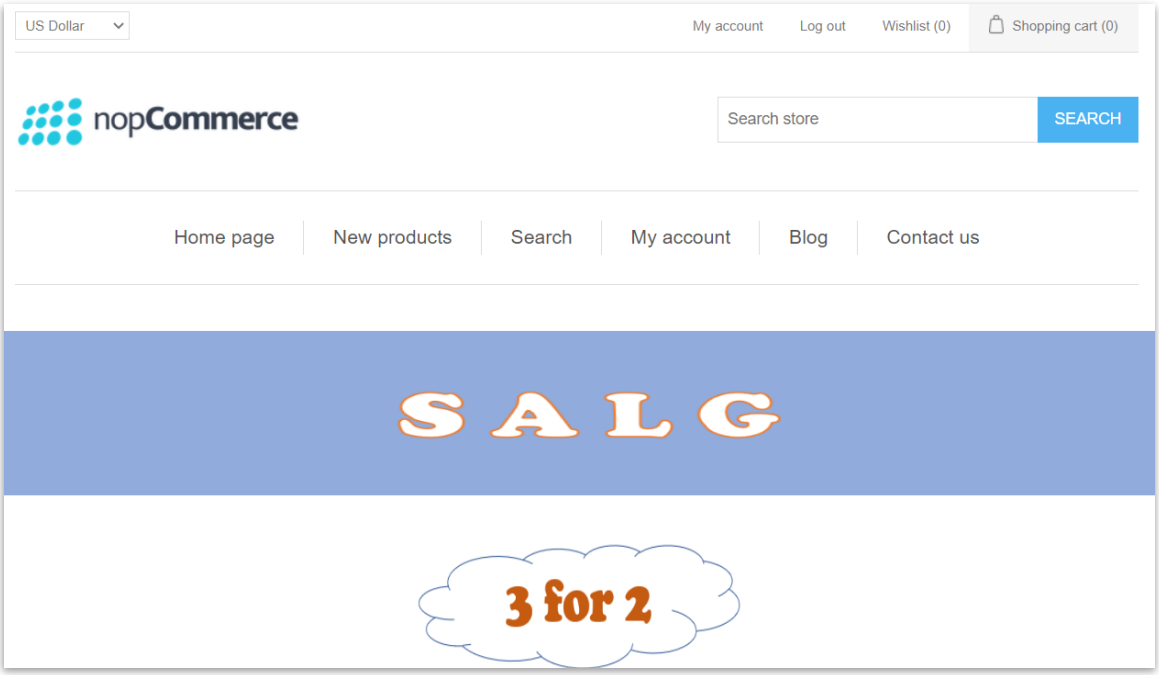
Figur 37: Drop-down liste med ulike widget soner

8.6 Lagre layout

Når brukeren er fornøyd med oppsettet av Tiles i rutenettet, kan *layouten* lagres. Layouten er rutenettets struktur av Tiles i ønsket plassering. Lagringsknappen befinner seg nederst til venstre i konfigureringssiden. Ved lagring, vil layouten plasseres i valgt widget sone. Figur 38 og figur 39 er illustrasjon av lagring og fremvisning av layout på kunde side.



Figur 38: Lagring av layout



Figur 39: Lagret layout i kunde side

9 DOKUMENTASJON AV KILDEKODE

For å se nyeste versjon av dokumentasjonen er det mulig å finne filen «Nop.Plugin.Digitroll.DigiTiles.xml» i mappen der nopCommerce ligger -> Presentation -> Nop.Web -> Plugins -> Digitroll.DigiTiles.

Figur 40 er et skjermbilde av dokumentasjonen på tidspunktet denne filen ble sist oppdatert.

```
<member name="M:DigiTiles.Components.DigitrollDigiTilesComponent.Invoke(System.String,System.Object)">
    <summary>
    This method is called by nopCommerce at startup and chooses where and what to show at the page.
    </summary>
    <param name="widgetZone"></param>
    <param name="additionalData"></param>
    <returns></returns>
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Controller.DigitrollDigiTilesController.Configure">
    <summary>
    Is called before configure-page, gets existing grid-model or creates new if necessary.
    </summary>
    <returns> Configure-view </returns>
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Controller.DigitrollDigiTilesController.Configure(System.Collections.Generic.List{Nop.Plugin.Digitroll.DigiTiles.Models.TileModel})">
    <summary>
    This method is activated when the user saves the layout
    Saves the lit of tiles in the given position
    </summary>
    <param name="newTilesList"> list of Tiles in the given position </param>
    <returns>Redirects to Configure(GET)</returns>
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Controller.DigitrollDigiTilesController.ShowOnWidgetZone(System.String)">
    <summary>
    This method changes the widgetZone-variable for the local grid
    </summary>
    <param name="widgetZone"> chosen widgetZone from Configure view </param>
    <returns> </returns>
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Controller.DigitrollDigiTilesController.Create(Nop.Plugin.Digitroll.DigiTiles.Models.ViewModel)">
    <summary>
    This method creates a tile on the grid
    </summary>
    <param name="viewModel"> parameters given from the Configure view, into a ViewModel</param>
    <returns> Sends model to appropriate view depending on tiletype </returns>
</member>0
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Controller.DigitrollDigiTilesController.DeleteAll">
    This method Deletes all tiles on the grid.
    Makes a new empty grid
    Is gonna be removed before handin.
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Controller.DigitrollDigiTilesController.Image(Nop.Plugin.Digitroll.DigiTiles.Models.ImageModel)">
    <summary>
    This method saves an image from the variables and files given
    from the user.Saves the grid at last, with image.
    Image is saved locally
    Presentation > Nop.Web > wwwroot
    </summary>
    <param name="image"> image model from view with variables</param>
    <returns> Redirectet to configure(GET) </returns>
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Controller.DigitrollDigiTilesController.Sletting(Nop.Plugin.Digitroll.DigiTiles.Models.TileModel)">
    <summary>
    Deletes a tile from the grid
    Saves the grid in the end, without the tile
    </summary>
    <param name="oldTile"> tile that is chosen to be deleted </param>
    <returns>model with updated grid, javascript, which prints out tiles and reloads current site (caches)</returns>
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Controller.DigitrollDigiTilesController.ChooseTileType">
    <summary>
    Opens the choose tile view when creating a new tile
    </summary>
    <returns> ChooseTile view</returns>
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Models.Extensions.GridModelExtensions.ToGridModelSerialized(Nop.Plugin.Digitroll.DigiTiles.Models.GridModel)">
    <summary>
    Receives a GridModel-object, serializes it and returns it.
    </summary>
    <param name="gridModel"> GridModel-object that needs to be seralized. </param>
    <returns> Json-object of gridmodel. </returns>
</member>
<member name="M:Nop.Plugin.Digitroll.DigiTiles.Models.Extensions.GridModelExtensions.ToGridModel(System.String)">
    <summary>
    Deserializes a JSON-object into a GridModel-object
    </summary>
    <param name="gridModelSerialized"> Json-object </param>
    <returns> Deseralized GridModel-object </returns>
</member>
<member name="T:Nop.Plugin.Digitroll.DigiTiles.Models.TileTypeEnum">
    <summary>
    Which tiletypes are available. Types added here will be able to be chosen by user when creating new tile.
    </summary>
</member>
<member name="T:DigiTilesClass.DigiTiles">
    <summary>
    Rename this file and change to the correct name.
    </summary>
</member>
<member name="P:DigiTilesClass.DigiTiles.HideInWidgetList">
    <summary>
    Gets a value indicating whether to hide this plugin on the widget list page in the admin area
    </summary>
</member>
<member name="M:DigiTilesClass.DigiTiles.GetWidgetViewComponentName(System.String)">
    <summary>
    Gets a name of a view component for displaying widget
    </summary>
    <param name="widgetZone">Name of the widget zone</param>
    <returns>View component name</returns>
</member>
<member name="M:DigiTilesClass.DigiTiles.GetWidgetZones">
    <summary>
    Gets widget zones where this widget should be rendered
    </summary>
    <returns>Widget zones</returns>
</member>
<member name="M:DigiTilesClass.DigiTiles.Install">
    <summary>
    Install the plugin
    </summary>
</member>
<member name="M:DigiTilesClass.DigiTiles.Uninstall">
    <summary>
    Uninstall the plugin
    </summary>
</member>
```

Figur 40: XML Dokumentasjon DigiTiles

10 REFERANSER

gridstack.js (2020) *gridstack.js*. Tilgjengelig fra: https://gridstackjs.com/ (Hentet: 8. Mai 2022)

Norges Blindeforbund (u. år) *Digital informasjon.* Tilgjengelig fra:

https://www.blindeforbundet.no/universell-utforming/digital-informasjon. (Hentet: 30. April 2022)

npm Docs (u. år) *About npm.* Tilgengelig fra: https://docs.npmjs.com/about-npm (Hentet: 08. Mai 2022)