VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Teemu Jaskari

HYBRIDIAPPLIKAATION LUONTI SIBERIAN CMS-ALUSTALLE

Tekniikka
2021

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Teemu Jaskari
Opinnäytetyön nimi	Hybridiapplikaation luonti Siberian CMS-alustalle
Vuosi	2021
Kieli	suomi
Sivumäärä	39
Ohjaaja	Timo Kankaanpää

Opinnäytetyön tilasi it-alan yritys, jonka tavoitteena oli perehdytysmateriaalin kokoaminen Siberian CMS-alustalle luotavalle moduulille. Opinnäytetyön materiaalin lisäksi luotiin esimerkkisovellus, johon ohjelmoitiin moduuli, jotta perehdytettävä voi opinnäytetyön yhteydessä tutkia valmista ja selkeää moduulia.

Moduuliin luotiin sekä hybridiapplikaatio- että hallintapaneelitoiminnallisuuksia kattamaan mahdollisimman laaja ympäristö, pitäen kuitenkin moduulin selkeänä. Hybridiapplikaatio toteutettiin Ionic Framework sekä AngularJS kehyksillä. Hallintapaneeli toteutettiin HTML:ää sekä JavaScriptiä käyttäen, joiden lisäksi käytössä oli jQuery-kirjasto. Back end on ohjelmoitu PHP:llä.

Opinnäytetyö mahdollistaa tulevaisuudessa sujuvan perehdytyksen yrityksen rekrytoinneissa. Esimerkkisovellukseen voidaan jatkokehittää erilaisia harjoitustehtäviä perehdytettävälle.

Avainsanat AngularJS, PHP, Ionic Framework, Hybridiapplikaatio

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Teemu Jaskari
Title	Creation of Hybrid-Application Module in Siberian CMS
Year	2021
Language	Finnish
Pages	39
Name of Supervisor	Timo Kankaanpää

The purpose of the thesis to write orientation material for creating a hybrid application module for Siberian CMS platform. The thesis was ordered by an IT company for the orientation of new employees. During this thesis, an example application was created with a brief module for learning and practice.

The example module has hybrid application and management panel features that help to understand the structure of Siberian module and how it is programmed. The front end of hybrid application is based on Ionic Framework and AngularJS. The management panel was implemented with HTML and JavaScript with jQuery library. The back end of module was programmed with PHP.

This thesis provides orientation how to program a module for Siberian CMS platform. The example application will be a good and safe place to practice and various exercises will be developed in the future.

Keywords AngularJS, PHP, Ionic framework, hybrid application

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

TERMIT JA LYHENTEET

SAE	Single-application edition, Siberianin alustaversio, jolla voi luoda yhden applikaation
Back end	Applikaation palvelinohjelma
Front end	Käyttäjälle näkyvä sovellus
Backoffice	Siberianin applikaation admin-työkalu
Editor	Siberianin hallintapaneeli
Framework	Ohjelmistokehys
HTML	Hypertext Markup Language, Kieli, jolla käyttöliittymä ohjelmoitu
PHP	Back end ohjelmointikieli
AngularJS	JavaScript kehys
JSON	Javascript Object Notation, tiedostomuoto
Hybridiapplikaatio	Mobiiliapplikaatio, jota pystytään käyttämään myös web-selaimella

1 JOHDANTO

Tämän opinnäytetyön on tilannut IT-alan yritys, joka tarvitsee perehdytysmateriaalin Siberian CMS-alustalla luotavalle hybridiapplikaatiolle sekä hallintatyökaluille. Opinnäytetyön luoman perehdytysmateriaalin lisäksi toteutetaan pieni esimerkkisovellus, jonka tarkoituksena on luoda perehdytettävälle henkilölle kattava ja selkeä ympäristö moduulin tutkimiselle.

Opinnäytetyön tarkoituksena on luoda suppean ohjelmointitaustan omaavalle henkilölle materiaali, jolla hän saa käsityksen, kuinka luodaan ja ohjelmoidaan moduuli Siberian CMS-alustalle, joka sisältää mobiiliominaisuuden sekä hallintapaneelin. Opinnäytetyön ohella luodun esimerkkisovelluksen on tarkoitus antaa perehdytysmateriaaliin tutustuvalle henkilölle selkeä kuvan moduulista sekä tarjota harjoitteluympäristö perehdytysvaiheessa.

Siberianin back end perustuu Zend Framework 1 kehykseen, jonka ohjelmointikieli on PHP, lisäksi Siberian on lisännyt omia kirjastojaan kehyksen päälle helpottamaan ohjelmointityötä. Moduulin hybridiapplikaatio perustuu Ionic ohjelmistokehityspakettiin sekä AngularJS kehykseen.

2 SIBERIAN CMS HYBRIDIAPPLIKAATIO KEHITYKSESSÄ

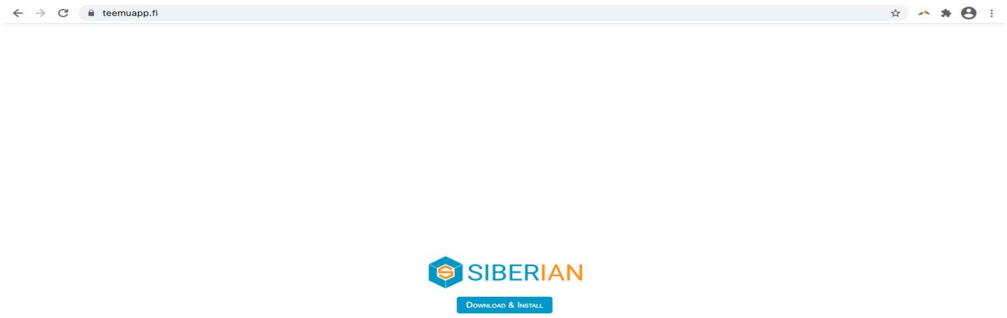
Siberian CMS on avoimen lähdekoodin sisällönhallintajärjestelmä hybridiapplikaatioiden kehittämisessä, ja sen on julkaissut yhtiö nimeltä Xtraball /1/. Ilmaisella avoimen lähdekoodin paketilla voi luoda yhden applikaation. Tämän lisäksi Siberian tarjoaa maksullisia monisovellus- ja alustaversioita. Siberian Single-app Edition (jäljempänä ”**Siberian SAE**”) toimii avoimen lähdekoodin lisenssin ”Open Software License 3.0” alla.

2.1 Palvelin

Malliapplikaatiota varten hankitaan verkkotunnus domainhotelli.fi-palvelusta. Tämän lisäksi malliapplikaatiota varten vuokrataan palvelin OVHCloud-palvelintarjoajalta. Muita samantapaisia palvelintarjoajia löytyy useita erilaisia ja eri hintaluokissa, mutta työssä päädyttiin valitsemaan OVHCloud, koska se oli halpa esimerkkisovellusta varten. Palvelimen verkkopalveluohjauspaneeliksi valittiin Plesk. Myös näitä verkkopalveluohjauspaneeleita löytyy useita vaihtoehtoja kuten cPanel, Virtualmin, WordPress ja VestaCP. Plesk valittiin, koska yrityksessä on käytetty kyseistä ohjelmistoa aikaisemmin. Pleskistä luotiin verkkotunnus, joka aikaisemmin hankittiin, jonka käyttöönotto oli erittäin helppoa Pleskistä.

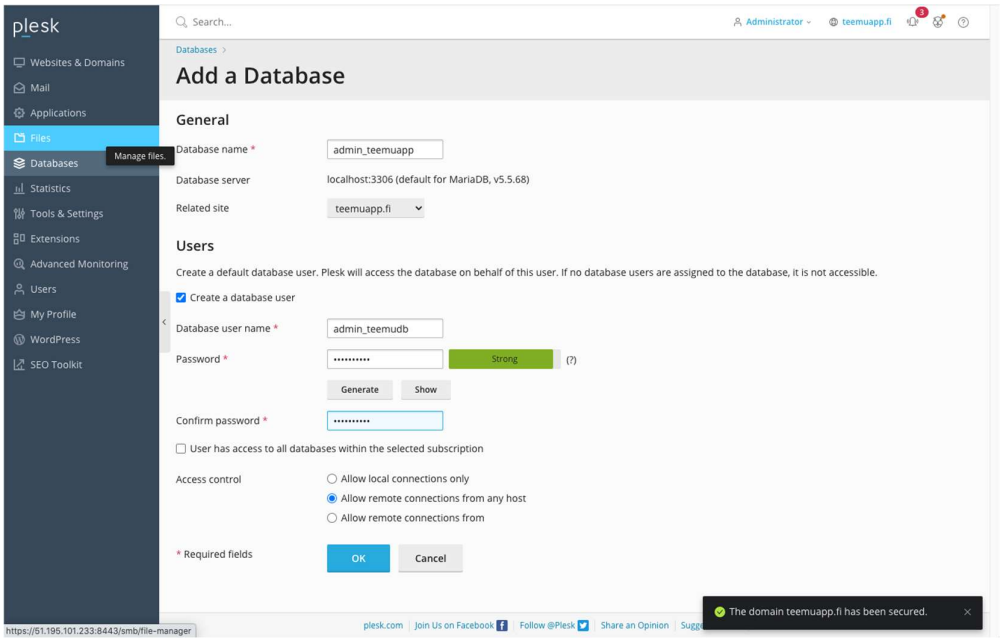
2.2 Siberian SAE

Verkkotunnuksen ja palvelimen aktivoinnin jälkeen pystyy Pleskin kautta asentamaan Siberian SAE:n, jonka asennustiedoston voi ladata ilmaiseksi osoitteesta https://www.siberiancms.com/download/. Tältä sivulta ladataan zip-tiedosto, joka sisältää index.php tiedoston, joka ladataan Pleskin File Managerin httpdocs kansioon ja siirrytään selaimella verkkotunnuksen osoitteeseen, josta Siberianin sivulta ladattu tiedosto aloittaa palvelimelle Siberian SAE:n asentamisen (**Kuvio 1.**).



Kuvio 1. Siberian SAE asennus.

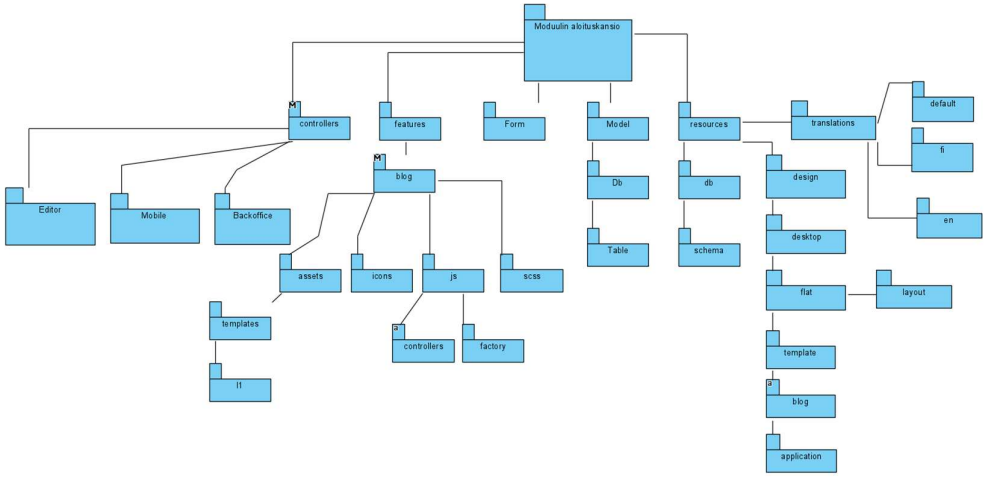
Nappia painamalla aloitetaan asennus ja käyttöehtojen hyväksynnän jälkeen asennus vaati tietokantayhteyden määrittämisen. Tietokanta luodaan Pleskistä kohdasta ”Add a Database”, johon määritettiin tarvittavat tiedot (**Kuvio 2**.)



Kuvio 2. Tietokannan määritys.

Tietokanta yhteyden määrittämisen jälkeen Siberian SAE aloittaa asennusprosessin, jonka jälkeen määritetään admin-käyttäjän tunnukset sekä applikaation nimi. Asennus on nyt valmis ja kirjautuminen onnistuu Siberian SAE:n backoffice:en luoduilla tunnuksilla osoitteessa verkkotunnus/backoffice.

2.3 Moduulin rakenne



Kuvio 3. Moduulin perusrakenne

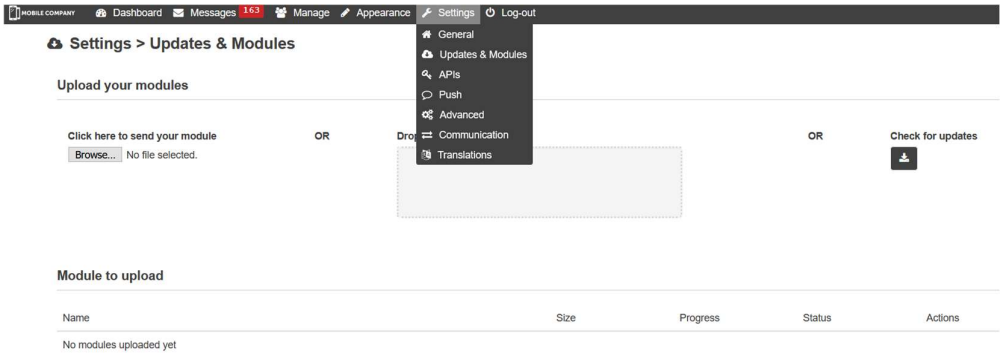
Kuviossa 3 on Siberianiin luotavan moduulin kansiorakenne, jossa controllers-kansiossa on back end PHP kontrolleritiedostot. Features-kansiossa on mobiiliapplikaation front end tiedostot alakansioissaan, ja resources-kansiossa on moduulin tietokantataulut luovat tiedostot sekä hallintapaneelin käyttöliittymän tiedostot. Tämän lisäksi tiedostopuun juuressa on init.php tiedosto, jonka avulla voidaan määrittää esimerkiksi Siberianin hallintapaneelin sivupalkkiin moduulin hallintatyökaluun johtava linkitys, josta lisää kappaleessa 4.3, ja package.json tiedosto, jossa on määritettynä moduulin tietoja, (**Kuvio 4**.) kuten moduulin nimi, kuvaus, tyyppi, moduulin versionumero sekä moduulin riippuvuudet.

```
{
  "name": "Blog",
  "description": "Blog module",
  "type": "module",
  "version": "0.0.5",
  "dependencies": {
    "system": {
      "type": "SAE",
      "version": "4.16.0"
    }
  }
}
```

Kuvio 4. package.json tiedosto.

2.4 Moduulin asentaminen alustalle

Siberianin alustalle voidaan asentaa moduuleita ja tämä tapahtuu Siberianin backoffice hallintapaneelista välilehdeltä Settings > Updates & Modules. Moduulin asentamiseksi alustalle moduulin kansiorakenne pakataan zip-tiedosto muotoon moduulin juurikansiosta, jonka jälkeen se ladataan alustalle (**Kuvio 5.**).



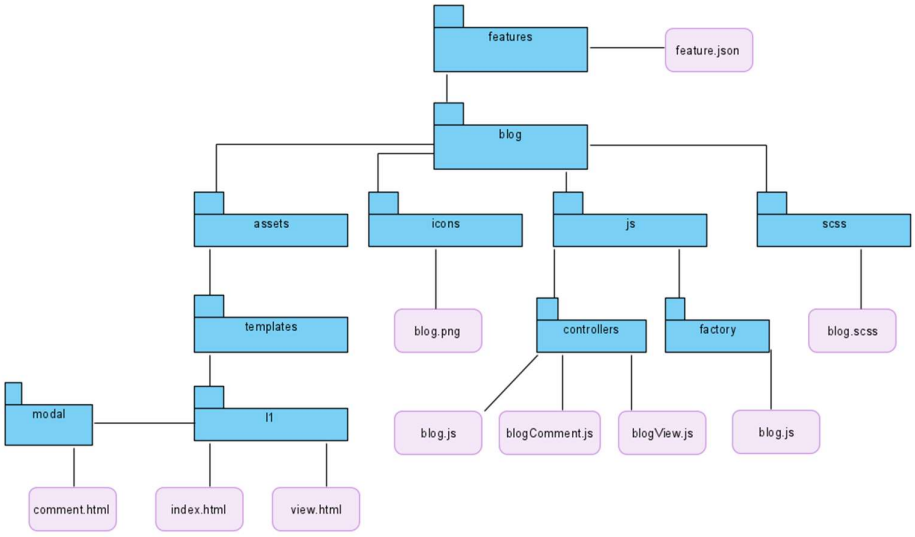
Kuvio 5. Moduulin asennus.

3 MOBIILI APPLIKAATIO

Siberian SAE:n mobiilisovellus perustuu Ionic sovelluskehykseen v1.0 sekä AngularJS v1.3, joten mobiilin front end on HTML sekä javascript koodia. Mobiilissa on myös Cordova laajennuksia ja näitä kaikkia sovelluskehyksiä on räätälöity Xtraballin toimesta. /2/

3.1 Mobiilin front end-rakenne

Siberianissa mobiilin front end tiedostot ovat moduulin juuresta alkavassa features kansiossa. (**Kuvio 6.**). Kansion features sisällä on moduulin niminen kansio, jonka sisälle jaotellaan eri alakansiot, näin pysyy selvä kansiorakenne. Alakansiot ovat selitettynä taulukossa 1.



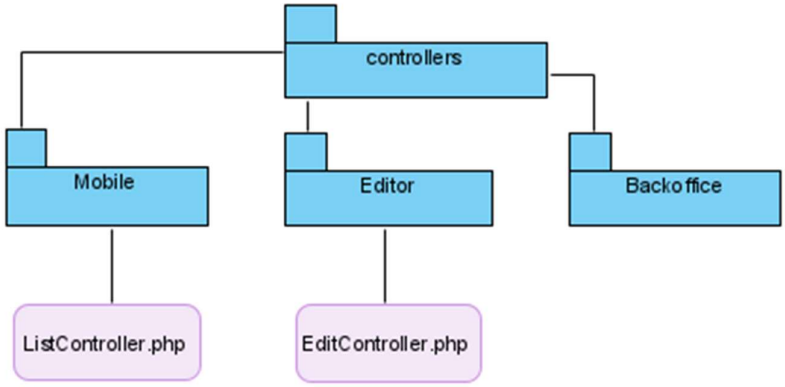
Kuvio 6. Mobiilin kansiorakenne.

Taulukko 1. Mobiiliapplikaatiorakenne

assets/templates/l1	Tähän kansioon sijoitetaan mobiili applikaation HTML-tiedostot
icons	Tähän kansioon sijoitetaan mobiiliapplikaatiossa käytettävät kuvat sekä kuvakkeet.
js	Tässä kansiossa on mobiiliapplikaation javascript tiedostot eri alakansiossa kuten ”controllers”, johon sijoitetaan AngularJS kontrollerit ja ”factory” johon sijoitetaan AngularJS factory-palvelu tiedostot. Tähän ”controllers” kansioon myös voi tehdä alakansiota, kuten ”directives” mikäli mobiiliapplikaatiossa tarvitaan direktiivejä.
scss	Tähän kansioon luodaan mobiiliapplikaatiossa käytettävät tyylit scss tai css tiedostomuodossa.

3.2 Moduulin back end-rakenne

Moduulin juuressa olevaan ”controllers” kansioon tulevat moduulin back end PHP luokat. Tämän lisäksi kansioon luodaan alakansioita, joilla saadaan selvyyttä mihin osaan moduulista kyseinen luokka kuuluu. (**Kuvio 7.**)



Kuvio 7. Moduulin back end kansiorakenne.

Mobile-kansiossa sijaitsevaan Blog_Mobile_ListController PHP-luokkaan luodaan julkiseksi määriteltyjä funktiota, joita kutsutaan mobiiliapplikaatiossa AngularJS factorysta. Funktiosta palautetaan aina paluusanoma JSON-objektina, jonka factory palauttaa AngularJS kontrollerille, jolloin funktiosta palautunutta tietoa voidaan käyttää käyttöliittymässä.

3.3 Mobiiliapplikaation feature.json

Moduulin mobiiliapplikaation perustietoja sekä tiedostoja ja niiden polkuja määritetään feature.json tiedostolla (**Kuvio 8**.). Tässä tiedostossa routes-taulukkoon määritetään applikaatiossa käytettävät eri tilat JSON-objekteina (**Taulukko 2.**).

```
{
    "name": "Blog",
    "code": "blog",
    "version": "0.0.1",
    "category": "integration",
    "model": "Blog_Model_Blog",
    "desktop_uri": "blog/application/",
    "routes": [
        {
            "root": true,
            "state": "blog-index",
            "controller": "BlogIndexController",
            "url": "blog/index/",
            "template": "l1/index.html",
            "cache": false
        },
        {
            "root": false,
            "state": "blog-view",
            "controller": "BlogViewController",
            "url": "blog/mobile_view",
            "template": "l1/view.html",
            "cache": false
        }
    ],
    "icons": [
        "icons/blog.png"
    ],
    "files": [
        "js/controllers/blog.js",
        "js/controllers/blogView.js",
        "js/controllers/blogComment.js",
        "js/factory/blog.js",
        "scss/blog.scss"
    ],
    "is_ajax": true,
    "compile": true,
    "use_account": false,
    "only_once": false
}
```

Kuvio 8. feature.json tiedosto

Taulukko 2. Tilan json-objekti

Tietueen nimi	**Tietueen määritelmä**	**Esimerkki arvo**
root	Moduulin aloitus tila.	Tosi tai epätosi
state	Tilan nimi	blog-index
controller	AngularJS kontrollerin nimi	BlogIndexController
url	Tilan verkko-osoite	blog/index/
template	HTML-tiedoston sijainti	l1/index.html
cache	Välimuisti	Tosi tai epätosi

Tähän JSON-objektiin määritetään root arvolla tosi tai epätosi onko kyseinen tila moduulin aloitustila. Tilojen välillä navigoidaan objektissa annetulla state parametrilla lähettämällä tämä arvo merkkijonona AngularJS kehyksen $state.go() funktioon. Tilalle määritetään oma AngularJS kontrolleri, joka ohjaa tilan toimintaa. Tilan verkko-osoite määritetään url-kenttään, johon voi lisäksi antaa parametrejä, joita tässä esimerkissä ei ole käytetty.

Tilassa käytettävän HTML-tiedoston sijainti määritetään template-kenttään templates kansiosta alaspäin. Viimeisenä JSON-objektiin määritetään cache arvolla tosi tai epätosi, jolla määritämme, tallennetaanko tila välimuistiin, jolloin siihen palatessa ei ajeta lataus funktiota uudelleen. Lisäksi JSON-objektiin asetetaan files-taulukko, johon määritetään moduulissa käytettävien tiedostojen kuten kontrollereiden polut.

3.4 Tiedon noutaminen tietokannasta

Tilaan tullessa ladataan tilan AngularJS kontrolleri, johon voidaan tehdä funktio, joka ajetaan kontrollerin käynnistys vaiheessa ja tässä funktiossa kutsumme moduulin factorya. LoadContent funktiossa kutsutaan Blog-nimisestä factorysta funktiota, jolta odotetaan vastauksena payload-objektia, joka sisältää PHP:lta palautetun JSON-objektin. Kun loadPosts-funktiota kutsutaan, se ajaa factoryssa olevan funktion, joka ottaa yhteyden $pwaRequestin get funktiolla palvelimelle ja palauttaa PHP luokalta tulevan vastauksen (**Kuvio 9.**).

```
//Blogi kirjoitusten lataus funktio
$scope.loadContent = function () {
    //Nollataan lataus iconin teksti ja näytetään se
    Loader.last_config = null;
    Loader.show();
    //Kutsutaan factory funktiota joka palauttaa payload objektin
    Blog.loadPosts()
        .success(function (payload) {
        //Asetetaan $scope muuttujaan tulleet blogi kirjoitukset
        $scope.posts = payload.posts;
        //Ilmoitetaan ionic päivitys funktiolle päivityksen olevan valmis
        $scope.$broadcast('scroll.refreshComplete');
    }).error(function(error) {
        //Logataan funktiosta tullut virhe
        console.log(error);
    }).finally(function() {
        //Piilotetaan lataus iconi
        Loader.hide();
    });
};
```

Kuvio 9. AngularJS blogikirjoitusten latausfunktio.

```
factory.loadPosts = function () {
    return $pwaRequest.get('blog/mobile_list/load-posts', {
        cache: false
    });
};
```

Kuvio 10. Funktio, joka kutsuu back end funktiota.

Merkkijonoksi asetetaan luokan nimi, joka tässä tilanteessa on Blog_Mobile_ListController ja tästä luokasta kutsutaan julkista funktiota nimeltä loadPostsAction. $pwaRequest get kutsuun tulee aina luokan nimen jälkeen funktion nimi ilman Action päätettä sekä isot kirjaimet funktion nimessä korvataan väliviivalla, jonka jälkeen tulee pienikirjain (**Kuvio 10.**).

```php
public function loadPostsAction()
{
    try {
        //Haetaan kaikki kirjoitukset tietokannasta
        $posts = (new Blog_Model_Post())->findAll();
        //Alustetaan palautus taulukko
        $returnData = [];

        //Pyöritetään läpi kaikki
        foreach($posts as $post){
            //Haetaan tietokannasta kaikki kommentit jotka liittyvät tähän kirjoitukseen
            //kirjoituksen tietokanta Id:llä
            $comments = (new Blog_Model_Comment())->findAll(["post_id" => $post->getId()]);
            //Alustetaan kommentti taulukko
            $commentsArray = [];
            foreach($comments as $comment){
                //Kerätään kommentti objektista kaikki tieto getData funktiolla
                $commentsArray[] = $comment->getData();
            }
            //Kerätään tiedot palautus taulukkoon
            $returnData[] = ['title' => $post->getTitle(),
                             'post_id' => $post->getId(),
                             'text' => nl2br($post->getText()),
                             'created_at' => $post->getCreatedAt(),
                             'comments' => $commentsArray,
                            ];
        }
        //Alustetaan payload taulukko
        $payload = [
            "success" => true,
            'posts' => $returnData,
        ];
    } catch (Exception $e) {
        $payload = [
            "error" => true,
            "message" => $e->getMessage()
        ];
    }
    //Palautetaan payload taulukko json muodossa
    $this->_sendJson($payload);
}
```

Kuvio 11. Back end PHP-funktio kaikkien kirjoitusten hakuun.

PHP-funktiot luokissa tehdään try-catch rakenteen sisälle, jolloin vikatilanteissa lähetetään virheviesti takaisin. PHP-funktiota luokkaan tehdessä funktion nimen loppuun kirjoitetaan Action Zend-kehyksen routeria varten (**Kuvio 11.**).

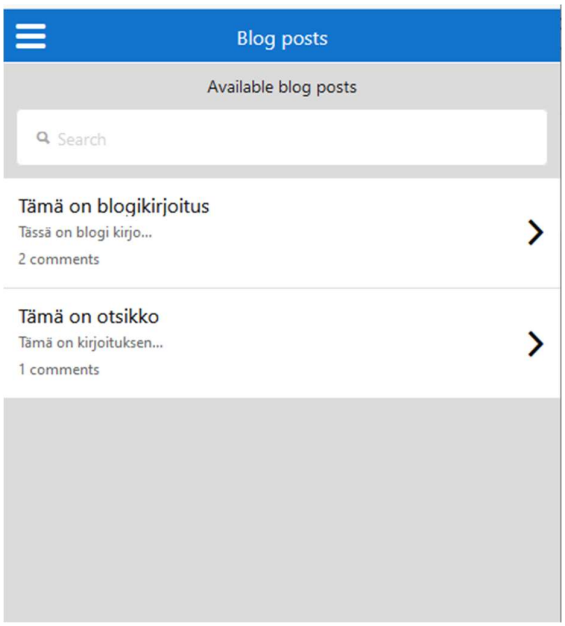
PHP-funktiossa oliosta voidaan kutsua Siberianin ydinfunktioita, kuten find, findAll ja getData, jotka periytyvät Siberianin ydinluokalta ja nämä käyttävät Zend-kehyksen funktioita. Nämä funktiot palauttavat tietokannasta noudettua tietoa. Funktioon getData voidaan myös antaa parametrinä tietokantataulun kolumnin nimi, jolloin se palauttaa vain tämän kolumnin tiedot.

3.5 Käyttäjän näkymä

HTML-koodilla näkymää (**Kuvio 13.**) tulostettaessa voidaan käyttää kontrollerissa scope-muuttujan sisälle tallennettua tietoa, joka tietokannasta haettiin. Näkymää tulostettaessa ei muuttujan eteen mainita ”$scope” alkua vaan kaikkea näkymässä käytettyä tietoa haetaan $scope-muuttujan sisältä (**Kuvio 12.**).

```
<!-- Määritetään ionic view -->
<ion-view sb-page-background class="module-blog">
    <!-- Asetetaan $scope.title:ssa oleva määritetty
    teksti näkymän yläreunaan otsikkopalkkiin -->
    <ion-nav-title>{{ title }}</ion-nav-title>
    <!-- Näkymän sisältö alkaa -->
    <ion-content>
        <!-- Luodaan Ionic:in alasveto päivitys
        joka aktivoi doRefresh funktion -->
        <ion-refresher on-refresh="doRefresh()"
                       pulling-text="{{ 'Pull down to refresh' | translate:'blog' }}"
                       refreshing-text="{{ 'Updating..' | translate:'blog' }}">
        </ion-refresher>
        <div style="width:100%">
            <div class="text-center">
                <h5>
                    {{ "Available blog posts" | translate:"blog" }}
                </h5>
            </div>
        </div>
        <!-- Haku joka suodattaa blogi kirjoituksista ne joissa on annettu parametri -->
        <label class="item item-input borderadius5" style="margin: 10px;">
            <i class="icon ion-search placeholder-icon"></i>
            <input type="text" ng-model="input.filter" placeholder='{{ "Search" | translate:"blog" }}'>
        </label>
        <div class="list">
            <!-- Toistetaan AngularJS:llä blogi kirjoituksia ja tulostetaan
            niistä jokaisesta oma kohde listaan ja annetaan sille klikkauksesta
            aktivoituva funktio goToPost johon parametrinä lähetään koko blogi kirjoitus-->
            <div ng-repeat="post in posts |filter:input.filter" ng-click="goToPost(post)">
                <a class="item item-icon-right">
                    <!-- Tulostetaan kirjoituksen otsikko -->
                    <h2>{{ post.title }}</h2>
                    <!-- Tulostetaan paragrafiin kirjoituksen teksti mutta vain 20 merkkiin asti -->
                    <p>{{ post.text | limitTo: 20 }}{{ post.text.length > 20 ? '...' : '' }}</p>
                    <!-- Tulostetaan kkirjoituksen kommenttien lukumäärä -->
                    <p>{{ post.comments.length }} {{ "comments" | translate:"blog" }}</p>
                    <i class="icon ion-chevron-right"></i>
                </a>
            </div>
        </div>
    </ion-content>
</ion-view>
```

Kuvio 12. Aloitusnäkymän HTML-koodi



Kuvio 13. Näkymä käyttäjälle aloitussivulla.

3.6 Tilanäkymän vaihto

Näkymään tulostetaan laatikko jokaiselle tietokannassa olevalle blogikirjoitukselle ja näihin liitetään ng-click attribuutti (**Kuvio 14.**), jota klikkaamalla ajetaan tilanvaihto funktio. Tähän funktioon lähetetään parametrinä klikattu blogikirjoitus, joka voidaan tallentaa Blog factory muuttujaan (**Kuvio 15.**), jolloin sen siirtäminen seuraavaan tilaan onnistuu.

```
<div ng-repeat="post in posts |filter:input.filter" ng-click="goToPost(post)">
```

Kuvio 14. AngularJS toistofunktio ja ng-click attribuutti

```
36      $scope.goToPost = function (post) {
37          //Asetetaan klikattu post-objekti factory:n muuttujaan
38          Blog.postToView = post;
39          //Siirrytään blog-view tilaan
40          $state.go("blog-view");
41      };
```

Kuvio 15. Tilanvaihtofunktio

Kun $state:n go-funktiolla siirrytään toiseen tilaan, on uuden tilan feature.json tiedostossa asetettu state-parametri lähetettävä merkkijonona funktiolle. Uudessa tilassa otetaan $scope-muuttujaan aikaisemmin factoryn muuttujaan asetettu blogikirjoitus, jolloin uudessa tilassa voidaan tulostaa näkymä tästä kirjoituksesta ilman, että tarvitsee käydä tietokannassa noutamassa kirjoitusta uudelleen.

3.7 Ionic modal

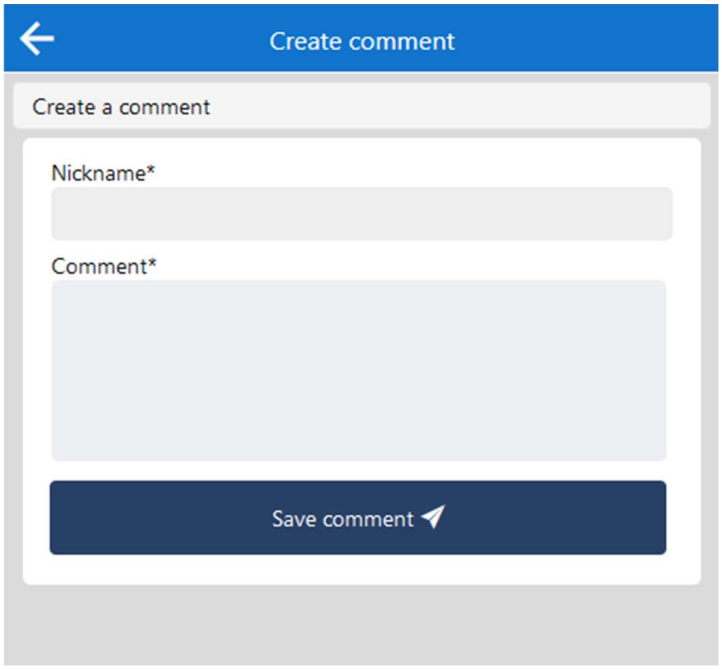
Ionicissa on mahdollisuus avata myös näkymä modal-ikkunassa, jolloin ei tarvitse siirtyä uuteen tilaan mahdollistaakseen uuden näkymän käyttäjälle. Modal-ikkunalle luodaan oma AngularJS kontrolleri sekä applikaatio puolen HTML-tiedostojen kansioon modal kansio, johon tehdään modal-ikkunan HTML-tiedosto. Modal-ikkunaa varten luodaan factory-tiedostoon funktio (**Kuvio 16**.), jolla ladataan modal-ikkunan HTML-tiedosto ja tuloksena saadaan kommentointinäkymä (**Kuvio 18**.).

```
40      factory.commentModal = function () {
41          Modal.fromTemplateUrl("features/blog/assets/templates/l1/modal/comment.html", {
42              //Luodaan $scope:en funktio jolla modal-ikkuna suljetaan
43              scope: angular.extend($rootScope.$new(true), {
44                  close: function () {
45                      factory._commentModal.hide();
46                  }
47              }),
48              //Lisätään liukumis animaatio oikealta vasemmalle
49              animation: "slide-in-right-left"
50          }).then(function (modal) {
51              //Tallennetaan modal factory muuttujaan
52              factory._commentModal = modal;
53              //Näytetään modal
54              factory._commentModal.show();
55              return modal;
56          });
57      };
58      return factory;
59  });
```

Kuvio 16. Kommentti modal-ikkunan avausfunktio

```
<ion-modal-view sb-page-background
                ng-controller="BlogCommentController"
                class="blog-comment-l1">
    <ion-header-bar class="bar-custom">
        <div class="buttons">
            <button class="button button-clear"
                    ng-click="close()">
                <i class="icon ion-android-arrow-back"></i>
            </button>
        </div>
        <h1 class="title">{{ pageTitle }}</h1>
    </ion-header-bar>
    <ion-content>
        <div class="container">
            <div class="item item-divider item-divider-custom margin-5 borderadius5">
                <h3 class="title forceColor">{{ "Create a comment" | translate:"blog" }}
                </h3>
            </div>
            <div class="customBG borderadius5">
                <p class="customP">
                    {{ 'Nickname' | translate:'blog' }}*
                </p>
                <input type="text" class="customInput borderadius5" ng-model="formData.sender">
                <p class="customP">
                    {{ 'Comment' | translate:'blog'}}*
                </p>
                <textarea type="text" class="text-area borderadius5" ng-model="formData.comment" rows="6">
                </textarea>
                <div class="buttons">
                    <div style="margin: 2vh; width:100%;">
                        <button class="button customButton" ng-click="postComment()">
                            {{ "Save comment" | translate:"blog" }}
                            <i class="icon ion-ios-paperplane"></i>
                        </button>
                    </div>
                </div>
            </div>
        </div>
    </ion-content>
</ion-modal-view>
```

Kuvio 17. Kommentointi modal-ikkunan HTML-koodi.



Kuvio 18. Kommentointinäkymä

3.8 Tiedon tallentaminen tietokantaan

Tietoa tallentaessa on hyvä kerätä kaikki tarvittava tieto scopen objektiin, joka nimetään formDataksi. Tähän muuttujaan voimme asettaa tietoa näkymästä antamalla syötteelle ng-model attribuutti (**Kuvio 17.**). Käyttäjän painaessa tallennusnappia kutsutaan ng-click attribuutilla kontrollerissa olevaa tallennusfunktiota (**Kuvio 19.**).

```
$scope.postComment = function() {
    //Näytetään lataus spinneri
    Loader.last_config = null;
    Loader.show();
    //Kutsutaan Blog factoryn funktiota johon asetetaan kerätty tieto
    Blog.postComment($scope.formData)
        .success(function(data){
        if(data.success) {
            Dialog.alert($translate.instant("Success", "blog"),
                         $translate.instant("Comment saved successfully", "blog"),
                         $translate.instant("OK"));

            //Asetetaan tallennettu kommentti kirjoituksen kommentti taulukkoon
            Blog.postToView.comments.push(data.comment);
            //Modal suljetaan
            $scope.close();
        }
    }).error(function(data) {
        //Funktion vikatilanteessa näytetään vika dialogi
        if(data && angular.isDefined(data.message)) {
            Dialog.alert($translate.instant("Error!", "blog"), data.message, $translate.instant("OK"));
        }
    }).finally(function() {
        $scope.isLoading = false;
        //Piilotetaan lataus spinneri
        Loader.hide();
    });
};
```

Kuvio 19. AngularJS kommentin tallennusfunktio

Funktiossa kutsumme Blog factory:n postComment funktiota, jolle annetaan formData-objekti, jossa on tallennettuna käyttäjän syöttämät tiedot. Funktiolta postComment odotetaan vastausta ja funktion ajon onnistuessa asetetaan tallennettu kommentti blogikirjoituksen kommenttitaulukkoon.

```
factory.postComment = function (data) {
    return $http({
        method: "POST",
        url: Url.get("blog/mobile_list/post-comment"),
        data: data,
        cache: false
    });
};
```

Kuvio 20. Kommentin tallennusfunktion kutsu factorysta

Tässä funktiossa kutsutaan AngularJS:n $http palvelulla (**Kuvio 20.**), jolla kommunikoidaan palvelimen kanssa. Tähän palveluun annetaan JSON-objekti, johon asetetaan metodi, back end funktion osoite, tiedot, sekä tallennetaanko funktiolta palautetut tiedot välimuistiin.

```
public function postCommentAction()
{
    try {
        //Haetaan factorystä lähetetty data
        $request = $this->getRequest();
        $data = $request->getBodyParams();

        //Tarkistetaan käyttäjän syöttämät tiedot
        if(!$data['sender'] || !$data['comment']){
            throw new \Siberian\Exception(p__('blog', 'Please provide required data!'));
        }
        //Haetaan kirjoitus johon kommentti on kirjoitettu
        $post = (new Blog_Model_Post())->find($data['post_id']);
        //Tarkistetaan onko kirjoitus vielä tietokannassa
        if(!$post->getId()){
            throw new \Siberian\Exception(p__('blog', 'Post not found!'));
        }
        //Luodaa kommentti luokasta uusi olio
        $newComment = new Blog_Model_Comment();
        //Asetetaan olioon syötetyt tiedot ja tallennetaan tietokantaan
        $newComment->setData($data)->save();
        //Kerätään palautettava payload
        $payload = [
            "success" => true,
            'comment' => $newComment->getData(),
        ];

    } catch (Exception $e) {
        $payload = [
            "error" => true,
            "message" => $e->getMessage()
        ];
    }
    //Palautetaan payload JSON-objektina
    $this->_sendJson($payload);
}
```

Kuvio 21. Kommentin tietokantaan tallennusfunktio back end:issa

Funktion alussa annetut syötteet tarkastetaan, jonka jälkeen haetaan kyseinen kirjoitus tietokannasta ja tarkistetaan sen olemassaolo. Tiedon tallentamiseksi luomme

uuden olion kommenttiluokasta, johon voidaan asettaa tieto Siberianin ydinluokasta periytyvällä setData funktiolla. Siberianin tallennusfunktiolla save tallennetaan tietokantaan objektin tiedot, joiden avain vastaa tietokantataulun kolumnin nimeä. (**Kuvio 21.**)

4 EDITOR-HALLINTAPANEELI

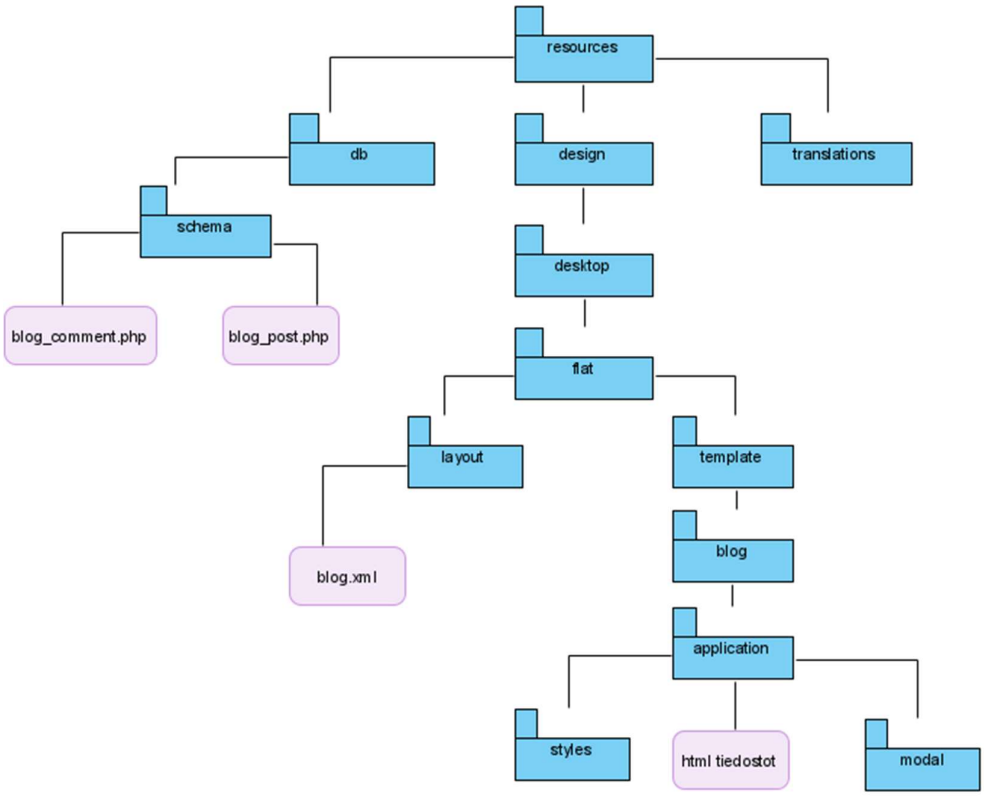
Siberianin hallintapaneelia käytetään mobiilin asetusten määrittämisessä, johon tarvitaan tunnukset ja ne luodaan backofficesta. Hallintapaneeliin voidaan tehdä erilaisia osioita, joista pystyy hallita mobiilissa olevan moduulin asetuksia tai tietoja riippuen moduulin luonteesta. Hallintapaneeli yleensä annetaan käyttöön asiakkaalle, joka hallinnoi luotua applikaatiota, kun taas backoffice on palveluntuottajan hallintatyökalu koko applikaatioon ja sen ydinasetuksiin.

Editorissa on erilaisia ominaisuuksia applikaation ulkoasun muokkaamiseen sekä sivu, josta valitaan applikaatiossa toimivat moduulit. Editorista löytyy myös paljon muita ominaisuuksia, joita tässä opinnäytetyössä ei käydä läpi, koska opinnäytetyön tarkoituksena on tutustuttaa lukijansa moduulin ohjelmointiin.

Editor hallintapaneelin käyttöliittymässä käytetään HTML-koodia, jonka apuna on Bootstrap 3.2.0 CSS-kehys sekä jQuery-kirjasto, joka on avoimen lähdekoodin JavaScript-kirjasto. Näiden lisäksi Editor hallintapaneelista löytyy myös useita muita Siberianin valitsemia lisäosia kuten SweetAlert, CKEditor, Chart.js ja DataTables. Hallintapaneelin back end on mobiilin tapaan PHP:tä, jossa on Zend Framework 1-kehys, johon on lisätty Siberianin omia toiminnallisuuksia.

4.1 Editor-hallintapaneelin kansiorakenne

Hallintapaneeliin tarvittavat tiedostot sijoitetaan resources-kansion alle (**Kuvio 22.**) jonka alta löytyvät myös tietokantarakenteen skeema sekä käännöskansiot. Hallintapaneelin kansiorakenteen viimeiseen kansioon, application, voidaan luoda useita alakansiota tiedostojen hallinnan parantamiseksi, mikäli moduuliin liittyy huomattava määrä tiedostoja hallintapaneelia ohjelmoidessa.

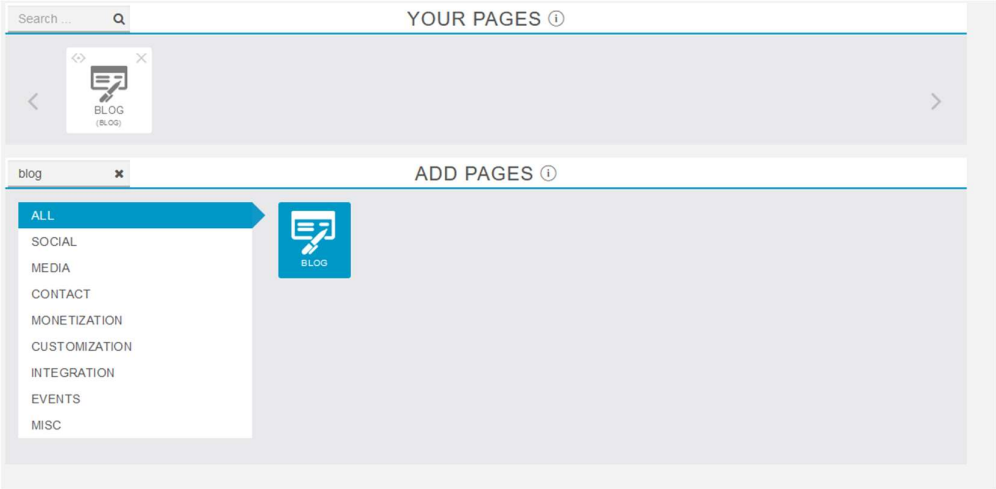


Kuvio 22. Editor hallintapaneelin kansiorakenne

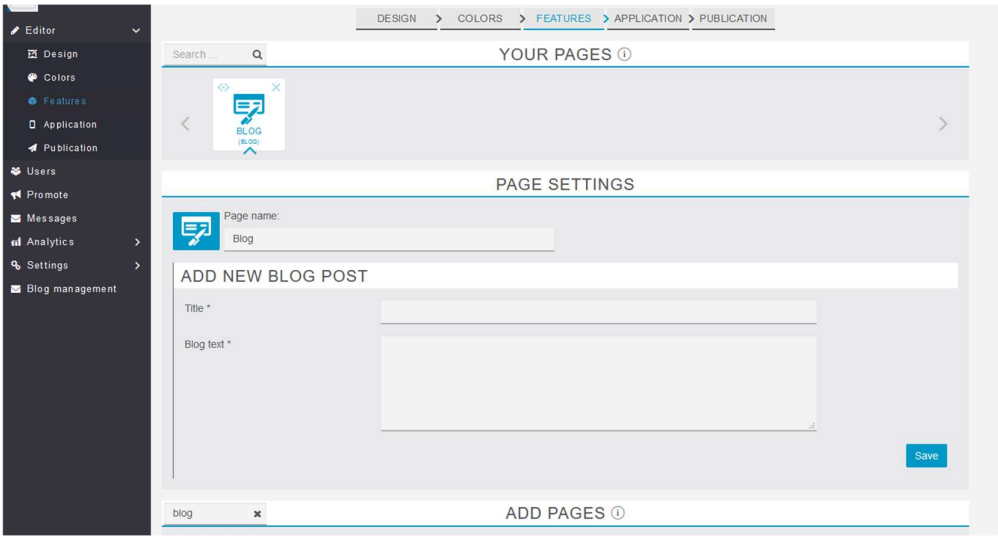
4.2 Feature-välilehden ohjelmointi

Applikaation moduulia ohjelmoimaan lähdettäessä tulee tehdä päätös, millainen hallintapaneeli kyseiselle moduulille tarvitaan. Yksi vaihtoehdoista on feature-välilehdellä sijaitseva applikaation moduulin lisäyssivu, kun moduuli tarvitsee vain vähäisen hallinnointiohjelman. Tätä voidaan hyödyntää, kuten tässä esimerkissä on blogikirjoituksen luonti ohjelmoitu kyseiseen paikkaan.

Kun applikaatioon on lisättynä haluttu moduuli, näkyy se editorissa ”features”-sivulla ”Your Pages”-valikossa (**Kuvio 23.**). Moduulia klikkaamalla voidaan avata moduuliin ohjelmoitu hallinnointiohjelma (**Kuvio 24.**).



Kuvio 23. Moduulin lisäys applikaatioon



Kuvio 24. Moduulin ohjelmoitu blogikirjoituksen julkaisu

Kuviossa 24 nähtävän ”Page Settings” sivun voi ohjelmoida pelkällä HTML ja jQuery javascript kirjastolla sekä myös Zend-sovelluskehyksestä löytyvällä Form-luokalla, jonka Siberian on räätälöinyt. Tässä esimerkissä luodaan lomake edellä mainitulla Form-luokalla. Form-luokka luodaan moduulin kansiorakenteen juureen kansioon nimeltä Form, jonka sisään luodaan PHP-luokka. Tässä PHP luokassa on julkinen funktio nimeltä init() jossa luodaan lomake (**Kuvio 25.**). Lomakkeelle annetaan action-attribuutti, jolla määritetään mihin PHP-funktioon lomake lähetetään.

```php
<?php
/**
 * Class Blog_Form_Blog
 */
class Blog_Form_Blog extends Siberian_Form_Abstract
{
    public function init()
    {
        //Alustetaan lomake
        parent::init();
        //Luodaan lomakkeelle form jonka toiminnaksi annetaan
        //funktion osoite joka tässä tilanteessa on saveAction
        //Blog_ApplicationControllerissa
        $this
            ->setAction(__path("/blog/application/save"))
            ->setAttrib("id", "form-blog");
        self::addClass("create", $this);
        //Luodaan lomakkeeseen syöte otsikolle
        $name = $this->addSimpleText("title", p__("blog", "Title"));
        //Asetetaan syöte pakolliseksi
        $name->setRequired(true);
        //Luodaan lomakkeeseen tekstialue blogi kirjoitukselle
        $text = $this->addSimpleTextarea("text", p__("blog", "Blog text"));
        $text->setRequired(true);
        //Luodaan tallennus nappi
        $this->addSubmit(p__("blog", "Save"))
            ->addClass("default_button")
            ->addClass("pull-right")
            ->addClass("submit_button");
    }

}
```

Kuvio 25. Lomake

Lomaketta varten luodaan PHTML-tiedosto application kansioon, joka määritetään layout-kansiossa olevassa XML-tiedostossa (**Kuvio 26.**), jonka kautta oikea PHTML-tiedosto tulostuu käyttäjälle. PHTML-tiedostossa voidaan luoda lomake luokasta olio, joka tulostetaan div-elementin sisään (**Kuvio 27.**).

```xml
    <blog_application_edit>
        <views>
            <content class="application_view_customization_features_edit_tabbareditor"
                     template="application/customization/features/edit/tabbar_editor.phtml" />
            <content_editor class="core_view_default"
                            template="blog/application/edit.phtml" />
        </views>
    </blog_application_edit>
```

Kuvio 26. XML-tiedostoon lisättävä osuus

```php
<?php
    $form = new Blog_Form_Blog();
?>
<div id="blog" class="">
    <div class="feature-block-add">
        <h3 class="title-editor no-border-radius title-feature-indent">
            <?php echo p__("blog", "Add new blog post"); ?>
        </h3>
        <div class="container-fluid subcontent content-feature">
            <?php echo $form; ?>
        </div>
    </div>
</div>
```

Kuvio 27. HTML-koodi lomakkeelle

Lomakkeen tallennusnappia painaessa lomakkeen tiedot lähetetään back end funktiolle, jossa uusi blogikirjoitus tallennetaan tietokantaan. Tallennusfunktiossa tiedot tarkistetaan, jonka jälkeen ne tallennetaan tietokantataulun kolumniin, johon tiedon avain täsmää. (**Kuvio 28.**).

```php
public function saveAction(){
    try {
        //Kerätään lomakkeen tiedot muuttujaan
        $request = $this->getRequest();
        $data = $request->getPost();
        //Alustetaan lomake olio
        $form = new Blog_Form_Blog();
        //Tarkistetaan syötetyt tiedot
        if ($form->isValid($data)) {
            //Alustetaan blogikirjoitus olio
            $blogPost = new Blog_Model_Post();
            //Asetetaan ja tallennetaan annetut tiedot tietokantaan
            $blogPost->setData($data)->save();
            //Kerätään palautettava payload
            $payload = [
                "success" => true,
                "message" => $blogPost->getTitle() . " " . p__("blog", "Succesfully created."),
            ];
        }else {
            //Jos tietojen vahvistus ei onnistu,
            //palautetaan virhe
            $payload = [
                "error" => true,
                "message" => $form->getTextErrors(),
                "errors" => $form->getTextErrors(true)
            ];
        }
    } catch (Exception $e) {
        $payload = [
            "error" => true,
            "message" => $e->getMessage(),
        ];
    }
    $this->_sendJson($payload);
}
```

Kuvio 28. Blogikirjoituksen tallennusfunktio

4.3 Sivupalkkiin kiinnitettävä moduuli

Mikäli moduulin hallinnointiin tarvitaan suurempi tila kuin feature-välilehdellä oleva tila, voidaan editoriin myös ohjelmoida sivupalkkiin osio, jolloin ohjelmoitu sivu aukeaa koko ikkunan kokoiseksi pois lukien sivupalkki. Tämä helpottaa tilannetta, kun moduulin hallinnoinnissa on useampia eri toiminallisuuksia. Tässä esimerkkisovelluksessa on blogikirjoitusten tarkastelu- sekä poisto-ominaisuus toteutettu sivupalkista aukeavalle hallintasivulle.

Sivupalkin alustamiseksi init.php tiedostoon moduulin juurikansiossa luodaan funktio, jota kutsutaan kuuntelemalla Siberianin hookkia nimeltä ”editor.left.menu.ready”, jolloin sivupaneeli on valmis ja siihen voidaan lisätä kyseisen moduulin välilehti (**Kuvio 29.**). Tässä funktiossa taas kutsutaan Post luokkaan luotua funktiota, jolla alustetaan uusi sivupalkin osio.

```
1  <?php
2  use Siberian\Hook;
3  $init = function ($bootstrap) {
4       Hook::listen("editor.left.menu.ready", "blog_nav", "dashboardNav");
5  };
6
7  function dashboardNav ($payload) {
8      return Blog_Model_Post::dashboardNav($payload);
9  }
```

Kuvio 29. Sivupalkin kuuntelu hookilla.

Funktiossa, jota init.php tiedostossa kutsutaan, luodaan moduulin nimellä taulukko, johon alustetaan moduulin sivupalkkiosion tiedot kuten kuviossa 30. Taulukon alustuksen jälkeen koko taulukko palautetaan, jolloin Siberianin ytimessä oleva funktio luo osion sivupalkkiin. Lisäksi layout-kansion sisällä olevaan XML-tiedostoon määritetään XML-koodilla tiedosto, joka tulostetaan näkymään (**Kuvio 31.**).

```php
public static function dashboardNav($editorTree)
{
    $currentUrl = str_replace(self::getBaseUrl(), "", self::getCurrentUrl());
    $editorTree["blog"] = [
        "hasChilds" => false,
        "isVisible" => true,
        "label" => p__("blog", "Blog management"),
        "id" => "sidebar-left-group-blog",
        "is_current" => ("/blog/application" === $currentUrl),
        "url" => __url("blog/application/list"),
        "icon" => "fa fa-envelope",
    ];

    return $editorTree;
}
```

Kuvio 30. Uuden sivupalkin osion alustus

```xml
<blog_application_list>
    <addLayout name="admin_menu" />
    <addLayout name="customization_default" />
    <base>
        <title>Blog Management</title>
    </base>
    <views>
        <content class="admin_view_default"
                 template="blog/application/list.phtml" />
        <customization_sidebar_left class="admin_view_default"
                                    template="application/customization/index/sidebar_left.phtml" />
    </views>
</blog_application_list>
```

Kuvio 31. XLM-koodi, jolla määritetään tiedosto, joka avataan sivupalkista

5 TIETOKANTA

Siberianissa tietokannan taulujen luonti tapahtuu skeematiedoston avulla, jota Siberian vertaa automaattisesti moduulin päivityksen yhteydessä olemassa olevaan tietokantatauluun ja luo puuttuvat kolumnit. Näillä skeematiedostoilla ei voi muokata tai poistaa olemassa olevia kolumneja.

Tietokannan alustamiseen vaaditaan skeematiedosto, joka luodaan resources-kansion alle db/schema. Tälle PHP skeematiedostolle annetaan nimeksi tietokantataulun nimi. Skeematiedostoon luodaan taulukko, jossa on taulun kolumnit sekä kolumnien tiedot (**Kuvio 32.**). Koska Siberian vaatii sille omistetun MySQL tai MariaDB tietokannan, on käytössä kaikki MySQL tietotyypit.

```php
<?php

$schemas = (!isset($schemas)) ? [] : $schemas;
$schemas["blog_comment"] = [
    "comment_id" => [
        "type" => "int(11) unsigned",
        "auto_increment" => true,
        "primary" => true,
    ],
    'post_id' => [
        'type' => 'int(11) unsigned',
        'foreign_key' => [
            'table' => 'blog_post',
            'column' => 'post_id',
            'name' => 'blog_comment_ibfk_1',
            'on_update' => 'CASCADE',
            'on_delete' => 'CASCADE',
        ],
        'index' => [
            'key_name' => 'KEY_POST_ID',
            'index_type' => 'BTREE',
            'is_null' => false,
            'is_unique' => false,
        ],
    ],
    "sender" => [
        "type" => "text",
        "charset" => "utf8",
        "collation" => "utf8_unicode_ci",
    ],
    "comment" => [
        "type" => "text",
        "charset" => "utf8",
        "collation" => "utf8_unicode_ci",
    ],
    "created_at" => [
        "type" => "datetime",
    ],
];
```

Kuvio 32. Skeematiedosto.

Tietokantataululle luodaan Model-kansioon PHP-luokka, joka perii Siberianin ydinluokan Core_Model_Default_Abstract. Tähän taulun luokkaan luodaan construct-funktio (**Kuvio 33.**) jolla määritetään tietokantataulu-luokka, jota tämä luokka käyttää.

```php
<?php
/**
 * Class Blog_Model_Comment
 */
class Blog_Model_Comment extends Core_Model_Default
{
    /**
     * Blog_Model_Comment constructor.
     * @param array $datas
     */
    public function __construct($datas = [])
    {
        parent::__construct($datas);
        $this->_db_table = 'Blog_Model_Db_Table_Comment';
    }
}
```

Kuvio 33. Kommentti-luokka.

Model-kansioon luodaan tietokanta luokille oma alikansiorakenne db/Table, johon tietokantataulujen luokat luodaan (**Kuvio 34.)**. Taulun luokka laajentaa Siberian ydin taululuokkaa Core_Model_Db_Table, joka taas laajentaa Zend-kehyksen luokkaa Zend_Db_Table_Abstract. Tämän Siberianin taululuokan kautta toimivat esimerkiksi funktiot, kuten getterit ja setterit sekä find ja findAll, jotka periytyvät Core_Model_Default luokalta.

```php
<?php
class Blog_Model_Db_Table_Comment extends Core_Model_Db_Table
{
    protected $_name = "blog_comment";
    protected $_primary = "comment_id";
}
```

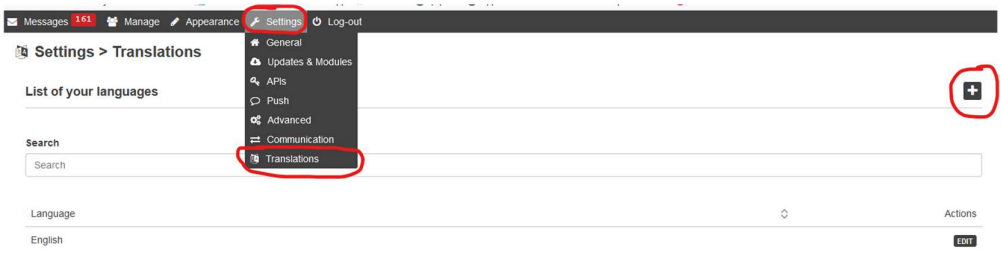
Kuvio 34. Kommentti-tietokantataululuokka

Tietokantataululuokkaan alustetaan taulun nimi sekä perusavain merkkijonoina. Tähän luokkaan voidaan ohjelmoida vaativampia tietokantahakuja joihin find ja findAll funktiot eivät kykene, näitä haku funktiota voidaan kutsua taululuokkaa käyttävästä luokasta.

6 KÄÄNNÖKSET

Siberianissa on mahdollista kääntää sekä hallintapaneeli että applikaatio usealle eri kielelle, jolloin applikaatio mobiililaitteella käytettäessä kääntyy laitteen järjestelmän kielelle ja hallintapaneeli kääntyy selaimen kielelle, jota on mahdollista vaihtaa hallintapaneelin oikeasta ylänurkasta.

Käännettävän kielen lisääminen tapahtuu Siberianin backofficesta valikon Settings alta kohdasta Translations. Sivun oikeasta yläreunasta löytyvästä lisäysnapista painamalla pääsee valitsemaan kielen, joka applikaation lisätään (**Kuvio 35.**).



Kuvio 35. Käännettävän kielen lisääminen

Lisäksi kuviossa 22 löytyvään translations-kansioon tehdään kielen kaksikirjaimisen koodin mukainen kansio, johon luodaan po-tiedosto moduulin nimellä. Tähän po-tiedostoon käännetään merkkijonot, joita hallintapaneelissa tai applikaatiossa käytetään. Applikaatiossa käytettävien käännösten eteen tulee laittaa merkintä *#, mobile* (**Kuvio 36.**).

```
msgid ""
msgstr ""
"Project-Id-Version: \n"
"Report-Msgid-Bugs-To: \n"
"Last-Translator: \n"
"Language-Team: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"POT-Creation-Date: 2019-03-28T15:24:21+01:00\n"
"PO-Revision-Date: 2019-03-28T15:24:21+01:00\n"
"Language: \n"

msgctxt "blog"
msgid "Blog management"
msgstr "Blogin hallinta"

#, mobile
msgctxt "blog"
msgid "Available blog posts"
msgstr "Saatavilla olevat blogiviestit"
```

Kuvio 36. Po-tiedoston esimerkki

Hallintapaneelissa merkkijonon kääntämiseksi käytetään Siberianin PHP-funktiota *p__(),* johon ensimmäisenä parametrinä annetaan konteksti, joka on kuviossa 36 ”msgctxt” ja toisena parametrinä annetaan käännettävä merkkijono, joka on po-tiedostossa ”msgid”, esimerkiksi *p__("blog", "Blog management")*.

Applikaatiossa HTML-tiedostoissa merkkijonojen kääntäminen tapahtuu kirjoittamalla *{{ "Available blog posts" | translate:"blog" }}* muotoon, jolloin käännettävä merkkijono tulee ensin ja tämän jälkeen konteksti. Kun käännetään merkkijonoja AngularJS kontrollerissa käytetään $translate factorya, joka sisällytetään kontrolleriin. Tästä factorysta kutsutaan instant()-funktiota, jolle annetaan ensimmäiseksi parametriksi käännettävä merkkijono ja toiseksi parametriksi konteksti esimerkiksi seuraavalla tavalla *$translate.instant("Available blog posts", "blog").*

7 POHDINTA

Opinnäytetyön ja perehdytysmateriaalin dokumentointi oli mielenkiintoinen projekti, joka loi hyviä haasteita esittää selkeästi Siberianin moduulin rakenne sekä sen ohjelmointi. Esimerkkisovellus joka opinnäytetyön ohella rakennettiin, luo hyvän pohjan yrityksen tuleville perehdytystarpeille. Esimerkkisovelluksessa annettiin kuitenkin vain pintaraapaisu Siberianin mahdollisuuksiin kehittää mobiiliapplikaatioita sekä hallintatyökaluja.

Opinnäytetyötä kirjoittaessa opin uusia näkökulmia, joita voidaan käyttää tulevaisuudessa perehdytystilanteissa. Esimerkkisovellus haastoi luomaan koodia, joka on selkeä, mutta kuitenkin monipuolinen ja josta perehdytettävä saa näkökulmia moduulin ohjelmoimiseen.

Opinnäytetyössä päästiin yrityksen asettamiin tavoitteisiin ja opinnäytetyötä sekä esimerkkisovellusta tullaan käyttämään perehdyttämiseen. Opinnäytetyötä tullaan jatkokehittämään keräämällä perehdytettäviltä tuntemuksia, kuinka oppiminen Siberianin moduulin luontiin olisi sujuva. Esimerkkisovellukseen tullaan kehittämään erilaisia harjoitustehtäviä, jolloin perehdytettävä saa harjoitella moduulin ohjelmointia selkeässä ympäristössä.

LÄHTEET

/1/ Xtraball – Xtraball. Viitattu 8.3.2021 https://xtraball.com/

/2/ Frameworks. Viitattu 8.3.2021 https://developers.siberiancms.com/stack/frameworks/#based-on-ionic-angularjs