LAB University of Applied Sciences

ASP.NET-pohjaisten julkaisujärjestelmien vertailu Avenla Oy:lle

LAB-ammattikorkeakoulu
Insinööri (AMK), Ohjelmistotekniikka
2022
Juho Leskelä

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Leskelä, Juho	Opinnäytetyö, AMK	2022
	Sivumäärä	
	35	

Työn nimi

ASP.NET-pohjaisten julkaisujärjestelmien vertailu Avenla Oy:lle

Tutkinto

Insinööri (AMK)

Toimeksiantajan nimi, titteli ja organisaatio

Teemu Parkkinen, web developer, Avenla Oy

Tiivistelmä

Työn tavoitteena oli ehdottaa ASP.NET -pohjaista käyttöönotettavaksi julkaisujärjestelmää työn toimeksiantajan, Avenla Oy:n, antamien kriteerien sekä teknisen tuen saatavuuden pohjalta. Avenla Oy:n antamat kriteerit olivat tunnettuus, ohjelmistokehys, hinta ja kehitettävyys. Kehitettävyyden alikriteereiksi määriteltiin julkaisujärjestelmän käyttämä API, ja mahdollisuus luoda omia moduuleja ja providereita. Vertailuun otettiin mukaan Microsoftin ASP.NET-pohjaisia julkaisujärjestelmiä, joihin löytyi ilmaista aineistoa. Lisäksi vertailuun otettiin mukaan jo käytössä oleva DNN-julkaisujärjestelmä. Vertailtavat järjestelmät olivat DNN, Umbraco, N2, Piranha CMS ja Orchard Core.

Kriteerejä arvioitiin julkaisujärjestelmien dokumentaation perusteella. Vertailua varten luotiin pisteytysjärjestelmä, jonka avulla kriteerejä vertailtiin. Kullekin kriteerille ja alikriteerille annettiin painokerroin, joka hyväksytettiin toimeksiantajalla. Painokertoimet kuvaavat kriteerien välistä tärkeysjärjestystä.

Lopuksi pohdittiin tulosten merkitystä ja itse luodun pisteytysjärjestelmän ongelmia tulosten kannalta. Vertailun tulosten perusteella ehdotetaan käyttöönotettavaksi Umbraco-julkaisujärjestelmää jo käytössä olevien julkaisujärjestelmien rinnalle.

Asiasanat

Julkaisujärjestelmä, ASP.NET, Microsoft

Abstract

Author(s)	Type of Publication	Published
Leskelä, Juho	Thesis, UAS	2022
	Number of Pages	
	35	

Title of Publication

Comparison of ASP.NET based CMS for Avenla Oy

Name of Degree

Engineer (UAS)

Name, title and organization of the client

Teemu Parkkinen, web developer, Avenla Oy

Abstract

Goal of the thesis was to suggest an ASP.NET based content management system based on criteria given by the commissioner, Avenla Oy, and availability of technical support. The criteria given by Avenla Oy were conspiciousness, software framework, price and developability. Developability was divided into subcriteria which were API used by the CMS and possibility of creating your own modules and providers. The compared CMS were Microsoft's ASP.NET based content management systems that had free materials available. In addition, DNN, which is already in use at Avenla, was put in the comparison. The compared systems were DNN, Umbraco, N2, Piranha CMS and Orchard Core.

The criteria were evaluated based on documentation of the CMS. A scoring system was created for comparing the systems. Each criterion and sub-criterion was given a score multiplier, which got approved by commissioner. Score multipliers depict order of significance of the criteria.

Finally, significance of the results and problems of using self-made scoring system were assessed. Based on the results it is suggested to start using Umbraco CMS alongside the content management systems that are already in use.

Keywords

Content management system, ASP.NET, Microsoft

Sisällys

Liitteet

Termit ja lyhenteet

.NET = Myös .NET Core, avoimen lähdekoodin ohjelmistokehys Windowsille, MacOS:lle ja Linuxille

.NET Framework = Microsoftin ohjelmistokehys Windowsille, .NET:in edeltäjä

API = Application Programming Interface, mahdollistaa kommunikaation sovellusten välillä

ASP.NET = avoimen lähdekoodin ohjelmistokehys dynaamisten verkkosovellusten kehitykseen .NET Frameworkilla

ASP.NET Core = ASP.NET:in uudempi, modulaarinen versio .NET Corelle

CMS = Sisällönhallintajärjestelmä, sovellus verkkosivun sisällön hallinnointiin ilman ohjelmointiosaamista

GraphQL API = API, joka käyttää GraphQL-kyselykieltä kommunikointiin

Razor = Tapa upottaa palvelinpuolen koodia HTML:n sekaan ASP.NET:issä

REST API = REpresentational State Transfer, REST-arkkitehtuurin kriteerit toteuttava API

SOAP API = Simple Object Access Protocol, protokolla palvelimen ja asiakasohjelman väliseen kommunikointiin

SPA = Single Page Application, yhden sivun verkkosivu, joka päivittää sivun sisällön sen sijaan, että ladattaisiin uusi sivu

1 Johdanto

Tämän opinnäytetyön toimeksiantaja Avenla Oy on ohjelmistoalan yritys, joka tuottaa digitaalisia palveluja muille yrityksille. Näihin palveluihin kuuluu julkaisujärjestelmälähtöiset verkkopalvelut, joiden toteutukseen käytetään .NET-pohjaisia DNN- ja Episerver-julkaisujärjestelmiä.

Julkaisujärjestelmät (CMS) ovat olennainen osa nykyaikaista web-kehitystä. Julkaisujärjestelmiä käyttää yli 73 miljoonaa verkkosivua, ja niitä käyttävien verkkosivujenmäärä on ollut kasvussa jo 2010-luvun alusta lähtien (Minaev 2022). Julkaisujärjestelmällä tarkoitetaan sovellusta, jonka avulla useat käyttäjät pystyvät samanaikaisesti muokkaamaan ja julkaisemaan verkkosivustolla (Optimizely).

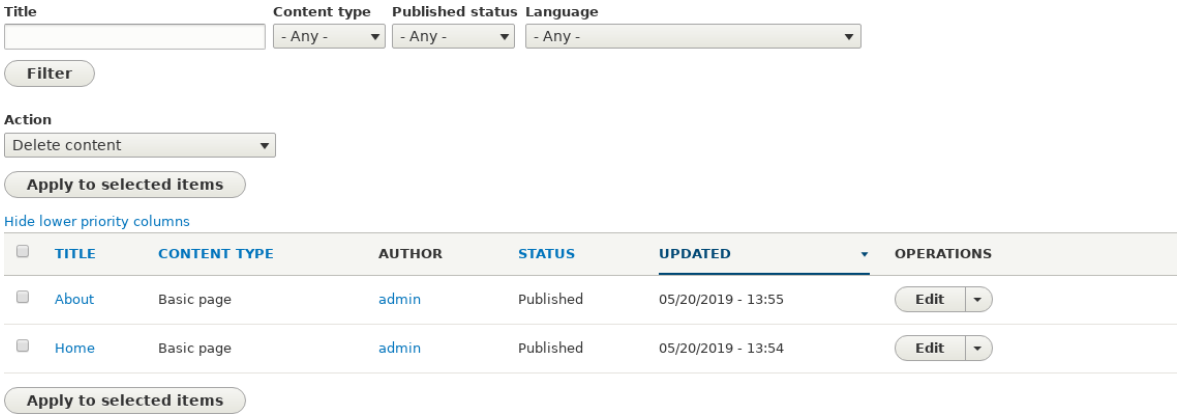
Työssä arvioidaan ja vertaillaan Microsoftin .NET-pohjaisia julkaisujärjestelmiä niiden hinnan, tunnettuuden, arkkitehtuurin ja kehitettävyyden pohjalta. Nämä valintakriteerit tulevat toimeksiantajalta. Lisäksi myös teknisen tuen saatavuus on katsottu arvioitavaksi kriteeriksi. Työn tavoitteena on ehdottaa julkaisujärjestelmää käyttöönotettavaksi Avenla Oy:llä, mikäli tutkituista järjestelmistä löytyy nykyään käytössä olevia parempi järjestelmä. Ehdotettu julkaisujärjestelmä joko korvaisi aiemmin käytössä olevat julkaisujärjestelmät tai toimisi niiden rinnalla.

Toimeksiantajan vaatimuksesta työssä tarkastellaan ainoastaan Microsoftin .NET-pohjaisia julkaisujärjestelmiä, joista on saatavilla dokumentaatio ja muu tarvittava aineisto ilmaiseksi. Myöskään itse toteutetun julkaisujärjestelmän mahdollisuutta ei tutkita tässä työssä, sillä järjestelmän tunnettuus on tärkeää verkkopalveluiden myytävyyden kannalta.

2 Julkaisujärjestelmä

2.1 Julkaisujärjestelmän arkkitehtuuri

Julkaisujärjestelmällä, jota kutsutaan myös sisällönhallintajärjestelmäksi (eng. Content management system, CMS), tarkoitetaan sovellusta, jonka avulla voidaan luoda, muokata, poistaa ja julkaista verkkosivun sisältöä ilman HTML- tai muuta ohjelmointiosaamista (Keene Systems). Sovelluksessa on graafinen käyttöliittymä (Kuva 1), jolla hallinnoidaan tietokantaa, jossa on verkkosivun sisältö sen sijaan että muokattaisiin yksittäisiä HTML-sivuja.

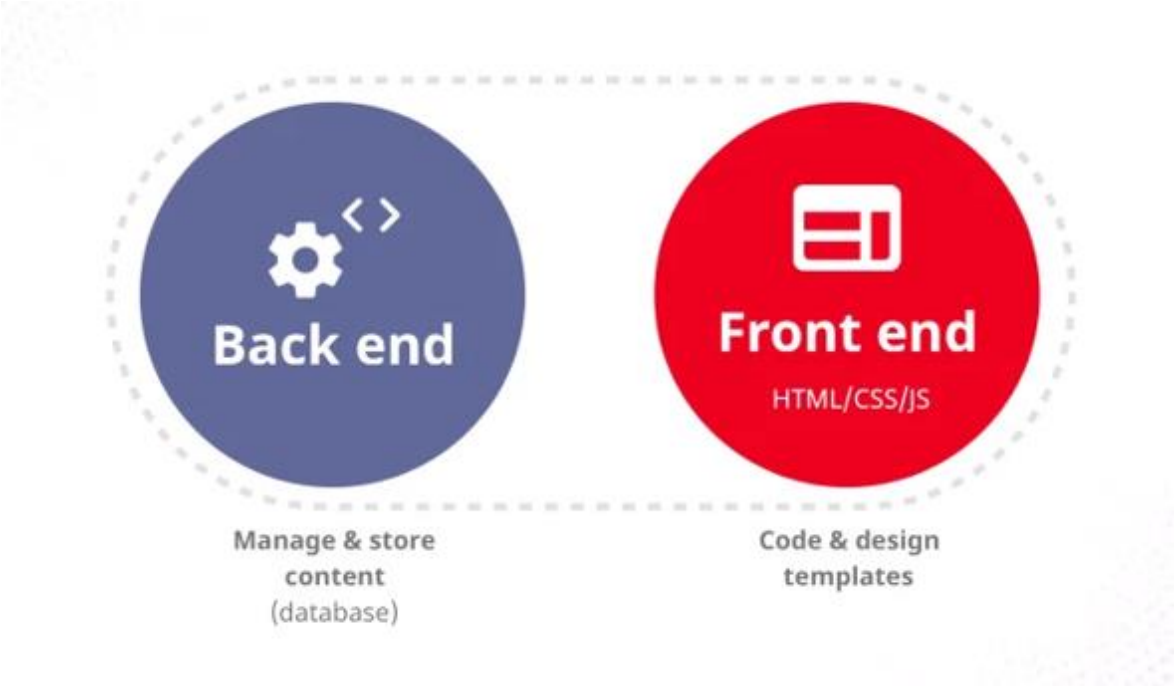


Kuva 1. Esimerkki julkaisujärjestelmän graafisesta käyttöliittymästä (Drupal 2021)

Julkaisujärjestelmiä voidaan luokitella sen perusteella, miten backend ja frontend ovat toteutettu ja kommunikoivat keskenään. Tätä kutsutaan julkaisujärjestelmän arkkitehtuuriksi. Joitain julkaisujärjestelmiä pystytään käyttämään useammilla eri arkkitehtuureilla, kuten esimerkiksi Orchard Corea (Orchard Core Documentation 2021).

2.1.1 Coupled CMS

Coupled (myös ”perinteinen”) julkaisujärjestelmässä kaikki sisältö eli kuvat ja tekstit käsitellään ja tallennetaan sivun backendiin. Toisin kuin muissa arkkitehtuurimalleissa, front- ja backend ovat suoraan yhteydessä toisiinsa. Kuviossa 1 on kuvattu tätä kiinteää kytköstä. Frontend vastaa sivuston sisällön näyttämisestä HTML-sivuilla. (Winkels 2019.)

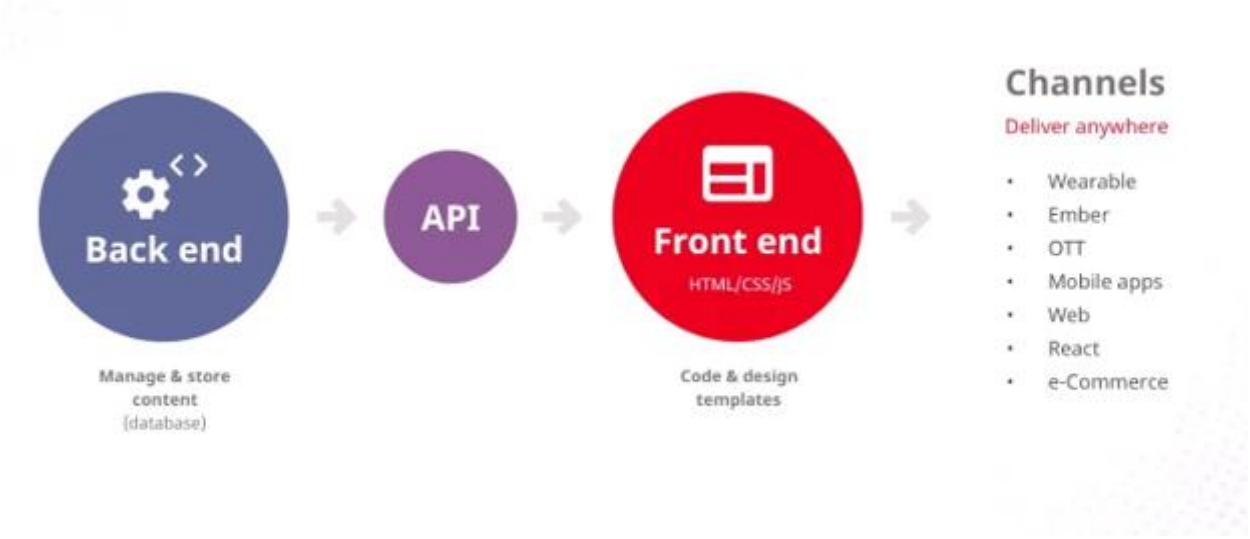


Kuvio 1. Coupled-julkaisujärjestelmäarkkitehtuuri (Brightspot 2018)

Järjestelmän muokattavuus rajoittuu verkkosivun kehittäjän näkökulmasta, sillä muutosten tekeminen vaatii sekä front- että backendin muokkaamisen. Tästä johtuen perinteiset julkaisujärjestelmät soveltuvat parhaiten yksinkertaisten blogien ja muiden tekstipohjaisten sivustojen luomiseen ja ylläpitoon. (Winkels 2019.) Esimerkiksi maailman eniten käytetty julkaisujärjestelmä (Southern 2021) WordPress on perinteinen julkaisujärjestelmä.

2.1.2 Decoupled CMS

Decoupled julkaisujärjestelmässä back- ja frontend ovat erotettu toisistaan. Aivan kuten perinteisessäkin julkaisujärjestelmäarkkitehtuurissa, myös decoupled CMS:n backendissä on verkkosivun sisältö, mutta backendin sisältö saadaan tietokannasta käyttämällä API-kutsuja. Tätä toimintaperiaatetta on havainnollistettu kuviossa 2.

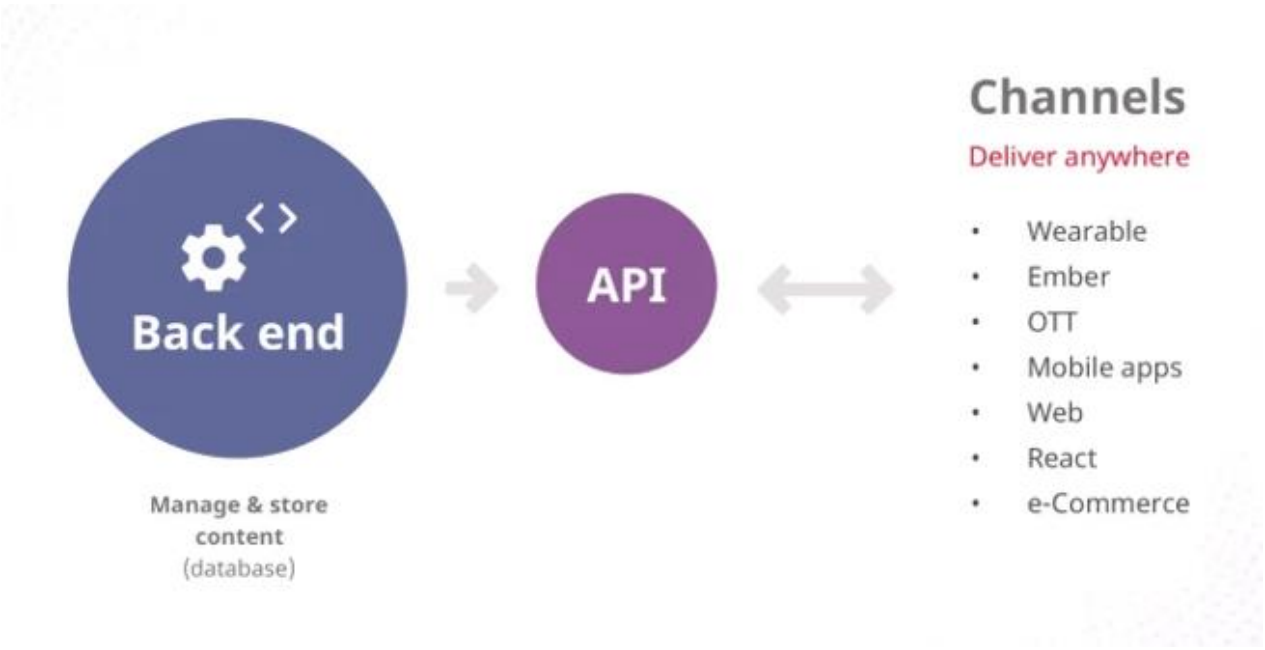


Kuvio 2. Decoupled-julkaisujärjestelmäarkkitehtuuri (Brightspot 2018)

Etuna decoupled-arkkitehtuurissa on, että frontendin koodi ei ole riippuvainen backendin koodista, jolloin voidaan esimerkiksi luoda eri käyttöliittymiä eri laitteille, kuten matkapuhelimille ja tietokoneille. Decoupled-julkaisujärjestelmän heikkoutena voidaan pitää sen korkeampaa osaamisvaatimusta verrattuna perinteiseen julkaisujärjestelmään. (Winkels 2019.) Esimerkiksi Drupal on decoupled-julkaisujärjestelmä.

2.1.3 Headless CMS

Kuten decoupled CMS:ssä, myös headless CMS:ssä front- ja backend ovat erotettu toisistaan ja niiden välinen kommunikaatio tapahtuu API-kutsujen avulla. Headless arkkitehtuurissa ei kuitenkaan varsinaisesti ole frontendiä osana julkaisujärjestelmää (Kuvio 3), vaan frontend tai frontendit ovat täysin erillisiä. (Winkels 2019.) Sekä decoupled että headless CMS ovat riippumattomia frontendistä. Näiden julkaisujärjestelmätyyppien ero on siinä, että decoupled on julkaisujärjestelmä, johon on lisätty API, kun taas headless- julkaisujärjestelmässä on rakennettu API ensin. Käytännössä tämä näkyy siinä, että decoupled-julkaisujärjestelmässä on enemmän ominaisuuksia valmiiksi asennettuna. (Umbraco a.)



Kuvio 3. Headless-arkkitehtuuri (Brightspot 2018)

Headless-julkaisujärjestelmillä on samat edut kuin decoupled-julkaisujärjestelmillä verrattuna perinteisiin julkaisujärjestelmiin, ja se antaa vielä enemmän vapauksia kehittäjille sisällön näyttämisessä kuin decoupled-arkkitehtuuri (Winkels 2019). Umbraco suosittelee headless-arkkitehtuuria, mikäli tarkoituksena on luoda useita frontendejä eri laitteille (Umbraco a). Arkkitehtuurin heikkoutena on kuitenkin se, että useiden frontendien luominen voi viedä huomattavan paljon aikaa (Winkels 2019).

2.2 Tyypillisiä ominaisuuksia

Julkaisujärjestelmiin kuuluu usein myös monia muita ominaisuuksia sisällön muokkaamisominaisuuksien lisäksi. Käyttäjille voidaan antaa eri käyttöoikeudet roolien avulla, kuten esimerkiksi admin ja author. Author voi muun muassa vain luoda, muokata ja poistaa omia kirjoituksiaan, kun taas administrator pystyy myös muokkaamaan toisten julkaisemia tekstejä ja muutoin hallinnoimaan sivustoa. (WordPress).

Muita ominaisuuksia ovat esimerkiksi hakukoneoptimointi (eng. search engine optimization, SEO), eCommerce ja lokalisointi. Hakukoneoptimoinnilla voidaan parantaa verkkosivun näkyvyyttä Googlessa ja muissa hakukoneissa. Näkyvyyteen vaikuttaa muun muassa sivun ulkonäkö ja metatagit (Climb). Termi eCommerce viittaa julkaisujärjestelmän mahdollisiin verkkokauppaominaisuuksiin, kuten tuotteiden hallinta ja ostosten maksaminen.

3 Julkaisujärjestelmän valintakriteerit

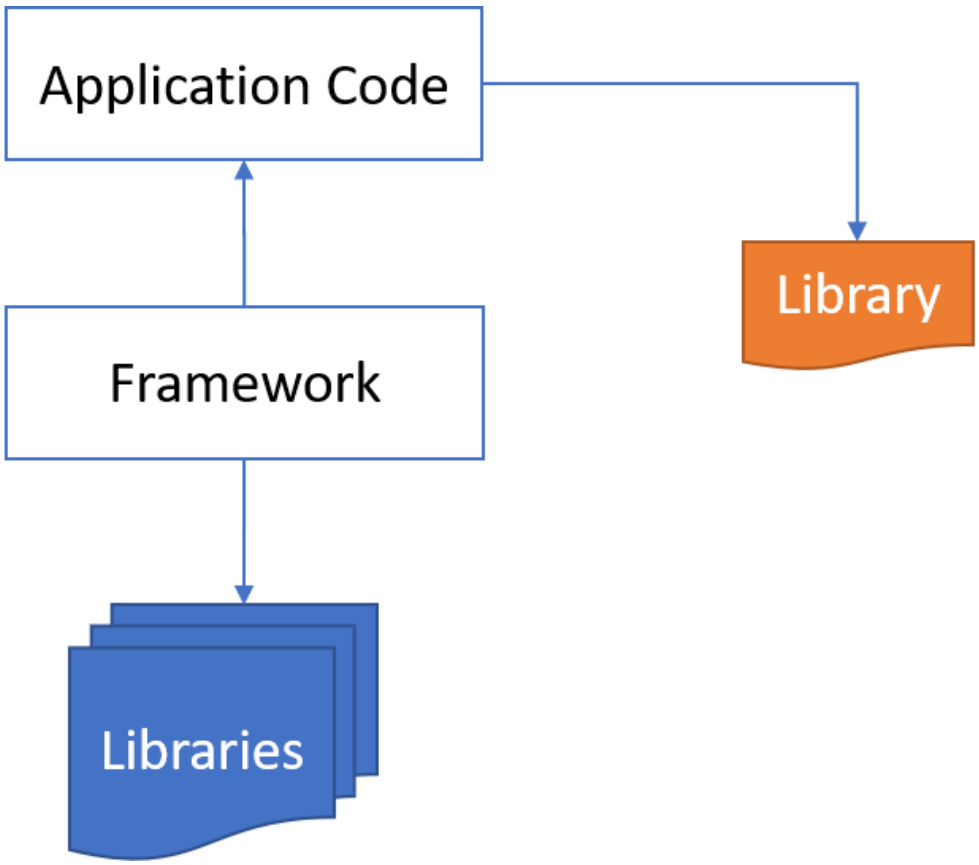
3.1 Kriteerit

Toimeksiantajan mukaan julkaisujärjestelmän sopivuuteen vaikuttavia tekijöitä ovat hinta, tunnettuus, ohjelmistokehys ja kehitettävyys. Toimeksiantajan mukaan hinnan ja tunnettuuden merkitys valinnassa on se, että kalliilla tai tuntemattomalla julkaisujärjestelmällä toteutettu ratkaisu voi olla hankala myydä asiakkaalle. Tunnettuutta arvioidaan saatavilla olevien tietojen perusteella. Nämä tiedot ovat latauskerrat, yhteisön suuruus, Google-hakujen määrä ja julkaisujärjestelmää käyttävien sivustojen määrä.

Lisäksi myös teknisen tuen saatavuus on katsottu arvioitavaksi tekijäksi, vaikka Salokankaan (2022) mukaan tukipalveluita ei ole juuri tarvittu. Tekniseksi tueksi lasketaan kaikki julkaisujärjestelmän ylläpitäjien tarjoamat tukipalvelut, kuten esimerkiksi kysymyksiin vastaaminen sähköpostitse. Keskustelupalstat sisältyvät tekniseen tukeen ainoastaan siinä tapauksessa, että keskustelupalsta on aktiivinen ja julkaisujärjestelmän kehittäjät osallistuvat keskusteluun.

3.2 Ohjelmistokehys

Ohjelmistokehys on runko, joka tarjoaa toteuttamattomia funktioita, joita kehys kutsuu. Ohjelmistokehyksen merkittävä ero ohjelmakirjastoihin on se, että kirjaston käyttäjä kutsuu ohjelmakirjaston funktiota, kun taas ohjelmistokehys vastaa siitä, milloin jotain funktiota kutsutaan. Royn (2022) mukaan tämä voi tarkoittaa esimerkiksi tapahtumiin (eng. event) perustuvaa funktiokutsua. Tätä kutsutaan ohjauksen kääntämiseksi (eng. Inversion of Control). (Kumar 2020). Kirjaston ja ohjelmistokehyksen välistä suhdetta on havainnollistettu kuviossa 4, josta näkyy, että ohjelmistokehys voi käyttää useita kirjastoja ja ohjauksen kääntäminen.



Kuvio 4. Ohjelmistokehyksen ja kirjaston välinen suhde (Roy 2022)

Julkaisujärjestelmän käyttämällä ohjelmistokehyksellä on merkitys järjestelmän elinkaaren kannalta, sillä mikäli järjestelmä on rakennettu toimimaan hyvin vanhalla ohjelmistokehyksellä, jota ei enää ylläpidetä, sitä ei välttämättä kannata ottaa käyttöön mahdollisen tuen loppumisen vuoksi. Tuen loppuminen tarkoittaa, että Microsoft ei enää korjaa, päivitä tai tarjoa teknistä tukea kyseiseen ohjelmistokehykseen (Microsoft c). Lisäksi vanhalla ohjelmistokehyksellä kehitetty sovellus on usein suorituskyvyltään hitaampi, eikä työntekijöitä välttämättä kannata opettaa käyttämään käytöstä poistuvia teknologioita.

3.2.1 ASP.NET Web Forms

ASP.NET Web Forms on Microsoftin vuonna 2002 (Microsoft a) julkaisema ohjelmistokehys, jonka avulla voi luoda ASP.NET-verkkosivuja Visual Studiolla. Ohjelmistokehyksen perusperiaatteena on erottaa toisistaan näkymä ja koodi jakamalla Web Forms kahteen osaan: ASPX-tiedostoihin ja kooditiedostoihin. ASPX-tiedostot vastaavat verkkosivun ulkoasusta, ja se koostuu HTML:stä ja skriptistä. Ohjelmointikielenä voidaan käyttää mitä tahansa .NET ohjelmointikieltä, kuten C#:ia. Web Formsin koodi suoritetaan palvelimella ja vastaus lähetetään selaimeen. (JavaTpoint.)

Web Forms on osa Microsoftin .NET Frameworkia, joka on osa Windows-käyttöjärjestelmää (Microsoft c). Tästä johtuen Web Forms tulee toimimaan vielä pitkään, vaikkei sen käyttöä välttämättä enää suositella uusiin projekteihin (Seng 2021). Syitä tähän on muun muassa se, että Web Forms ei tule olemaan osa .NET:iä, ja sitä tuetaan vain Windowsilla, toisin kuin monia uudempia ohjelmistokehyksiä (Microsoft a). Lisäksi Web Forms on myös suorituskyvyltään huomattavasti heikompi kuin esimerkiksi ASP.NET MVC (Sukesh 2015).

3.2.2 ASP.NET MVC

ASP.NET MVC on Microsoftin vuonna 2013 julkaisema avoimen lähdekoodin framework (Tyler 2021). ASP.NET MVC hyödyntää MVC-ohjelmistosuunnittelumallia, jossa websovellus on jaettu malliin (model), näkymään (view) ja ohjaimeen (controller). Käytännössä näkymät tehdään Razorilla, jonka avulla voidaan upottaa C#- tai VB-koodia HTML-sivuun. Malli ja ohjain kirjoitetaan myös C#:lla tai VB:lla (Tutorialspoint). Malli kuvaa tiedon rakennetta, näkymä näyttää tiedon ja ohjain vastaa verkkosovelluksen käyttäjän antamista syötteistä (Tyler 2021).

ASP.NET MVC on vielä tuettu eikä Microsoft ole antanut virallista tuen loppumispäivämäärää (Microsoft b). Toisaalta MVC:n viimeinen päivitys oli marraskuussa 2018, mikä on yksi syy, miksi sitä ei suositella uusiin projekteihin. MVC:ssä ei myöskään ole cross-platform tukea kuten ASP.NET Coressa, ja se on suorituskyvyltään Corea huonompi (Abto Software).

3.2.3 ASP.NET Core

ASP.NET Core on Microsoftin vuonna 2016 kehittämä avoimen lähdekoodin cross-platform ohjelmistokehys verkkosovellusten kehittämiseen. ASP.NET Core 3.x ja myöhemmät versiot vaativat .NET Coren. ASP.NET Core on uudelleen suunniteltu versio ASP.NET:istä, joka sisältää edellä esitellyt Web Formsin, MVC:n sekä Web Pages:n. Perusperiaatteena on modulaarisuus, eli ohjelmistokehys itsessään on kevyt, ja siihen voi lisätä toiminnallisuutta NuGet-pakettien avulla. Arkkitehtuuristen muutosten tuomia etuja ASP.NET:iin verrattuna ovat muun muassa parempi suorituskyky ja toimivuus eri käyttöjärjestelmillä. (Roth ym. 2021.)

.NET:n, joka Microsoftin mukaan sisältää ASP.NET Coren, uusimman version 6 tuki lakkaa marraskuussa 2024. Tämä ei kuitenkaan tarkoita, että ASP.NET Core olisi poistumassa käytöstä, sillä .NET versioiden 7 ja 8 julkaisuajankohdiksi on arvioitu marraskuu 2022 ja marraskuu 2023. (Microsoft b.)

3.3 Kehitettävyys

Julkaisujärjestelmän kehitettävyydelle ei ole yleisesti käytettyä määritelmää eikä arviointitapaa. Kehitettävyydellä tarkoitetaan tässä työssä julkaisujärjestelmän tarjoamia työkaluja ominaisuuksien lisäämiseksi järjestelmään. Tähän kuuluu esimerkiksi se, kuinka helposti pystytään siirtämään koodia asiakkuudesta toiseen. (Salokangas 2022.)

Näitä työkaluja ovat Avenlan mukaan erilaiset API-rajapinnat ja mahdollisuus luoda omia moduuleja ja providereita. Yleisesti moduulilla tarkoitetaan ohjelman osaa, joka tuo ohjelmaan jotain toiminnallisuutta (Technopedia). Julkaisujärjestelmien asiayhteydessä moduulilla tarkoitetaan uudelleenkäytettäviä komponentteja, joista verkkosivu koostuu. Esimerkiksi Umbraco-julkaisujärjestelmässä verkkosivun ylänavigointipalkki voi olla moduuli. (O'Reilly.) Provider on suunnittelumalli, jonka avulla voidaan helpottaa ohjelman muokkaamista riippuvuusinjektion avulla (Nance 2017).

3.3.1 API

API on lyhenne sanoista Application Programming Interface. API mahdollistaa kahden ohjelman välisen kommunikoinnin. Toimintaperiaatteena on, että asiakasohjelma (eng. client) tekee pyynnön API:lle, joka lähettää pyynnön palvelimelle. Palvelin lähettää vastauksen API:n kautta asiakasohjelmalle. (IBM Cloud Education 2020.) API:a hyödynnetään esimerkiksi säätietojen hakemisessa kolmannen osapuolen palvelusta syöttämällä postinumero, ja vastauksena tulisi kyseisen alueen lämpötilatiedot (Red Hat 2020).

Ei ole olemassa yksiselitteisesti parasta API-tyyppiä, sillä eri API:illa on eri hyvät ja huonot puolet ja eri tilanteissa voi olla järkevämpää käyttää esimerkiksi SOAP:ia kuin REST API:a. API:n tyypillä on merkitystä ohjelman suorituskyvyn ja turvallisuuden kannalta. (Altvater 2017.) Taulukossa 1 on tiivistetty työn kannalta olennaisten API:en ominaisuudet.

	SOAP	REST	GraphQL
Julkaistu	90-luvun lopulla	2000-luvulla	2015
Formaatit	XML	XML, JSON, HTML, YAML, muut	JSON
Idea	Yksi endpoint URI datan hakemiseen ja muokkaamiseen	Data esitetään resurs-seina URI:n avulla HTTP-protokollaa käyttäen	Asiakasohjelma määrittelee haettavan datan kyselykielen avulla
Merkittävimmät edut	- Tunnettu -Yksi endpoint -Vahvat tietotyypit -Useita protokollia (HTTP, SMTP, FTP)	- Joustava datan formatointi - Kevyt - Useita dataformaatteja -Tilaton	- Joustava datan haku - Kevyt - Vahvat tietotyypit -Tehokas datan haku
Merkittävimmät haitat	- Ylimääräisen datan haku - Runsassanainen - Ei välimuistin käyttömekanismia - Tehoton	- Ylimääräisen datan haku - Heikot tietotyypit - HATEOAS monimutkainen toteutus	- Ei välimuistin käyttömekanismia - Versioinnin puute

Taulukko 1. Erilaisten API:en ominaisuudet, heikkoudet ja vahvuudet (mukailtu Camparanio 2019)

REST

REST API (myös RESTful API) on REST-arkkitehtuuria noudattava API. REST tulee sanoista Representational State Transfer. Asiakasohjelma tekee REST API -pyynnön palvelimelle HTTP-protokollaa käyttäen. Pyyntö voidaan tehdä käyttäen JSON:ia, HTML:ää, XLT:tä, Pythonia, PHP:ta tai tavallista tekstiä. (Red Hat 2020.)

Jotta API-kutsu olisi RESTful, sen täytyy täyttää viisi kriteeriä. Ensinnäkin täytyy olla selkeä jako asiakas- ja palvelinohjelmiin. Toiseksi asiakas- ja palvelinohjelman välisen kommunikoinnin täytyy tapahtua ”tilattomasti” (eng. stateless) Tämä tarkoittaa, että palvelin ei tallenna tietoja asiakasohjelmasta. Kolmantena vaatimuksena on, että REST API:n tulee käyttää välimuistia hyödykseen usein haettujen resurssien löytämisen nopeuttamiseksi. REST API:in kuuluu myös palvelimien välinen hierarkia, joka ei näy asiakasohjelmalle. Viimeisenä kriteerinä on, että API käyttää yhdenmukaista rajapintaa (uniform interface), mikä tarkoittaa, että jokainen API-pyyntö sisältää tarvittavat tiedot pyynnön käsittelyyn, ja jokainen vastaus palvelimelta sisältää tarpeeksi tietoa, jotta asiakasohjelma pystyy ymmärtämään

vastauksen. Yhdenmukaiseen rajapintaan kuuluu myös HATEOAS, joka on lyhenne sanoista Hypermedia As The Engine Of Application State. Käytännössä tämä tarkoittaa, että palvelin kertoo vastauksessaan, miten haettuja tietoja pystyy muokkaamaan. Kuvassa 2 on esimerkki HATEOAS:in mukaisesta vastauksesta. Lisäksi on myös kuudes vapaaehtoinen kriteeri, jonka mukaan palvelimen pitää pystyä lähettämään koodia asiakasohjelman suoritettavaksi. (Avraham 2017; Red Hat 2020.)

```
HTTP/1.1 200 OK
Content-Type: application/+json
Content-Length: ...
{
   "payroll": {
       "employee_number": "employee_123",
       "salary" : 1000,
       "links": {
           "increment": "/payroll/employee_123/increment",
           "decrement": "/payroll/employee_123/decrement",
           "close": "/payroll/employee_123/close"
       }
   }
}
```

Kuva 2. Esimerkki HATEOAS:ista (GeeksForGeeks 2020)

SOAP

SOAP API on protokolla asiakasohjelman ja palvelimen väliseen kommunikointiin. SOAP tulee sanoista Simple Object Access Protocol. SOAP API on ohjelmointikielestä riippumaton ja se käyttää XML:ää tiedon siirrossa alla olevan esimerkin mukaisesti (Kuva 3). SOAP-pyyntö voi sisältää neljä osaa: soap:Envelope, soap:Header, soap:Body ja soap:Fault, tosin vain envelope ja body ovat pakollisia. (Stoplight.)

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetUser>
      <m:UserId>123</m:UserId>
    </m:GetUser>
  </soap:Body>
</soap:Envelope>
```

Kuva 3. Esimerkki SOAP-pyyntö (Stoplight)

SOAP Envelopen on tarkoitus erottaa SOAP muista XML-dokumenteista. Envelopen sisään tulee kaikki muut mahdolliset osat (Stoplight). Headerillä voidaan liittää sovelluskohtaista kontekstitietoa (esim. tietoturva, salaus) SOAP-pyyntöihin ja vastauksiin (IBM 2022). SOAP Body sisältää haettavat tiedot. Kuvan 3 esimerkissä body sisältää tarvittavat tiedot käyttäjän, jonka ID on 123, tietojen hakemiseen. SOAP Fault on tarkoitettu virheviestejä varten, jos API-kutsu epäonnistuu. Fault kertoo, mikä virhe tapahtui ja missä, sekä SOAP noden roolin, jossa virhe tapahtui. (Stoplight.)

GraphQL

GraphQL API on GraphQL-kyselykieleen perustuva API. Palvelinohjelmassa on skeemoja (eng. schema), jotka määrittelevät, mitä dataa asiakasohjelmat voivat hakea palvelimelta (Red Hat 2019). Tämä eroaa REST API:sta, jossa palvelimella on määritetty, mitkä kyselyt ovat sallittuja endpointtien avulla. GraphQL API:ssa on vain yksi endpoint, johon tehdään POST-pyyntöjä. Pyynnön vartalossa (eng. body) on GraphQL kysely kuvan 4 mukaisesti. GraphQL:n etuna REST:iin verrattuna on se, että pystytään hakemaan vain se data, mitä tarvitaan. (Academind 2019.)

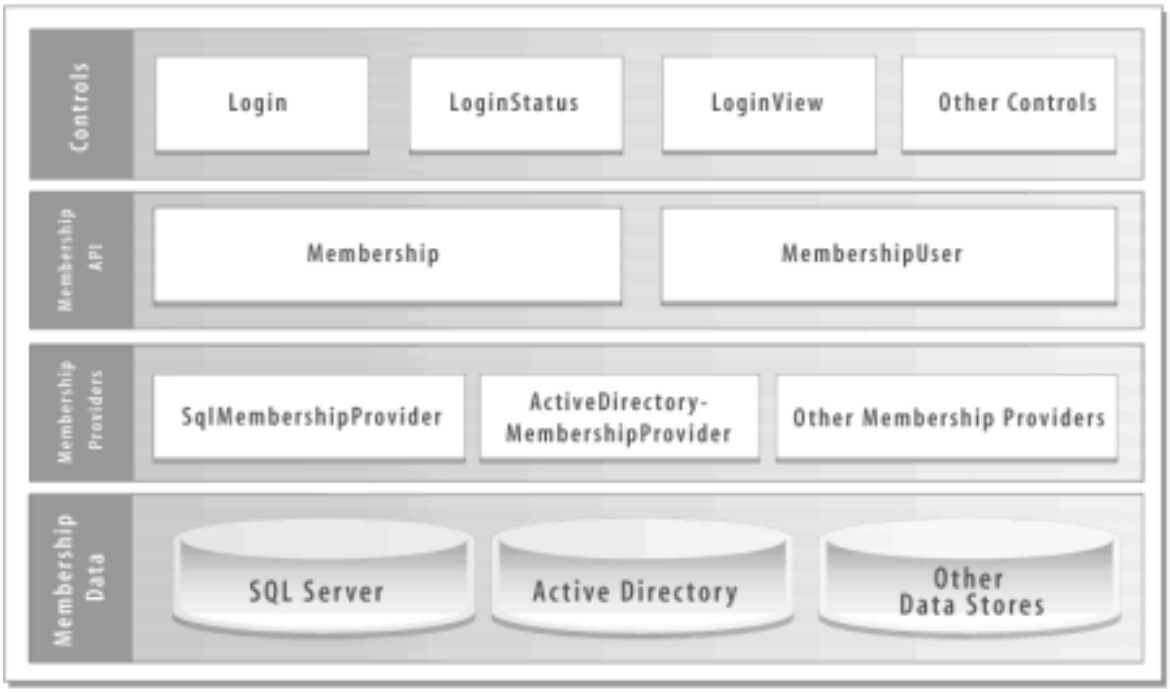
```
{
  human(id: "1000") {
    name
    height(unit: FOOT)
  }
}
```

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 5.6430448
    }
  }
}
```

Kuva 4. Esimerkki GraphQL-pyynnöstä ja vastauksesta (GraphQL)

3.3.2 Provider

Provider on Microsoftin kehittämä suunnittelumalli ASP.NET 2.0:ssa. Provider on moduuli, joka toimii rajapintana palvelun ja datalähteen välillä (Kuva 5). Kuvassa 5 palveluna on membership, joka vastaa käyttäjiien autentikoinnista. Provider määrittelee, mistä data haetaan. ActiveDirectoryMembershipProvider voisi esimerkiksi olla itse tehty provider, joka hakee tiedot Active Directorysta. Provider-mallin ideana on tehdä omien providereiden käyttämisestä vaivattomampaa, ja providerin vaihtaminen onnistuu vähäisillä muutoksilla ohjelman konfiguraatioon. (Microsoft 2011.)



Kuva 5. ASP.NET 2.0 Provider-malli (Microsoft 2011)

Riippuvuusinjektio

Riippuvuusinjektio (eng. dependency injection, DI) on suunnittelumalli, jonka avulla ohjelmien kehittäjät pystyvät tekemään moduuleja, jotka ovat vähemmän riippuvaisia toisista moduuleista ja mahdollistaa toteutuksien vaihtamisen vähemmällä ohjelman muokkaamisella (Nance 2017). Kuvassa 6 IndexModel-luokka on suoraan riippuvainen MyDependency-luokasta. Tämänlaisia riippuvuuksia tulee välttää, sillä esimerkin IndexModel-luokkaa joudutaan muokkaamaan, jos halutaan vaihtaa MyDependency-luokan toteutusta. Lisäksi ohjelman yksikkötestaus voi olla hankalaa. (Larkin ym. 2022.) Salokankaan (2022) mukaan riippuvuusinjektiota voidaan käyttää providereiden luomiseen.

```
public class MyDependency
{
    public void WriteMessage(string message)
    {
        Console.WriteLine($"MyDependency.WriteMessage called. Message: {message}");
    }
}

public class IndexModel : PageModel
{
    private readonly MyDependency _dependency = new MyDependency();

    public void OnGet()
    {
        _dependency.WriteMessage("IndexModel.OnGet");
    }
}
```

Kuva 6. Esimerkki riippuvuudesta luokkien välillä (Larkin ym. 2022)

Yllä olevan esimerkin ongelma voidaan ratkaista tekemällä MyDependency-luokasta rajapinta IMyDependency, jonka MyDependency toteuttaa kuvan 7 mukaisesti (Larkin ym. 2022). ASP.NET Core-pohjaisissa sovelluksissa niin sanottu IoC-kontti voidaan konfiguroida Startup.cs-tiedoston ConfigureServices-metodissa. IoC tulee sanoista Inversion of Control, ja se viittaa siihen, että siirretään komponenttien alustus komponentin käyttäjältä konttiin (eng. container). Ennen .NET Corea riippuvuusinjektio toteutettiin kolmannen osapuolen ohjelmakirjastoilla. (Nance 2017.)

```
public interface IMyDependency
{
    void WriteMessage(string message);
}

public class MyDependency : IMyDependency
{
    public void WriteMessage(string message)
    {
        Console.WriteLine($"MyDependency.WriteMessage Message: {message}");
    }
}
public class Index2Model : PageModel
{
    private readonly IMyDependency _myDependency;

    public Index2Model(IMyDependency myDependency)
    {
        _myDependency = myDependency;
    }

    public void OnGet()
    {
        _myDependency.WriteMessage("Index2Model.OnGet");
    }
}
```

Kuva 7. Esimerkki riippuvuusinjektiosta (Larkin ym. 2022)

4 Vertailu

4.1 Pisteytys

Julkaisujärjestelmien vertailua varten julkaisujärjestelmät pisteytettiin hinnan, tunnettuuden, kehitettävyyden ja teknologian ajantasaisuuden perusteella. Kriteereille annettiin painokertoimia, jotka hyväksytettiin Avenlan edustajilla. Painokertoimet kuvaavat eri kriteerien tärkeyttä Avenla Oy:lle. Alla olevassa taulukossa 2 on kuvattu eri arviointikriteereistä annetut pisteet ja niiden painokertoimet. Kriteeristä tai alikriteeristä tulee nolla pistettä myös silloin, kun kyseistä tietoa ei ole saatavilla.

Kehitettävyys

Kehitettävyyden kokonaispisteet koostuvat kolmesta eri alikriteeristä, jotka ovat julkaisujärjestelmän API:n tyyppi, mahdollisuus luoda ja käyttää omia providereita ja moduuleja. API:n, omien providerien ja moduulien painokertoimet ovat 2, koska nämä vaikuttavat huomattavasti siihen, miten julkaisujärjestelmään pystytään lisäämään toiminnallisuutta. API:en pisteytykset tulevat Avenlan toiveista. JSON REST API.sta annetaan kaksi pistettä, SOAP API:sta yksi piste ja kaikista muista nolla pistettä.

Hinta

Hinta pisteytettiin ainoastaan sen perusteella, onko julkaisujärjestelmä ilmainen vai maksullinen. Julkaisujärjestelmä luokiteltiin ilmaiseksi, mikäli siitä on saatavilla ilmainen versio, vaikka siitä olisi myös maksullisiakin versioita. Julkaisujärjestelmien maksullisia versioita ja palveluita on käsitelty luvuissa 4.2.1–4.2.5. Hinnan painokertoimeksi valittiin yksi. Vaikka hinta ei vaikuta julkaisujärjestelmän toimintaan, sillä on merkitys liiketoiminnan kannalta, ja siksi sille valittiin korkeampi painokerroin kuin tekniselle tuelle ja tunnettuudelle.

Tekninen tuki

Koska teknisiä tukipalveluita ei ole juurikaan tarvittu Avenlalla, niiden arviointi tehtiin suppealla asteikolla pisteytyksen suhteen. Saatavilla olevia tukipalveluita ja niiden hintoja on käsitelty laajemmin sanallisesti julkaisujärjestelmien käsittelyluvuissa 4.2.1–4.2.5, mutta ne eivät vaikuta pisteytykseen. Teknisen tuen olemassa olemisesta annettiin 0,5 pistettä, kun taas sen puuttumisesta tuli 0 pistettä. Koska Avenlalla tukipalveluita ei ole koettu kovin tärkeäksi, teknisen tuen painokertoimeksi laitettiin 0,5.

Tunnettuus

Tunnettuuden pisteytys perustuu tärkeimmästä vähiten tärkeään siihen, kuinka moni verkkosivu käyttää kyseistä julkaisujärjestelmää, latauskertoihin, yhteisön suuruuteen, ja Google-hakujen määrään. Pisteet annettiin vertailujärjestyksen mukaan, eli käytetyin julkaisujärjestelmä sai eniten pisteitä, ja vähiten käytetty sai vähiten. Tämä suhteellinen pisteytystapa valittiin siksi, että C# julkaisujärjestelmillä ei ole kovin suurta markkinaosuutta, joten vertailu esimerkiksi WordPressiin saattaisi johtaa siihen, että kaikki vertailtavat järjestelmät saisivat nolla pistettä. Tunnettuuden painokertoimeksi valittiin 0,5, sillä sitä on hyvin vaikea arvioida objektiivisesti, eikä kaikkiin alikriteereihin löytynyt tietoja. Tunnettuus ei myöskään vaikuta julkaisujärjestelmän toimintaan.

Ohjelmistokehys

Ohjelmistokehys pisteytettiin teknologian nykyaikaisuuden perusteella. Vanhimmasta teknologiasta eli ASP.NET Web Forms:ista annettiin nolla pistettä, ASP.NET MVC:stä yksi piste ja ASP.NET Coresta kaksi pistettä. Ohjelmistokehyksen painokertoimeksi asetettiin 3, koska sillä on suuri merkitys julkaisujärjestelmän suorituskyvyn ja kehitettävyyden kannalta.

Kriteeri	0p	0,5p	1p	1,5p	2p	Painokerroin
Kehitettävyys						
API	Muu	-	SOAP	-	REST	2
Provider	Ei	-	On	-	-	2
Moduuli	Ei	-	On	-	-	2
Hinta	Maksullinen	-	Ilmainen	-	-	1
Tekninen tuki	Ei	On	-	-	-	0,5
Tunnettuus	5.	4.	3.	2.	1.	0,5
Ohjelmistokehys	Web Forms	-	MVC	-	Core	3

Taulukko 2. Julkaisujärjestelmien arviointikriteerien pisteytysperusteet

4.2 Julkaisujärjestelmien tarkastelu

Vertailuun valittiin Microsoftin .NET-pohjaisia julkaisujärjestelmiä, joihin oli saatavilla aineistoa ilmaiseksi. Myös muita johdannossa mainittuja valintakriteerejä otettiin huomioon

tarkasteltavien julkaisujärjestelmien valinnassa. Lisäksi vertailuun otettiin jo Avenlalla käytössä oleva DNN-julkaisujärjestelmä. Mukaan on otettu myös N2CMS esimerkiksi huonosti dokumentoidusta ja vanhentuneesta julkaisujärjestelmästä.

Alla olevassa taulukossa 3 on esitetty perusteet tunnettuudesta annetuille pisteille kullekin arvioitavalla julkaisujärjestelmälle. Niiden julkaisujärjestelmien kohdalla, joissa on viiva, ei löytynyt kyseistä tietoa. Liitteessä 1 on kuvattu julkaisujärjestelmien Googlehakujen määrää.

	DNN	Umbraco	Orchard Core	N2CMS	PiranhaCMS
Sivujen määrä	750 000	-	-	-	-
Lataukset	8 milj.	5 milj.	-	-	-
Yhteisön koko	>1milj.	>200 000	-	-	-
Googlehaut	2.	1.	3.	5.	4.
Järjestysnumero	1.	2.	3.	5.	4.

Taulukko 3. Tunnettuuden järjestyksen perusteet

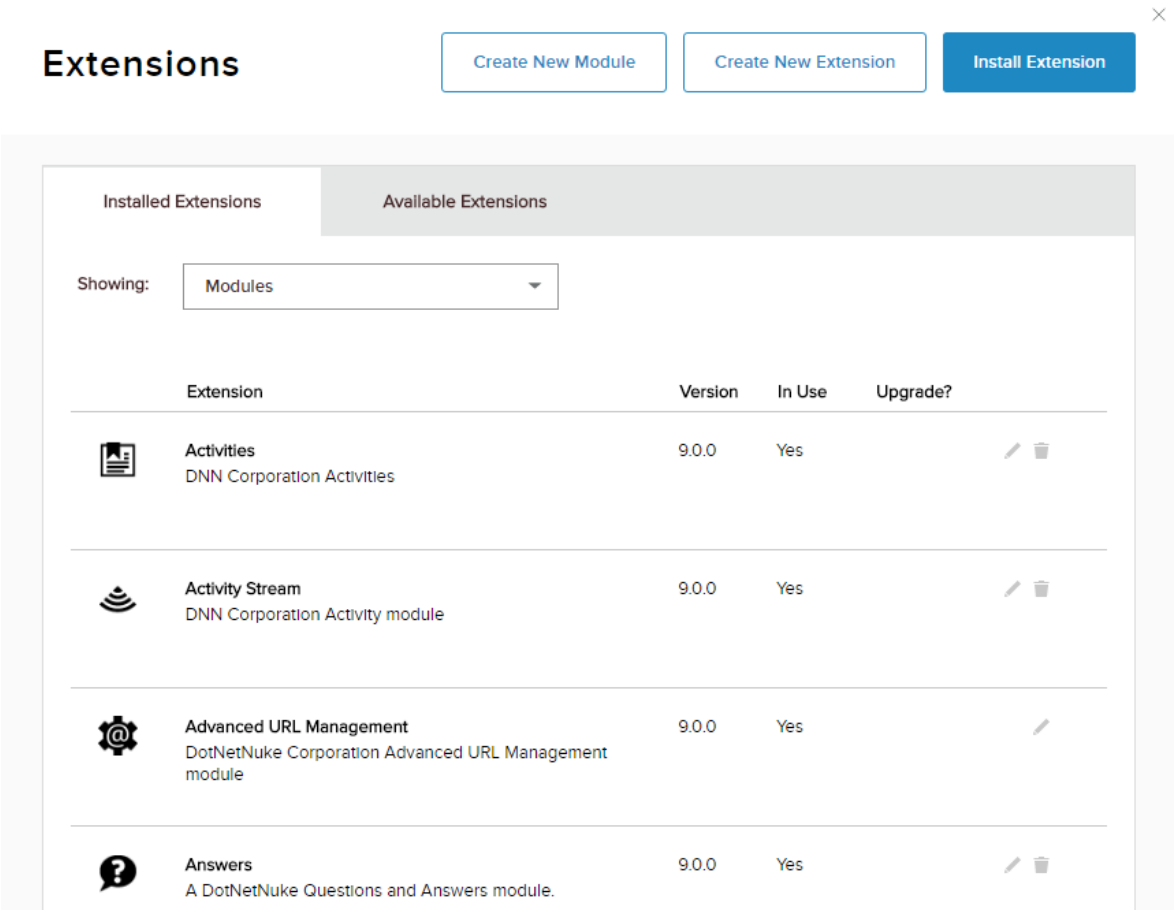
4.2.1 DNN

DNN Platform (ennen DotNetNuke) on avoimen lähdekoodin ASP.NET Web Forms-pohjainen julkaisujärjestelmä (DNN a). DNN-yhteisössä on ollut jo vuosia keskustelua julkaisujärjestelmän päivittämisestä .NET Corelle, tosin mitään konkreettista aikataulua ei ainakaan vielä vuonna 2020 ollut (Sellers 2020). DNN:ää käyttää yli 750 000 verkkosivua, ja sillä on yli 8 miljoonaa latauskertaa ja yli miljoona yhteisön jäsentä (Github a) DNN:llä on myös kaksi maksullista julkaisujärjestelmää, DNN Evoq Content ja DNN Evoq Engage (DNN b). Julkaisujärjestelmän ilmaisversiota päivitetään edelleen (Github a).

DNN:ään on saatavilla dokumentaatio, keskustelupalsta ja Evoq-asiakkaille on myös saatavilla asiakaspalvelua (DNN c). Dokumentaatio ei tosin välttämättä ole aivan ajan tasalla, sillä siellä on muun muassa linkkejä ohjeisiin, joissa käsitellään vanhaa 4.3.4 versiota ja yli kymmenen vuotta vanha blogi. DNN käyttää REST API:a datan hakemiseen (Github a).

Julkaisujärjestelmän dokumentaation mukaan on mahdollista luoda omia providereita (DNN d) ja moduuleja (DNN e). Omien providereiden luonti onnistuu DNN 9.4 ja sitä uudemmilla versioilla .NET Coren riippuvuusinjektio-ominaisuutta hyödyntäen, vaikka DNN onkin Web

Forms -pohjainen julkaisujärjestelmä (Hoefling 2019). Moduulit voivat olla Web Forms, MVC tai SPA (Single Page Application framework, kuten ReactJS) moduuleja. Yksinkertaisten moduulien luomiseen voi käyttää DNN Module Creatoria, joka on osa julkaisujärjestelmän hallintapaneelia. Tällöin Visual Studiota ei tarvita. Luodut ja testatut moduulit voidaan asentaa julkaisujärjestelmän asetuksissa (Kuva 8) (DNN f). (DNN e.) DNN:n pisteet on esitetty taulukossa 4.



Kuva 8. DNN:n moduulien asennussivu (DNN f)

Kriteeri	Pisteet	Peruste	Painokerroin	Loppu-pisteet
Kehitettävyys				
API	2	REST	2	4
Provider	2	On	2	4
Moduuli	2	On	2	4
Hinta	1	Ilmainen	1	1
Tekninen tuki	0,5	On	0,5	0,25
Tunnettuus	2	1.	0,5	1
Ohjelmistokehys	0	Web Forms	3	0
Yhteensä	9,5			14,25

Taulukko 4. DNN Platformin pisteet eriteltynä

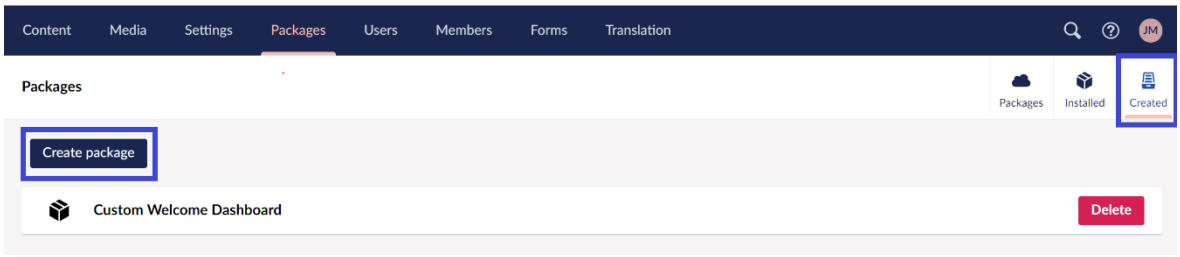
4.2.2 Umbraco

Umbraco CMS on avoimen lähdekoodin ASP.NET Core -pohjainen perinteinen julkaisujärjestelmä (Umbraco a). Umbraco on ladattu yli 5 miljoonaa kertaa (Poudel 2022) ja sen yhteisöön kuuluu yli 220 000 aktiivista jäsentä (Umbraco a). Umbraco on ilmainen, mutta lisätukea saa maksamalla Professional tai Enterprise versiosta. Umbraco Professional maksaa 7500 $ vuodessa ja tarjoaa lisätukea yhden verkkosivun kehittämiseen. Umbraco Enterprisen hinta riippuu verkkosivujen määrästä, ja se tarjoaa lisäksi myös apua arkkitehtuurin suunnittelussa ja ohjelmoinnissa. (Umbraco b.) Githubissa saatavilla olevaa lähdekoodia päivitetään edelleen (Github b).

Julkaisujärjestelmään löytyy tekninen dokumentaatio, keskustelupalsta, yhteisön luomia avoimen lähdekoodin laajennuksia sekä kolmannen osapuolen luomia laajennuksia. Laajennuksia (eng. extension) kutsutaan Umbracossa paketeiksi. Ilmaisia paketteja on yli 1200. Myös Umbraco tarjoaa paketteja erillistä maksua vastaan ja osana maksullisia Professional ja Enterprise versioita. (Umbraco c.) Myös omien pakettien (moduulien) luominen on mahdollista Umbraco Backofficessa (Kuva 9), joka on Umbracon backend tai komentokehotteessa komennolla

```
dotnet new umbracopackage --name CustomWelcomeDashboard
```

, joka luo tyhjän paketin, jolla on nimi. Luodut paketit voidaan julkaista NuGet-paketteina, jolloin ne voidaan asentaa Visual Studiossa tai komentokehotteella kuten kaikki muutkin NuGet-paketit. (Umbraco d.) Umbraco käyttää REST API:a (Umbraco e). Umbracossa omien providereiden luominen onnistuu, sillä se hyödyntää ASP.NET Coren riippuvuusinjektiota (Umbraco f). Umbracon pisteet on eritelty taulukossa 5.



Kuva 9. Moduulin luonti Umbracossa (Umbraco d)

Kriteeri	Pisteet	Peruste	Painokerroin	Loppu-pisteet
Kehitettävyys				
API	2	REST	2	4
Provider	2	On	2	4
Moduuli	2	On	2	4
Hinta	1	Ilmainen	1	1
Tekninen tuki	0,5	On	0,5	0,25
Tunnettuus	1,5	2.	0,5	0,75
Ohjelmistokehys	2	Core	3	6
Yhteensä	11			20

Taulukko 5. Umbracon pisteet eriteltynä

4.2.3 N2

N2CMS on täysin ilmainen avoimen lähdekoodin ASP.NET-pohjainen julkaisujärjestelmä (N2CMS a). N2:ta voidaan käyttää sekä Web Forms- että MVC-projekteissa lataamalla tarvittavat NuGet-paketit tai Github-projektin (N2CMS Read The Docs). N2:n Github-projektia

on viimeksi päivitetty marraskuussa 2018 (Github c), joten julkaisujärjestelmää ei luultavasti tulla päivittämään ASP.NET Corelle.

N2 julkaisujärjestelmään löytyy tekninen dokumentaatio N2CMS:n verkkosivulta, tosin se ei vaikuta kovin kattavalta, ja sisältää muun muassa rikkinäisiä latauslinkkejä lisäosiin, joita on yhteensä 13 (N2CMS b). Lisäksi dokumentaatiota on olemassa myös muilla sivuilla, kuten Confluence-sivulla (Confluence) mutta sekin on puutteellinen sisällöltään. N2CMS:n pisteet on eritelty taulukossa 6.

Kriteeri	Pisteet	Peruste	Painokerroin	Loppu-pisteet
Kehitettävyys				
API	0	Ei tiedossa	2	0
Provider	0	Ei	2	0
Moduuli	0	Ei	2	0
Hinta	1	Ilmainen	1	1
Tekninen tuki	0	Ei	0,5	0
Tunnettuus	0	5.	0,5	0
Ohjelmistokehys	1	MVC	3	3
Yhteensä	2			4

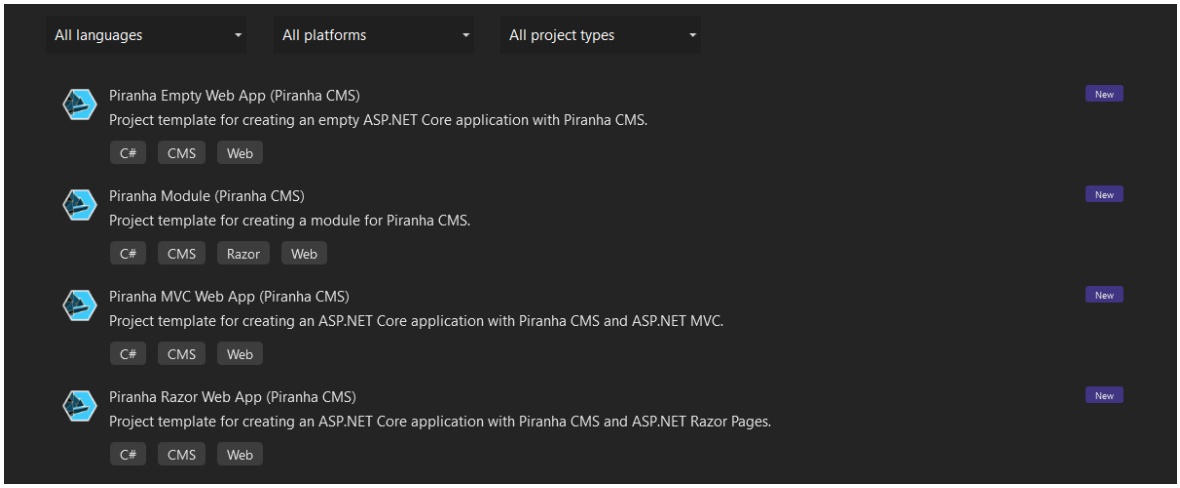
Taulukko 6. N2CMS pisteet eriteltynä

4.2.4 Piranha CMS

Piranha CMS on kevyt avoimen lähdekoodin julkaisujärjestelmä .NET 6:lle, eli julkaisujärjestelmän käyttämä ohjelmistokehys on ASP.NET Core. Arkkitehtuuriltaan julkaisujärjestelmä on decoupled, tosin sitä pystyy käyttämään myös perinteisenä julkaisujärjestelmänä tai headless-arkkitehtuurilla Piranhan pakettien avulla. (Piranha CMS a.) Piranha CMS:n pystyy asentamaan jo olemassa olevaan projektiin tai uuteen projektiin käyttäen valmiita projektipohjia Visual Studiossa (Kuva 10). Projektipohjat voidaan asentaa komentokehotteessa komennolla

```
dotnet new -i Piranha.Templates
```

, jolloin projektipohjat näkyvät Visual Studiossa. (Piranha CMS b) Piranhan lähdekoodia päivitetään edelleen Githubissa (Github d).



Kuva 10. Piranha CMS:n projektipohjat Visual Studiossa

Piranha CMS:ään on tekninen dokumentaatio, ja teknistä tukea voi saada julkaisujärjestelmän Github- ja Gitter-sivuilta (Github d). Omien moduulien luonti on mahdollista NuGet-pakettina tai projektina, jos paketti tulee vain omaan käyttöön (Kuva 10). Moduulien tulee toteuttaa Piranha.Extend.IModule -interface, jotta sen pystyy lisäämään Startup.cs -tiedostossa. (Piranha CMS c) REST API:n käyttämiseksi sisällönhallintaan on Piranha.WebApi-paketti (Piranha CMS d). Piranhassa esimerkiksi autentikointiproviderin pystyy korvaamaan omalla providerilla luomalla toteutuksen ISecurity-interfacesta. Luotu provider voidaan rekisteröidä ConfigureServices-metodissa seuraavalla kutsulla .NET Coren riippuvuusinjektiota hyödyntäen:

```
services.AddSingleton<Piranha.ISecurity, MyCustomAuthService>();
```

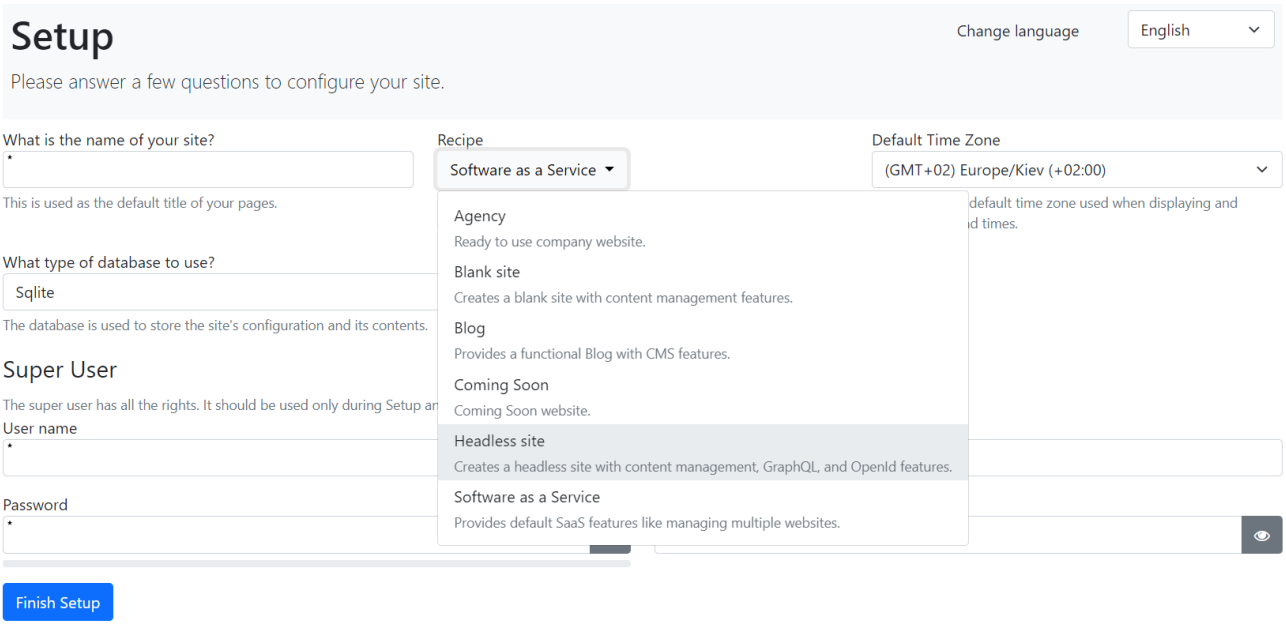
(Piranha CMS e.) Taulukossa 7 on eritelty Piranha CMS:n pisteet.

Kriteeri	Pisteet	Peruste	Painokerroin	Loppu-pisteet
Kehitettävyys				
API	2	REST	2	4
Provider	2	On	2	4
Moduuli	2	On	2	4
Hinta	1	Ilmainen	1	1
Tekninen tuki	0,5	On	0,5	0,25
Tunnettuus	0,5	4.	0,5	0,25
Ohjelmistokehys	2	Core	3	6
Yhteensä	10			19,5

Taulukko 7. Piranha CMS:n pisteet eriteltynä

4.2.5 Orchard Core

Orchard Core CMS on Frameworkin päälle rakennettu avoimen lähdekoodin julkaisujärjestelmä (Github e). Orchard Core CMS tukee perinteistä- (Orchard käyttää tästä termiä ”full”), decoupled- ja headless-julkaisujärjestelmäarkkitehtuuria. (Orchard Core Documentation 2021) Julkaisujärjestelmän käyttötapa ja arkkitehtuuri voidaan määritellä Setup-sivulla (Kuva 11). Käytettäessä Orchard Corea decoupled- arkkitehtuurilla frontendin voi tehdä esimerkiksi ASP.NET Core MVC:llä ja Razor pages:illä. Orchard Core on täysin ilmainen.



Kuva 11. Orchard Coren määrittelysivu.

Orchard Coreen on tekninen dokumentaatio, ja Github-projektia päivitetään edelleen (Github e). Tukea voi saada Github-keskustelupalstan lisäksi myös Orchard Coren Gitter-sivulta. Orchard Coreen on myös saatavilla NuGet-paketteja toiminnallisuuden lisäämiseksi, kuten esimerkiksi OrchardCore SEO, joka sisältää ominaisuuksia hakukoneoptimointiin (NuGet Must Haves). Myös omien moduulien luominen on mahdollista käyttäen joko komentokehotteessa komennolla

```
dotnet new ocmodulecms
```

tai luomalla OrchardCore.Modules kansioon .NET Standard Library-projektin ja lisäämällä siihen OrchardCore.Module.Targets NuGet-paketti. (Marcussen & Savard). Dokumentaation mukaan Orchard Core API toimii GraphQL:llä (Brent & Savard 2021), mutta oman REST API:n lisäämisen pitäisi myös olla mahdollista (Paterson 2022). Orchard Core hyödyntää ASP.NET Coren riippuvuusinjektiota omien providereiden käytössä. Orchard Coren pisteet on eritelty taulukossa 8.

Kriteeri	Pisteet	Peruste	Painokerroin	Loppu-pisteet
Kehitettävyys				
API	0	GraphQL	2	0
Provider	2	On	2	4
Moduuli	2	On	2	4
Hinta	1	Ilmainen	1	1
Tekninen tuki	0,5	On	0,5	0,25
Tunnettuus	1	3.	0,5	0,5
Ohjelmistokehys	2	Core	3	6
Yhteensä	8,5			15,75

Taulukko 8. Orchard Coren pisteet eriteltynä

4.3 Tulokset

DNN:n heikkoutena oli sen käyttämä vanhentunut teknologia, eikä julkaisujärjestelmän kehittäjillä ollut selkeää suunnitelmaa sen päivittämisestä .NET Corelle. N2:n pistemäärät olivat kaikilla osa-alueilla huonot, joten sitäkään ei tarvitse harkita mahdolliseksi uudeksi julkaisujärjestelmäksi. Tulosten tulkinnassa täytyy myös huomioida pisteytysjärjestelmän subjektiivisuus. Painokertoimet kuvastavat ainoastaan eri kriteerien keskeistä tärkeysjärjestystä. Työn tulosten kannalta valituilla painokertoimilla ei kuitenkaan ollut kovin suurta merkitystä, sillä kärkikaksikko olisi ollut sama, vaikka painokertoimia olisikin muokattu, kunhan kriteerien tärkeysjärjestys pysyy samana. Kuten taulukosta 9 näkyy, Umbraco olisi saanut korkeimmat pisteet (11), ja Piranha olisi ollut toisena.

Kriteeri	DNN	Umbraco	Orchard Core	N2CMS	Piran-haCMS
Kehitettävyys	6	6	4	0	6
Hinta	1	1	1	1	1
Tekninen tuki	0,5	0,5	0,5	0	0,5
Tunnettuus	2	1,5	1	0	0,5
Ohjelmistokehys	0	2	2	1	2
Yhteensä	9,5	11	8,5	2	10

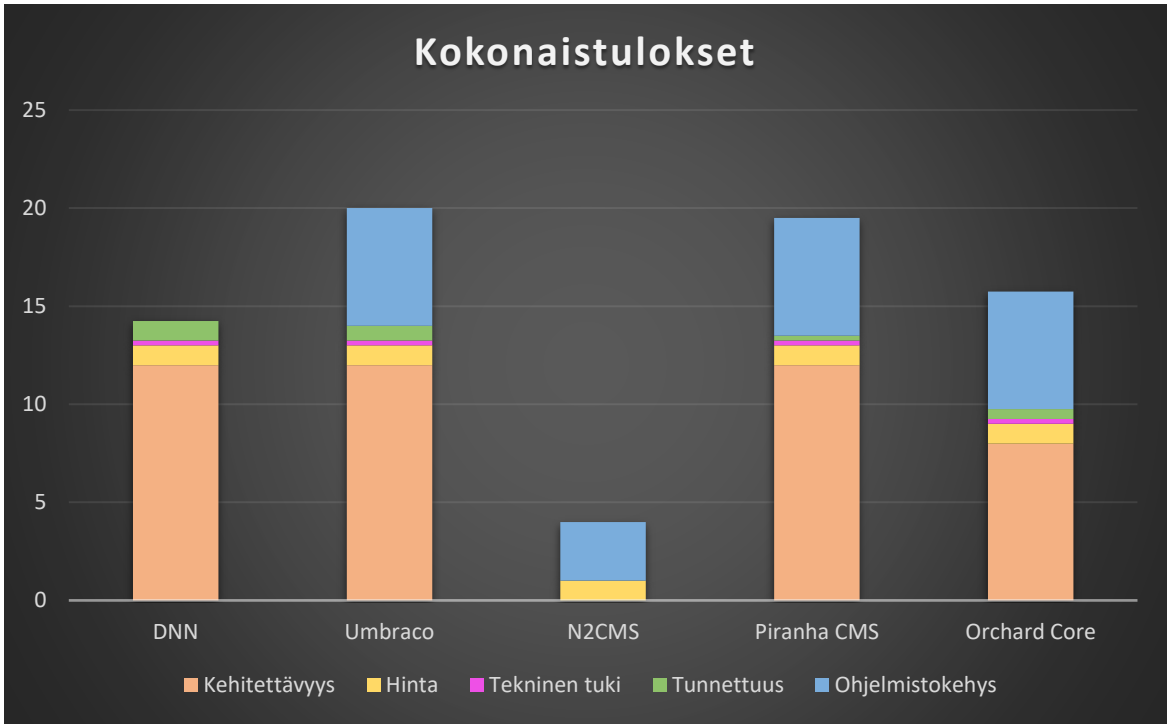
Taulukko 9. Vertailun tulokset ilman painokertoimia

Kokonaispisteet on esitetty taulukossa 10 ja kuvaajassa 1. Vertailusta eniten pisteitä sai Umbraco CMS, tosin toiseksi tullut Piranha CMS hävisi ainoastaan tunnettuudesta saamatta jääneen 0,5 pisteen vuoksi, eli toiminnallisuudeltaan ne ovat hyvin lähellä toisiaan vertailtavien kriteerien suhteen. Molemmissa julkaisujärjestelmissä moduuleja luodaan NuGet-paketteina., tosin Umbracossa moduuleja voidaan luoda myös Backofficessa, kun taas Piranhassa moduuleja voidaan luoda myös Visual Studiossa. Myös teknisen tuen laadussa on eroja, sillä Piranha CMS:ssä on vain Gitter-sivu ja Github-keskustelupalsta, kun taas Umbracossa on chat-palvelu, sähköpostiyhteys asiakaspalveluun, Github-keskustelupalsta sekä Umbracon oma keskustelupalsta. Lisäksi erona on, että Umbraco tarjoaa erilaisia lisäpalveluita maksua vastaan, kun taas Piranha on täysin ilmainen. Umbracoon on yli 1200 ilmaista pakettia (moduulia) toiminnallisuuden lisäämiseksi, kun taas Piranhalle tätä määrää ei ole ilmoitettu.

Orchard Core menetti pisteitä toiminnallisuuden osalta ainoastaan REST API:n puuttumisesta. Kuten aikaisemmin jo mainittiin, Patersonin (2022) mukaan Orchard Coreen pystyisi kuitenkin lisäämään REST API:n. Tällöin julkaisujärjestelmä saisi 19,75 pistettä, joka olisi toiseksi korkein pistemäärä. Dokumentaatiota REST API:n käyttöön Orchard Coressa ei tosin löytynyt. Kuten Piranha CMS:ssä, myöskään Orchard Coressa ei ole maksullisia lisäpalveluita. Omien moduulien lisääminen onnistuu, tosin moduulien uudelleenkäytöstä muissa projekteissa ei vaikuttanut löytyvän tietoa. Teknisen tuen laadun puolesta Orchard Core vastaa Piranhaa, eli sillä on Gitter-sivu ja Github-keskustelupalsta. Orchardissa ei myöskään ole minkäänlaisia maksullisia lisäpalveluita.

Kriteeri	DNN	Umbraco	Orchard Core	N2CMS	Piran-haCMS
Kehitettävyys	12	12	8	0	12
Hinta	1	1	1	1	1
Tekninen tuki	0,25	0,25	0,25	0	0,25
Tunnettuus	1	0,75	0,5	0	0,25
Ohjelmistokehys	0	6	6	3	6
Yhteensä	14,25	20	15,75	4	19,5

Taulukko 10. Vertailun kokonaistulokset



Kuvaaja 1. Vertailun kokonaistulokset

5 Yhteenveto ja pohdinta

Työn tavoitteena oli ehdottaa Avenla Oy:lle ASP.NET-pohjaista julkaisujärjestelmää käyttöön joko aikaisemmin käytettyjen julkaisujärjestelmien rinnalla tai niiden korvaajana. Julkaisujärjestelmien arviointikriteerit olivat tunnettuus, hinta, ohjelmistokehys, kehitettävyys ja teknisen tuen saatavuus. Nämä kriteerit tulivat teknisen tuen saatavuutta lukuun ottamatta toimeksiantajalta. Kehitettävyys jaettiin kolmeen alikriteeriin, julkaisujärjestelmän käyttämään API:in, sekä mahdollisuuteen luoda omia moduuleja ja providereita. Nämä alikriteerit tulivat myös Avenlalta.

Julkaisujärjestelmäehdotuksen aikaansaamiseksi luotiin pisteytysjärjestelmä, jonka avulla kaikki kriteerit ja alikriteerit pisteytettiin. Kriteereille annettiin myös painokertoimet, jotka kuvaavat kriteerien suhteellista tärkeyttä. Painokertoimet hyväksytettiin toimeksiantajalla.

Vertailussa korkeimmat pisteet sai Umbraco-julkaisujärjestelmä, joka sai täydet pisteet toiminnallisuuden kannalta merkittävissä arvioitavissa kriteereissä. Umbraco menetti pisteitä ainoastaan tunnettuudessa, johtuen työssä käytetystä suhteellisesta pisteytystavasta tunnettuudessa. Piranha CMS jäi toiseksi pelkästään tunnettuudesta menetettyjen pisteiden vuoksi, joten sekin on harkitsemisen arvoinen vaihtoehto. Toisaalta Umbraco oli Piranhaa parempi osa-alueilla, joita ei pisteytetty, kuten teknisen tuen laadussa, ja toisin kuin Piranhassa, Umbracossa on myös mahdollisuus ostaa lisätukea.

Työn tulosten hyödynnettävyyden kannalta voidaan pitää ongelmallisena pisteytyksen painokertoimien subjektiivisuutta, valittuja arviointikriteerejä ja tunnettuuden arviointia. Painokertoimia säätämällä voidaan saada hyvin erilaiset tulokset. Ilman kertoimia DNN olisi saanut kolmanneksi korkeimmat pisteet, mutta Umbraco olisi saanut eniten pisteitä, ja Piranha olisi ollut edelleen toisena. Näin ollen voidaan sanoa, että valitut painokertoimet eivät vaikuttaneet siihen, mikä julkaisujärjestelmä saa eniten pisteitä.

Toisena työn tulosten kannalta ongelmallisena asiana on valitut kriteerit, vaikka kriteerit tulevatkin toimeksiantajalta. Kriteerit eivät varsinaisesti ole yleisesti käytettyjä ”hyvän” julkaisujärjestelmän arvioitavia ominaisuuksia. Laajentamalla näitä kriteerejä esimerkiksi lisäämällä teknisen tuen laadun ja julkaisujärjestelmän arkkitehtuurin arvioitaviin kriteereihin olisi voitu saada erilaiset tulokset, tai ainakin selkeämpi ero korkeimpien pisteiden saaneiden julkaisujärjestelmien välille.

Kolmantena, mutta suhteellisen vähäisenä ongelmana työn tulosten kannalta on tunnettuuden pisteytys. Julkaisujärjestelmien tunnettuudelle ei ole yleisesti hyväksyttyä mittaustapaa tai määritelmää. Lisäksi kaikille vertailtaville julkaisujärjestelmille ei löytynyt vertailukelpoisia

tietoja, kuten sitä, kuinka moni verkkosivu käyttää kyseistä julkaisujärjestelmää. Tämä ei tosin juurikaan vaikuta tuloksiin, sillä tunnettuudella on pieni painokerroin.

Vertailun tulosten perusteella ehdotetaan käyttöönotettavaksi Umbraco-julkaisujärjestelmää. Umbracossa pitäisi olla toimeksiantajan vaatimusten kannalta tärkeät ominaisuudet, joten siihen ei tarvinne tehdä muutoksia, kuten esimerkiksi REST API:n lisääminen. Koska järjestelmän opettelu vie varmasti jonkin verran aikaa, sitä ei voi tässä vaiheessa suositella korvaamaan aiemmin käytössä olleita DNN- ja Episerver-julkaisujärjestelmiä. Mikäli Umbraco otetaan käyttöön Avenlalla, seuraava askel voisikin olla julkaisujärjestelmän kouluttaminen henkilöstölle.

Lähteet

Abto Software. ASP.NET MVC vs ASP.NET Core — Why Use It for Web Development? Viitattu 30.11.2021. Saatavissa https://www.abtosoftware.com/blog/why-use-asp-net-mvc-framework-for-web-application-development

Academind. 2019. REST vs GraphQL - What's the best kind of API?. Youtube-video. Viitattu 7.3.2022. Saatavissa https://www.youtube.com/watch?v=PeAOEAmR0D0

Altvater, A. 2017. SOAP vs. REST: The Differences and Benefits Between the Two Widely-Used Web Service Communication Protocols. Stackify. Viitattu 9.5.2022. Saatavissa https://stackify.com/soap-vs-rest/

Avraham, S. B. 2017. What is REST — A Simple Explanation for Beginners, Part 2: REST Constraints. Medium. Viitattu 15.1.2022. Saatavissa https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-2-rest-constraints-129a4b69a582

Brent, Savard, J. 2021. GraphQL. Viitattu 7.3.2022. Saatavissa https://docs.orchard-core.net/en/dev/docs/reference/modules/Apis.GraphQL/

Brightspot. 2018. The pros and cons of headless CMS, decoupled CMS and coupled CMS architecture. Viitattu 12.4.2022. Saatavissa https://www.brightspot.com/learn/articles/de-coupled-cms-and-headless-cms-platforms

Camparanio, R. 2019. SOAP and GraphQL should we give it a REST? Viitattu 2.5.2022. Saatavissa https://medium.com/dvt-engineering/soap-and-graphql-should-we-give-it-a-rest-fc8fafee6cc7

Climb. What does SEO ready mean? Viitattu 11.4.2022. Saatavissa https://climbcreative.co.uk/blog/what-does-seo-friendly-mean

Confluence. n2cms. Viitattu 23.11.2021. Saatavissa https://n2cmsdocs.atlassian.net/wiki/spaces/N2CMS/overview

DNN a. DNN Open Source Platform Technology. Viitattu 24.1.2022. Saatavissa https://www.dnnsoftware.com/platform/start/architecture

DNN b. Products. Viitattu 22.1.2022. Saatavissa https://www.dnnsoftware.com/products

DNN c. Customer Support. Viitattu 24.1.2022. Saatavissa https://www.dnnsoftware.com/services/customer-support

DNN d. Providers reference. Viitattu 24.1.2022. Saatavissa https://www.dnnsoftware.com/docs/developers/providers.html#top-providers__authprov

DNN e. Creating a Module. Viitattu 24.1.2022. Saatavissa https://www.dnnsoftware.com/docs/developers/creating-modules/create-module.html

DNN f. Installing Modules. Viitattu 25.1.2022. Saatavissa https://www.dnnsoftware.com/docs/developers/extensions/install-extension.html

Drupal. 2021. Editing a content item. Viitattu 20.12.2021. Saatavissa https://www.drupal.org/docs/user_guide/en/content-edit.html

GeeksForGeeks 2020. HATEOAS and Why It's Needed in Successful REST API? Viitattu 9.5.2022. Saatavissa https://www.geeksforgeeks.org/hateoas-and-why-its-needed-in-restful-api/

Github a. Viitattu 20.1.2022. Saatavissa https://github.com/dnnsoftware/Dnn.Platform

Github b. Viitattu 23.11.2021 Saatavissa https://github.com/umbraco/Umbraco-CMS

Github c. Viitattu 23.11.2021. Saatavissa https://github.com/n2cms/n2cms

Github d. Viitattu 15.3.2022. Saatavissa https://github.com/PiranhaCMS/piranha.core

Github e. Viitattu 17.5.2021 Saatavissa https://github.com/OrchardCMS/OrchardCore

Google Trends. Viitattu 28.3.2022. Saatavissa https://trends.google.com/trends/explore?cat=31&q=piranha%20cms,n2cms,Umbraco,DNN,Orchard%20core

GraphQL. Queries and Mutations. Viitattu 8.3.2022. Saatavissa https://graphql.org/learn/queries

Hoefling, A. 2019. DNN Dependency Injection: .NET Core. Andrew Hoefling's blog. Viitattu 11.4.2022. Saatavissa https://www.andrewhoefling.com/Blog/Post/dnn-dependency-injection

IBM. 2022. SOAP Headers. Viitattu 26.4.2022. Saatavissa https://www.ibm.com/docs/en/baw/20.x?topic=services-soap-headers

IBM Cloud Education. 2020. Application Programming Interface. IBM. Viitattu 6.12.2021. Saatavissa https://www.ibm.com/cloud/learn/api

JavaTpoint. ASP.NET Web Forms. Viitattu 29.11.2021. Saatavissa https://www.javatpoint.com/asp-net-web-form-introduction

Kumar, S. 2020. Difference Between Library and Framework. C#Corner. Viitattu 18.4.2022. Saatavissa https://www.c-sharpcorner.com/uploadfile/a85b23/framework-vs-library/

Larkin K., Smith S. & Dahler, B. 2022. Dependency injection in ASP.NET Core. Microsoft. Viitattu 9.4.2022. Saatavissa https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0

Marcussen, D. & Savard, J. 2021. Code Generation Templates. Orchard Core Documentation. Viitattu 14.12.2021. Saatavissa https://docs.orchardcore.net/en/dev/docs/getting-started/templates/#create-a-new-cms-module

Microsoft. 2011. ASP.NET 2.0 Provider Model: Introduction to the Provider Model. Viitattu 9.4.2022. Saatavissa https://docs.microsoft.com/en-us/previous-versions/aa479030(v=msdn.10)#goals-of-the-provider-model

Microsoft a. An introduction to Blazor for ASP.NET Web Forms developers. Viitattu 29.11.2021. Saatavissa https://docs.microsoft.com/en-us/dotnet/architecture/blazor-for-web-forms-developers/introduction

Microsoft b. ASP.NET Support Policy. Viitattu 30.11.2021. Saatavissa https://dotnet.microsoft.com/platform/support/policy/aspnet

Microsoft c. .NET Framework Support Policy. Viitattu 29.11.2021. Saatavissa https://dotnet.microsoft.com/platform/support/policy/dotnet-framework

Minaev, A. 2022. CMS Market Share 2022: Usage Statistics of Popular Technologies. First Site Guide. Viitattu 2.5.2022. Saatavissa https://firstsiteguide.com/cms-stats/

N2CMS a. Features. Viitattu 23.11.2021. Saatavissa https://n2cms.com/features.html

N2CMS b. Add-ons. Viitattu 14.3.2022. Saatavissa https://n2cms.com/add-ons.html

N2CMS Read The Docs. Setting up your environment. Viitattu 23.11.2021. Saatavissa https://n2cms.readthedocs.io/setting-up-your-environment/

Nance, J. 2017. .NET Core Dependency Injection. Stackify. Viitattu 9.4.2022. Saatavissa https://stackify.com/net-core-dependency-injection/

NuGet Must Haves. Top 20 NuGet orchardcore Packages. Viitattu 23.11.2021. Saatavissa https://nugetmusthaves.com/Tag/orchardcore

Optimizely. What is a content management system (CMS)? Viitattu 2.5.2022. Saatavissa https://www.optimizely.com/optimization-glossary/content-management-system/

O'Reilly. FINDING AND INSTALLING UMBRACO MODULES. Viitattu 14.12.2021. Saatavissa https://www.oreilly.com/library/view/umbraco-users-guide/9780470560822/chap1-sec21.html

Orchard Core Documentation. 2021. Orchard Core. Viitattu 20.12.2021. Saatavissa https://docs.orchardcore.net/en/latest/

Paterson, M. 2022. OrchardCMS/OrchardCore. Gitter-keskustelu. Lähetetty 7-8.3.2022.

Piranha CMS a. What Is Piranha. Viitattu 21.3.2022. Saatavissa https://piranhacms.org/docs/master/basics/what-is-piranha

Piranha CMS b. Project Templates. Viitattu 21.3.2022. Saatavissa https://piranhacms.org/docs/master/basics/project-templates

Piranha CMS c. Modules. Viitattu 21.3.2022. Saatavissa https://piranhacms.org/docs/master/extensions/modules

Piranha CMS d. Packages. Viitattu 21.3.2022. Saatavissa https://piranhacms.org/docs/v7/basics/packages

Piranha CMS e. Authentication. Viitattu 22.3.2022. Saatavissa https://piranhacms.org/docs/master/extensions/authentication

Poudel, D. 2022. Top and Best .NET CMS Platforms | .Net Developer Choice. Ourtechroom. Viitattu 28.3.2022. Saatavissa https://ourtechroom.com/fix/top-best-net-cms-platforms/

Red Hat. 2019. What is GraphQL? Viitattu 7.3.2022. Saatavissa https://www.redhat.com/en/topics/api/what-is-graphql

Red Hat. 2020. What is a REST API? Viitattu 15.1.2022. Saatavissa https://www.redhat.com/en/topics/api/what-is-a-rest-api

Roth, D., Anderson, R. & Luttin, S. 2021. Introduction to ASP.NET Core. Microsoft. Viitattu 6.12.2021. Saatavissa https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0

Roy, S. 2022. The Difference Between a Framework and a Library. Baeldung. Viitattu 18.4.2022. Saatavissa https://www.baeldung.com/cs/framework-vs-library

Salokangas, T. 2022. Juhon oppari (C# julkkari). Teams-keskustelu. Lähetetty 31.1-29.3.2022

Sellers, M. 2020. The Technical Future of DNN. DNN Blogs. Viitattu 22.1.2022. Saatavissa https://dnncommunity.org/blogs/Post/7625/The-Technical-Future-of-DNN

Seng, E. 2020. Web Forms in the Age of .NET 5/6+: Planning for the Long Term. Viitattu 29.11.2021. Saatavissa https://blog.inedo.com/dotnet/net5-web-forms

Southern, M. 2021. WordPress Powers 39.5% of All Websites. Search Engine Journal. Viitattu 16.11.2021. Saatavissa https://www.searchenginejournal.com/wordpress-powers-39-5-of-all-websites/391647/

Stoplight. SOAP API. Viitattu 15.1.2022. Saatavissa https://stoplight.io/api-types/soap-api/

Sukesh, M. 2015. ASP.NET MVC vs ASP.NET WebForm Performance Comparison. Code Project. Viitattu 30.11.2021. Saatavissa https://www.codeproject.com/Articles/864950/ASP-NET-MVC-vs-ASP-NET-WebForm-Performance-Compari

Technopedia. Module. Viitattu 25.1.2022. Saatavissa https://www.techopedia.com/definition/3843/module

Tutorialspoint. ASP.NET MVC Razor. Viitattu 30.11.2021. Saatavissa https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_razor.htm

Tyler, C. 2021. ASP.NET MVC Tutorial for Beginners: What is, Architecture. Guru99. Viitattu 30.11.2021. Saatavissa https://www.guru99.com/asp-net-mvc-tutorial.html

Umbraco a. What is a Headless CMS? Viitattu 22.11.2021. Saatavissa https://umbraco.com/knowledge-base/headless-cms/

Umbraco b. Umbraco - the friendly, open source .NET CMS. Viitattu 22.11.2021. Saatavissa https://umbraco.com/umbraco-cms-pricing/

Umbraco c. Umbraco Extensions. Viitattu 22.11.2021. Saatavissa https://umbraco.com/why-choose-umbraco/umbraco-extensions/

Umbraco d. Creating a Package. Viitattu 25.1.2022. Saatavissa https://our.umbraco.com/documentation/extending/Packages/Creating-a-Package/

Umbraco e. Umbraco API Controllers. Viitattu 17.1.2022 Saatavissa https://our.umbraco.com/documentation/reference/routing/webapi/

Umbraco f. Inversion of Control / Dependency injection Viitattu 11.4.2022. Saatavissa https://our.umbraco.com/documentation/reference/using-ioc/

Winkels, N. 2019. The Definitive Guide to CMS Architecture. Web Dev Zone. Viitattu 15.11.2021. Saatavissa https://dzone.com/articles/the-definite-guide-to-cms-architecture

WordPress. Roles and Capabilities. Viitattu 17.5.2022. Saatavissa https://wordpress.org/support/article/roles-and-capabilities/

Liite 1. Julkaisujärjestelmien tunnettuus Google-hakujen perusteella (Google Trends)