Niklas Rötkönen

Angular-kurssi Moodleen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka, monimuoto

Insinöörityö

15.5.2024

Tiivistelmä

Tekijä:	Niklas Rötkönen
Otsikko:	Angular-kurssi Moodleen
Sivumäärä:	34 sivua
Aika:	15.5.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Osaamisaluepäällikkö Janne Salonen

Opinnäytetyössä oli tavoitteena suunnitella ja luoda Angular-perusteet-verkkokurssi Metropolia ammattikorkeakoulun Moodle-oppimisympäristöön. Verkkokurssin tavoitteena oli tarjota opiskelijalle mahdollisuus itsenäisesti tutustua Angular-ohjelmistokehykseen, sen peruskäsitteisiin ja muihin tarvittaviin työkaluihin. Opinnäytetyön yhteydessä opin myös itse lisää Angularista ja sen ominaisuuksista ja eroista muihin sovelluskehyksiin.

Opinnäytetyön teoriaosuudessa tutustutaan Angulariin ja sen ominaisuuksiin sekä vertaillaan eroja Angularin ja AngularJS:n välillä. Lopuksi tutustutaan Angularin käytössä tarvittaviin teknologioihin. Teoria pohjautuu kirjallisuuteen, verkkoartikkeleihin ja viralliseen dokumentaation.

Oppinäytetyön lopputuloksena luotiin Angular-kurssi Moodle-ympäristöön. Kurssin suorittanut opiskelija saa peruskäsityksen Angular-ohjelmistokehyksestä sekä sen yhteydessä tarvittavista työkaluista ja ominaisuuksista. Kurssin jälkeen opiskelijan odotetaan osaavan hyödyntää Angularia omissa projekteissaan.

Avainsanat: Angular, TypeScript, Moodle, ohjelmistokehys

Abstract

Author:	Niklas Rötkönen
Title:	Angular course to Moodle
Number of Pages:	34 pages
Date:	15 May 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Software Engineering
Supervisors:	Janne Salonen, Head of school

The goal of the thesis was to design and create an Angular basics online course for the Moodle learning environment of the Metropolia University of Applied Sciences. The purpose of the online course is to provide students with the opportunity to independently learn about the Angular web application framework, its fundamental concepts, and other necessary tools. As part of the thesis, I also learned about Angular myself and the different features compared to other frameworks.

In the theoretical part of the thesis, Angular and its features are introduced, as well as comparing the differences between Angular and AngularJS. It also covers the technologies required for using Angular. The theory is based on literature, online articles and official documentation.

As a result of the thesis, an Angular course was created in the Moodle environment. Upon completion of the course, the student gains a basic understanding of the Angular web application framework and the tools and features needed in conjunction with it. After the course, the student is capable of utilizing Angular in their own projects.

Keywords: Angular, TypeScript, Framework, Moodle

Sisällys

Lyhenteet

HTML: *Hyper Text Markup.* Merkintäkieli, jota käytetään verkkosivujen rakenteen määrittämiseen.

CSS: *Cascading Style Sheets.* Kieli, jota käytetään verkkosivujen ulkoasun ja tyylin määrittämiseen.

MVW: *Model-View-Whatever.* Arkkitehtuurimalli, jota käytetään ohjelmistosuunnittelussa.

MVC: *Model-View-Controller.* Arkkitehtuurimalli, jota käytetään ohjelmistosuunnittelussa.

MVVM: *Model-View-ViewModel,* Arkkitehtuurimalli jotta käytetään erityisesti käyttöliittymäpohjaisissa sovelluksissa.

RxJS: *Reactive Extensions for JavaScript.* JavaScript-kirjasto, joka tarjoaa työkaluja reaktiivisen ohjelmoinnin toteuttamiseen.

DOM: *Document Object Model.* Tapa, jolla selain esittää HTML-dokumentin rakenteen.

SPA: *Single Page Application.* Verkkosovelluksen tyyppi, joka ladataan kokonaan yhteen HTML-sivuun.

DI: *Dependency Injection.* Suunnittelumalli, joka käsittelee riippuvuuksien hallintaa ohjelmistokehityksessä.

CLI: *Command Line Interface.* Käyttöliittymä, joka mahdollistaa vuorovaikutuksen tietokoneen kanssa komentorivin kautta.

ECMA: *European Computer Manufacturers Association.* Järjestö, joka kehittää ja ylläpitää standardeja tietotekniikan alalla.

NPM: *Node Package Manager.* Paketinhallintajärjestelmä JavaScript-ohjelmointikielelle.

XML: *Extensible Markup Language.* Merkintäkieli, jota käytetään tietojen esittämiseen hierarkkisessa rakenteessa.

SVG: *Scalable Vector Graphics.* XML-pohjainen merkintäkieli, jota käytetään kuva- ja grafiikkatiedostojen luomiseen ja esittämiseen verkkosivuilla.

W3C: *World Wide Web Consortium.* Kansainvälinen standardointiorganisaatio.

1 Johdanto

Nykypäivän digitaaliaikana ohjelmointitaidot ovat tulleet yhä tärkeämmiksi opiskelijoille. Näitä taitoja tarvitaan menestymiseen teknologiaveroisessa maailmassa. Verkkokehityskehysten nopean edistymisen myötä on ratkaisevan tärkeää, että oppilaitokset mukauttavat opetussuunnitelmansa varustaakseen opiskelijat tarvittavilla teknologiataidoilla, joita tarvitaan menestyäkseen nykyaikaisessa työympäristössä. Angular, Googlen ylläpitämä laajalti käytetty JavaScript-ohjelmistokehys, on noussut näkyväksi työkaluksi dynaamisten, reagoivien ja yksisivuisen verkkosovellusten rakentamiseen.

Opinnäytetyössä oli tarkoituksena vastata tarpeeseen ja luoda kattava Angular-ohjelmistokehyksen-verkkokurssi kolmannen asteen opiskelijoille Metropolia Ammattikorkeakoulun Moodle-oppimisympäristöön. Verkkokurssin avulla opiskelijat saavat mahdollisuuden oppia perusohjelmoinnin käsitteitä ja hankkia samalla käytännön kokemusta verkkosovellusten rakentamisesta Angular-ohjelmistokehyksellä.

Verkkokurssin tavoitteena on tarjota opiskelijalle mahdollisuus itsenäisesti tutustua Angular-ohjelmistokehykseen, sen peruskäsitteisiin ja muihin tarvittaviin työkaluihin. Kurssi on tarkoitettu opiskelijoille, jolla on jo kokemusta HTML, CSS ja JavaScript-kielistä. Kurssi on opiskelijoille valinnainen ja sen aikana opiskelija opastetaan asteittain kurssin läpi.

Työn teettäjänä toimi Metropolia Ammattikorkeakoulu, joka on Suomen suuriin ammattikorkeakoulu. Se aloitti toimintansa 1. elokuuta 2008. Metropolia tarjoaa monipuolisia koulutuksia eri aloille ja on tunnettu myös kansainvälisyydestään. Metropoliassa on tarjolla neljä eri pääkoulutushaaraa: kulttuuriala, sosiaali- ja terveysala, tekniikka ja liiketalous.

Opinnäytetyön teoriaosuudessa tutustutaan Angulariin ja sen taustaan. Lisäksi tutustutaan Angularin eri ominaisuksiin kuten mm. komponentteihin, palveluihin

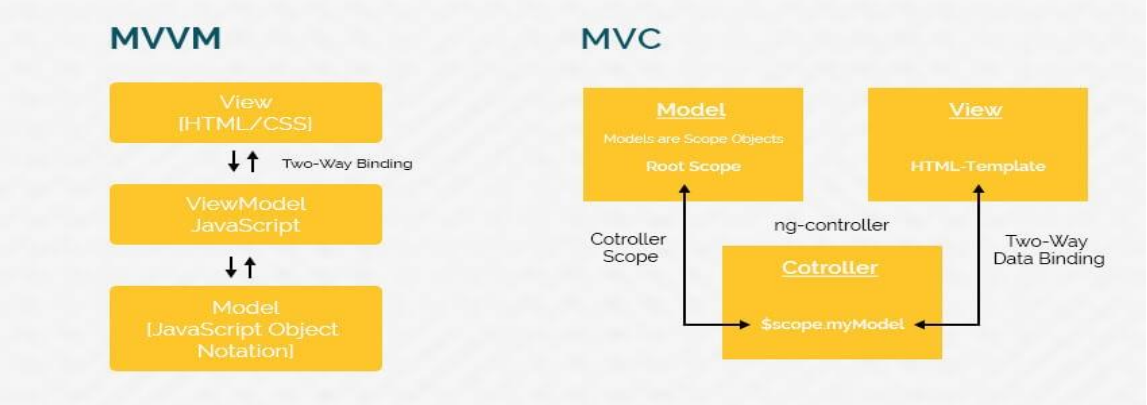
ja moduuleihin. Lopuksi tutustutaan Angularissa tarvittaviin teknologioihin kuten Node.js ja TypeScript. Teoria pohjautuu kirjallisuuteen ja viralliseen dokumentaatioon.

2 Angular

2.1 AngularJS

Googlen kehittämä AngularJS on tehokas lähdekoodin JavaScript-ohjelmistokehys, joka on suunniteltu yksinkertaistamaan dynaamisten yksisivuisten verkkosovellusten rakentamista. AngularJS sai alkunsa vuonna 2009 Google Feedback -projektista. Googlen työntekijä Miško Heveryn väitti pystyvänsä uudelleen kirjoittamaan 17,000 koodiriviä kahdessa viikossa käyttämällä harrastusprojektiaan, avoimen lähdekoodin kirjastoa GetAngularia. Hevery myöhästyi viikolla mutta silti hämmästytti kaikki, kun koodirivit putosivat 17,000:sta vain 1500:aan. (Seshadri & Green 2013.) Tästä alkoi AngularJS:n kehitys Miško Heveryn ja Adam Abronsin toimesta. Ensimmäinen vakaa versio AngularJS:stä julkaistiin 2010. Vuonna 2010 julkaistu kehys sai nopeasti suosiota kehittäjien keskuudessa tehokkaiden ominaisuuksiensa ja intuitiivisen lähestymistapansa ansiosta. AngularJS kirjoitettiin JavaScriptillä, koska JavaScript oli sen aikakauden ensisijainen ohjelmointikieli verkkosovelluksien luomiseen.

AngularJS mullisti verkkokehityksen ottamalla käyttöön MVW (Model-View-Whatever) -arkkitehtuurin konseptin. Mikä merkitsee, että AngularJS:ssä voidaan käyttää joko MVC (Model-View-Controller) tai MVVM (Model-View-Viewmodel) -arkkitehtuuria. Nämä arkkitehtuurit auttavat kehittäjiä koodin järjestämisessä, vastuualueiden erottamisessa ja ylläpidettävyyden parantamisessa.



Kuva 1 MVVM- ja MVC-arkkitehtuurit

AngularJS:llä on ollut suuri vaikutus verkkokehitykseen ja se on antanut kehittäjille mahdollisuuden luoda dynaamisia ja interaktiivisia verkkosovelluksia helposti ja nopeasti. AngularJS tarjoaa kattavan ohjelmistokehyksen verkkosovellusten rakentamiseen, sillä se sisältää ominaisuuksia, kuten kaksisuuntainen tiedonsidonta, direktiivit, riippuvuuden lisäys, reititys ja palvelut. Sen intuitiivinen suunnittelu, kattava dokumentaatio ja aktiivinen yhteisön tuki teki siitä suositun kehittäjille maailmanlaajuisesti. AngularJS on tullut käyttöikänsä päähän ja sen tuki on lopetettu. AngularJS:n viimeinen versio (1.8.3) julkaistiin 7.huhtikuuta 2022 ja se korvautui uudelleenkirjoitetulla versiolla Angular 2, jota kutsutaan pelkästään Angulariksi.

2.2 Angular 2

Uudelleenkirjoitettu Angular 2 -ohjelmistokehys edustaa merkittävää kehitystä edeltäjästään, AngularJS:stä. Vuonna 2016 julkaistu Angular 2 esitteli kehyksen täydellisen uudelleenkirjoituksen, joka sisältää nykyaikaiset verkkokehityskäytännöt ja tarjoaa modulaarisemman, skaalautuvamman ja tehokkaamman ratkaisun verkkosovellusten rakentamiseen.

Angular 2 on Googlen kehittämä kattava ohjelmistokehys dynaamisten ja monipuolisten verkkosovellusten rakentamiseen. Neton [2024] mukaan Angular on turvallinen valinta ohjelmointikehyksenä Googlen tukemisen takia, erityisesti suurille yrityksille. Toisin kuin edeltäjänsä, JavaScriptiin perustuva AngularJS, Angular 2 on rakennettu TypeScriptillä, JavaScriptin supersarjalla, joka lisää kieleen staattista kirjoittamista ja muita edistyneitä ominaisuuksia. Angular 2:ssa on korjattu useita AngularJS:n puutteita ja se mukautuu nykyaikaisen verkkokehityksen parhaisiin käytäntöihin.

Angular 2 edustaa merkittävää harppausta verkkokehityksessä, tarjoten nykyaikaisen ja kattavan kehyksen kestävien verkkosovellusten rakentamiseen. Sisäänrakennetut työkalut ja ominaisuudet helpottavat responsiivisten ja mobiilioptimoitujen käyttöliittymien rakentamista. Angularin ominaisuuksiin kuuluu mm. komponenttipohjainen arkkitehtuuri, parantunut suorituskyky, TypeScript-

integraatio, riippuvuusinjektio, alustariippumattomuus, komentoliittymä sekä selain- että mobiilituki. Näiden ansiosta Angular 2 antaa kehittäjille mahdollisuuden luoda skaalautuvia, ylläpidettäviä ja tehokkaita sovelluksia verkkoon.

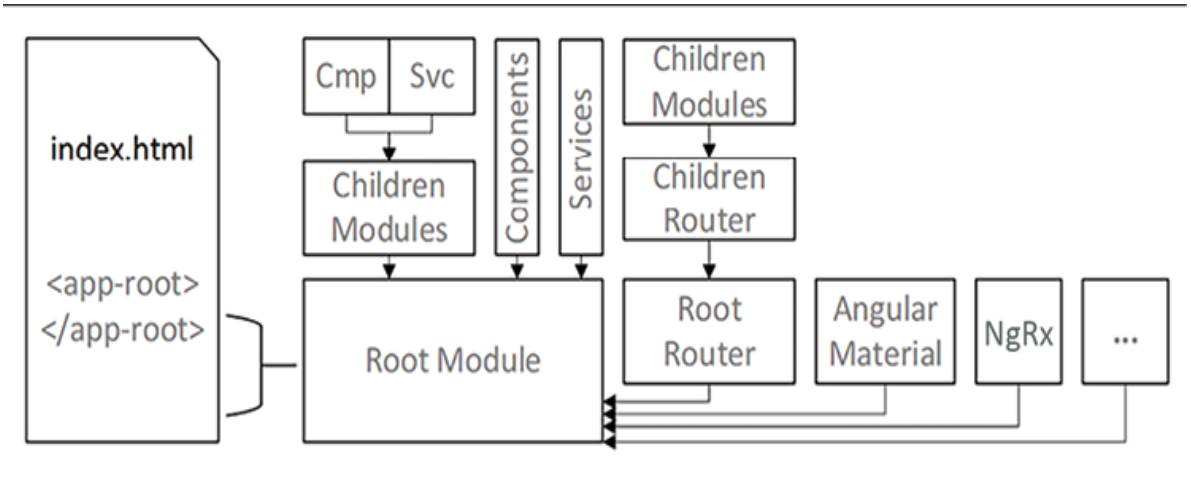
Angular on edistynyt paljon siten vuodesta 2016. Uusin versio Angularista (17.3.0) julkaistiin 13. maaliskuuta 2024. Yksi Angularin suurimmista eduista on sen säännöllinen 6 kuukauden välein tapahtuva pääjulkaisu. Tämän ajanjakson sisälle mahtuu noin 1–3 pienempää version julkaisua. Ulucanin [2024] mukaan tällä voi kuitenkin olla haittapuolia, ominaisuuksia ei ole testattu tai ne ovat keskeneräisessä tilassa vähäisen ajan vuoksi. Niin kauan kuin Google jatkaa Angularin kehittämistä, tulevaisuus näyttää valoisalta, vaikka muita kehyksiä kehitetään jatkuvasti.

3 Angularin ominaisuudet

3.1 Modulaarinen arkkitehtuuri

Angularin modulaarinen arkkitehtuuri, joka tunnetaan myös nimellä komponenttipohjainen arkkitehtuuri on suunnittelutapa, jossa sovellus koostuu itsenäisistä ja uudelleenkäytettävistä, riippuvuuksista, komponenteista ja palveluista. Nämä ovat usein organisoitu ja julistettu moduuleissa paremman organisoinnin takaamiseksi. Angular on moderni ohjelmointikehys, joka korostaa voimakkaasti tätä arkkitehtuuria skaalautuvien, ylläpidettävien ja laajennettavien sovellusten rakentamisessa.

Modulaarinen arkkitehtuuri, joka perustuu komponentteihin ja moduuleihin, on olennainen osa Angular-kehitystä. Omaksumalla tämän arkkitehtuurin kehittäjät voivat rakentaa skaalautuvia, ylläpidettäviä ja laajennettavia sovelluksia. Komponentit kapseloivat käyttöliittymäelementtejä ja niihin liittyvää logiikkaa. Moduulit puolestaan järjestävät toiminnallisuuden yhtenäisiksi yksiköiksi, mikä edistää uudelleenkäytettävyyttä, kapselointia ja riippuvuuden hallintaa. Tämä lähestymistapa ei ainoastaan paranna kehitystehokuutta, vaan myös parantaa Angular-sovellusten yleistä laatua ja luotettavuutta.



Kuva 2 Angularin käynnistyprosessi (Uluca 2024)

3.2 Komponentit

Angular komponentit ovat Angular-sovellusten käyttöliittymien rakennuspalikoita. Komponentti kapseloi osan käyttöliittymää ja siihen liittyvää logiikkaa. Komponentti koostuu TypeScript-luokasta, joka sisältää komponentin logiikan ja ominaisuudet sekä mallin, joka määrittää komponenttinäkymän HTML-rakenteen. Luokka on koristeltu @Component-dekoraattorilla, joka sisältää metatietoja kuten komponentin valitsijan, mallin ja tyylin.

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'test';
}
```

Kuva 3 Angular-komponentti

TypeScript-luokassa ominaisuudet sisältävät tietoja, jotka on sidottu mallin dynaamista renderöintiä varten. Menetelmät puolestaan määrittävät komponentin toiminnan ja käsittelevät käyttäjien vuorovaikutuksia.

HTML-malli määrittää komponentin käyttöliittymän asettelun ja rakenteen. Se sisältää mm. datasidoksia, ohjeita ja tapahtumasidoksia dynaamisen sisällön hahmottamiseksi. Angularin tehokas mallinnusmoottori mahdollistaa eri ominaisuuksia kuten interpoloinnin, ominaisuuksien sitomisen, tapahtumien sitomisen ja rakennedirektiivit. Nämä helpottavat monipuolisten ja interaktiivisten käyttöliittymien luomista.

Valitsija on kuin HTML-tunniste, joka tunnistaa mallien sisältämät komponentit. Se mahdollistaa komponenttien käytön toisissa komponenttien malleissa.

Valitsijalla on keskeinen rooli Angular-kehityksessä tarjoamalla deklaratiivisen syntaksin sovelluksen käyttöliittymän rakenteen määrittelemiseksi.

Angular-komponenteilla on keskeinen rooli Angular-sovellusten rakentamisessa, minkä ansiosta kehittäjät voivat luoda dynaamisia, interaktiivisia ja skaalautuvia käyttöliittymiä helposti ja nopeasti. Komponentteja voi koota ja upottaa toisiinsa monimutkaisten käyttöliittymien luomiseksi. Komponentit ovat suuri syy miksi Angular täyttää nykyaikaisen verkkokehityksen vaatimukset.

3.3 Moduulit

Angular-moduulit ovat Angular-sovelluksien perusrakenteita. Ne toimivat säiliöinä, jotka järjestävät ja kapseloivat komponentteja, palveluita, direktiivejä ja muita toiminallisuuksia. Ne palvelevat toimintojen eristämistä ja tekevät sovelluksista modulaarisempia, ylläpidettävämpiä ja skaalautuvampia.

Angular-moduulit ovat pohjimmiltaan TypeScript-luokkia, joihin on merkitty @NgModule-dekoraattori. Tämä dekoraattori tarjoaa metatietoja, jotka kuvaavat moduulin komponentteja kuten direktiivejä ja palveluita. Moduulit voivat tuoda toimintoja muista moduuleista tuontitaulukon avulla, mikä mahdollistaa resurssien jakamisen sovelluksen eri osien välillä.

Angular-moduuleita on kahta päätyyppiä, ominaisuusmoduulit ja juurimoduulit. Ominaisuusmoduulit ryhmittelevät yhteen liittyviä toimintoja kuten käyttäjän todennusta tai tuotelistauksia, kun taas juurimoduulit edustavat sovelluksen aloituspistettä.

Yksi Angular-moduulien tärkeimmistä eduista on niiden kyky hallita riippuvuuksia. Moduulit ilmoittavat tarjoamansa palveluntarjoajat, jolloin palvelut ja muut resurssit ovat käytettävissä koko sovelluksen ajan. Tämä edistää löysää kytkentää ja helpottaa huoltoa ja testausta.

Angular-moduulit tukevat myös ”laiskalatausta”, tekniikkaa, joka parantaa sovelluksen suorituskykyä lataamalla moduuleja vain silloin, kun niitä tarvitaan. Tämä

on erityisen hyödyllistä suurissa sovelluksissa, joissa on monia ominaisuuksia, koska se lyhentää alkulatausaikaa lataamalla moduuleja asynkronisesti.

Angular-moduulit ovat välttämättömiä Angular-sovellusten strukturoinnissa. Ne edistävät modulaarisuutta, kapselointia ja riippuvuuden hallintaa, mikä helpottaa monimutkaisten sovellusten rakentamista ja ylläpitoa parantaen samalla suorituskykyä ja skaalautuvuutta. Angular on kuitenkin siirtymässä pois moduulien käytöstä. Tilalle on tuotu itsenäisiä komponentteja, joilla on lisäominaisuuksia. Se on komponentin ja NgModulen yhdistelmä. Custodian [2022] mukaan itsenäisten komponenttien lisäys Angularin on merkittävin parannus Angularissa vuosiin.

3.4 Direktiivit

Angular-direktiivit laajentavat HTML-syntaksia ja lisäävät elementteihin ja komponentteihin dynaamista käyttäytymistä. Niiden avulla kehittäjät voivat luoda uudelleenkäytettäviä, modulaarisia koodinpätkiä, jotka voivat käsitellä DOM:ia, soveltaa tyylejä, käsitellä tapahtumia ja olla vuorovaikutuksessa tietojen kanssa. Angular tarjoaa useita erityyppisiä direktiivejä, joista jokainen palvelee tiettyä tarkoitusta. Sisäänrakennettujen direktiivien lisäksi Angular antaa kehittäjille mahdollisuuden luoda omia mukautettuja direktiivejä. Mukautetut Direktiivit voidaan luoda @Directive-dekoratorilla ja ne voivat olla joko attribuutti direktiivejä tai rakenteellisia direktiivejä.

3.4.1 Komponentti direktiivi

Komponentti direktiivit ovat periaatteessa yksinkertaisia komponentteja mallin kanssa. Ne on koristeltu @Component-dekoratorilla, joka koostuu mallista, siihen liittyvistä CSS-tyyleistä ja TypeScript-koodista. Ne kapseloivat osan käyttöliittymästä sen käyttäytymisen kanssa ja niitä voidaan käyttää uudelleen koko sovelluksessa. Komponentit ovat yleisin Angular-kehityksessä käytetty direktiivityyppi.

3.4.2 Attribuutti direktiivi

Attribuutti direktiivit muokkaavat DOM:in elementin käyttäytymistä tai ulkonäköä ottamalla käyttöön lisäkäyttäytymisiä tai tyylejä. Niitä käytetään attribuutteina HTML-elementeissä ja ne voivat reagoida elementin ominaisuuksien muutoksiin. Esimerkkejä Angularin attribuutti direktiiveistä ovat NgClass, NgStyle ja NgModel. NgModel-direktiiviä käytetään esimerkiksi ensisijaisesti syöttökenttien arvojen sitomiseen komponentin, eli se mahdollistaa kaksisuuntaisen tiedonsidonnan näkymän ja komponentin välillä.

3.4.3 Rakeenteelliset direktiivit

Rakenteelliset direktiivit muokkaavat DOM:in rakennetta lisäämällä tai poistamalla elementtejä tietyin ehdoin. Niiden etuliitteenä on tähti (*) mallin syntaksissa. Esimerkkejä Angularin rakenteellisista direktiiveistä ovat *NgIf, *NgFor ja *NgSwitch. *NgIf-direktiivi esimerkiksi mahdollistaa elementin näyttämisen tai piilottamisen perustuen annettuun ehtoon.

3.5 Reititys

Angular-reititys viittaa toimintoon, jolla navigoidaan eri näkymien tai sivujen välillä yksisivuisessa sovelluksessa (SPA). Angular tarjoaa tehokkaan ja joustavan reititysjärjestelmän, jonka avulla kehittäjät voivat määrittää navigointipolut, ladata komponentteja dynaamisesti ja käsitellä reittiin liittyviä toimintoja kuten reittiparametreja ja ehtoja.

Angularin reititys mahdollistaa navigoimisen ilman koko sivun uudelleenlatausta. Tämä tarjoaa sujuvan käyttökokemuksen, joka muistuttaa perinteisiä monisivuisia verkkosivuja. Reitityksen avulla sovellus voidaan järjestää erillisiin näkymiin tai komponentteihin, joista jokainen vastaa tietystä ominaisuudesta tai toiminnallisuudesta. Reitittimen määritelmä koostuu tyypillisesti polusta ja komponentista.

```
export const routes: Routes = [
    { path: "", component:HomeComponent},
    { path: "Shopping-list", component:ShoppingListComponent}
];
```

Kuva 4 Reitityksen määritys

RouterOutlet-direktiivi <router-outlet> toimii paikanvaraajana, jolla Angular renderöi dynaamisesti sopivan komponentin nykyisen reitin perusteella. RouterOutlet on osa RouterModule-moduulia, joka pitää importoida RouterOutletiä käytettäessä. Navigointi sovelluksen eri näkymien välillä onnistuu routerLink-direktiivin avulla, joka tavallisesti upotetaan esim. <a>-elementtiin.

3.6 Palvelut

Angular-palvelut ovat peruskäsite, jota käytetään uudelleenkäytettävien toimintojen kapseloimiseen. Palvelut voidaan jakaa sovelluksen komponenttien ja muiden palveluiden kesken. Ne tarjoavat tavan keskittää yleisiä tehtäviä, kuten tietojen hakemista, liiketoimintalogiikkaa ja vuorovaikutusta ulkoisten sovelluksiittymien kanssa. Angular-palvelut ovat TypeScript-luokkia, jotka on koristeltu @Injectable()- dekoraatiolla. Tämä dekoraattori ilmaisee, että luokalla voi olla omat riippuvuutensa, joita on injektoitava tai että luokka voidaan injektoida muihin luokkiin riippuvuutena.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class ProductService {
  itemArray: { content: string, completed: boolean}[] = [
    {content: "Jauheliha",completed: false},
    {content: "Kurkku",completed: false},
    {content: "Tomaatti",completed: false}
    ];
  addItem(itemText: string){
    this.itemArray.push({content: itemText, completed: false});
  }
  getItems(){
    return this.itemArray;
  }
  deleteItem(index: number){
    this.itemArray.splice(index,1);
  }
  productComplete(index: number){
    this.itemArray[index].completed = !this.itemArray[index].completed
  }
  editProduct(index: number, newProduct: string) {
    if (newProduct.trim()) {
      this.itemArray[index].content = newProduct;
    }
  }
}
```

Kuva 5 Kurssiin luotu palvelu

Palvelut edistävät koodin uudelleenkäyttöä ja järjestävät sitä pienempiin, modulaarisiin osiin, mikä helpottaa koodin ymmärtämistä ja ylläpitoa. Palvelut kapseloivat yhteisiä toimintoja, joita voidaan käyttää useissa sovelluksen komponenteissa tai moduuleissa. Palvelut myös mahdollistavat selkeän erottelun tarjoamalla keskitetyn sijainnin jaetuille toiminnoille, mikä vähentää päällekkäisyyksiä ja parantaa koodin ylläpidettävyyttä.

3.7 Riippuvuusinjektio

Angularin riippuvuusinjektio (DI) on suunnittelumalli, joka helpottaa riippuvuuksien luomista ja hallintaa sovelluksen sisällä. Se on tekniikka, jossa yksi objekti injektoi riippuvuudet toisesta objektista sen sijaan, että objekti loisi riippuvuutensa suoraan. Angularissa riippuvuusinjektio vastaa tarvittavien riippuvuuksien injektoimisesta komponenteille, palveluille, direktiiveille ja muille sovelluksen objekteille.

Kehittäjät voivat hyödyntää DI:tä Angularissa käyttämällä @Injectable-dekoraattoria injektoitavan palvelun merkitsemiseen. @Injectable-dekoraattori tarvitsee joko palveluntarjoajan tai sen voi tarjota suoraan komponenttiin. Palveluntarjoajalla voi määrittää, mistä Angular tarjoaa palvelun tai objektin esim. juuri injektorista. Angular CLI:tä käytettäessä palveluntarjoaja on oletuksena juuri injektorissa. Komponenttitasolla riippuvuudet tarjotaan @Component- dekoraattorissa provider-listan avulla. Komponentti tarvitsee konstruktor-luokan riippuvuuksien ilmoittamiseen tai inject-menetelmän riippuvuuden hyödyntämiseen.

```
constructor(private productService: ProductService){
  this.products= this.productService.getItems();
}
```

Kuva 6 Palvelun injektointi komponenttiin

Pohjimmiltaan Angularin riippuvuusinjektio toimii erottamalla komponentit riippuvuuksistaan ja tarjoaa vankan ja joustavan mekanismin riippuvuuksien hallintaan Angular-sovelluksissa. Riippuvuusinjektio edistää Angularissa modulaarisuutta, ylläpidettävyyttä ja testattavuutta. Riippuvuusinjektion avulla kehittäjät voivat rakentaa kestäviä ja skaalautuvia sovelluksia helposti.

3.8 Kaksisuuntainen datasidonta

Angularin kaksisuuntainen tiedonsidonta on tehokas ominaisuus, joka mahdollistaa tietojen synkronoinnin mallin ja näkymän välillä. Tämä tarkoittaa sitä että

malliin tehdyt muutokset näkyvät automaattisesti näkymässä ja päinvastoin ilman erityistä tapahtumakäsittelyä tai DOM-manipulaatiota.

Kaksisuuntainen tiedonsidonta yksinkertaistaa kehitysprosessia poistamalla manuaalisen DOM-käsittelyn tai tapahtumien käsittelyn tarpeen. Kehittäjät voivat keskittyä sovelluksensa logiikan rakentamiseen, tarvitsematta huolehtia käyttöliittymän manuaalisesta päivittämisestä. Lisäksi se parantaa käyttökokemusta antamalla välitöntä palautetta ja varmistamalla, että käyttöliittymä pysyy synkronoituna taustalla olevien tietojen kanssa.

Angular mahdollistaa kaksisuuntaisen tiedonsidonnan ominaisuussidonnalla ja tapahtumasidonnalla. Ominaisuuksien sidonta muodostaa yhteyden komponenttiluokasta näkymään, jolloin data voi virrata mallista malliin. Elementin ominaisuuden sitominen merkitään hakasuluilla [], mikä identifioi ominaisuuden kohteen. Tapahtumasidonta puolestaan mahdollistaa tietojen virtauksen näkymästä komponenttiluokkaan tallentamalla käyttäjän tapahtumia, kuten näppäinpainalluksia tai hiiren napsautuksia. Elementin tapahtuman sitominen merkitään kaarisuluilla (), joiden sisään syötetään tapahtuma kuten esim. click. Sidonnan jälkeen määritellään mikä menetelmä suoritetaan tapahtuman seurauksena. Angularissa käytetään ensisijaisesti [(ngModel)] -direktiiviä kaksisuuntaiseen tiedonsidontaan. [(ngModel)] -direktiivi yhdistää ominaisuuden sidonnan ja tapahtumansidonnan.

```
<h4>Ostoslista</h4>
<div class="create-new-item-container">
    <input class="new-item-input" type="text" placeholder="Lisää uusi tuote..." [(ngModel)]="itemName">
    <button type="submit" class="create-new-item-button" (click)="addItem()"><fa-icon [icon]="faPlus"></fa-icon></button>
</div>
```

Kuva 7 Ominaisuus- ja tapahtumasidonta

Yhteenvetona voidaan todeta, että Angularin kaksisuuntainen tiedonsidonta on tehokas ominaisuus, joka yksinkertaistaa dynaamisten verkkosovellusten kehittämistä. Se luo yhteyden mallin ja näkymän välillä ja siten mahdollistaa reaaliaikaisen synkronoinnin. Angularin avulla kehittäjät voivat luotaa että käyttöliittymä heijastaa automaattisesti taustalla olevia tietoja muutoksesta.

4 Olennaiset teknologiat Angular-sovelluksen kehittämiseen

4.1 JavaScript

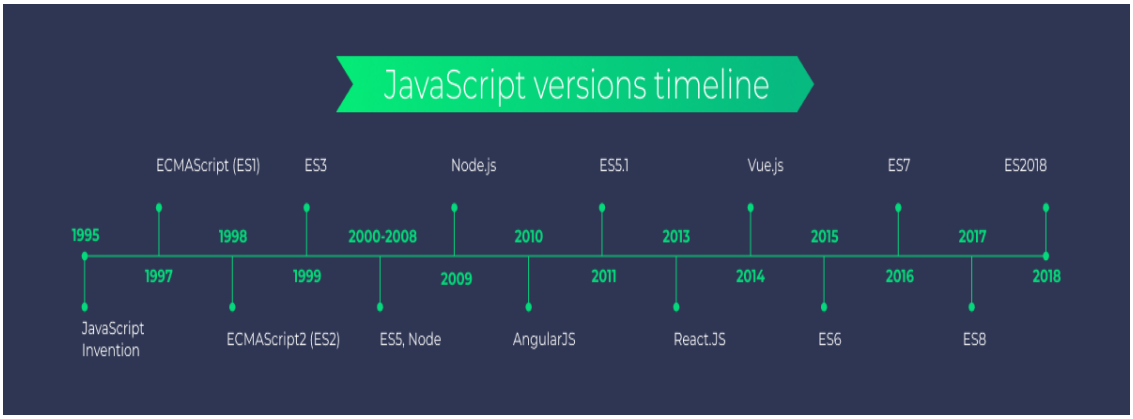
Dynaamisten ja interaktiivisten verkkosivujen tekoon tarvitaan JavaScriptiä, HTML:n ja CSS:n rinnalle. JavaScript on monipuolinen ja dynaaminen ohjelmointikieli, jota käytetään ensisijaisesti web-kehitykseen. Sen avulla kehittäjät voivat luoda interaktiivisia ja kiinnostavia verkkosivustoja lisäämällä verkkosivuille toimintoja ja interaktiivisuutta. JavaScript on kehittynyt yhdeksi maailman laajimmin käytetyistä ohjelmointikielistä.

JavaScript sai alkunsa Netscapessa, Brendan Eichin toimesta vuonna 1995. JavaScript nousi keskeiseksi kieleksi verkkokehityksessä, koska se pystyi parantamaan staattisia HTML-sivuja dynaamisella sisällöllä ja vuorovaikutteisuudella. Netscape 2 oli yksi ensimmäisistä selaimista, joka hyödynsi JavaScriptiä selaimessa. Alun perin JavaScript tunnettiin nimellä Mocha, sitten LiveScriptinä, ja lopuksi se muutettiin JavaScriptiksi Java-kielen suosion vuoksi. ECMA-standardiorganisaatio otti JavaScriptin hoitoonsa ja määritteli sen avulla ECMAScript-kielimäärityksen. ECAMScript julkaistiin vuonna 1997, joka auttoi vakiinnuttamaan kielen ja mahdollisti sen kehityksen monilla eri alustoilla. Viimeisin ECMAScripti (ES13) julkaistiin 2023.

JavaScriptiä tukevat kaikki yleisimmät verkkoselaimet, joten se on keskeinen työkalu verkkokehityksessä. McFedries [2023] mukaan JavaScript on ollut jonkin aikaa verkon universaali kieli. JavaScriptin avulla kehittäjät voivat muokata verkkosivun DOM:ia, jolloin he voivat muuttaa dynaamisesti sisältöä, muokata tyylejä ja reagoida käyttäjien toimiin. Tämä interaktiivisuus on välttämätöntä nykyaikaisten verkkosovellusten luomisessa.

Lisäksi JavaScriptillä on runsas ekosysteemi kehyksiä ja kirjastoja, jotka laajentavat sen ominaisuuksia. Kehykset kuten Angular, React ja Vue.js, tarjoavat kehittäjille työkaluja ja malleja verkkosovellusten rakentamiseen. Kirjasto kuten jQuery yksinkertaistaa yleisiä tehtäviä, kuten DOM-käsittelyä ja AJAX-pyyntöjä. Viime vuosina JavaScript on laajentunut verkkoselaimen ulkopuolelle Node.js:n

kaltaisten alustojen ansiosta. Node.js:n avulla kehittäjät voivat käyttää JavaScriptiä palvelinpuolella, jolloin kehittäjät voivat rakentaa verkkosovelluksen yhdellä kielellä.



Kuva 8 JavaScript versiot

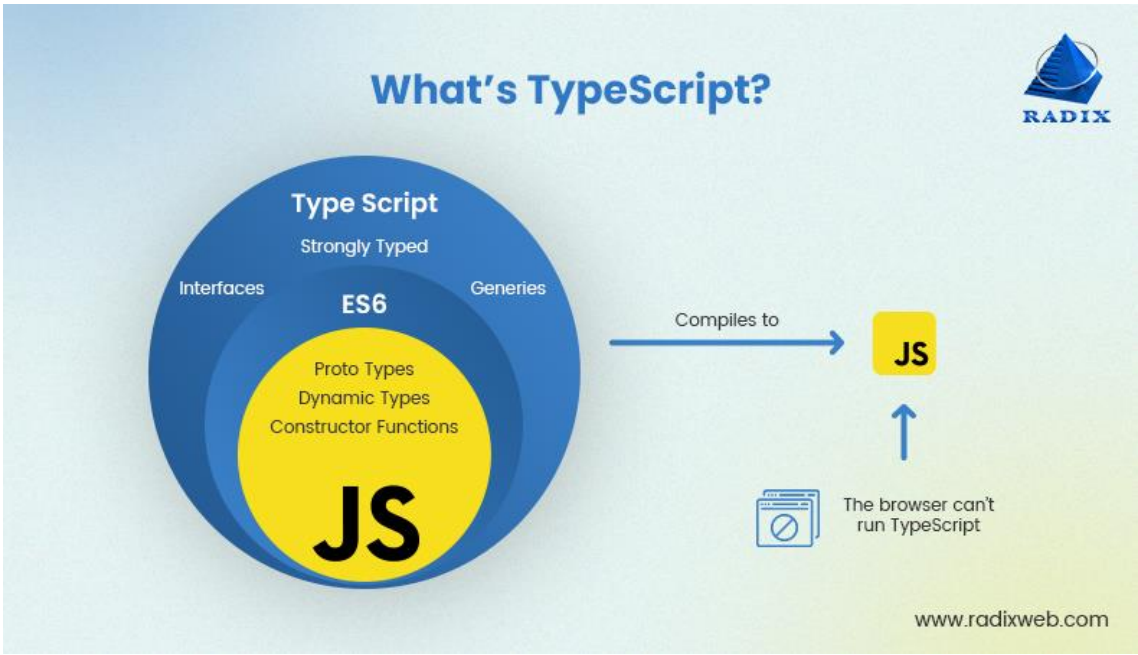
Vahvuuksistaan huolimatta JavaScriptillä on myös haasteensa. Sen asynkronisuus voi johtaa monimutkaiseen koodiin. Lisäksi voi ilmetä selaimen yhteensopivuusongelmia, koska selaimet tulkitsevat koodia eri tavalla. Tämä edellyttää kehittäjien kirjoittavan koodia eri ympäristöjä varten. JavaScriptillä on kuitenkin suuri yhteisö, joka vastaa aktiivisesti näihin haasteisiin tarjoamalla resursseja, parhaita käytäntöjä ja työkaluja niiden voittamiseksi.

Kaiken kaikkiaan JavaScript on edelleen suuri osa verkkokehityksen maailmaa. Sen monipuolisuus, yksinkertaisuus ja eloisa ekosysteemi tekevät siitä välttämättömän työkalun dynaamisten ja interaktiivisten verkkokokemusten luomiseen. Verkkokehityksen kehittyessä JavaScript pysyy todennäköisesti keskeisenä kielenä verkkosovellusten kehityksessä.

4.2 TypeScript

Angular käyttää avoimen lähdekoodin TypeScript-ohjelmointikieltä, joka luotiin vastauksena tarpeeseen kehittää suurempia JavaScript-sovelluksia. Microsoftin kehittämä TypeScript julkaistiin 2012 ja sen loi Anders Hejlsberg. TypeScript

laajentaa JavaScriptiä lisäämällä siihen tyypityksen ja muita ominaisuuksia, jotka tekevät koodin kirjoittamisesta turvallisempaa ja helpompaa ylläpitää. Beattie-Hoodin [2023] mukaan lähes kaikki suuret projektit ovat siirtyneet TypeScriptiin koska se tarjoaa dokumentoinnin ja staattisen testauksen. Koska TypeScript on osa JavaScriptiä kehittäjät voivat myös hyödyntää uusimpia JavaScript ominaisuuksia ja kirjastoja. TypeScript-koodi käännetään JavaScript-koodiksi ennen kuin sitä voidaan suorittaa web-selaimessa tai missään JavaScript-suoritusympäristössä. Tämä mahdollistaa TypeScriptin käytön nykyisissä JavaScript-ympäristöissä, koska selaimet ja muut JavaScript-alustat eivät ymmärrä suoraan TypeScriptiä. Kehittäjät voivat kirjoittaa ja ylläpitää koodia TypeScriptillä hyödyntäen sen etuja, kun taas sovellus suoritetaan lopulta JavaScriptillä.



Kuva 9 TypeScript käännösprosessi

Angularia käytettäessä TypeScriptin yleisimmät erityisominaisuuksia ovat:

- Dekoraattorit – Angularissa käytetään dekoraattoreita luokkien ja niiden jäsenten määrittelyyn. Näitä käytetään esimerkiksi komponenttien, palveluiden ja moduulien määrittelyyn. Dekoraattorit ovat

olennainen osa Angularin toimintaa ja mahdollistavat metadatan liittämisen luokkiin ja niiden jäseniin.

- Rajapinnat – TypeScriptin rajapinnat mahdollistavat vahvan tyypityksen ja koodin dokumentoinnin. Angular-kehittäjät hyödyntävät rajapintoja määritelläkseen sopimuksen, jonka luokat voivat toteuttaa. Tämä auttaa varmistamaan, että koodi noudattaa tiukasti määriteltyä rakennetta ja käyttäytymistä.

- Yleistäminen (Generics) – Generics mahdollistavat tyyppiturvalliset ratkaisut muuttuville tietotyypeille. Angularissa ne voivat olla erityisen hyödyllisiä esimerkiksi palveluiden ja komponenttien luomisessa, joissa tarvitaan joustavuutta tietotyyppien suhteen.

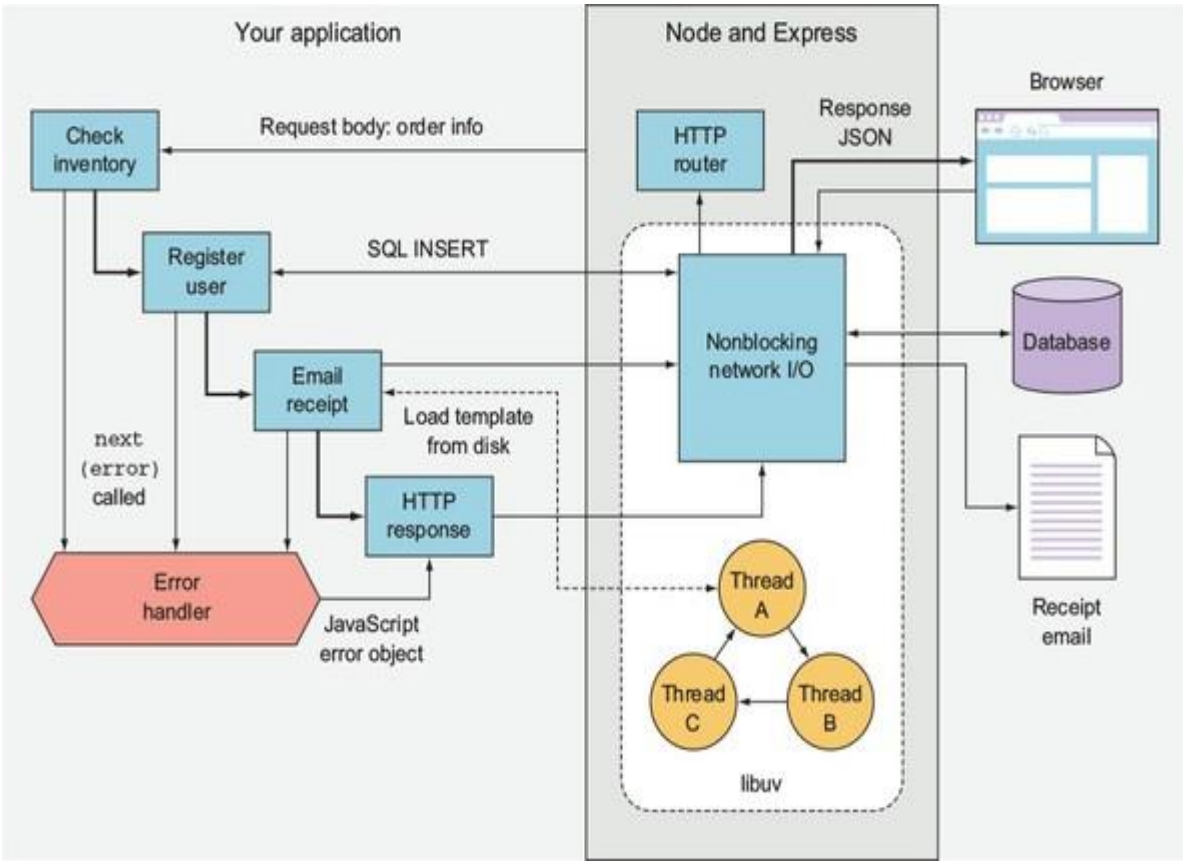
- Moduulit – TypeScriptin moduulit tarjoavat tavan jakaa koodi osiin, mikä auttaa pitämään sovelluksen rakenteen jäsenneltynä ja helpottaa koodin uudelleenkäyttöä. Angularissa moduulit ovat keskeinen käsite sovelluksen organisoimisessa ja riippuvuuksien hallinnassa.

4.3 Node.js

Perinteisesti JavaScriptiä käytettiin pääasiassa asiakaspuolen skriptaukseen web-selaimissa. Avoimen lähdekoodin Node.js:n myötä JavaScriptiä voidaan kuitenkin nyt käyttää myös palvelinpuolen koodin kirjoittamiseen. Node.js tarjoaa kehittäjille mahdollisuuden käyttää sama ohjelmointikieltä sekä asiakas- että palvelinpuolen kehityksessä, mikä helpottaa koodin ylläpitoa ja parantaa kehityksen tehokkuutta. Herronin [2020] mukaan yhden ohjelmointikielen käyttö sekä palvelimen että asiakkaan puolella on ollut pitkäaikainen tavoite verkkokehityksessä ja sillä on lukuisia mahdollisia etuja. Node.js:ää käytetään laajalti palvelinpuolen verkkosovellusten rakentamiseen. Node.js on alun perin Ryan Dahlin kehittämä ohjelmistokehitysalusta. Dahl esitteli Node.js:n ensimmäisen kerran vuonna 2009 ja siitä lähtien siitä on tullut merkittävä osa verkkokehitystä.

Node.js:n mukana tulee npm, paketinhallinta JavaScript-kirjastoille ja -työkaluille. Npm:n avulla voi helposti asentaa, hallita ja jakaa uudelleenkäytettäviä koodipaketteja, mikä helpottaa kolmannen osapuolen moduulien ja kirjastojen lisäämistä Node.js-projekteihin. Node.js on monialustainen ja toimii useissa

käyttöjärjestelmissä, mukaan lukien Windows, macOS ja Linux. Se on monipuolinen valinta sovellusten kehittämiseen eri ympäristöissä. Node.js:ssä on laaja ja eloisa avoimen lähdekoodin kirjastojen ja puitteiden ekosysteemi, joka laajentaa sen toimintoja ja yksinkertaistaa yleisiä tehtäviä. Tämä ekosysteemi sisältää kirjastoja verkkokehityksen (esim. Express.js), tietokantakäyttöön (esim. MongoDB), todennukseen (esim. Passport.js) sekä moneen muuhun tarkoitukseen.



Kuva 10 Tyypillinen Node.js verkkosovellus (Meck 2017)

Vaikka Angularin käyttäminen ilman Node.js:ää on teknisesti mahdollista, se vaatisi huomattavan määrän työtä ja saattaa rajoitta kehittäjien kykyä hyödyntää Angular-kehityksen mukana tulevaa rikasta ekosysteemiä ja työkaluja. Angular CLI käyttää Node.js:ää taustalla monissa toiminnoissaan. Esimerkiksi se hyödyntää npm:ää projektiin liittyvien riippuvuuksien hallintaan ja paketointiin. Lisäksi Angular CLI voi käyttää Node.js-pohjaisia palvelimia kehitystarkoituksiin.

Syitä, miksi tulisi käyttää Node.js:ää Angular-projektissa:

- Yhteensopivuus – Angular on kehitetty TypeScriptillä, joka on JavaScriptin supersetti. Node.js mahdollistaa sekä TypeScriptin että JavaScriptin käytön, mikä tekee siitä luonnollisen valinnan Angularin backend-palvelimelle.

- Rakennusprosessi – Node.js tarjoaa monipuolisen ekosysteemin työkaluja, kuten npm:n, joka helpottaa riippuvuuksien hallintaa ja projektin rakentamista. Angular-projektin rakentamiseen voi sisältyä monia vaiheita, kuten kääntäminen, pakkaaminen ja testaus. Node.js voi helpottaa näiden prosessien automatisointia.

- Palvelinpuolen logiikka – Vaikka Angular onkin erinomainen front-end-kehitykseen, monet sovellukset tarvitsevat myös palvelinpuolen logiikkaa, kuten tietokantayhteyksiä, tiedostojen käsittelyä tai kolmansien osapuolien rajapintojen käyttöä. Node.js voi toimia tässä roolissa tarjoamalla helpon tavan luoda palvelinsovelluksia JavaScriptillä.

- Yhteisö ja tuki – Node.js ja Angular ovat molemmat suosittuja ja laajalti käytettyjä teknologioita, joten niille on saatavilla runsaasti resursseja, oppaita, kirjastoja ja yhteisötukea. Käyttämällä näitä teknologioita yhdessä, voi hyödyntää näiden yhteisöjen tarjoamaa tietotaitoa ja apua.

4.4 Angular CLI

Angular-tiimin tarjoama tehokas Angular CLI (Command-line interface) komentorivityökalu helpottaa Angular-sovellusten luomista, rakentamista, testaamista ja käyttöönottoa. Se yksinkertaistaa kehitysprosessia tarjoamalla joukon komentoja, jotka automatisoivat toistuvia tehtäviä ja tekevät yleisistä työnkuluista paljon sujuvampia. Angular CLI ilmestyi vuonna 2017 Angular 6-version yhteydessä.

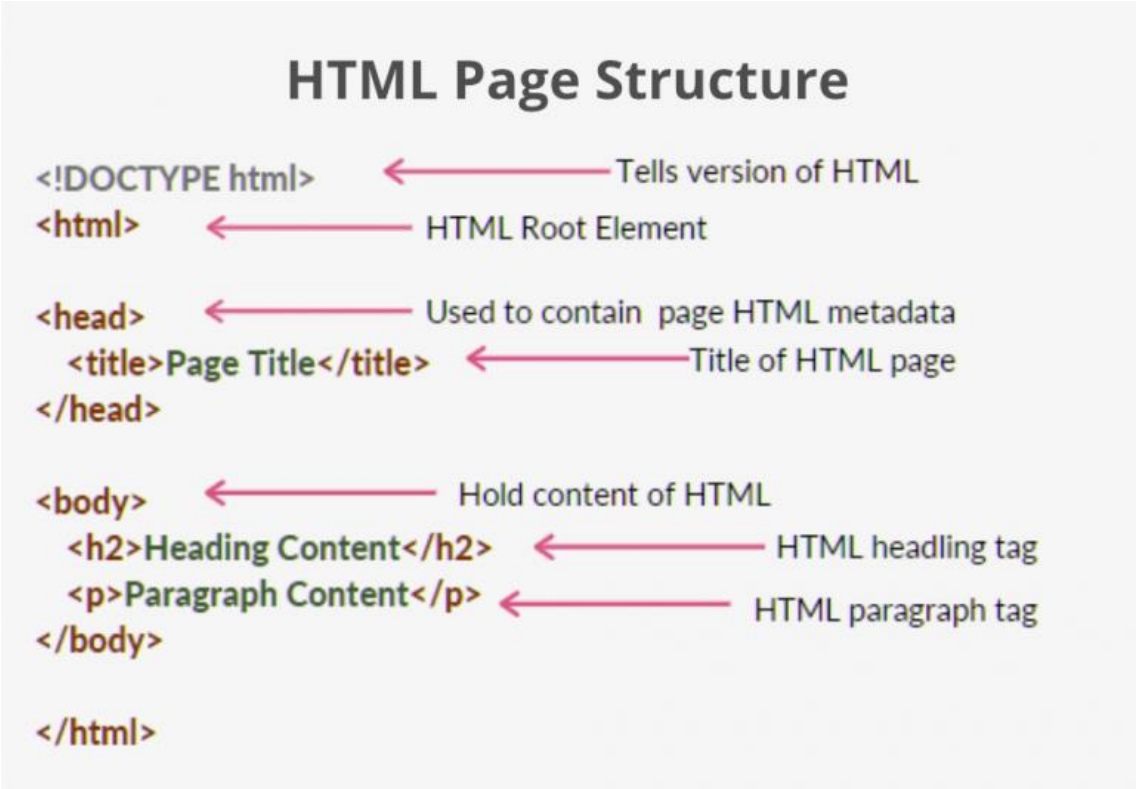
CLI:n avulla voidaan luoda uusia Angular-projekteja nopeasti ja helposti käyttämällä ”ng new <projektin-nimi>” -komentoa. Se määrittää projektin rakenteen, määritystiedostot ja alkuperäiset riippuvuudet. CLI antaa myös mahdollisuuden valita eri tyylitiedostojen välillä projektia luodessa. CLI tarjoa komennon ”ng

generate <ominaisuuden nimi>” erilaisten Angular-ominaisuksien luomiseen kuten, komponenttien, palveluiden, direktiivien ja moduulien. Sisäänrakennetun kehityspalvelimen avulla kehittäjät voivat käynnistää kehitysympäristön paikallisesti helposti ja nopeasti komennolla ”ng serve”. Kehityspalvelin luo ympäristön paikallisesti ja tarjoaa reaaliaikaisen uudelleenlatauksen. Jasmine ja Karma integraatioiden avulla kehittäjät voivat suorittaa yksikkö- ja integraatiotestejä projektissaan käyttämällä ”ng test” -komentoa. Projektiriippuvuuksia hallitaan npm:n (Node Package Manager) avulla, riippuvuuksien asentamiseen, päivittämiseen ja poistamiseen. Lopuksi Angular CLI:n avulla kehittäjät voivat suorittaa myös Angular-sovellusten kääntämisen tuotantokäyttöä varten käyttämällä komentoa ”ng build”. Tämä komento optimoi ja pakkaa sovelluksen tiedostot pienentäen niiden kokoa ja parantaen suorituskykyä.

Angular CLI on korvaamaton työkalu Angular-kehitykseen ja se tarjoaa kehittäjille tarvittavat työkalut ja työkulut Angular-sovellusten tehokkaaseen rakentamiseen, testaamiseen ja käyttöönottoon. Sen helppokäyttöisyys, automaatio-ominaisuudet ja muut kattavat ominaisuudet tekevät siitä arvokkaan työkalun Angularin kanssa työskenteleville kehittäjille.

4.5 HTML

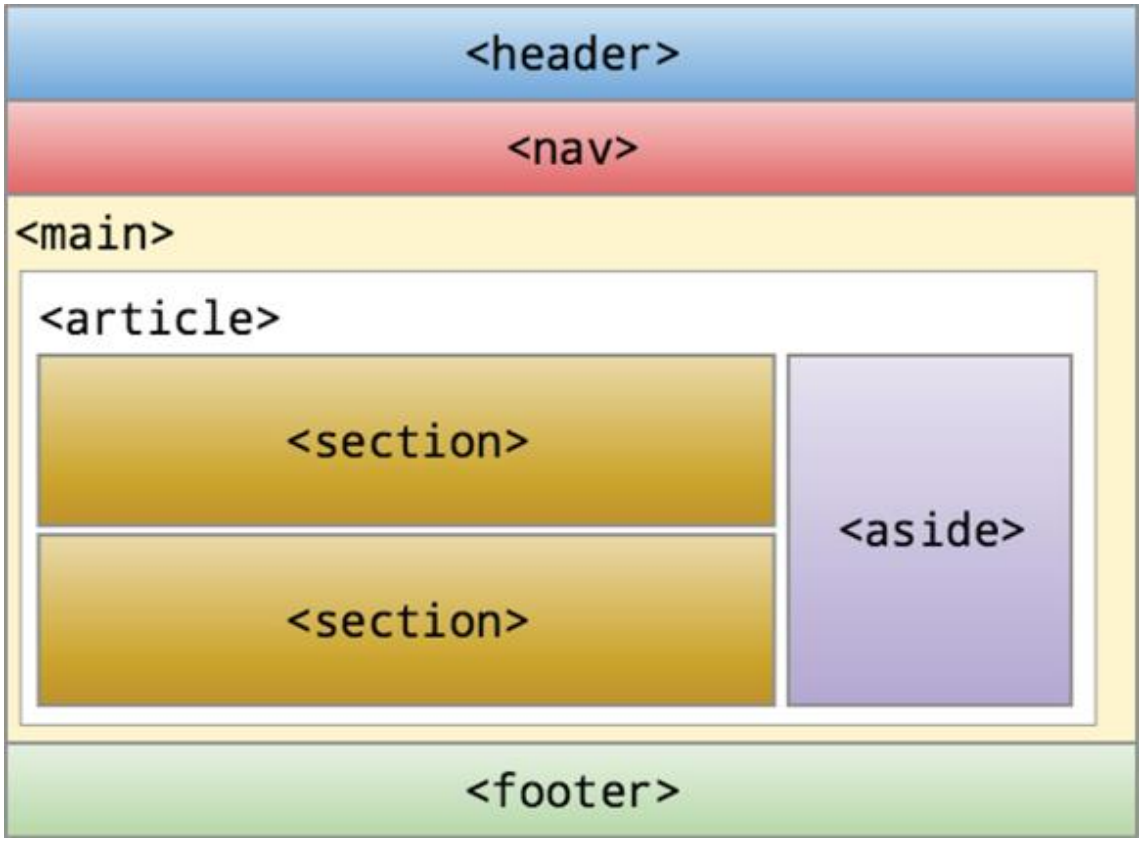
Ensimmäinen askel matkalla kohti verkkosivujen luomista on HTML (HyperText Markup Language) osaaminen. HTML on verkkosivujen kuvauskieli, joka määrittelee sivun rakenteen ja sisällön. HTML-elementtien avulla luodaan tekstiä, kuvia, linkkejä ja muita elementtejä, jotka muodostavat verkkosivun. HTML-tiedostot koostuvat elementeistä, joita kuvataan avautuvilla ja sulkevilla tageilla ja ne toimivat pohjana verkkosivujen visuaaliselle ja toiminnalliselle rakenteelle. HTML on olennainen osa World Wide Webin toimintaa ja sen avulla verkkosivut ovat yhteensopivia eri selainten ja laitteiden kanssa.



Kuva 11 HTML-sivun rakenne

Tim Berners-Lee loi ensimmäisen HTML-version (1.0) vuonna 1993 helpottaakseen asiakirjojen jakamista ja muotoilua World Wide Webissä. Tämän jälkeen on julkaistu viisi eri versiotta HTML:stä. Vuonna 1995 julkaistiin HTML:n toinen versio, joka toi mukanaan monia uusia ominaisuuksia, kuten taulukot ja linkkien upottamisen kuviin. HTML 3.2, julkaisuvuonnaan 1997, toi mukanaan vielä enemmän mahdollisuuksia verkkosivujen suunnitteluun, kuten tukemalla CSS-tyylitiedostoja ja tarjoamalla tarkempaa tietoa käyttäjille käyttämättömistä ominaisuuksista. HTML 4.01 julkaistiin vuonna 1999 ja se toi mukanaan XML-tuen sekä muita parannuksia ja korjauksia aiempiin versioihin verrattuna. Vuonna 2000 julkaistiin XHTML 1.0, joka oli HTML:n ja XML:n yhdistelmä, mikä teki siitä tiukemmin määritellyn ja yhteensopivamman tulevaisuuden web-standardien kanssa. Viimeisin suuri HTML-standardiversio ennen XHTML:n hylkäämistä oli HTLM5, joka julkaistiin vuonna 2014. HTML5 toi mukanaan paljon uusia ominaisuuksia kuten multimediaelementit (audio ja video), grafiikat (Canvas ja SVG) sekä semanttisen merkinnän, joka teki verkkosivujen sisällön

ymmärrettävämmäksi koneille. HTML5 on ollut keskeinen tekijä modernin verkkokehityksen kehityksessä ja sen jälkeen on ollut useita pienempiä päivityksiä, kuten HTML 5.1, HTML 5.2 ja viimeisin HTML5.3. Gor [2023] mukaan vaikka HTML5 on joustava, hän silti suosittelee muutaman ylimääräisen koodirivin kirjoittamista parhaiden käytäntöjen ylläpitämiseksi koodin paremman ylläpidon ja ymmärtämisen helpottamiseksi. Siksi koodin kirjoittaminen vanhaan tyyliin on hyödyllistä.



Kuva 12 Abstrakti näkymä HTML5:n semanttisista sivurakenteen tunnisteista (McFedries 2023)

Jokainen HTML-tiedosto koostuu sarjasta elementtejä, joista jokainen on ympäröity kulmasuluilla (<>). Ne määrittelevät verkkosivun rakenteen ja sisällön. HTML-elementti koostuu aloitustunnisteesta, sisällöstä ja lopputunnisteesta. Aloitustunniste sisältää elementin nimen kulmasulkeissa ja se voi sisältää myös attribuutteja, jotka tarjoavat lisätietoja sekä määrittelevät sisällön tyylin ja

toiminnallisuuden elementissä. Lopputunniste on samanlainen kuin aloitustunniste, mutta elementin nimen edessä on vinoviiva (/).

HTML:n helppouden takia se on ihanteellinen valinta aloitteleville kehittäjille. Sen yksinkertaisen syntaksin ja selkeän rakenteen ansiosta uuden sivun luominen on vaivatonta. Kehittäjät voivat nopeasti oppia perusasiat, kuten elementtien lisäämisen, tekstisisällön muotoilun ja kuvien upottamisen. Lisäksi HTML:n avulla kehittäjät voivat helposti integroida muita verkkokehityksen-teknologioita, kuten CSS ja JavaScript, mikä laajentaa sivustojen mahdollisuuksia ja ulkoasua.

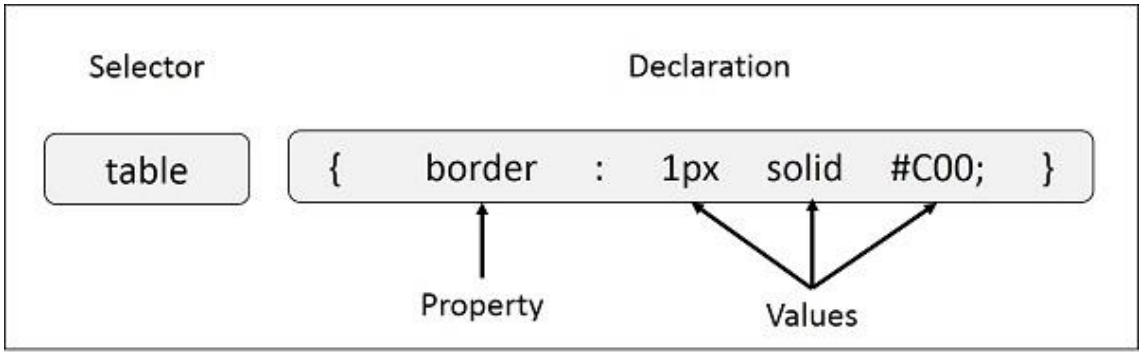
4.6 CSS

CSS (Cascading Style Sheets) on olennainen osa upeiden ja toimivien verkkosivujen luomista. Kun HTML määrittelee verkkosivujen rakenteen ja sisällön, CSS taas tarjoaa välineet sivun ulkoasun muokkaamiseen ja visuaalisen ilmeen luomiseen. CSS mahdollistaa tekstien, kuvien, taustojen ja muiden elementtien tyylien määrittämisen, mikä antaa sivuille persoonallisuutta ja houkuttelevuutta. CSS-tyylit määritellään valitsimilla ja ominaisuuksilla, jotka ohjaavat elementtien ulkoasua ja sijoittelua sivulla. Tämä tekee CSS:stä voimakaan työkalun, joka antaa kehittäjille hallinnan sivustojen ulkoasuun ja parantaa käyttäjien käyttökokemusta. Yhdessä HTML:n kanssa CSS muodostaa vahvan perustan verkkosivujen suunnittelulle ja toteutukselle, varmistaen sivustojen yhteensopivuuden eri selainten ja laitteiden kanssa ja tarjoaa samalla houkuttelevan ja intuitiivisen käyttökokemuksen kävijöille.

CSS syntyi tarpeesta erottaa dokumenttien rakenteellinen sisältö niiden ulkoasusta World Wide Webissä. Vuonna 1996 W3C julkaisi CSS1-standardin, joka mahdollisti verkkosivujen ulkoasun hallinnan keskitetysti. CSS1 toi mukanaan perusominaisuuksia kuten fonttien ja värien hallinnan, mutta sen käyttö oli vielä melko rajoitettua. Seuraavaksi vuonna 1998 julkaistiin CSS2, joka toi mukanaan monipuolisempia tyylien hallintamahdollisuuksia, kuten sijaintiin ja koon määrittämisen. CSS2 mahdollisti myös paremman taulukoiden ja listojen muotoilun.

Vuonna 2011 julkaistiin CSS3, joka toi mukanaan valtavan määrän uusia ominaisuuksia ja parannuksia. CSS3:n myötä kehittäjät saivat käyttöönsä muun muassa animaatiot, varjostukset, monimutkaiset muotoilut ja jopa 3D-transformaatiot. CSS:n kehitys on jatkunut vilkkaana ja siitä on tullut olennainen osa modernia verkkokehitystä. Se on mahdollistanut verkkosivujen ulkoasun ja käyttökokemuksen hienosäädön entistä tarkemmin, mikä on tehnyt verkkosisällöstä visuaalisesti houkuttelevampaa ja käyttäjäystävällisempää. Jatkuvat CSS-päivitykset ja uusien ominaisuuksien lisääminen ovat varmistaneet, että CSS pysyy ajan tasalla ja tarjoaa kehittäjille työkaluja innovatiivisten responsiivisten verkkosivujen luomiseen.

Jokainen CSS-tiedosto koostuu säännöistä ja määrityksistä, jotka määrittelevät verkkosivun ulkoasun ja visuaalisen tyylin. CSS-säännöt koostuvat valitsijoista ja deklaraatioista. Valitsijat kohdistavat säännöt tiettyihin HTML-elementteihin, luokkiin tai tunnisteisiin (ID), kun taas deklaraatiot määrittävät valitun elementin ulkoasun ominaisuudet, kuten värit, fontit, marginaalit ja koot. CSS-sääntö alkaa valitsijalla, joka voi olla HTML-elementin nimi, luokka tai ID. Tämän jälkeen sääntö sisältää aaltosulkeissa ({}) deklaraatioita, jotka määrittelevät valitun elementin ulkoasun. Deklaraatiot koostuvat ominaisuudesta ja sen arvosta, erotettuna kaksoispisteellä, esimerkiksi ”border : 1px solid #C00;” asettaa reunuksen paksuudeksi 1 pikseliä ja värin punaiseksi.



Kuva 13 CSS syntaksi

CSS:n avulla kehittäjät voivat erottaa sisällön rakenteelliset osat visuaalisesti, mikä tekee koodista helpommin ylläpidettävän ja muokattavan. Lisäksi CSS

mahdollistaa verkkosivujen ulkoasun muokkaamisen ilman, että itse HTML-rakennetta tarvitsee muuttaa, mikä helpottaa sivuston päivitystä ja kehitystä ajan mittaan.

5 Kurssin toteutus

Tavoitteena oli luoda Angular perusteet -verkkokurssi Metropolia Ammattikorkeakoulua varten. Aloitin tutustumalla Angulariin ja sen ominaisuuksiin. Jouduin syventämään tietoani TypeScriptistä, koska Angular käyttää ensisijaisesti TypeScriptiä. Tämän jälkeen ryhdyin työstämään kurssia Metropolian Moodle-oppimisympäristöön. Kurssi alkaa teorialla, jonka jälkeen siirrytään koodaustehtäviin. Angularin oppimiseen vaaditaan perustason osaaminen HTML-, CSS- ja JavaScript-kielistä. Kurssi toteutettiin Angular 17-versiolla. Kurssin lopussa on monivalintakoe, joka kattaa eri osa-alueet kysymyksissään.

5.1 Osio 1

Ensimmäisessä osiossa opiskelijat käyvät läpi Angular-ohjelmistokehyksen taustaa, TypeScriptiä sekä Angularin asennusta sekä tarvittavia työkaluja sen käyttämiseen. Opiskelijat saavat hieman käsitystä Angularin historiasta, kenen kehittämä Angular on, mihin sitä käytetään ja esimerkkejä sen käytöstä. On myös tärkeä, että opiskelijat tiedostavat erot eri Angular versioiden välillä (Angular 1, Angular 2), jotta ei tule väärinkäsityksiä kurssin aikana.

Tämä Jälkeen opiskelijat saavat hieman käsitystä itse Angularista ja sen tärkeimmistä ominaisuuksista. Koska kurssissa vaaditaan perustason osaamista JavaScriptistä, niin on tarvetta syventää tietämystä TypeScriptistä koska Angular hyödyntää sitä. Lopuksi opiskelijat oppivat Angularin asennuksen sekä tarvittavat työkalut kuten esimerkiksi Angular CLI ja Node.js käytön ja asennuksen. Kurssin aikana opiskelijat hyödyntävät Microsoftin avoimen lähdekoodin koodieditoria Visual studio codea. Tässä samalla opiskelija luo ensimmäisen projektinsa Angular CLI:tä hyödyntäen.

5.2 Osio 2

Seuraavassa osiossa opiskelija käy läpi Angular-projektin kansiorakennetta ja eri tiedostojen merkitystä. Tämän jälkeen opiskelija saa hieman tietoa Angularin toiminnasta editoimalla oletus komponenttia ja sen tyylitiedostoa. Tämän tiedon avulla opiskelija on valmis luomaan kurssin projektin (Ostoslista-sovellus) luomalla kaksi komponenttia, Home- sekä Ostoslista-komponentin.

Tämän osion aikana opiskelija saa käsityksen komponentin rakenteesta ja siihen liittyvistä tiedostoista kuten HTML-mallista tai CSS-tiedostosta. Angularissa on myös tärkeä oppia liittämään komponentteja itse sovellukseen, joten se käsitellään samassa yhteydessä. Home-komponentin sisällön lisäyksen jälkeen on aika luoda ensimmäinen Angular reititys Ostoslista-komponenttiin. Tämän jälkeen opiskelija lisää sisältöä Ostoslista-komponenttiin malliin ja työstää sovelluksen toimintaa lisäämällä TypeScriptiä komponentin TypeScript-tiedostoon. Tässä samalla opiskelija oppii käyttämään Angularin sisäänrakennettuja moduuleita, direktiivejä ja muita ominaisuuksia.

5.3 Osio 3

Tässä osiossa opiskelija oppii luomaan ensimmäisen Angular-palvelun ja ymmärtämään sen käyttötarkoituksen. Samalla opiskelija oppii käsittelemään riippuvuuksien injektointia ja sitä, miten injektoituja ominaisuuksia käytetään toisessa tiedostossa. Lopuksi opiskelija luo viimeisen komponentin projektia varten (Tuotelista-komponentin).

Ennen tätä osiota opiskelijalla on kaikki koodit ja tiedot yhdessä komponentissa. Tämä takia palvelun käyttö on tarpeellista, koska eri komponentit saattavat käyttää samoja tietoja, kuten esimerkiksi taulukoita tiedon tallentamiseen. Opiskelija lisää pari metodia palveluun, tuotteen lisäämisen sekä tuotteiden esittämiseen. Tämän työn jälkeen opiskelija oppii injektoimaan palvelun Ostoslista-komponenttiin. Seuraavaksi opiskelija luo Tuotelista-komponentin, jotta lisätyt tuotteet voidaan näyttää sovelluksessa. Palvelun injektoiminen luotuun

komponenttiin on pakollista, jotta sovellus toimii. Opiskelija työstää Tuotelista-komponenttia lisäämällä sisältöä HTML-malliin käyttämällä ”*ngFor” -direktiiviä ja toimintoja TypeScript-tiedostoon, kuten (esim. poistomenetelmä). Tämän osion aikana opiskelija saa paremman käsityksen siitä, kuinka SPA toimii sekä palveluiden merkityksen Angular-projektissa.

5.4 Osio 4

Tässä osiossa opiskelija luo eri menetelmiä tuotelista-komponenttiin ja palveluun, sekä yhdistää tuotelistan menetelmät HTML-malliin hyödyntäen erilaisia Angularin sisäänrakennettuja ominaisuuksia. Lopputuloksena opiskelija on luonut ostoslistasovelluksen hyödyntäen Angularia, mutta ilman ulkoasua.

Opiskelija aloittaa luomalla menetelmän tuotteen tilan muuttamiseen tuotelista-komponenttiin sekä palveluun. Tämän jälkeen opiskelija oppii hyödyntämään Angularin ominaisuussidontaa ja tapahtumansidontaa, menetelmien sitomiseen HTML-malliin. Tuotetta pitäisi myös pystyä muokkaamaan, joten opiskelija luo tarpeelliset menetelmät sitä varten. Tässä osiossa opiskelija joutuu hyödyntämään paljon TypeScriptiä. Lopuksi opiskelija luo tuotelista-komponenttiin uuden näkymän muokkaamista varten, jossa hyödynnetään eri direktiivejä sekä tapahtumansidontoja.

5.5 Osio 5

Projektin aikana opiskelija ei ole muokannut sovelluksen ulkoasua, joten tässä osiossa opiskelija oppii hyödyntämään CSS:ää Angular-sovelluksessa. Tämän myötä opiskelija saa paremman käsityksen sovelluksesta tyylin lisäyksen jälkeen, sillä sovellus näytti alun perin hyvin yksinkertaiselta. Opiskelija oppii neljä eri tapaa lisätä CSS:ää Angular-sovellukseen, jonka jälkeen opiskelija lisää tyylit jokaiseen komponenttiin. Lopuksi opiskelija oppii lisäämään kolmannen osapuolen kirjaston, Font Awesomen, ja hyödyntämään sen tarjoamia ikoneja sovelluksessa.

Kuva 14 Opiskelijan luoman projektin lopputulos.

5.6 Osio 6

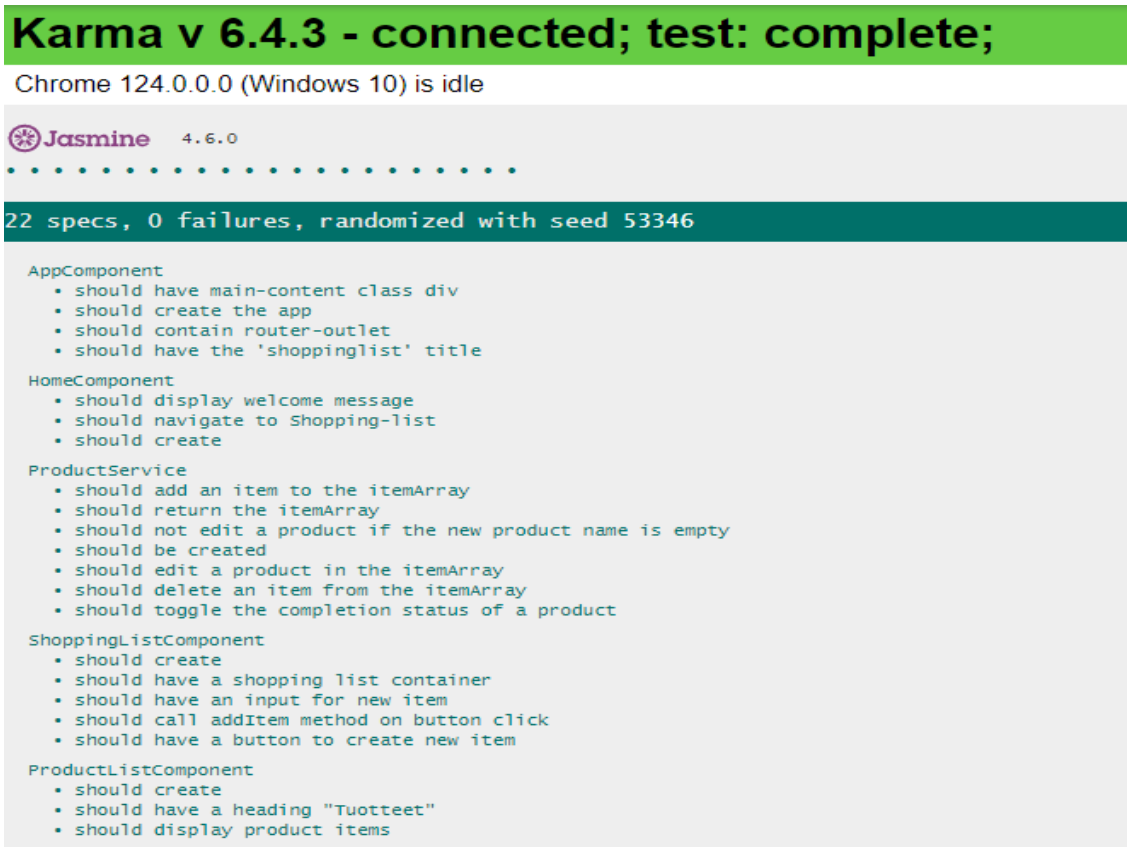
Viimeisessä osiossa opiskelija käy läpi, kuinka tehdä yksikkötestejä ja hyödyntää Angular CLI:n tarjoamia testaustyökaluja, kuten Karmaa ja Jasminea. Samalla opiskelija saa käsityksen siitä, miksi yksikkötestaus on hyödyllistä ja mitä etuja siitä on.

Osio alkaa kertomalla Angular CLI:n tarjoamista testaustyökaluista sekä Angularin luomista TypeScript-testitiedostoista. Tämän aikana opiskelija saa perustiedot testitiedostojen Jasmine-skiripteistä ja niiden rakenteista, minkä jälkeen

testausympäristöä testataan oletusskripteillä komennolla ”ng test”. Lopuksi opiskelija luo jokaiselle Angular-sovelluksen komponentille ja palvelulle testiskriptit, joiden tuloksena pitäisi olla yhteensä 22 luotua testiä.

5.6.1 Karma

Angular CLI tarjoaa sisäänrakennetun tuen yksikkötestaukseen. Karmaa ja Jasminea käyttämällä Angular-sovellusten testien määrittäminen ja suorittaminen on helppoa. Karma on työkalu asiakaspuolen JavaScriptin testaamiseen. Karma suorittaa testejä oikealla selaimella toimivalla sovelluksella, mikä auttaa havaitsemaan ja korjaamaan mahdollisia yhteensopivuusongelmia ja muita virheitä. Karma tarjoaa reaaliaikaisen palautejärjestelmän, joka päivittää testitulokset automaattisesti aina, kun koodia muokataan. Tämä nopeuttaa kehitysprosessia ja auttaa varmistamaan sovelluksen laadun ja luotattavuuden. Angular CLI:ssä voi hyödyntää ”ng test” -komentoa kätevään testaamiseen.



Kuva 15 Karmalla ajetut kurssin testit selaimessa.

5.6.2 Jasmine

Jasmine on toinen tärkeä työkalu Angular-sovellusten testaamisessa. Se on testauskehys, joka on suunniteltu erityisesti JavaScript-sovellusten yksikkö- ja integraatiotestaukseen. Jasmine tarjoaa kehittäjille helppokäyttöisen syntaksin testien kirjoittamiseen ja suorittamiseen. Yksi Jasmine-testauksen tärkeimmistä eduista on sen deklaratiivinen luonne. Testit kirjoitetaan kuvailevassa muodossa (describe-it), mikä tekee testien tarkoituksen ja odotettujen tulosten ymmärtämisestä helppoa.

```
describe('ShoppingListComponent', () => {
  let component: ShoppingListComponent;
  let fixture: ComponentFixture<ShoppingListComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [ShoppingListComponent]
    })
    .compileComponents();

    fixture = TestBed.createComponent(ShoppingListComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should have a shopping list container', () => {
    const shoppingListContainer = fixture.nativeElement.querySelector('.shopping-list-container');
    expect(shoppingListContainer).toBeTruthy();

  });

  it('should have an input for new item', () => {
    const inputElement = fixture.nativeElement.querySelector('.new-item-input');
    expect(inputElement).toBeTruthy();
  });
```

Kuva 16 Esimerkki kurssin aikana luoduista Jasmine-skpriteistä.

6 Yhteenveto

Opinnäytetyössä suunniteltiin ja kehitettiin Angular-verkkokurssi Metropolian Moodle-ympäristöön. Tavoitteena on julkaista kurssi monimuoto-opintoihin, vapaasti valittaviin opintoihin, avoimiin AMK-opintoihin ja väyläopintoihin. Kurssi on osoittautunut toimivaksi ja toimii perustana Angular-ohjelmistokehykseen. Sisällön tarkistamiseen ja loppukokeen suorittamiseen osallistui yksi Metropolian opiskelijaa.

Kurssilla käsitellään Angular-perusteita sekä sen eri ominaisuuksia ja kuinka niitä voi hyödyntää omissa sovelluksissa. Koska kyseessä on JavaScript-ohjelmistokehys, opiskelijan odotetaan hallitsevan verkkokehityksen perusteet, kuten HTML-, CSS- ja JavaScript-kielet. Kurssi on tarkoitettu opettamaan Angular-ohjelmistokehyksen perusteet ja samalla kertaamaan eri tarvittavien ohjelmointikielten osaamista.

Opiskelija suorittaa kurssin asteittain osio kerrallaan. Selkeästi kirjoitettua materiaalia on helppo seurata. Kurssissa käytetyt koodimuuttujat ovat selkeästi nimetty ja helppo ymmärtää. Kurssi sisältää lukuisia havainnollistavia kuvia, jotka tehostavat oppimista.

Työn teoria oli suurelta osin minulle tuntematonta, joten minun oli syvennettävä tietämystäni Angularista käyttäen kirjallisuutta, verkkoartikkeleita ja virallisia dokumentaatioita. Lisäksi kurssin suunnittelemine ja toteuttaminen Moodle-oppimisympäristöön tulivat minulle uusina kokemuksina.

Kurssia on syytä päivittää tulevaisuudessa, sillä Angular saa pääversiopäivityksen noin joka kuudes kuukausi. Tämä tarkoittaa sitä, että kurssin sisältöön voi tulla muutoksia, joten on tärkeä pitää materiaali ajan tasalla vastaamaan uusimpia Angularin ominaisuuksia ja parhaita käytäntöjä. Lisäksi voisi harkita backend-osion integroimista kurssiin, jotta opiskelijat saavat kokonaisvaltaisen käsityksen siitä, miten Angular toimii yhdessä backendin kanssa. Toinen päivitysidea olisi GitHubin käytön integroiminen kurssille.

Lähteet

Beattie-Hood, Ben. 2023. Modern TypeScript: A Practical Guide to Accelerate Your Development Velocity. Apress.

Custodia, Henrique. 2022. Angular Standalone Components: Say Goodbye To NgModules. Verkkoaineisto. Medium. <https://medium.com/@henriquecustodia/angular-standalone-components-say-goodbye-to-ngmodules-f3979661bc31> 17.2.2023. Luettu 15.4.2024

Gor, Varun. 2023. Creating Responsive Websites Using HTML5 and CSS3: A Perfect Reference for Web Designers. Apress.

Herron, David. 2020. Node.js Web Development. Packt Publishing.

McFedries, Paul. 2023. HTML, CSS & JavaScript All-in-One For Dummies. For Dummies.

Meck, B. Railich, N. Holowaychuk, T. Cantelon, M. Oxley, T. Young, A. 2017. Node.js in Action, Second Edition. Manning Publications.

Neto, Alavaro Camillo. 2024. Angular Design Patterns and Best Practices. Packt Publishing.

Seshadri, S. ja Green, B. 2013. AngularJS. O'Reilly Media, inc.

Uluca, Doguhan. 2024. Angular for Enterprise Applications – Third Edition. Packt Publishing.