Opinnäytetyö AMK

Tietojenkäsittelyn koulutus

2019

Lauri Keinänen

HAHMONTUNNISTUSTA HYÖDYNTÄVÄN VERKKOSOVELLUKSEN SUUNNITTELU JA TOTEUTUS

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ AMK | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittelyn koulutus

2019 | 37 sivua

Lauri Keinänen

HAHMONTUNNISTUSTA HYÖDYNTÄVÄN VERKKOSOVELLUKSEN SUUNNITTELU JA TOTEUTUS

Sekä hahmon-, että kasvojentunnistus ovat tärkeitä ja alati kehittyviä osia modernia yhteiskuntaa. Niiden hyödyt ulottuvat turvallisuudesta ja henkilöllisyyden verifioinnista aina viihteeseen ja arkiseen hyötykäyttöön. Teknologian jatkuvan kehityksen myötä sekä hahmon-, että kasvojentunnistus ovat saaneet useat erilaiset kasvot. Tänä päivänä voidaan arvioida jopa ihmisen sen hetkistä mielialaa pelkän kasvojentunnistusprotokollan avulla. Vaikka hyöty- ja viihdearvoa löytyy kosolti, on kasvojentunnistukseen liitettynä myös paljon Orwellilaisia pelkoja.

Tässä opinnäytetyössä suunitellaan sekä ohjelmoidaan toimiva verkkosovellus, joka havaitsee hahmon sovellukseen ladattavasta kuvasta. Sovellus on kehitetty React-kehitysympäristössä ja lopputuloksena on toimiva hahmontunnistusohjelma.

ASIASANAT:

Kasvojentunnistus, Käyttäjäkokemus, Käyttöliittymä

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology

2019 | 37 pages

Lauri Keinänen

DESIGNING AND PROGRAMMING A FACIAL DETECTION APP

Face detection and recognition are an important and ever-evolving parts of a modern community. Their benefits range from security and indentity verification to entertainment and everyday use. With the continuous development of technology, facial detection and recognition have many new forms. Today, even the current mood of the human being can be evaluated by the mere face recognition protocol. Although, the value of entertainment is enormous when it comes to face recognition, there are also many Orwellian fears and misconceptions attached to face detection and recognition technology.

In this thesis I am designing and developing a facial detection app, and diving in depth into how one can create something similar on their own, what it requires and how it can be achieved. This thesis is done as a case-study in conjunction with empiristic and theoretical research. The application was developed with React framework in Javascript, HTML and CSS languages. The result of the study is a full functioning stand-alone facial detection app.

KEYWORDS:

Facial recognition, User Interface, User Experience

SISÄLTÖ

KUVAT

KÄYTETYT LYHENTEET JA SANASTO

MF	Mobile First
RD	Responsive Design
CCTV	Valvontakamera
DOM	Document Object Model
VR	Virtual Reality
Parent	Elementin isäntä
Child	Elementin jälkeläinen
Framework	Runko
Natural Swipe Area	Luonnollinen käyttöalue älypuhelimen ruudulla
CSS	Tyylitiedosto
HTML	Hypertext Markup Language
API	Application Programming Interface

1 JOHDANTO

Jokainen meistä näkee päivittäisellä tasolla kymmeniä, jopa satoja erilaisia kasvoja. Ihmisaivot ovat yllättävän hyvät muistamaan sekä tunnistamaan jo nähtyjä kasvoja. Aivot tunnistavat kasvot lähes 100% tarkkuudella, ja alle 200 millisekunnissa pystymme päättelemään onko kyseessä ystävä vai vihollinen *(UCSB 2013)*. Läheisten ihmisten kasvot tunistetaan 100% tarkkuudella, sillä olettamuksella ettei henkilö sairasta muistiin tai havainnointikykyyn liittyvää sairautta. Mutta miten on konenäön laita? Kameroita on kaikkialla, eikä yksikään meistä pääse lähikauppaansa pidemmälle tulematta taltioiduksi. Valvontakameralaitteet on suunniteltu sekä yksilöiden että yritysten turvaksi, ja niihin on helppoa asentaa jälkikäteen sekä hahmon- että kasvojentunnistusohjelmia, jotta henkilöllisyys voidaan tulkita pelkän kuvan avulla hyödyntäen tietotekniikkaa ja henkilötietokantoja. Ongelmaksi kuitenkin nousee luotettavuus. Voidaanko syytön ihminen todeta syylliseksi, kun tekoäly tunnistaa väärän henkilön rikospaikalta tai voiko joku sinua muistuttava avata älypuhelimesi lukituksen omilla kasvoillaan?

Ensimmäiset muistot hahmontunnistuksesta monella ovat mitä luultavimmin digi- ja järjestelmäkameroista. Kun digitaaliset kamerat yleistyivät, niihin alkoi tulla kuluttajillekin tutuksi ominaisuus, jossa kamera automaattisesti etsii kuvasta kasvot. Tämä ei kuulosta lähtökohtaisesti pelottavalta tai erikoiselta, sillä kamera ei tunnistanut muuta kuin kasvot – eli se osasi etsiä kuvasta pään muodon, nenän, silmät, kulmakarvat ja suun. Jännittävämmäksi se muuttuu silloin, kun tekoäly kykenee analysoimaan edellä mainittuja ominaisuuksia niin tarkasti, että se pystyy päättelemään kenelle ihmiselle kyseiset kasvot kuuluvat vertaamalla kuvan tietoja henkilötietokantaan.

Edellä mainitun tapainen hahmontunnistusohjelma on verrattaen yksinkertainen rakentaa. Tässä opinnäytetyössä tutkimusmetodina on hyödynnetty tapaustutkimusta, jossa käydään läpi yksityiskohtaisesti kyseisen ohjelman rakentaminen alusta loppuun sekä pohditaan eettisiä ongelmia ja ennakkoluuloja sekä empiiriseltä että teoreettiselta pohjalta, joita varsinainen kasvojentunnistus saattaa yhteiskunnassa aiheuttaa

2 SOVELLUKSEN SUUNNITTELU

Tämän suhteellisen yksinkertaisen hahmontunnistusohjelman rakentamiseen käytän apunani Photoshopia sivuston ulkoasun- ja rakenteen suunnitteluun. Varsinainen toteutus tehdään pääosin React- ja Redux-frameworkeja hyödyntämällä. Verkkosovellus tulee sisältämään oman palvelimen sekä tietokannan. Sivustolle pystyy luomaan oman henkilökohtaisen tilin, jolla sovellukseen voi kirjautua. Lopputuote tulee olemaan täysin responsiivinen, ja siihen voidaan ladata mikä tahansa kuva, josta sovellus tunnistaa kuvassa mahdollisesti olevat kasvot.

2.1 Käytettävyys

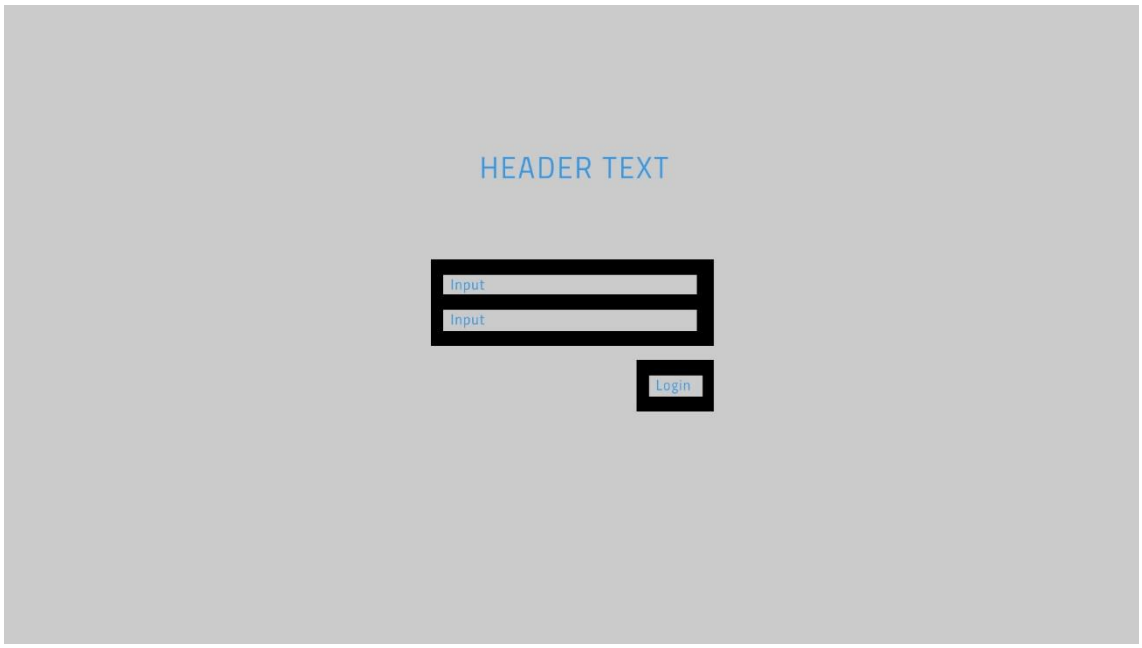
Koska sovelluksesta on tulossa responsiivinen, täytyy sen myös toimia mobiililaitteilla. Tämä on huomioitava jo rautalankamallia rakennettaessa. Tässä esimerkissä ei ole paljoa ajateltavaa skaalautuvuuden kannalta, mutta tahdon paneutua erikseen *Mobile First* sekä *Responsive Design* lähtökohtien eroihin. Viittaan jatkossa edellä mainittuihin termeihin lyhenteillä MF (Mobile First) ja RD (Responsive Design) luettavuuden helpottamiseksi.

Yksinkertaisuudessaan eroavaisuus voidaan selittää seuraavalla lauseella: MF tekniikalla sivusto tehdään kirjaimellisesti mobiililaitteiden käytettävyys edellä, kun taas RD-lähestymistavassa pidetään mobiililaitteiden käytettävyys mielessä. RD on yleisempää B2B-yritysten puolella, jossa verkkosivuston on oltava informatiivinen. Hyvin jäsennelty sisältö on hyödyllistä myös hakukoneoptimoinnin kannalta. Tämä lähestymistapa on loistava kun tiedetään, että valtaosa selaajista tulevat sivustolle kannettavalla- tai pöytätietokoneella. Sen sijaan MF tähtää ulkoasultaan sekä sisällöltään siihen, että sivusto vaikuttaa mobiilisovelluksen tapaiselta. Se on helposti luettavissa mobiililaitteelta, sisältää vähemmän tekstiä, latautuu nopeasti ja kaikki tarvittava käytettävyyden kannalta löytyvät *Natural Swipe Arean* sisältä. Natural Swipe Area on havainnollistettu kuvassa 1. *(Drawn Digital 2018)*

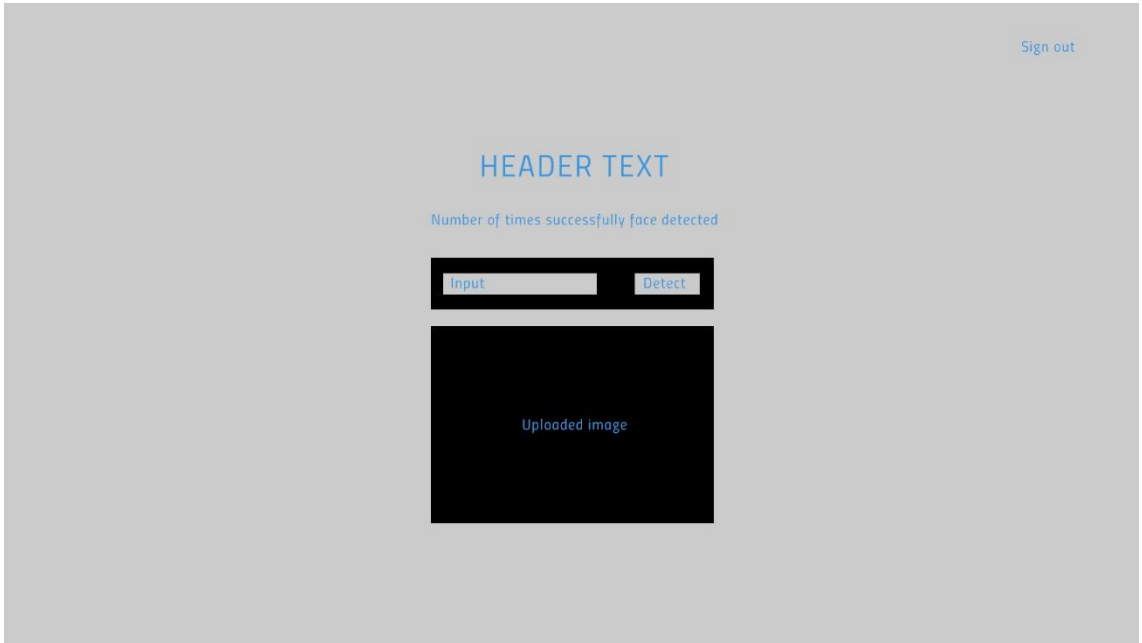
Kuva 1. Oikean käden peukalon natural swipe area.

2.2 Rautalankamalli

Sivuston rakenne on yksinkertainen ja selkeä. Se sisältää kirjautumisalueen, jossa käyttäjältä kysytään sähköpostiosoite sekä salasana. Tämän jälkeen avautuu uusi näkymä, jossa laatikon sisällä on tilavaraus syötteelle (URL) sekä painike, jota painamalla ohjelma analysoi kuvan ja tunnistaa siinä mahdollisesti olevat kasvot. Lisäksi käyttäjä näkee kuinka monesti ohjelma on onnistunut tunnistamaan kasvot kuvasta sekä mahdollisuus kirjautua ulos. Rautalankamallin tarkoituksena on luoda nopeasti sivuston skeema valmiiksi. Rautalankamallit esiteltynä kuvissa 2 ja 3.



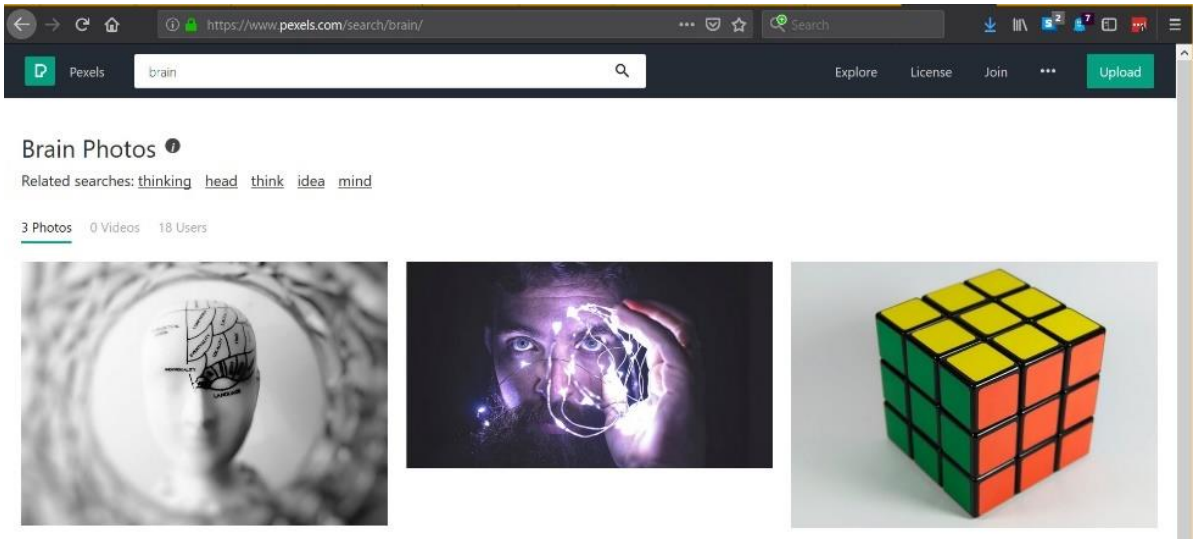
Kuva 2. Rautalankamalli verkkosovelluksen kirjautumissivusta



Kuva 3. Rautalankamalli verkkosovelluksen toimintasivusta

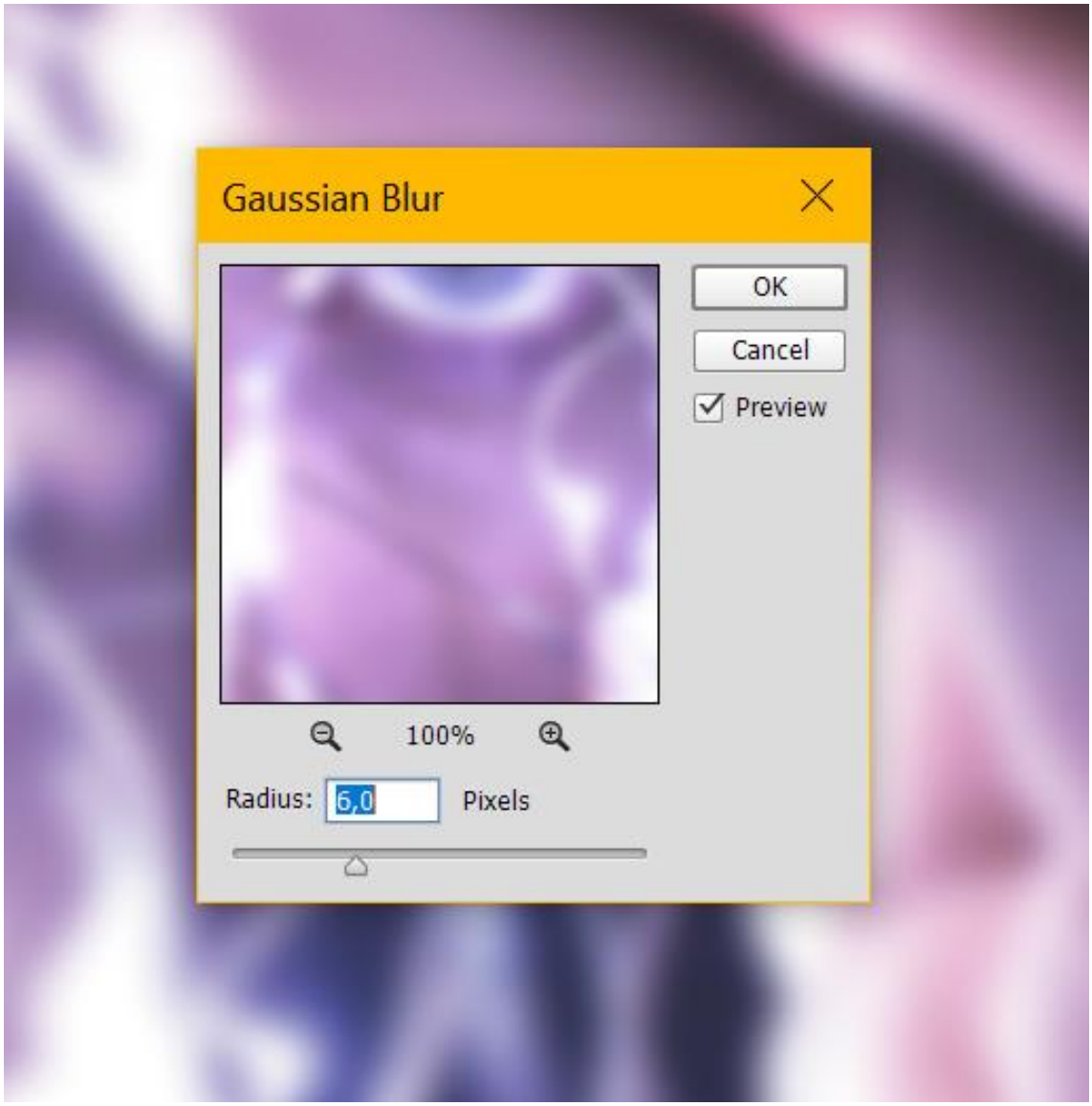
2.3 Graafisen ilmeen viimeistely

Kun rautalankamalli on todettu toimivaksi ja käytännölliseksi, voidaan lähteä miettimään ulkoasua. Ulkoasun kannalta merkittävää on sivuston käyttötarkoitus, sisältö ja käyttäjäryhmä. Koska kyseessä on verkkosovellus, joka hyödyntää tekoälyä, lähden rakentamaan ulkoasua sen pohjalta. Verkko on täynnä ilmaisia kuvapankkeja ja tähän tarkoitukseen käytin itse Pexels.com sivustoa. Tekoälystä ensimmäisenä mieleeni tulevat aivot, joten hakusanan ”Brain” avulla päädyin valitsemaan alla olevasta kuvasta keskimmäisen (kuva 4).



Kuva 4. Taustakuvan hankinta Pexels.com sivustolta

Koska taustakuva ei kuitenkaan ole täysin relevantti sovelluksen sisällön kannalta ja se on kovin hektinen, on sitä syytä hieman muokata, jottei taustakuva vie käyttäjältä kaikkea huomiota. Kuvaan lisätään Photoshopissa 6.0px sumennusta. Tähän tarkoitukseen käytin Gaussian-sumennus työkalua (kuva 5).

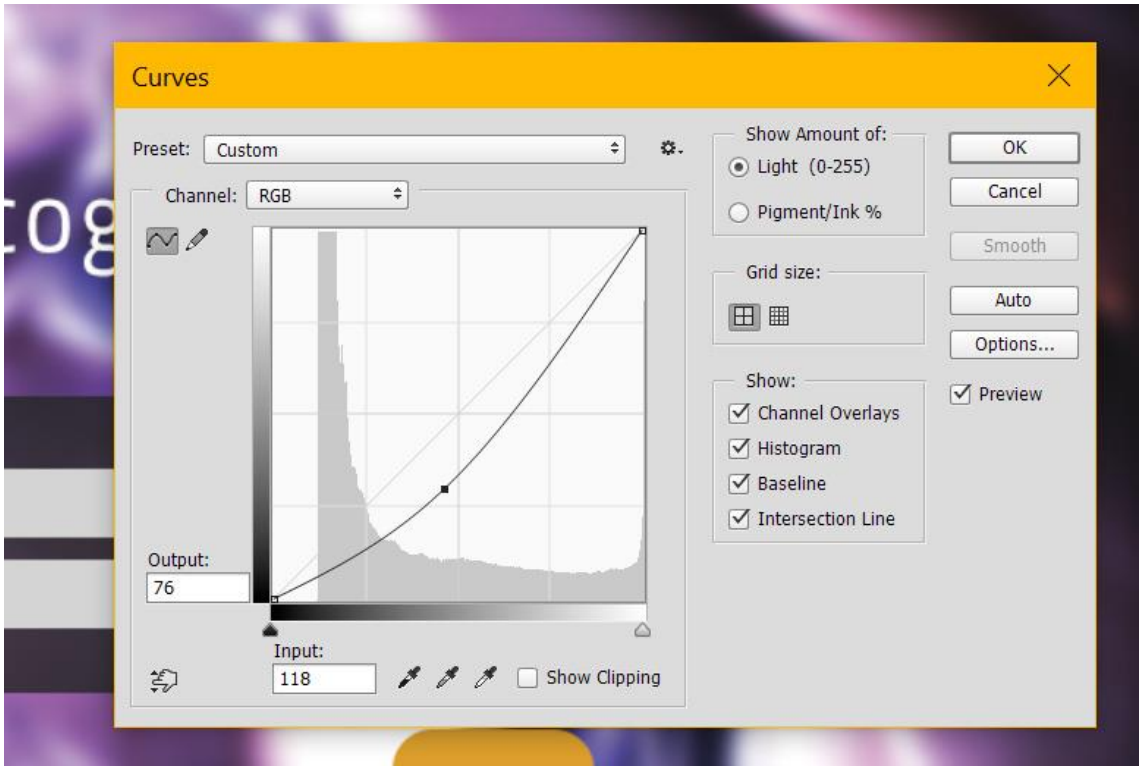


Kuva 5. Gaussian-sumennuksen lisääminen

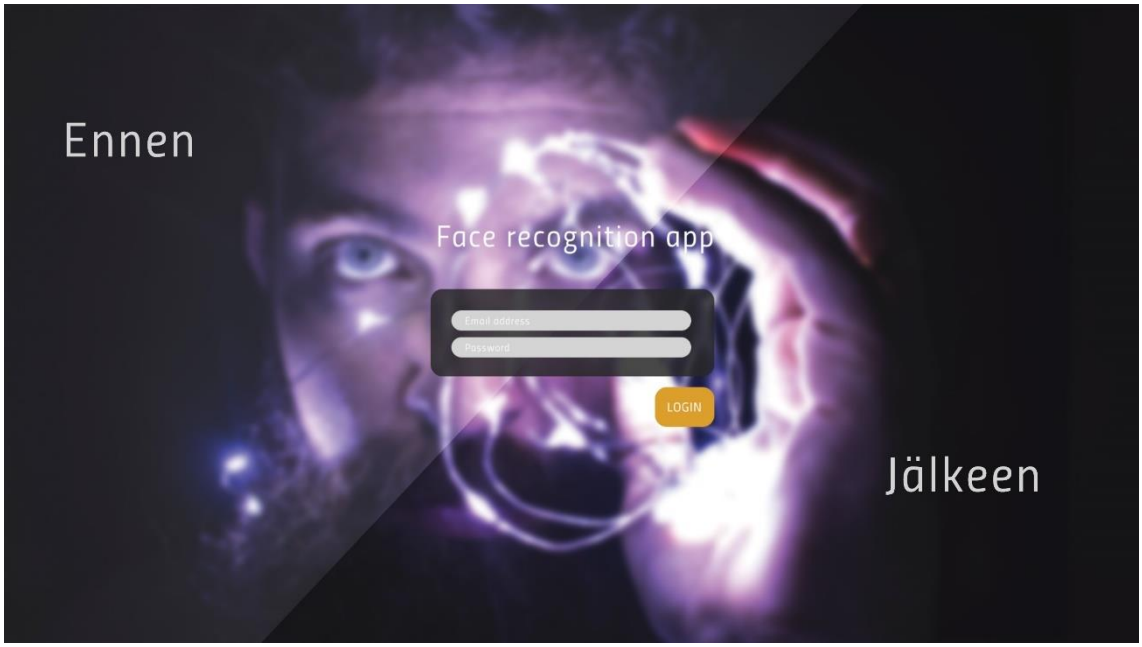
Seuraavassa vaiheessa lisätään tekstit sekä laatikkoelementit. Rautalankamallista poiketen päädyin käyttämään pyöristettyjä laatikoita, joissa reunojen pyöristys on 20 pikesliä. Tämä saavutettiin käyttämällä *Rounded Rectangle Tool* -työkalua tavallisen *Rectangle Tool* -työkalun sijasta, ja asettamalla reunojen pyöristykselle arvoksi 20px (kuva 6).

Kuva 6. Reunojen pyöristys

Kirjautumisalueen taustaväriksi valikoitui #282828, jolle annettiin 15% läpinäkyvyyttä. Syötelaatikoiden väriksi puolestaan #d4d4d4 ja kirjautumispainikkeen väriksi #dc9f2a. Koska halusin hyödyntää valkoista väriä teksteissäni, oli kuvasta tehtävä hieman dramaattisempi ja tummempi. Tämä saavutettiin hyödyntämällä Curves-työkalua (kuva 7). Tehostuksen ero havainnollistettu kuvassa 8.



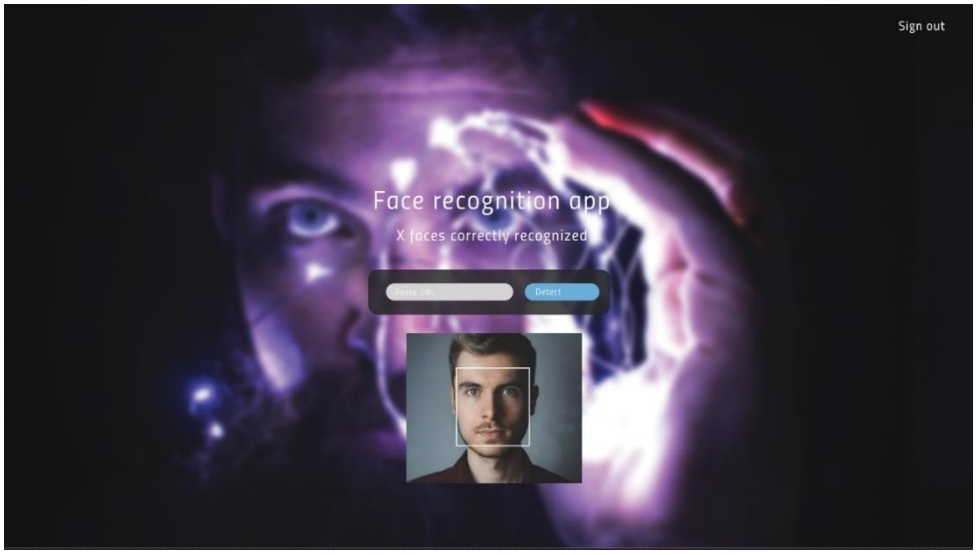
Kuva 7. Dramatiikan lisääminen Curves-työkalulla



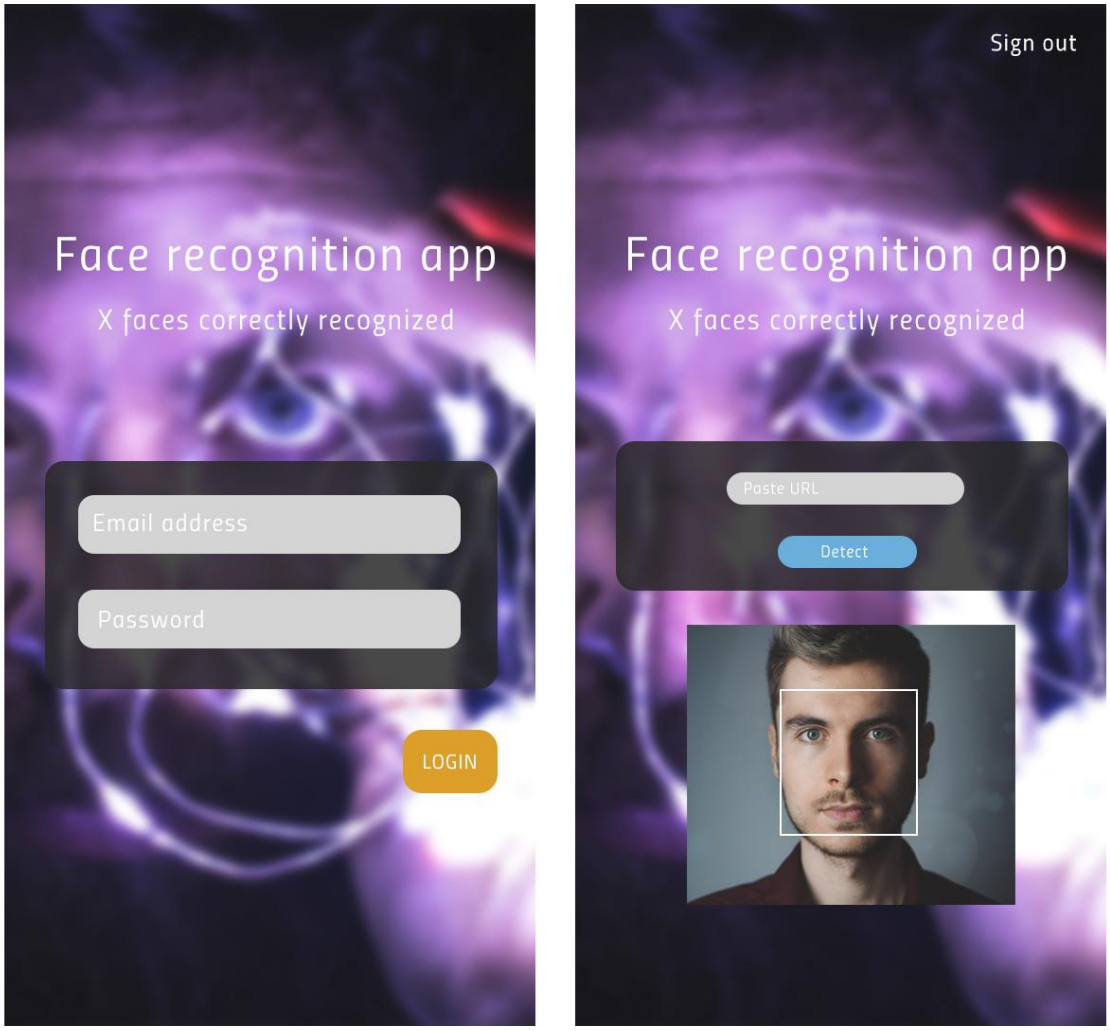
Kuva 8. Ennen ja jälkeen Curves-työkalun käyttämistä

Sama työskentelykaava suoritetaan myös toimintasivun ulkoasulle. Eroavuutena havaitsemistoiminnon käynnistävän painikkeen taustaväriksi on asetettu #6aafda sekä lisätty havainnekuva alaosioon, johon varsinaisessa sovelluksessa tulostuu käyttäjän syöttämä kuva. Kuvan päälle piirtyy laatikko hahmon kohdalle. Esimerkkinä käytetty henkilökuva on niinikään hankittu Pexels.com -sivustolta (kuva 9).

Tässä vaiheessa voimme todeta, että ulkoasu on suunniteltu loppuun. On huomioitava, että sivusto toimii RD:nä, joten mobiiliulkoasun suhteen on vielä otettava kantaa. Vähäisten elementtien ansiosta mobiiliulkoasu tulee olemaan todella helppo ja suoraviivainen, joten kovin drastisia muutoksia ei tulla tarvitsemaan suunnittelun eikä ohjelmoinninkaan puolelta. Elementit siirtyvät näppärästi luonnolliselle käytettävyysalueelle eikä ero verratten työpöytäversioon ole suuri (kuva 10).



Kuva 9. Toimintasivun ulkoasu



Kuva 10. Sivuston mobiilinäkymä

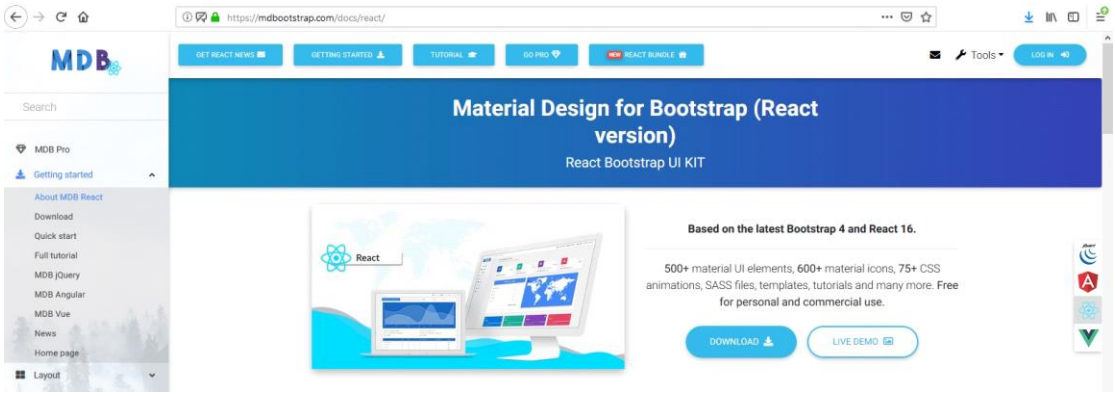
3 KEHITYSYMPÄRISTÖ

Sovelluksen kehittäminen toteutettiin React frameworkia hyödyntämällä. React on framework, joka helpottaa sovelluskehittäjän työtä. Sivustoja ja applikaatioita voidaan rakentaa komponenttimallilla, joka yksinkertaisuudessaan tarkoittaa sitä, että jo kertaalleen tehtyjä koodiosioita voidaan hyödyntää useaan kertaan pelkästään viittaamalla kyseiseen komponenttiin. Reactin on suunnitellut Facebook, joka myös ylläpitää kyseistä kirjastoa. Se on otettu ensimmäisen kerran käyttöön vuonna 2011 (avoimena lähdekoodina vuodesta 2013) ja tuorein repository on aina ladattavissa GitHubista (github.com/facebook/react). Reactin viralliset verkkosivut löytyvät osoitteesta www.reactjs.org. Kilpailevia, mutta yhtälailla hyviä frameworkeja ovat Angular.js sekä Vue.js.

Mitä vaihtoehtoja on ollut ennen Reactia? Yhtenä hyvin tunnettuna vaihtoehtona mainittakoon JQuery, joka oli (ja on edelleen) kovin monimutkainen framework. JQuerylla oli hyvin hankalaa seurata muutoksia DOM:ssa (Document Object Model), sillä muutosten seuraaminen oli hyvin hankalaa ja hidasta, joka edelleen aiheutti ongelmia, kun kehitysalusta skaalautui suureksi ja ohjelmoivan tiimin jäsenmäärä kasvoi. React kehitettiin tämän ongelman ratkaisemiseksi. Tavoitteena oli luoda framework, jonka käytös on helposti ennakoitavissa ja skaalautuu suuriin kokonaisuuksiin, ja jossa sovelluksen hallinointi tiimien kesken on yksinkertaista ja suoraviivaista. Vaikka React kehitettiinkin ratkomaan verkkosivujen ja -sovellusten ongelmia, voidaan Reactia käyttää myös älypuhelinten natiivisovellusten, tietokoneohjelmien sekä VR-sovellusten kehittämiseen. Hyvänä ja kaikille tuttuna esimerkkinä muun muassa Netflix käyttää Reactia, etenkin palvelinpuolellaan (*Medium 2019*). React mahdollistaa sovelluksen legomaisen rakentamisen - tiimit kehittävät yhtä osa-aluetta, joka voidaan helposti liittää osaksi suurempaa kokonaisuutta. Reactia voidaan ajatella komponenttimallina. Komponentti tarvitsee rakentaa vain kerran, ja sen jälkeen sitä voidaan hyödyntää sellaisenaan loputtomasti ilman, että koodia tarvitsee uudelleenkirjoittaa.

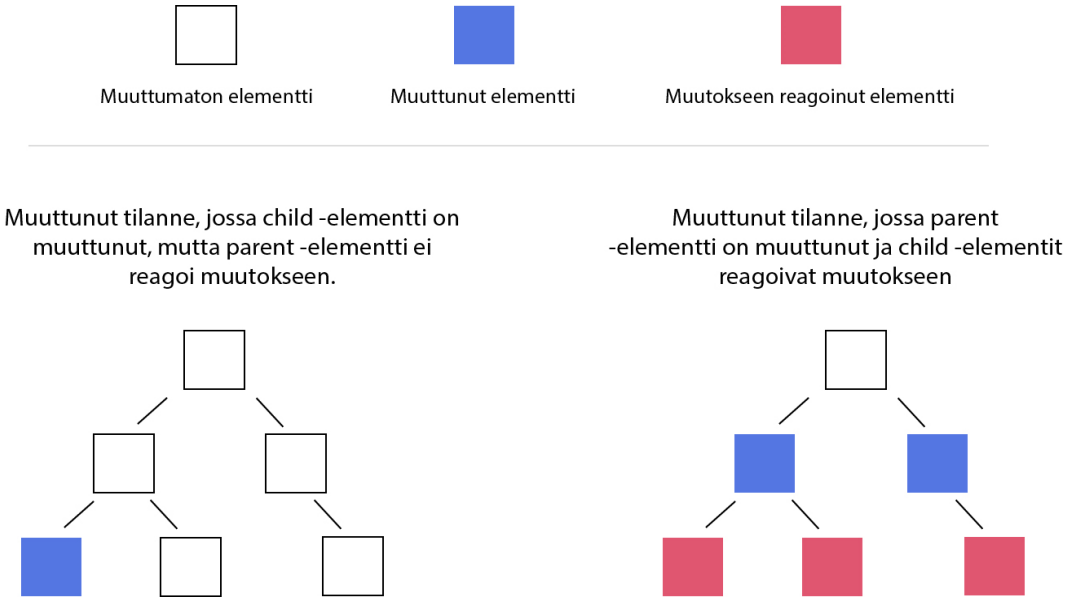
Reactia ympäröi myös valtava yhteisö. Valmiita komponentteja on vapaasti ladattavissa tuhansia, joten kaikkea ei tarvitse enää itse kirjoittaa. Joku on ohjelmoinut sen jo valmiiksi, ja voit vapaasti hyödyntää näitä komponentteja omissa sovelluksissasi. Yhtenä hyvänä esimerkkinä käytän MD-Bootstrap -sivustoa (www.mdbootstrap.com), josta kuka

tahansa voi käydä hakemassa itselleen ja tarpeisiinsa sopivia React komponentteja. Komponentit eivät ole pelkästään HTML- ja CSS-tiedostoja, joilla hallitaan sivuston sisälötä sekä ulkoasua, vaan oikeita toiminnallisuuksia, kuten kirjautumislomakkeita ja kalentereja (kuva 11).



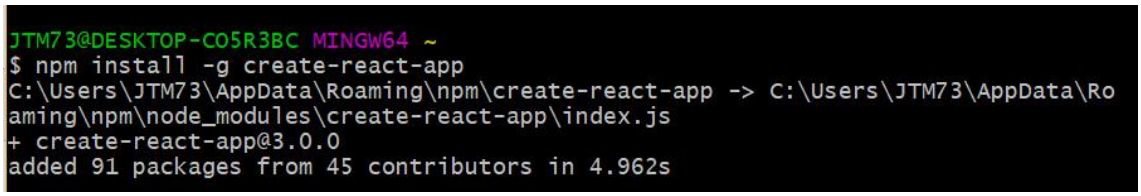
Kuva 11. MD Bootstrap

Kaiken tämän kokonaisuuden voi ajatella yksisuuntaisena datavirtana. Data kulkee ylhäältä alas päin ja täten "isännät" ovat kytköksissä "lapsiinsa", mutta ei koskaan ylhäältä alas päin, eli "lapset" eivät ole kytköksissä "isäntiinsä". Jatkossa viittaan isäntiin termillä parent ja lapsiin termillä child. Alla olevassa kuvassa havainnollistetaan yksisuuntaisen datavirran toiminnallisuutta (kuva 12).



Kuva 12. Yksisuuntainen datavirta

Kehittäjän kannalta edukkainta on se, että minkä tahansa ongelman kohtaatkin on jollain muulla todennäköisesti ollut se jo aiemmin, ja kirjoittanut ratkaisun siitä jollekin yhteisöalustalle. Tämä tekee kaikkien sovelluskehityksestä vilkkaampaa ja yksinkertaisempaa. React sovelluksen asentaminen tapahtuu terminaalin kautta helposti komennolla *npm install -g create-react-app*, jonka jälkeen varsinainen projekti voidaan luoda kätevästi komennolla *create-react-app application-name*, jossa *application-name* vastaa projektisi nimeä. Oletusnäkymään päästään komennolla npm-start, joka avaa selaimeen paikallispalvelimen *localhost:3000* (kuva 13).



Kuva 13. Reactin asentaminen.

3.1 Redux

Redux helpottaa sovelluksen tilanteen hallintaa. Yksinkertaisemmin selitettynä se auttaa kehittäjää hallitsemaan näytettävää dataa ja kuinka käyttäjät siihen reagoivat. Reduxin on kehittänyt Dan Abramov (*Three Devs and a Maybe 2018*). Hän halusi minimalistisen API:n (Application Programming Interface), joka parantaa sovelluksen struktuuria, ja jossa on hyödyllisemmät ja paremmat työkalut. Kokoa Reduxilla on vain 2 kilobitin verran. Reduxia kannattaakin käyttää, kun kehitettävä sovellus muuttuu todella kompleksiksi ja vaativaksi kokonaisuudeksi. Sivuamme Reduxia vielä myöhemmin tämän opinnäytetyön sovelluskehitykseen keskittyvässä kappaleessa.

4 SOVELLUKSEN KEHITTÄMINEN

Aikaisemmissa osioissa kävimme läpi hahmontunnistussovellukseen tarvittavien komponenttien toiminnallisuudet sekä niiden ulkoasut. Voimme nyt alkaa näiden tietojen ja määritysten perusteella muodostamaan sovellusta toimivaksi kokonaisuudeksi. Ohjelmointityössä helppolukuisuuden kannalta käytän muuttujissa englanninkielisiä nimiä sekä termejä.

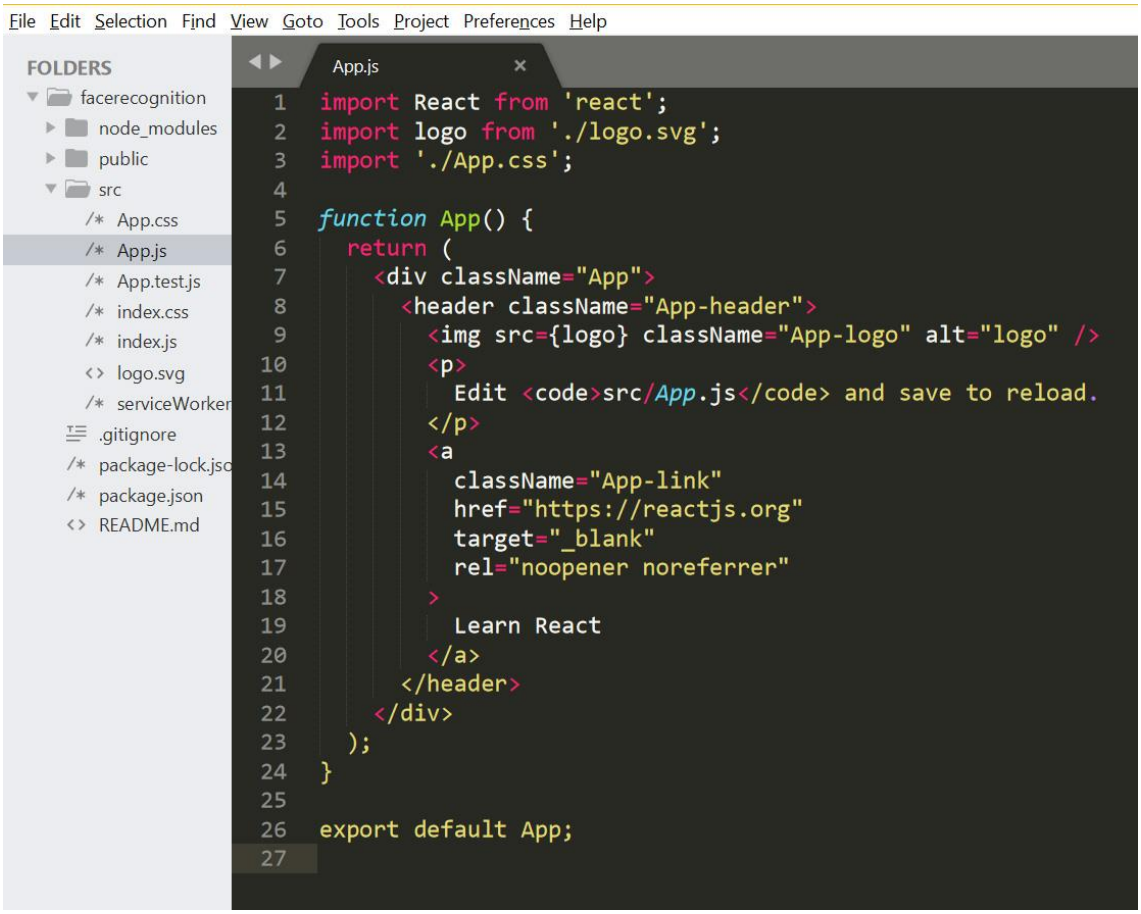
4.1 React-projektin luominen

Ensimmäiseksi tarkistetaan, että Reactista on ajantasalla oleva versio asennettuna tietokoneelle. Tämä voidaan tehdä komennolla *create-react-app -version.* Seuraavaksi sovellus alustetaan terminaalissa komennolla *create-react-app facerecognition*. Tämä komento luo meille projektitiedoston nimellä *"facerecognition"*, jonka alle voimme rakentaa hahmontunnistusohjelmamme (kuva 14).

```
JTM73@DESKTOP-CO5R3BC MINGW64 ~
$ create-react-app --version
3.0.0

JTM73@DESKTOP-CO5R3BC MINGW64 ~
$ create-react-app facerecognition

Creating a new React app in C:\Users\JTM73\facerecognition.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...

+ react@16.8.6
+ react-dom@16.8.6
+ react-scripts@3.0.0
added 1392 packages from 726 contributors and audited 878612 packages in 52.366s
found 0 vulnerabilities


Success! Created facerecognition at C:\Users\JTM73\facerecognition
Inside that directory, you can run several commands:
```

Kuva 14. React-projektin luominen

Käynnistäessämme sovelluksen komennolla *npm-start*, saamme näkymäksi Reactin oletussivun, sillä muutoksia yllä olevan kuvan koodiin ei ole vielä tehty. Tässä vaiheessa poistetaan tarpeettomia tiedostoja, kuten oletuslogo (logo.svg), sekä oletuskoodia, jota projektissa ei tulla tarvitsemaan (kuva 15).



Kuva 15. Projektin oletusnäkymä

4.2 Tachyonit

Tachyon on CSS-työkalu, joka mahdollistaa nopeasti latautuvan ja täysin responsiivisen rajapinnan nopean kehityksen. Kirjaston koko on alle 14kb ja se on äärimmäisen hyvin dokumentoitu. Tachyonien tarkoitus on keventää kehittäjän työtaakka valmiilla, ennaltamääritetyillä lyhenteillä, joiden avulla voidaan rakentaa elementtejä nopeasti (kuvat 16, 17 ja 18). (*Tachyons.io 2019)*.

```
13
14  /** Tavallinen CSS määritys elementin reunojen lisäämiselle
15      joka luo 1px levyisen, tasaisen mustan reunuksen **/
16
17  .esimerkki {
18
19      border: 1px solid black;
20  }
```

Kuva 16. Tavallinen CSS-määritys

```
16
17  /* Vastaava esimerkki Tachyonina */
18   <div className="ba">
19
```

Kuva 17. Vastaava määritys Tachyonilla

Kuva 18. Tarkemmat Tachyon-spesifikaatiot reunuksista

Tachyons.io sivusto tarjoaa valmiita komponentteja kenen tahansa käytettäväksi. Hyödynnän näitä valmiita komponentteja omassa työssäni työmäärän keventämiseksi sekä nopeuttamiseksi. Sivuhuomiona mainittakoon, että Tachyon ei kuitenkaan korvaa CSS:n käyttöä. Useita valmiita komponentteja löytyy laajalle skaalalle käyttötarkoituksia, mutta viimekädessä **spesifit** ulkoasulliset muutokset tehdään edelleen CSS-tyylejä hyödyntäen.

4.3 Komponenttilista

Perinteisin keinoin sivustoa luotaessa on totuttu manipuoloimaan *index.html* sekä *styles.css* (jossa *styles* = tyylitiedostosi nimi) tiedostoja. Reactia käytettäessä manipuoloimme komponentteja, jotka tuodaan (import) projektin App.js tiedostoon. Tämä helpottaa työskentelyä usealla eri tavalla; jos haivataan korjattavaa, tiedetään heti missä komponentissa korjattava asia sijaitsee, virheellistä koodia ei tarvitse etsiä tuhansien rivien seasta ja komponentit ovat monistettavissa, tarkoittaen sitä että niitä voidaan käyttää useassa eri paikassa helposti vain tuontikomentoa hyödyntämällä. Valmiin sovelluksen komponenttien tuontilista havainnollistettu kuvassa 19.

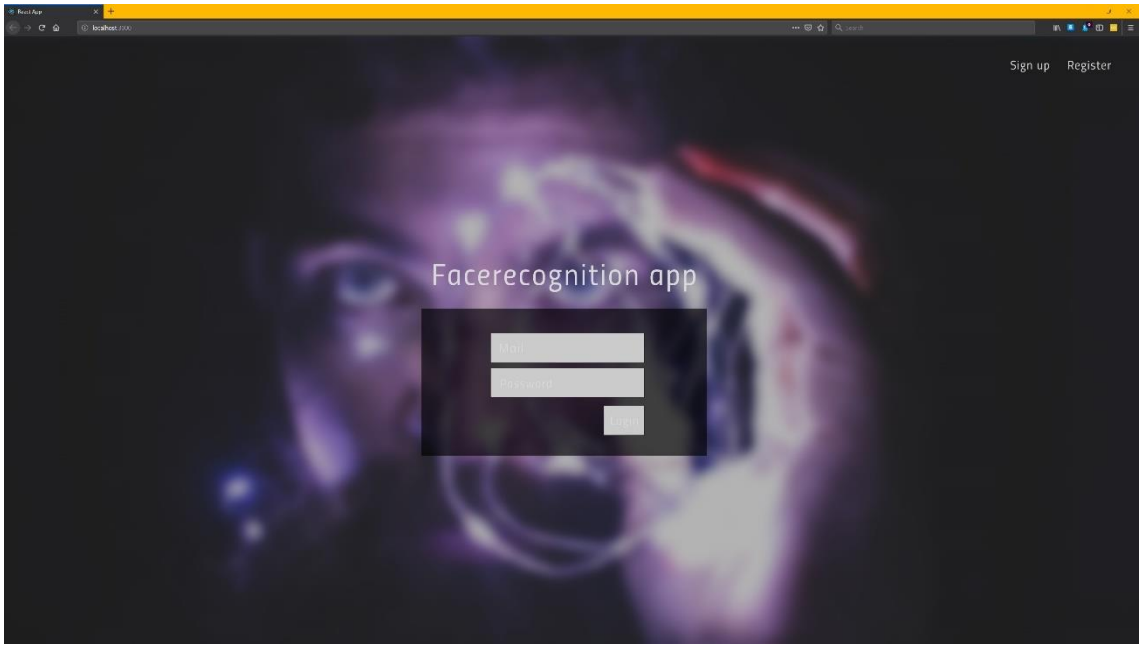
```
1  import React, { Component } from 'react';
2  import Clarifai from 'clarifai';
3  import FaceRecognition from './components/FaceRecognition/FaceRecognition';
4  import Navigation from './components/Navigation/Navigation';
5  import Signin from './components/Signin/Signin';
6  import Register from './components/Register/Register';
7  import Logo from './components/Logo/Logo';
8  import ImageURL from './components/ImageURL/ImageURL';
9  import './App.css';
```

Kuva 19. Komponenttien tuontilista

Oman kokemukseni mukaan komponenttirakentaminen on huomattavasti nopeampaa, sillä kehityksen aikana helposti yhtä tiedostoa manipuloidessa unohtaa millä rivillä jokin tietty elementti sijaitsee, riippumatta siitä kuinka huolellisesti työtään on dokumentoinut. Projektin skaalautuessa suureksi muuttujien nimien määrä alkaa olla kosminen, eikä niitä tästä syystä johtuen voi *find* toiminnolla etsiäkään. Sen sijaan komponenttilistaa tutkiessa on helppoa löytää oikea komponentti, olettaen että ne on järkevästi nimetty. Hyvä ja selkeä nimeämislogiikka kannattaakin pitää mielessä koko projektin ajan paitsi itseään, myös muita tiimiläisiä varten.

4.4 Sivuston päänäkymä

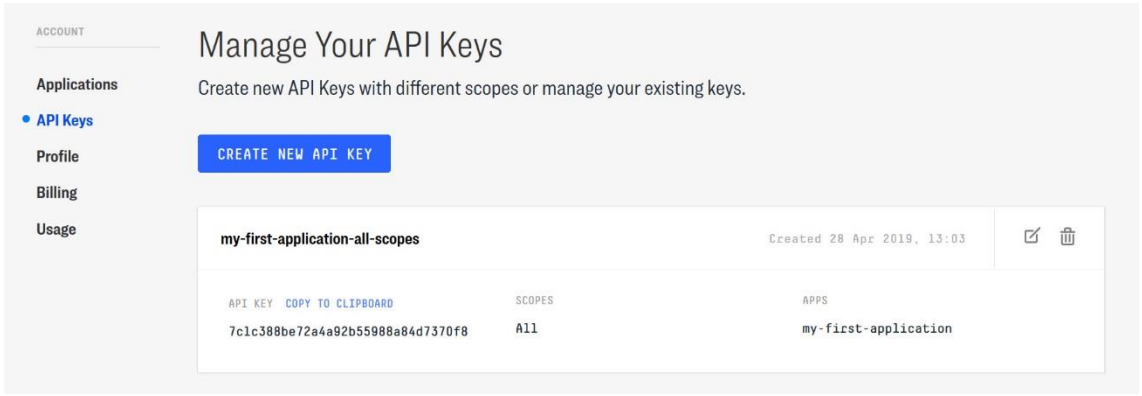
Tässä vaiheessa projektia olemme alustaneet kaikki tarvittavat komponentit pääsivun näkymän kannalta. Jokainen komponentti hoitaa omaa rooliaan ja ne ovat määritelty erikseen omiin tiedostoihinsa. Päänäkymä havainnollistettu kuvassa 20, ja se vastaa ennalta määritettyä suunitelmaa lukuunottamatta hienosäätöjä, kuten reunojen pyöristyksiä.



Kuva 20. Sovelluksen päänäkymä paikallispalvelimella ennen hienosäätöjä

4.5 Clarifai

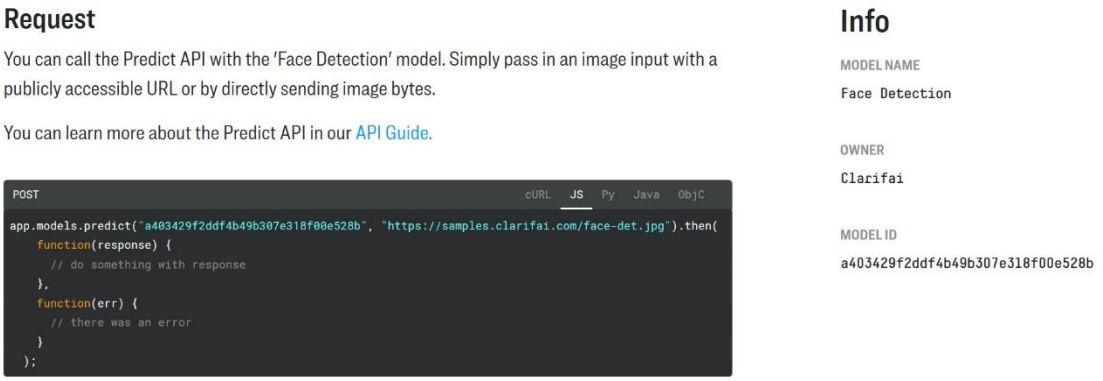
Hahmon tunnistamiseen olen valinnut Clarifai rajapinnan. Clarifai tarjoaa ilmaisena palveluna 5000 kuvan analysoinnin kuukaudessa. Kilpailevina palveluina mainittakoon Amazon sekä Megviin Face++. Clarifai valikoitu joukosta lähinnä helppokäyttöisyyden sekä keveyden vuoksi. Sivustolle tarvitsee vain rekisteröityä saadakseen API-avaimen (kuva 21), jonka avulla voit käyttää Clarifain hahmontunnistusrajapintaa. Sivusto löytyy osoitteesta www.clarifai.com. (*Clarifai 2019*).



Kuva 21. Clarifai API-avain

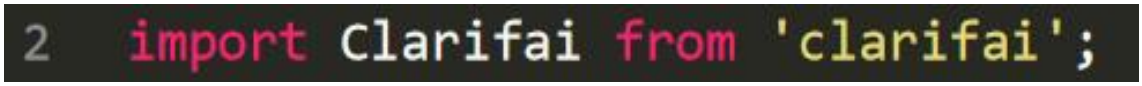
4.5.1 Rajapinnan käyttöönotto

Rajapinnan käyttöönottoon löytyvät erinomaiset dokumentaatiot suoraan Clarifain verkkosivuilta. Tässä tapauksessa kiinnostaa erityisesti JavaScript dokumentaatio (kuva 22).



Kuva 22. API:n käyttöönotto

Jotta rajapintaa voidaan käyttää, tulee huomioida, että projektimme ei tunnista *app.modle-predict('arguments').then* komentoa toistaiseksi, vaan sitä tulee soveltaa sitä omaan käyttöömme. Virallinen dokumentaatio antaa ohjeeksi asentaa rajapinta komennolla *npm install clarifai*, jonka jälkeen se tulisi lisätä sovellukseen perinteisin keinoin komennolla *const Clarifai = require('clarifai')*. Reactilla työskennellessä tahdomme kuitenkin käyttää *import* komentoa (kuvat 23, 24 ja 25).



Kuva 23. Clarifain implementointi

Kuva 24. Oman API-avaimen lisääminen

Client Installation Instructions

JS Py Java C# ObjC Php cURL

```
// The JavaScript client works in both Node.js and the browser.


// Install the client from NPM

npm install clarifai

// Require the client

const Clarifai = require('clarifai');

// initialize with your api key. This will also work in your browser via http://browserify.org/

const app = new Clarifai.App({
 apiKey: 'YOUR_API_KEY'
});

// You can also use the SDK by adding this script to your HTML:

<script type="text/javascript" src="https://sdk.clarifai.com/js/clarifai-latest.js"></script>
```

Kuva 25. Clarifain dokumentaatio sovelluksen asentamisesta ja implementoinnista

4.5.2 Kuvan esittäminen

Tavoitteena on luonnollisesti nähdä kuva, josta kasvoja pyritään tunnistamaan. Sen lisääminen ja keskittäminen on toteutettu FaceRecognition.js komponentilla hyödyntäen Tachyoneja sekä CSS-tyylejä (kuva 26).

```
1  import React from 'react';
2  import './FaceRecognition.css';
3
4  const FaceRecognition = ({ imageUrl, box }) => {
5    return (
6      <div className='center ma'>
7        <div className='absolute mt2'>
8          <img id='inputimage' alt='' src={imageUrl} width='500px' heigh='auto'/>
9          <div className='bounding-box' style={{top: box.topRow, right: box.rightCol
10       </div>
11     </div>
12   );
13 }
14
15 export default FaceRecognition;
```

Kuva 26. Tunnistettavan kuvan esittäminen ja keskittäminen

4.5.3 Tunnistelaatikko

Clarifai ei automaattisesti luo laatikkoa tunnistetun hahmon ympärille, vaan meidän tulee tehdä se manuaalisesti. Tätä prosessia varten tulee luoda oma funktio, joka laskee tunnistettun hamon sijainnin kuvan koordinaatistossa, ja piirtää tämän alueen ympärille laatikon. Tämä on saavutettu kuvan 27 mukaisella funktiolla.

```
61
62  calculateBoxLocation = (data) => {
63    const clarifaiFace = data.outputs[0].data.regions[0].region_info.bounding_box;
64    const image = document.getElementById('inputimage');
65    const width = Number(image.width);
66    const height = Number(image.height);
67    return {
68      leftCol: clarifaiFace.left_col * width,
69      topRow: clarifaiFace.top_row * height,
70      rightCol: width - (clarifaiFace.right_col * width),
71      bottomRow: height - (clarifaiFace.bottom_row * height)
72    }
73  }
74
```

Kuva 27. Hahmon sijainnin laskeminen

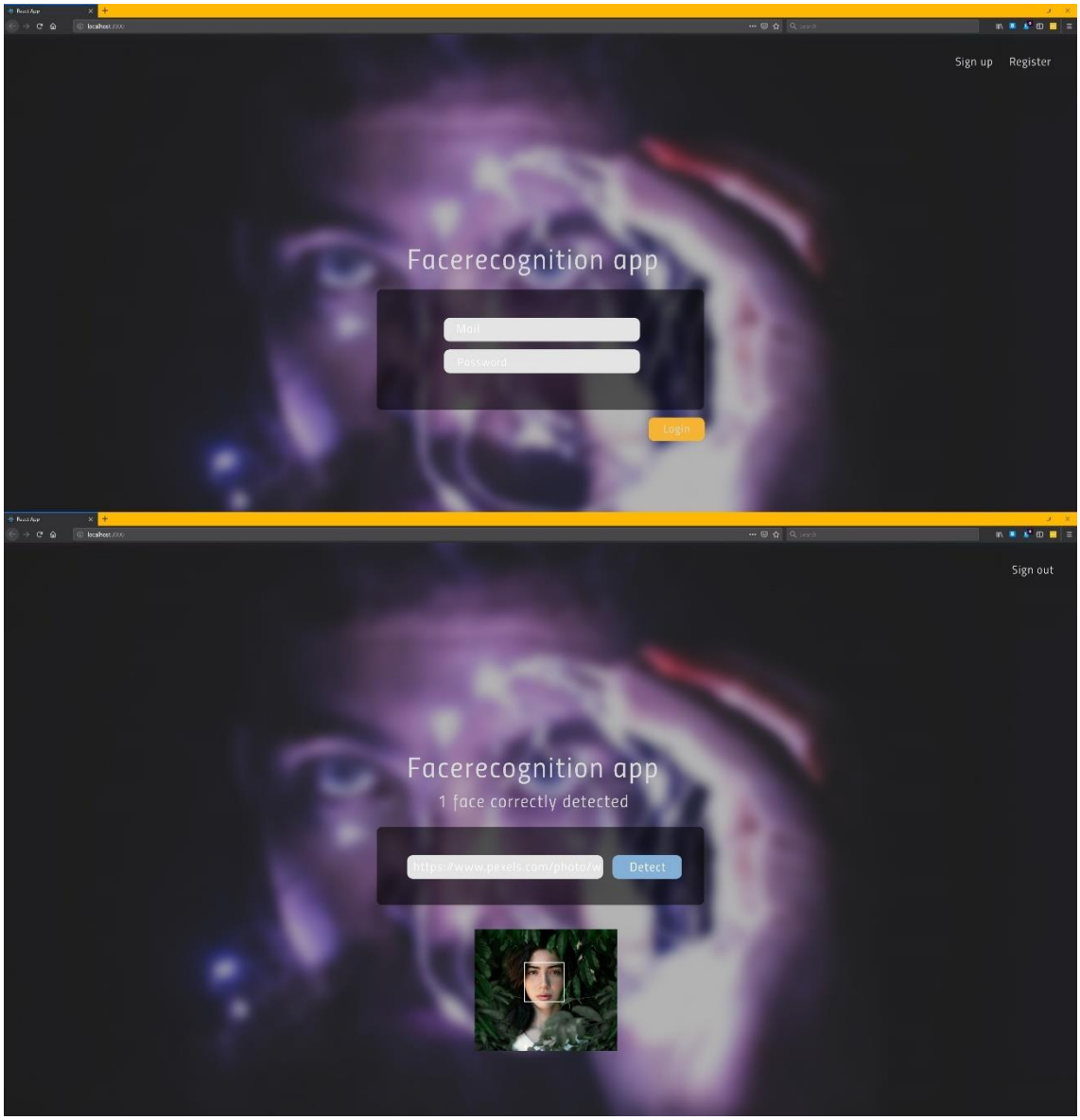
Funktiota kutsutaan erikseen, kun *”Detect”*-painiketta painetaan (kuva 28).

```
101          }
102          this.displayFaceBox(this.calculateBoxLocation(response))
103        })
```

Kuva 28. Funktion kutsuminen

4.6 Sivuston kokonaisnäkymä

Tässä vaiheessa projektiamme olemme saaneet komponentit valmiiksi, tehtyä ulkoasulliset hienosäädöt sekä muokattua Clarifain rajapintaa omiin käyttötarpeisiin sopivaksi. Hienosäädöt ovat olleet läihinnä CSS-tyylimäärityksiä, kuten reunojen pyöristystä ja elementtien kokojen muuttamista. Kokonaisuudessaan projekti paikallispalvelimella näyttää kuvan 29 mukaiselta.



Kuva 29. Kirjautumisnäkymä sekä tunnistusnäkymä paikallispalvelimella

Sovellus tunnistaa kuvasta kasvot ja voimme todeta, että sovellus toimii toivotulla tavalla

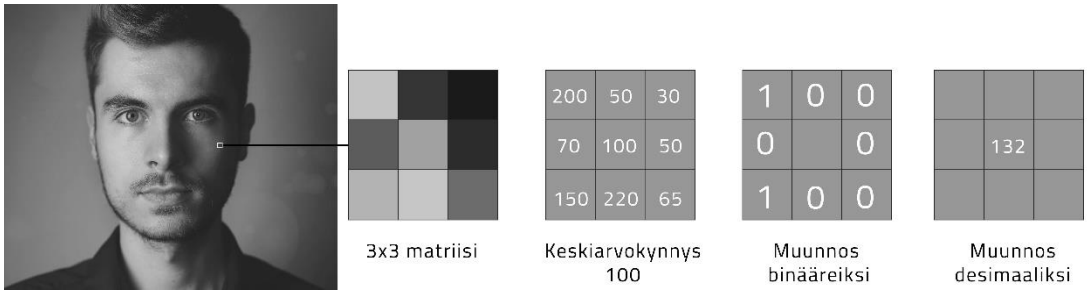
5 HAHMONTUNNISTUSALGORITMIT

Erilaisia algoritmeja hahmon tunnistamiseen löytyy useita. Tästä syystä luettelen muutamia hyvin tunnetuja ja pitkään käytettyäjä algoritmejä, joista parhaiten keskitymme Local Binary Patterns Historgarms (LBPH) algoritmiin, joka on kehitetty vuonna 1996 *(Pietikäinen 2010).* Jatkossa viittaan kyseiseen algoritmiin LBPH-lyhenteellä. Näitä sivumaininnan arvoisia algoritmeja ovat Eigenfaces, Fisherfaces, Scale Invariant Feature Transform (SIFT) sekä Speed Up Robust Features (SURF).

Local Binary Pattern (LBP) on hyvin yksinkertainen, kuitenkin varsin tehokas algoritmi, joka käy kuvan pikseli pikseliltä läpi 3x3 matriiseina. Tämä matriisi sisältää yhdeksän solua, joista jokainen solu sisältää sävyarvoja. Matriisin voi ajatella vaikkapa ruutupaperin ruudukkona. Yhdistämällä LBP ja Histogram of Oriented Gradients (HOG) saadaan tuloksista huomattavasti luotettavimpia. Tämä johtuu siitä, että yhdistämällä LBP sekä histogramit, kasvokuvat voidaan esittää yksinkertaisena datavektorina.

5.1 LBP-toiminto

Algoritmi kääntää kuvan mustavalkoiseksi, jotta sen on helpompi analysoida kuvaa. Tämän jälkeen kuva käydään pikseli pikseliltä läpi 3x3 matriiseina. Jokaisen ruudun keskipisteen sekä sen naapureiden keskiarvo esitetään sävyn intesiteettinä asteikolla 0-255, jossa 0 on tummin mahdollinen ja 255 kirkkain mahdollinen sävy. Kun nämä arvot ovat määritetty, annetaan jokaiselle naapurille uusi arvo binäärinä, jossa 1 on sama tai suurempi kuin matriisin keskipisteen arvo, ja jossa 0 on pienempi kuin matriisin keskipisteen arvo. Kun koko kuva on käyty pikseli kerrallaan läpi tällä metodilla on algoritmilla käytettävissään kuva, joka koostuu vain ja ainoastaan binääriluvuista. Jokainen matriisi käydään vielä erikseen läpi, jonka aikana binäärit käännetään desimaaleiksi ja sijoitetaan lopuksi matriisin keskelle. Binääreiden luku tapahtuu vasemmalta oikealle sarake kerrallaan (OpenCV 2019). Kuvan 30 esimerkin mukaisesti binääriluvuksi tulisi siis 10000100, jonka desimaali on 132.



Kuva 30. Algoritmin opettaminen

5.2 Algoritmin opettaminen

Algoritmille tulee syöttää paljon dataa. Data esitetään useina kuvina henkilöistä, jotka halutaan tunnistaa. Kuville on syytä antaa myös ID-numero, jotta algoritmi tunnistaa mihin henkilöön sen tulisi viitata palauttaessaan arvon. Saman henkilön kuvilla on oltava sama ID-numero.

5.3 Parametrit

LBPH käyttää neljää erilaista parametria. Esitän parametrit englanniksi helppolukuisuuden ja yleisen käyttötavan vuoksi. Nämä neljä parametria ovat *radius, neighbors, grid X* sekä *grid Y*.

- *Radius* viittaa säteeseen, jota käytetään rakentamaan matriisin koko. Oletusarvo on tavallisesti 1. Tämä tarkoittaa sitä, että keskipisteen ympäriltä kuljetaan yhden pikselin verran jokaiseen suuntaan. *Radiuksen* keskellä on matriisin keskimmäinen solu.

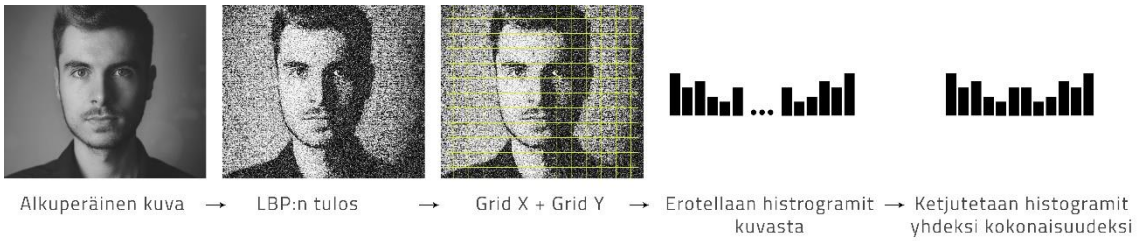
- *Neighbours* viittaa keskimmäistä solua ympäröiviin soluihin. Oletusarvo tavallisimmin kahdeksan, kun *radiuksen* arvo on yksi.

- *Grid X* viittaa solujen lukumäärä horisontaalisella akselilla. Mitä enemmän soluja, sitä laajempi vektorin ulottuvuus.

- *Grid Y* viittaa solujen lukumäärä vertikaalisella akselilla. Mitä enemmän soluja, sitä laajempi vektorin ulottuvuus.

5.4 Histogrammit

Histogrammi erotetaan siitä kuvasta, joka on luotu LBP-toiminnon avulla. Tässä vaiheessa algoritmi hyödyntää *Grid X* ja *Grid Y* parametreja jakamaan kuvan useaan eri soluun. Jokaisen solun sisältämä data käydään läpi ja niistä luodaan histogrammi. Kun kaikki solut on käyty läpi, ketjutetaan nämä histogrammit yhteen. Lopputuloksena on yksi suuren kokoinen histogrammi, joka edustaa alkuperäistä kuvaa paremmin datatasolla (kuva 31).



Kuva 31. Histogrammin ketjutus

Kun algoritmi on opetettu käyttäytymään toivotulla tavalla, on jäljellä ainoastaan varsinainen kasvojen tunnistaminen. Algoritmin tehtävänä on verrata kahta histogrammia keskenään. Tähän voidaan hyödyntää esimerkiksi eukliidista etäisyyttä tai khiin neliötä, joiden perusteella algoritmi pystyy päättelemään onko kuvassa oleva henkilö tunnistettu vai ei. Näihin pureutuminen sopii esimerkiksi jatkotutkimukseen, jossa käydään läpi matemaattiset aspektit kasvojentunnistuksessa.

6 KASVOJENTUNNISTUKSEEN LIITTYVÄT ENNAKKOLUULOT

Hahmontunnistusta käytetään useisiin eri käyttökohteisiin. Kaikille tutuimpia ovat ehkäpä Snapchatin ja Instagramin kasvofilterit, joissa ohjelma tunnistaa käyttäjän kasvot ja luo niihin hauskoja virtuaalisia elementtejä, kuten vääristää, vanhentaa tai nuorentaa. Tekniikka ulottuu kuitenkin myös valvontaan. Monien nykyisten älypuhelinten lukitus on avattavissa pelkillä kasvoilla. Sitä on mahdollista myös käyttää vaikkapa kotioven avaamiseen ja siihen tarkoitettuja itseasennettavia IoT-laitteita myydään kuluttajille esimerkiksi Amazonissa.

Ihmisten huoli kuitenkin nousee esiin, kun hamontunnistuksen sijaan aletaan puhua kasvojentunnistuksesta, ja ymmärrys kasvojentunnistustekniikan yleisyydestä ja käyttötarkoituksista leviää. Jotkut saattavat kokea sen jopa ahdistavana, kun kävellessäsi kadulla kaupan nurkassa oleva valvontakamera tunnistaa sinut henkiötasolla. On siis selvää, että tämän päivän teknologia herättää ihmisissä Orwellilaisia pelkoja. Teknologiat on kuitenkin lähtökohtaisesti suunniteltu parantamaan ihmisten sekä yritysten turvallisuutta.

6.1 Etiikka

Lausahdus ”isoveli valvoo”, jolla viitataan ihmisten valvontaan valtion toimesta, lienee kaikille varsin tuttu. Vuosituhannen alussa asia oli lähinnä vitsi, mutta tänä päivänä todellisuus ei ole kovin kaukana, sillä tekoälyllä varustettuja tekniikoita voidaan helposti sulauttaa vanhoihin CCTV-kameroihin *(Safr 2019).* Meidät voidaan tunnistaa kävelykaduilta ja ruokakaupoista – joskus jopa keskeltä kymmenien tuhansien ihmisten konserttiakin.

Kiinassa on 170 miljoonaa valvontakameraa maanlaajuisesti ja tarkoituksena lisätä kapasiteettia 400 miljoonalla seuraavan kolmen vuoden aikana *(BBC 2018).* BBC:n mukaan Kiinan poliisi on käyttänyt kasvojentunnistustekniikkaa paikantaakseen ja pidättääkseen miehen, joka on ollut keskellä 60 000 ihmisen konserttia.

Moinen uutinen voi olla monelle shokeeraavaa, toiset taas saattavat ajatella, että ”hyvä juttu, itseä ei haittaa kun ei minulla ole mitään salattavaakaan”. Rohkenen silti epäillä,

että kukaan ei tahtoisi oman kotinsa jokaiseen nurkkaan kameraa, jonka kautta kuka tahansa voisi katsella tekemisiään.

Suurimmaksi huoleksi monella varmasti nousee tunnistamisen tarkkuus. Voiko mikään olla 100% luotettava? Tämä puolestaan herättänee huolta siitä, voisiko tulla vahingossa tekoälyn tuoman kasvojentunnistustekniikan kautta tunnistetuksi rikokseen, jota ei oikeasti ole tehnyt ja jos näin käy, kuka on vastuussa? Kasvojentunnistusta on alettu käyttämään myös taistelussa masennusta vastaan. Carnegie Mellonin yliopistossa vuonna 2015 käyttöön otettu MultiSense analysoi ihmisen kasvoja ja pyrkii päättelemään tämän mielialoja *(LTI 2015).* Nyt ilmoille nouseekin kysymys, voiko mahdollisuutena olla myös väärän diagnoosin saaminen?

Kolmantena haluan nostaa esiin tekoälyn yleisesti, vaikka tämä pohdinta ei suoraan liitykään juuri kasvojen tunnistukseen. Itsestään ajavat autot eivät ole enää kaukaisen tulevaisuuden haaveita, vaan niitä on ollut liikenteessä jo muutaman vuoden ajan. Itsestään ajavat autot voidaan jakaa kuuteen eri tasoon (*IoT For All 2017)*.

- Taso 0: Ei automaatiota. Kuljettaja hoitaa kaiken itse.

- Taso 1: Kuljettajan avustus. Auto avustaa kuljettajaa hieman, mutta kuljettaja hoitaa itsenäisesti kiihdyttämisen, jarruttamisen sekä huolehtii ympärillä tapahtuvien asioiden huomioimisesta.

- Taso 2: Osittainen automaatio. Auto voi avustaa joissain osa-alueissa, kuten kiihdyttämisessä tai ohjauksessa. Näissäkin päävastuu on kuitenkin kuljettajalla.

- Taso 3: Ehdollinen automaatio. Tällä tasolla auto huomioi ympäristöä. Se käyttää hyödykseen erilaisia sensoreita, joista yhtenä LIDAR. Kuljettajan vastuu on edelleen kriittinen, mutta auto voi hoitaa esimerkiksi jarrutamisen.

- Taso 4: Korkea automaatio. Auto on kykenevä ohjaamaan, kiihdyttämään, jarruttamaan sekä monitoroimaan tietä ja liikennettä. Tällä tasolla auto ilmoittaa kuljettajalle, että olosuhteet ovat turvalliset pilotoinnin käyttämiselle.

- Taso 5: Täysi automaatio. Auto on kykenevä kulkemaan liikenteessä ilman kuljettajan apua täysin itsenäisesti.

Itsestään ajaviin autoihin liittyy valtavan paljon ennakkoluuloja, mutta sitäkin enemmän eettisiä kysymyksiä. Kuvitellaan tilanne, jossa kuljettaja ajaa kapeaa tietä liukkaalla talvisella kelillä. Vastaan tulee rekka, joka menettää hallinnan ja on tulossa kohti. Törmäys on välttämätön, ellei henkilöauto ohjaa itseään oikealla olevaan ojaan. Juuri samassa kohdassa on jalankulkija tien oikeassa reunassa. Valittavaksi jää pelastaako automatisoitu henkilöauto kuljettajansa ohjaamalla ajoneuvon oikealle, mutta samalla todennäköisesti surmaten jalankulkijan vai pelastaako tekoäly jalankulkijan, mutta surmaa kuljettajansa?

Näiden eettisten ongelmien parissa autotehtaat joutuvat painimaan päivittäin. Kuka haluaisi ostaa auton, joka ei pelasta omistajaansa? Toisaalta samaan aikaan voidaan pohtia kuka tahtoisi ostaa auton, joka päättää ajaa viattoman jalankulkijan yli pelastaakseen kuljettajansa hengen, mutta jättäen kuljettajan miettimään olisiko voinut tehdä jotain toisin säästääkseen tämän jalankulkijan hengen ja kärsimään mahdollisisita henkisistä ongelmista pitkäksi aikaa?

7 KESKUSTELU

Varsinaisen kasvojentunnistusohjelman luominen alusta alkaen itse ei todellakaan ole mikään läpihuuto juttu. Tekoälylle tulee syöttää tuhansia, mielellään kymmeniä tuhansia kuvia, joiden perusteella se oppii tunnistamaan mitä sen pitäisi havaita. Tämä johtaa myös herkästi syrjintään; mikäli ohjelmalle syötettäisiin kymmeniä tuhansia kuvia kaukaasialaisista miehistä, mutta vain muutamia satoja kuvia afrikkalaisista naisista, tunnistaisi tekoäly kaukaasialaiset miehet luonnollisesti paljon tarkemmin ja luotettavammin, sillä tätä dataa on syötetty tekoälylle enemmän analysoitavaksi. Sen sijaan valmiin tekoälyrajapinnan lisääminen omaan ohjelmaan on varsin yksinkertaista ja suoraviivaista, ja tämä mahdollistaakin monien omien luovien projektien rakentamisen varsin helposti. Tämä oli myös oman opinnäytetyöni päämärä sovellussuunnittelun ja eettisen pohdinnan ohella. Tältä kantilta katsottuna on vain muutamien vuosien päässä sellainen todellisuus, jossa meidät voidaan tunnistaa kaduilta koska tahansa. Se on samaan aikaan sekä helpottavaa etenkin rikollisuuden kannalta, mutta myös ahdistavaa oman yksityisyyden kannalta.

Kiinan poliisi on jo ottanut käyttöönsä älylasit, jotka näyttävät tavallisilta aurinkolaseilta, mutta kätkevät sisäänsä kosolti kehittynyttä teknologiaa. Ne skannaavat ympäristöä analysoiden ihmisten kasvoja, ja etsivät sieltä potentiaalisia rikoksia. Lasit ovat yhteydessä valtion tietokantaan, jossa niitä verrataan etsintäkuulutettujen rikollisten kasvoja vastaan. Tunnistaessaan rikollisen poliisi saa välittömästi tietoonsa paitsi henkilön nimen, myös kotiosoitteen (*TechCrunch 2018).* Luultavimmin kyseessä ovat vain niin kutsutut ”kuumat” rikolliset, jotka ovat syyllistyneet vakavampiin rikoksiin kuin karkkikaupasta näpistelyyn, mutta siitä huolimatta on hieman karvoja pystyttävää ajatella kuinka pitkälle teknologiamme on jo kehittynyt.

Puhuttaessa tekoälystä yleisemmällä tasolla jättäen pelkän kasvojentunnistuksen omaan arvoonsa, moni keksii omasta elämästään varmasti useammankin osa-alueen, jossa tekoälyn tarkkuus aiheuttaa huolta. Se voi olla kasvojentunnistusta hyödyntävä lukitus puhelimessa tai kotiovessa, ehkä oman autonomisen auton eettisten tilanteiden ratkaisutavat. Joka tapauksessa ei ole typerää ajatella, että tekoäly ei ole meitä viisaampi – vielä. Tekoäly on juuri niin älykäs kuin se on opetettu olemaan. Vaikka itseoppivia tekoälyjä on jo olemassa, se ei tarkoita sitä että se kykenisi ajattelemaan kuin ihminen. Kasvojentunnistusta hyödyntävän puhelimen lukituksen voi ainakin omassa Android-

pohjaisessa älypuhelimessani ohittaa omien kasvojeni valokuvalla, sillä konenäkö ei kykene erottamaan onko kyseessä valokuva vai ei – se näkee ainoastaan kasvot jotka vastaavat niitä arvoja jotka sille on asetettu. Vastaavasti aiemmin kertomassani itsestään ajavien ajoneuvojen eettisessä ongelmassa tekoäly ei ole tarpeeksi älykäs päättämään kenen henki on arvokkaampi. Eikä sen mielestäni pitäisikään. Toistaiseksi koneet eivät ole inhimillisiä eivätkä ne osaa analysoida tunteita saatika tuntea niitä, eivätkä täten voisi mielestäni päättää elämää ja kuolemaa koskevia ongelmia.

Ainakin vielä toistaiseksi tekoälyle kyllä on *kyllä* ja ei on *ei*. Ihmiselle se voi olla *ehkä*. Ihmistä ja inhimillisyyttä tarvitaan edelleen ja vaikka teknologia kehittyy päätähuimaavaa vauhtia, toivon itse ettei niiden tarve katoaisi koskaan, sillä jokainen ihminen ajattelee asiat ja tilanteet eri tavalla tehden sekä järki- että tunnepohjaisia ratkaisuja. Tekoäly tekee sen siten, kuin se on opetettu, ja täten vastaa vain muutaman ihmisen ennalta määritettyihin arvoihin.

Tämän opinnäytetyön pohjalta voisi kehittää useammankin jatkotutkimuksen. Henkilökohtaisesti minua kiinnostaisi tutkia tekoälyn tarkkuutta, ja suhteutettuna opinnäytetyöni sisältöön, se tarkoittasi kuinka tarkasti tekoäly pystyy tunnistamaan konkreettiset kasvot. Miten tekoäly pystyy päättelemään, onko kyseessä oikea ihminen vai valokuva? Oletettavasti tähän tarvittaisiin konenäkö, joka pystyy analysoimaan tilaa myös syvyysakselilla, mutta pystyisikö tätäkin menetelmää huijaamaan parallax-tekniikalla, jossa syvyyden illuusio luodaan useasta valokuvasta, jotka on asetettu eri syvyyksille? Mitä pidemmälle näitä asioita ja dilemmoja pohtii, sitä enemmän herää uusia kysymyksiä, jotka niin ikään herättävät uusia kysymyksiä.

Jään mielenkiinnolla odottamaan mitä seuraavat viisi, saati kymmenen, vuotta tuovat tullessaan ja missä vaiheessa voimme oikeasti luottaa siihen, että tekoäly on oikeasti tarpeeksi älykäs tekemään etenkin niitä tunnepohjaisia ratkaisuja. Tätä opinnäytetyötä kirjoittaessani luotan toistaiseksi enemmän omiin aivoihini.

LÄHTEET

BBC 2018. Chinese Man Caught by Facial Recognition at Pop Concert. Viitattu 13.4.2019. https://www.bbc.com/news/world-asia-china-43751276

Clarifai 2019. Transforming Enterprises with Computer Vision AI. Viitattu 28.5.2019. https://clarifai.com/

Drawn Digital 2018. Choose Your Strategy: Mobile-First Web Design vs. Responsive Web Design. Viitattu 28.4.2019. https://darwindigital.com/mobile-first-versus-responsive-web-design/

LTI 2015. Morency's Multimodal Research Helps Doctors Diagnose Mental Illness. Viitattu 17.5.2019. https://www.lti.cs.cmu.edu/news/morencys-multimodal-research-helps-doctors-diagnose-mental-illness

Medium 2019. A Netflix Performance Case Study. Viitattu 5.5.2019. https://medium.com/dev-channel/a-netflix-web-performance-case-study-c0bcde26a9d9

OpenCV 2019. Face Recognition with OpenCV. Viitattu 22.5.2019. https://docs.opencv.org/2.4/modules/contrib/doc/facerec/

Pietikäinen, M. 2010. Local Binary Patterns. Viitattu 3.5.2019. http://www.scholarpedia.org/article/Local_Binary_Patterns

Safr 2019. Implementing Facial Recognition to a CCTV. Viitattu 28.4.2019. https://www.safr.com

TechCrunch 2018. Chinese Police Are Using Smart Glasses to Identify Potential Suspects. Viitattu 2.4.2019. https://techcrunch.com/2018/02/08/chinese-police-are-getting-smart-glasses/

Tachyons.io 2019. Using Tachyons for Fast Responsive Design. Viitattu 28.4.2019. https://tachyons.io/

Three Devs and A Maybe 2018. The History of React and Flux with Dan Abramov. Viitattu 27.5.2019 https://threedevsandamaybe.com/the-history-of-react-and-flux-with-dan-abramov/

UCSB 2013. Face Identification Accuracy is in the Eye (and Brain) of the Beholder. Viitattu 23.5.2019. https://www.news.ucsb.edu/2013/013586/face-identification-accuracy-eye-and-brain-beholder-ucsb-researchers-say