Opinnäytetyö AMK

Tieto- ja viestintätekniikka

2020

Tatu Riihimäki

VISUAALISEN ETÄVALVONTARATKAISUN PROTOTYYPPI

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2020 | 42

Tatu Riihimäki

VISUAALISEN ETÄVALVONTARATKAISUN PROTOTYYPPI

Esineiden internetin kasvaessa etävalvonnan tarve ja mahdollisuudet lisääntyvät. Olemassa olevat etävalvontaratkaisut ovat usein kalliita ja käytettävissä vain yhdessä käyttökohteessa. Lisäksi etävalvonnan mahdollistaminen saattaa vaatia ehjän laitteen vaihdon uuteen, etävalvottavaan malliin, joka ei ole kestävän kehityksen mukaista.

Opinnäytetyössä luotiin prototyyppi etävalvonnan mahdollistavasta ratkaisusta, jolla voidaan valvoa mitä tahansa visuaalisesti eli kameralla valvottavaa asiaa. Tavoitteena oli luoda edullinen, ekologinen ja joustava ratkaisu, joka voisi toimia alustana räätälöidymmille etävalvonta-ratkaisuille mahdollistamalla vaivattoman integroinnin muihin sovelluksiin.

Etävalvontaratkaisu toteutettiin luomalla edullinen ESP32-CAM-pohjainen kameralaite ja REST-ohjelmointirajapintainen palvelinpuoli pilvialustalle. Lisäksi käyttöliittymä etävalvontatiedon katseluun toteutettiin Angular-sovelluskehystä käyttäen.

Prototyypin onnistuneesta toteutuksesta saatiin toivotusti havaintoja jatkokehityksen tueksi. REST-ohjelmointirajapinta todettiin erittäin hyväksi ja toimivaksi ratkaisuksi palvelinpuolelle sen mahdollistaessa tavoitellun joustavuuden erinomaisesti. ESP32-CAM osoittautui myös erinomaisesti tavoitteet täyttäväksi järjestelmäpiiriksi. Työssä nousseiden havaintojen ansiosta jatkokehitys on mahdollista toteuttaa hyvin ja keskittää olennaisiin asioihin.

ASIASANAT:

Etävalvonta, prototyyppi, REST, API, ESP32, IoT

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2020 | 42

Tatu Riihimäki

PROTOTYPE OF A VISUAL REMOTE MONITORING SYSTEM

The internet of things is growing rapidly presenting new needs and opportunities for remote monitoring. Existing remote monitoring solutions tend to be expensive and applicable to specific use cases only. Additionally, to enable remote monitoring, many existing solutions require replacing an existing and fully functional device with a new one which is not sustainable.

In this thesis, a prototype of a remote monitoring solution was developed. The aim was to develop an affordable, ecological, and a flexible solution that allows for monitoring of everything that can be visually monitored. Furthermore, the solution was to work as a platform for more tailored monitoring solutions, and to enable effortless integration to other systems.

The developed remote monitoring solution consists of an ESP32-CAM based monitoring device, a server with a REST API running in the cloud, and a web application user interface developed with the Angular framework.

The successful development of the prototype led to important and significant observations and findings to support the future development of the remote monitoring system. The REST API approach on the server-side proved to be an excellent architectural choice as it allows for easy extension and integration. Also, the camera module, ESP32-CAM, performed well and achieved the set objectives. Overall, the prototype and the thesis achieved its goals. The remote monitoring solution was developed and the process yielded valuable insight for the future development of the remote monitoring system.

KEYWORDS:

Remote monitoring, prototype, REST, API, ESP32, IoT

SISÄLTÖ

SANASTO JA KÄYTETYT LYHENTEET

API	Ohjelmointirajapinta (Application Programming Interface)
CORS	Mekanismi, jolla rajataan vieraiden sovellusten oikeuksia palvelimen resursseihin (Cross-Origin Resource Sharing)
CRUD-operaatio	Viittaa toimintoihin luo, lue, päivitä, poista (Create, Read, Update, Delete)
FTDI	Future Technology Devices International on skotlantilaisen yrityksen nimi, jolla yleisesti kuitenkin viitataan yrityksen valmistamiin adapterikomponentteihin, joilla USB-signaali voidaan muuttaa toiseen muotoon
GCP	Googlen pilvipalvelu (Google Cloud Platform)
HTML	Hypertekstin merkintäkieli ja tiedostomuoto (HyperText Markup Language)
HTTP	Hypertekstin tiedonsiirtoprotokolla (HyperText Transfer Protocol)
HTTPS	HTTP-protokolla suojatun yhteyden yli (HTTP Secure)
JSON	JavaScript-komentosarjakielen olion muotoinen tiedostomuoto (JavaScript Object Notation)
JWT	JSON-käyttöoikeustietue (JSON Web Token)
PaaS	Palvelumuoto, jossa palveluntarjoaja tarjoaa alustapalvelua (Platform as a Service)
REST API	REST-ohjelmistoarkkitehtuuriin pohjautuva ohjelmointirajapinta (Representational State Transfer Application Programming Interface)
SoC	Järjestelmäpiiri (System on a Chip)
SPA	Arkkitehtuurimalli, jossa palvelin jakaa vain yhden HTML-dokumentin, joka sisältää kaiken toimintalogiikan (Single-Page Application)
SSID	Lähiverkon tunnus (Service Set Identifier)
SQL	Kyselykieli tiedonhakuun relaatiotietokannoista (Structured Query Language)
URL	Kertoo resurssin osoitteen internetissä (Uniform Resource Locator)

1 JOHDANTO

1.1 Tausta

Tänä päivänä yhä useammat laitteet ovat etävalvottavissa esineiden internetin (engl. Internet of Things, IoT) kasvaessa kovaa vauhtia. Monet ratkaisut ovat kuitenkin kalliita, laite- ja valmistajakohtaisia, sekä ne on sidottu yhteen ainoaan käyttökohteeseen. Esimerkiksi vesimittarin etävalvonnan ratkaisut sopivat vain tähän tiettyyn tehtävään ja edellyttävät monissa tapauksissa koko mittarin vaihtoa, joka ei ehjän mittarin tapauksessa ole kestävän kehityksen mukaista.

Ajatus kamerapohjaisesta etävalvontaratkaisusta syntyi, kun kesämökin vesimittarin luku muistui mieleen toistuvasti vasta kotimatkalla. Toinen, merkittävämpi tekijä oli kesämökillä aikaisemmin sattunut vesivahinko, joka olisi ollut havaittavissa tässä työssä toteutetulla ratkaisulla. Vesivahinko olisi toki voitu havaita myös markkinoilta löytyvällä ratkaisulla. Markkinoilla olevat ratkaisut ovat kuitenkin kalliita ja ne on sidottu yhteen ainoaan käyttökohteeseen.

Kesämökkiympäristössä nousee jatkuvasti esiin kohteita, joita toivoisi voivan valvoa etänä ja edullisesti. Tässä työssä toteutetulla etävalvontaratkaisulla on tarkoitus valvoa ainakin vesimittaria, lisärakennuksen kattoa vuotojen varalta sekä hirsiseinien vääntymiä liitoskohtiin merkittyjen vaakaviivojen avulla.

Vaivattoman etävalvonnan tarve ilmeni myös Iltalehden uutisessa, jossa kerrottiin miehen saaneen suuren vesilaskun ilmoitettuaan väärän vesimittarilukeman vesilaitokselle. Kävi ilmi, että mies oli lukenut vesimittarilukeman ylösalaisin, koska mittarit ovat usein hämärissä ja hankalissa paikoissa lukemisen kannalta. (Kiviniemi 2019.)

1.2 Tavoite

Opinnäytetyön tavoitteeksi asetettiin prototyypin luonti kamerapohjaisesta etävalvontaratkaisusta, jolla useat käyttäjät voivat etävalvoa mitä tahansa ei-reaaliaikaisesti ja visuaalisesti valvottavaa kohdetta. Tavoitteena oli kehittää ekologinen,

edullinen ja joustava ratkaisu, jonka toiminnallisuutta voidaan jatkaa ja soveltaa moneen eri valvontakohteeseen. Vaikka tässä opinnäytetyöraportissa esimerkkikäyttökohteena käytetäänkin vesimittarin etävalvontaa, ei ratkaisu ole sidottu vain siihen.

Etävalvonnan mahdollistaman vaivattoman seurannan tavoite on johtaa aktiivisempaan kulutuksen seurantaan, joka puolestaan pyrkii johtamaan vastuullisempaan ja tietoisempaan kulutuskäyttäytymiseen. Etävalvonnan tavoite on myös mahdollistaa nopeampi reagointi mahdollisiin häiriötilanteisiin.

Joustavuuden tavoite on mahdollistaa etävalvontaratkaisun laajentaminen tarjoamalla alusta, jonka päälle voidaan luoda räätälöidympiä ratkaisuja, kuten erilaisia konenäkösovelluksia valvontakuvien jatkokäsittelyyn.

1.3 Olemassa olevat ratkaisut

Täysin tavoitteen kaltaista etävalvontaratkaisua ei opinnäytetyötä edeltäneessä selvityksessä löydetty. Kohdatut etävalvontaratkaisut olivat hintavia ja sidottuja yhteen käyttökohteseen. Olemassa olevista etävalvontaratkaisuista esitellään tässä kaksi, jotka ovat toiminnaltaan ja tavoitteiltaan lähimpänä työssä toteutettua.

LeakLook on konenäköön perustuva lisälaite, joka asennetaan olemassa olevan vesimittarin päälle, jolloin vältytään mittarin vaihdolta. Lisälaitteen avulla vesimittarilukema voidaan lukea etänä käyttölittymäsovelluksen kautta (Leaklook.io [n.d.].)

LeakLook on erinomainen ratkaisu vesimittarin etävalvontaan, mutta eroaa tässä opinnäytetyössä toteutetusta hinnan ja monikäyttöisyyden osalta. LeakLook lisälaite maksaa 199 € ja on soveltuva vain vesimittarien valvontaan (Leaklook.io [n.d.]).

internetkamerat ovat toinen etävalvonnan mahdollistava ratkaisu. Internetkamerat ovat hintavia, mutta käyttökohteiltaan erittäin joustavia. Kameralla voidaan valvoa lukemattomia erilaisia kohteita. Yksinkertaisimpien internetkameroiden hinnat ovat kuitenkin korkeita. Noin 60 €:sta ylöspäin (Suomen Turvatuote [n.d.]). Lisäksi internetkamerat ynnä muut valvontakamerat ovat usein liian kookkaita moniin käyttökohteisiin.

Tässä opinnäytetyössä toteutettu etävalvontaratkaisu on matalamman tason ratkaisu, jonka päälle edellä mainittujen kaltaiset räätälöidymmät ratkaisut voitaisiin sen joustavuuden ja jatkettavuuden ansiosta rakentaa.

1.4 Rajaukset

Tämän opinnäytetyöraportin tavoite on kuvata toteutettu etävalvontaratkaisun prototyyppi pääasiassa arkkitehtuuri- ja toimintatasolla. Etävalvontaratkaisun kannalta oleellista ei ole kehityksessä käytetyt työkalut tai ohjelmointikielet vaan se, mitä niillä on toteutettu ja millä tavalla. Tästä syystä kehityksessä käytetyt työkalut esitellään vain lyhyesti ja syvällinen läpikäynti on rajattu työn ulkopuolelle. Lisäksi matalan tason ohjelmakooditoteutukset jäävät esitetyn toimintatason ulkopuolelle ja osien toiminta kuvataan luettavammassa muodossa kuvaajin.

1.5 Raportin rakenne

Ensimmäiseksi raportti käsittelee etävalvontaratkaisun kannalta keskeisiä periaatteita ja teknologioita luvussa 2. Näiden keskeisten teknologioiden ja periaatteiden käyttöä sekä merkitystä etävalvontaratkaisun arkkitehtuurissa käsitellään luvussa 3. Lisäksi luvussa kuvataan koko ratkaisukokonaisuus ennen ratkaisun yksittäisiin osiin paneutumista myöhemmissä luvuissa. Luvusa 4 esitellään lyhyesti työssä käytetyt työkalut ennen etävalvontaratkaisun osien teknisen toteutuksen kuvausta luvussa 5. Teknisen toteutuksen pohjalta nousseita havaintoja ja jatkokehitysideoita käsitellään luvussa kuusi. Lopuksi tulokset tiivistetään lyhyessä yhteenvetoluvussa.

2 KESKEISET PERIAATTEET JA TEKNOLOGIAT

2.1 Keskeiset aihealueet

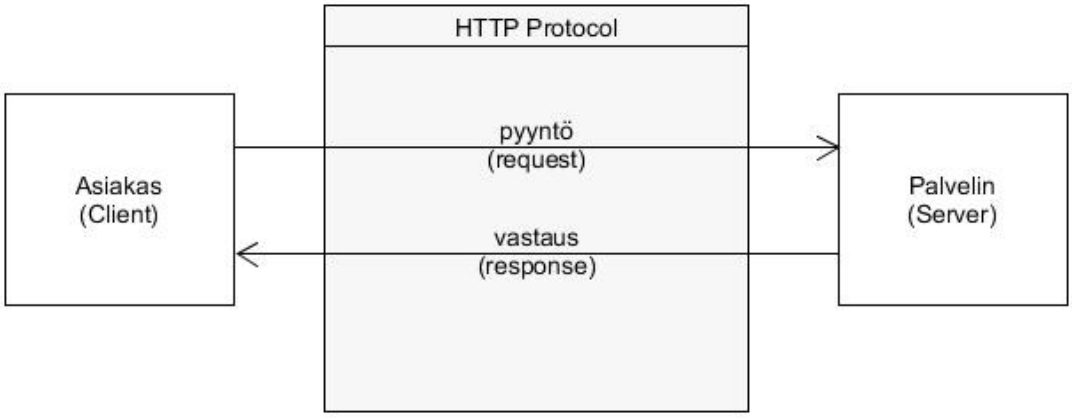
Työssä toteutetun etävalvontajärjestelmän tiedonsiirto rakentuu HTTP-protokollan ja REST API:n varaan, joten niiden perusteiden ymmärtäminen on oleellista kokonaiskuvan ja yksittäisten järjestelmän osien toiminnan ymmärtämiseksi. Lisäksi julkinen ohjelmointirajapintaratkaisu edellyttää CORS-mekanismin, autentikoinnin ja autorisoinnin periaatteiden, sekä JWT-käyttöoikeustietueiden toiminta-ajatuksen ymmärrystä.

2.2 HTTP-protokolla

HTTP-protokolla perustuu pyyntöihin (engl. request) ja vastauksiin (engl. response). Asiakas (engl. client) lähettää HTTP-pyynnön internetin yli palvelimelle (engl. server), eli HTTP-pyyntöjä käsittelevälle tietokoneohjelmistolle, joka käsittelee pyynnön ja lähettää asiakkaalle vastauksen. (Fielding & Reschke 2014a, 6–7.)

Asiakkaasta ja palvelimesta koostuvaa arkkitehtuuria kutsutaan asiakas-palvelin arkkitehtuuriksi tai malliksi (engl. client-server model) (Puder et al. 2006, 12–14). Tässä mallissa nimitykset *asiakas* ja *palvelin* kuvastavat rooleja, joita ohjelmat toimittavat tietyssä kommunikaatioyhteydessä. Todellisuudessa palvelinohjelmisto voi toimia myös asiakkaan roolissa lähettäessään pyynnön toiselle palvelimelle, eivätkä näiden roolien edustajat rajoitu vain verkkoselaimiin ja verkkopalvelimiin. Esimerkiksi verkkoyhteydellä varustettu valvontakamera voi toimia asiakkana tai palvelimena riippuen siitä, onko se pyynnön lähettäjä vai pyyntöön vastaaja (Fielding & Reschke 2014a, 6–8). Yksinkertainen esimerkki asiakas-palvelin muotoisesta kommunikaatiosta on www-selaimen ja palvelimen välinen kommunikaatio. WWW-selaimen hakukenttään voidaan kirjoittaa halutun WWW-sivuston osoite muodossa: ”http://www.osoite.fi”, jolloin selain, tässä tapauksessa asiakas, lähettää pyynnön palvelimelle, johon osoite ”http://www.osoite.fi” osoittaa. Pyynnön vastaanotettuaan palvelin käsittelee sen ja lähettää asianmukaisen vastauksen. Selaimen lähettämän pyynnön tapauksessa

vastauksen viestiosa on usein HTML-sivu, joka selaimessa esitetään vastauksen saavuttua. Asiakas-palvelin-malli on esitetty kuvassa 1.



Kuva 1. Asiakas-palvelin-malli.

HTTP-pyyntö koostuu metatietoa sisältävästä otsakeosiosta (engl. header), sekä varsinaisesta viestiosasta (engl. body). Otsakeosion alussa määritellään haluttu HTTP-metodi, sekä polku pyynnön kohteeseen. Määritelty HTTP-metodi yhdessä muun metatiedon kanssa kertoo palvelimelle pyynnön halutusta käsittelytavasta. (MDN Contributors [n.d.]a.)

GET-metodi on pääasiallinen tiedonhakumetodi, johon palvelimen oletetaan vastaavan halutulla resurssilla vastauksen viestiosassa (Fielding & Reschke 2014b, 23–24).

HEAD-metodi on on kuin GET-metodi, mutta palvelin ei vastaa kuin otsakeosiolla, eikä vastaus ei sisällä viestiosaa (Fielding & Reschke 2014b, 24).

POST-metodia käytetään, kun halutaan lähettää tietoa palvelimelle. Esimerkiksi HTML-sivulla täytetyn lomakkeen tiedot. POST-metodia käytettäessä lähetetyt tiedot lähetetään pyynnön viestiosassa. POST-metodia käytetään yleisesti myös ohjeistamaan palvelinta tallentamaan lähetetyt tiedot. (Fielding & Reschke 2014b, 25–26; MDN Contributors [n.d.]a.)

PUT-metodilla palvelinta pyydetään tallentamaan uusi resurssi tai päivittämään olemassaoleva pyynnön viestiosassa toimitetulla (Fielding & Reschke 2014b, 25–28).

DELETE-metodi ohjeistaa palvelimen poistamaan resurssin (Fielding & Reschke 2014b, 28–29).

OPTIONS-metodilla palvelimelta pyydetään tietoa tuetuista HTTP-metodeista (Fielding & Reschke 2014b 30–31). Palvelin ei välttämättä tue jokaista mahdollista HTTP-metodia, jolloin tuettujen metodien kysely voi olla tarpeen.

Palvelimen vastauksen ensimmäinen rivi sisältää aina HTTP-tilakoodin, joka kertoo pyynnön käsittelyn tilasta. Tilakoodit ovat muodoltaan kolminumeroisia ja ne jakautuvat viiteen kategoriaan:

- 1xx – informatiiviset tilakoodit.
- 2xx – kertovat onnistuneesta pyynnön käsittelystä.
- 3xx – liittyvät pyynnön uudelleenohjaukseen.
- 4xx – asiakkaan pyynnöstä johtuvat virheet.
- 5xx – palvelimen sisäiset virheet.

(Fielding & Reschke 2014b, 47–64.)

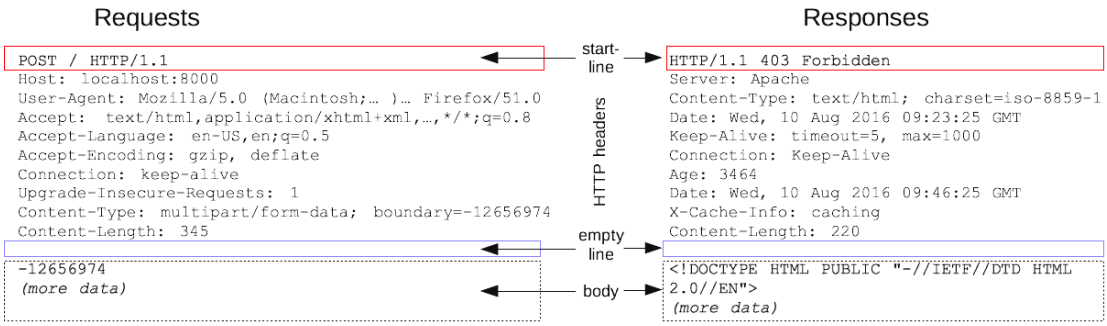
Tilakoodeja on lukematon määrä. Tämän opinnäytetyön kannalta keskeisimpiä tilakoodeja ovat seuraavat:

- **200 OK** – kertoo pyynnön onnistuneesta käsittelystä.
- **201 Created** – indikoi onnistunutta resurssin luontia pyynnön seurauksena. Usein POST tai PUT pyynnön vastauksen tilakoodi.
- **400 Bad Request** – palvelin ei pystynyt käsittelemään pyyntöä asiakkaan pyynnön sisältämän virheen vuoksi - esimerkiksi kirjoitusvirheestä johtuen.
- **401 Unauthorized** – pyynnön lähettäjää ei ole autentikoitu ja pyydetty resurssi vaatii sen.
- **403 Forbidden** – pyynnön lähettäjä on autentikoitu, mutta lähettäjällä ei ole oikeutta pyydettyyn resurssiin.
- **404 Not Found** – pyydettyä resurssia ei löydy palvelimelta.

- **408 Request Timeout** – pyynnön käsittely kesti liian kauan, joten palvelin keskeytti sen.
- **500 Internal Server Error** – virhe tapahtunut palvelimen päässä.

(Fielding & Reschke 2014b, 47–64.)

Esimerkki HTTP-kommunikaation pyynnöstä ja vastauksesta asiakkaan ja palvelimen välillä on esitetty kuvassa 2.



Kuva 2. Esimerkki HTTP-pyynnöstä ja vastauksesta (MDN Contributors [n.d.]a).

HTTPS

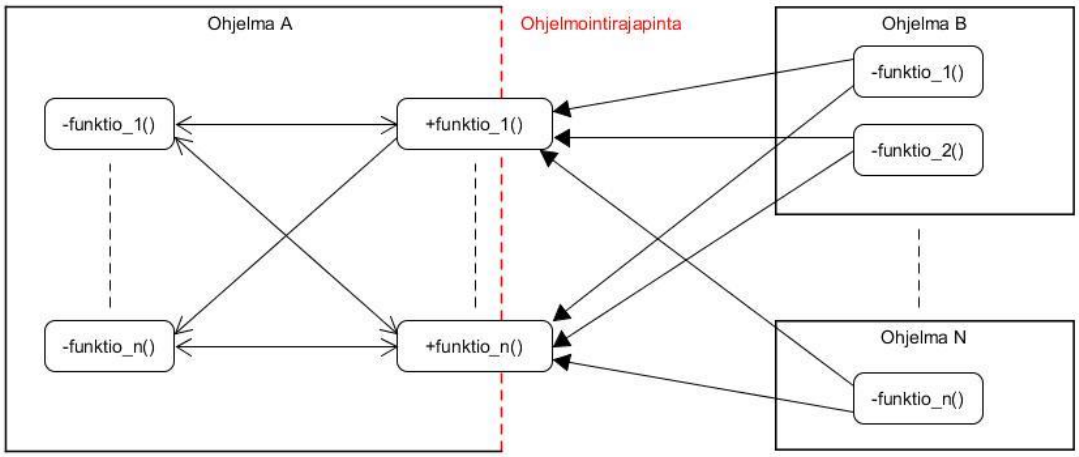
HTTP-protokolla on lähtökohtaisesti suojaamaton ja sen välityksellä lähetetyt pyynnöt ja vastaukset täysin luettavassa muodossa tiedonsiirtovaiheessa. Tämä tietoturvan puute osoittautui aikanaan ongelmaksi, kun arkaluontoista tietoa alettiin lähettämään internetin välityksellä. (Rescorla 2000, 1.)

HTTPS:ää käytettäessä tiedonsiirto salataan TLS-salausprotokollaa (Transport Layer Security) käyttäen, jolloin HTTP-viestit eivät ole enää ulkopuolisten luettavissa (Rescorla 2018, 5). Tätä syvällisempi HTTPS:n ja TLS:n ymmärrys ei tämän opinnäytetyön kannalta ole oleellista.

2.3 REST-ohjelmointirajapinta-arkkitehtuuri

Ennen REST-ohjelmointirajapintojen käsittelyä on syytä ymmärtää mitä ohjelmointirajapinnat yleisesti ottaen ovat. Ohjelmointirajapinnat, eli API:t mahdollistavat

ohjelmistoratkaisuiden välisen kommunikaation ja toiminnan (Conrad et al. 2016, 193). Esimerkiksi tietokoneohjelman ajo vaatii kommunikointia käyttöjärjestelmän kanssa sen ohjelmointirajapinnan kautta. Ohjelmointirajapinnan kautta voidaan pyytää esimerkiksi käyttöjärjestelmältä vapaan muistialueen osoitetta ajetun ohjelman käyttöön. Ohjelmointirajapinnoissa on siis kyse toisen ohjelman toiminnallisuuden käytöstä tavoitellun lopputuloksen saavuttamiseksi toisessa ohjelmassa. API ei kuitenkaan viittaa yksinomaan ajossa olevan ohjelmiston julkisiin funktioihin, vaan sillä voidaan tarkoittaa myös valmiin koodikirjaston funktioita, joita kirjaston sisällyttämässä lähdekoodissa voidaan käyttää. Ohjelmointirajapinnan periaate on kuvattu kuvassa 3.



Kuva 3. Ohjelmointirajapinnan toimintaperiaate.

REST on Roy Fieldingin kehittämä ja väitöskirjassaan esittelemä, HTTP-protokollan innoittama ohjelmistoarkkitehtuurityyli hajautettujen ohjelmistojen väliseen kommunikointiin, joka tuo hyvänlaatuisia rajoitteita rajapintojen toteutukseen, jotka oikein ja täysin toteutettuina tuovat monenlaisia hyötyjä niin asiakkalle kuin palvelimelle (Fielding 2000, 76). Saavutettuja hyötyjä käsitellään edempänä rajoitekohtaisesti. Ennen REST:n rajoitteiden käsittelyä, täytyy ymmärtää mitä ovat resurssit, resurssin esitykset ja resurssimetodit sekä miten URL-osoitteet liittyvät niihin.

Resurssilla tarkoitetaan tiedon abstraktiota, ja se voi olla esimerkiksi kuva, HTML-dokumentti tai kuvaus henkilöstä JSON-muodossa (Fielding 2000, 88–90). REST-arkkitehtuurimallia noudattavan API:n URL-osoitteet ovat resurssitunnisteita (engl.

resource identifier), jotka osoittavat aina tiettyyn resurssiin (Fielding 2000, 88–90). Esimerkiksi URL ”http://osoite/kuvat” on resurssitunniste, joka osoittaa kuvat-resurssiin palvelimella.

REST API antaa asiakkaan pyyntöön vastauksena esityksen (engl. representation) resurssin tilasta (engl. state) pyynnön hetkellä. Tästä tulee arkkitehtuurin nimi Representational State Transfer. (Fielding 2000, 90–92.)

URL-osoitteiden kautta voidaan suorittaa CRUD-operaatioita REST-rajapinnan resursseihin. Arkkitehtuuri ei tarkasti määrää miten HTTP-metodeita tulee käyttää, kunhan niitä käytetään yhdenmukaisesti ja loogisesti. Yleisimmin REST-ohjelmointirajapinnoissa kohtaa HTTP-metodeita käytettävän seuraavasti: GET-metodilla haetaan resurssi tai sen esitys, POST-metodilla luodaan uusi, PUT-metodilla päivitetään olemassaoleva, ja DELETE-metodilla poistetaan.

REST-arkkitehtuurin täydellinen toteuttaminen ja näin ollen kaikkien hyötyjen saaminen edellyttää seuraavien vaatimusten noudattamista järjestelmän toteutuksen osalta: asiakas-palvelin-mallin mukainen kommunikaatiomalli, tilattomuus, välimuistitettavuus, rajapinnan yhdenmukaisuus, ja kerroksisuus (Fielding 2000, 78–84).

Asiakas-palvelin-malli tarkoittaa, että kommunikaatiossa on kaksi toisistaan riippumatonta osapuolta, mikä puolestaan mahdollistaa näiden kehittymisen erikseen (Fielding 2000, 78).

Vaatimus tilattomuudesta (engl. stateless) tarkoittaa, että kaikki asiakkaan palvelimelle lähettämät pyynnöt ovat itsenäisiä, eivätkä ne ole sidoksissa muihin pyynöihin. Tämän johdosta kaikki pyynnön käsittelyyn vaadittu tieto välitetään pyynnön mukana palvelimelle. Tilattomuudella saavutetaan hyötyjä palvelimen päässä. Palvelintoteutus voi olla yksinkertaisempi ja kevyempi, kun sen ei tarvitse tallentaa asiakkaan tilaa, jolloin saavutetaan paremmin skaalautuva palvelin. Saavutetun hyödyn hintana on kuitenkin suuremmat pyynnöt, joissa joudutaan toistuvasti lähettämään samaa tietoa. (Fielding 2000, 78–79.)

Välimuistitettavuudella (engl. cache) tarkoitetaan, että palvelimen tulee vastauksessaan eksplisiittisesti määritellä, voiko asiakas tallentaa vastauksen välimuistiin. Välimuistin

käytöllä voidaan vähentää pyyntöjen määrää palvelimelle. Asiakkaan pyytäessä resurssia, joka löytyy jo välimuistista, luetaan se sieltä uuden pyynnön lähettämisen sijaan. Välimuistin käytöllä saavutetaan parempi suorituskyky ja nopeampi pääsy resurssiin toimintavarmuuden hinnalla. Välimuistiin tallennettu vastaus saattaa erota resurssin todellisesta tilasta, joka saataisiin pyynnön lähettämisellä palvelimelle. (Fielding 2000, 79–81.)

Vaatimus yhdenmukaisesta rajapinnasta tarkoittaa, että rajapinnan tulee olla kaikin puolin yhtenäinen ja looginen (Fielding 2000. 81–82). Rajapintaa käyttävät asiakkaat käyttävät sitä samalla tavalla ja saavat yhdenmukaiset vastaukset (Fielding 2000. 81–82). REST-arkkitehtuuri ei tarkasti määrittele yhdenmukaisen rajapinnan toteutusvaatimuksia matalalla tasolla. Yhdenmukaisen rajapinnan tulee kuitenkin olla nimensä mukaisesti yhtenäinen tarkoittaen esimerkiksi sitä, että kaikkia palvelimen resursseja käsitellään yhtäläisesti samoilla HTTP-metodeilla. HTTP-metodit PUT ja POST molemmat lähettävät tietoa palvelimelle ja niitä voidaan molempia käyttää samaan tarkoitukseen, mutta REST API:n ei tulisi käyttää niitä epäjohdonmukaisesti, eli päivittää jotain resurssia PUT-metodilla ja toista POST-metodilla. Yhtenäisyyden piiriin lukeutuvat myös URL-osoitteiden ja vastausten muodot. URL-osoitteiden tulee noudattaa yhtenäistä rakennetta kaikissa rajapinnan resursseissa ja rajapinnan antaa vastaukset samassa muodossa mahdollisuuksien mukaan. Rajapinta ei saa tarpeettomasti vastata yhdestä URL-osoitteesta XML:llä ja toisesta JSON:lla.

Kerroksisuudella tarkoitetaan, että asiakas-palvelin-mallin osapuolet eivät tiedä kuin itsestään ja kommunikaatioketjun seuraavavasta linkistä (Fielding 2000, 81–84). Asiakkaan lähettäessä pyynnön palvelimelle se ei tiedä, eikä sen kuulukaan tietää, vaatiiko pyynnön käsittely palvelimelta sen lähettämistä ensin eteenpäin toiselle palvelimelle.

Nämä viisi vaatimusta ja rajoitetta täyttävää rajapintaa voidaan kutsua REST-ohjelmointirajapinnaksi (Fielding 2008).

2.4 CORS-periaate

Tietoturvasyistä asiakassovellukset, kuten verkkoselaimet, asettavat rajoitteita pyyntöihin, jotka kohdistuvat eri osoitteeseen, kuin missä asiakassovellus itse on ajossa (van Kesteren 2014). Käytännössä tämä tarkoittaa sitä, että osoitteesta *http://www.asiakas.fi* ei voida pyytää resurssia osoitteessa *http://www.palvelin.fi*. Ainoastaan pyynnöt osoitteeseen *http://asiakas.fi* ovat lähtökohtaisesti sallittuja. Usein julkisissa ohjelmointirajapinnoissa palvelin on konfiguroitu niin, että siltä voidaan pyytää resursseja mistä tahansa asiakassovelluksesta.

Pyynnön mahdollistaminen on käytännössä hyvin yksinkertaista. Palvelimen tulee vain sisällyttää vastaukseensa otsake *Access-Control-Allow-Origin*, jonka arvo on pyytävän asiakkaan alkuperä (engl. origin). Esimerkiksi: ”Access-Control-Allow-Origin: http://www.asiakas.fi” mahdollistaa resurssien pyynnöt mainitusta asiakkaan alkuperästä. Mikäli otsakkeelle annetaan arvo ”*”, ovat kaikki alkuperät sallittuja. Tämä on yleinen arvo julkisissa ohjelmointirajapinnoissa, joihin tulee voida lähettää pyyntöjä mistä tahansa.

Tämän opinnäytetyön kannalta riittää, että ymmärtää CORS:in olevan keino, jolla eri osoitteissa ajettavilta asiakasohjelmilta voidaan joko evätä, tai sallia pyynnönlähetys palvelimelle.

2.5 JWT-autorisointi

Ohjelmien käsitellessä käyttäjäkohtaista tai muuten arkaluontoista materiaalia, on pääsy resursseihin suojattava ulkopuolisilta ja varmennuttava pyytäjän oikeudesta suorittaa pyydetty toiminto resurssille (Auth0 [n.d.]). Oikeuksien tarkastelussa puhutaan autentikoinnista ja autorisoinnista tai todennuksesta ja valtuutuksesta (Auth0 [n.d.]). Todennuksella tarkoitetaan varmistamista, että pyynnön lähettäjä todella on kuka tämä väittää olevansa. Yleensä tämä tapahtuu käyttäjätunnuksen ja salasanan avulla. Valtuutuksella puolestaan tarkoitetaan oikeuksia, joita todennetulla käyttäjällä on (Auth0 [n.d.]). Esimerkiksi autentikoidulla, eli todennetulla, käyttäjällä voi olla valtuudet lukea

ja muokata resurssia, mutta ei poistaa, kun taas toisella autentikoidulla käyttäjällä voi olla kaikki edellämainitut valtuudet. Tämä on autentikoinnin ja autorisoinnin ero.

JWT (JSON Web Token) on palvelimen luoma ja allekirjoittama käyttöoikeustietue (engl. access token), joka keveytensä ansiosta soveltuu erinomaisesti käytettäväksi HTTP-protokollan kanssa (Jones et al. 2015, 4). JWT on autorisointiväline, joka voidaan antaa autentikoidulle käyttäjälle osoittamaan, että tällä on tiettyjä oikeuksia.

Käytännössä JWT on merkkijono, joka koostuu otsakkeesta (engl. header), tietosisällöstä (engl. payload), ja allekirjoituksesta (engl. signature). Merkkijono on muotoa: *otsake.tietosisältö.allekirjoitus*. (Jones et al. 2015, 6–11; JWT.io [n.d.].)

JWT:n otsakeosa kertoo käyttöoikeustietueen ominaisuuksista ja se pitää yleensä sisällään tiedon käyttöoikeustietueen tyypistä, sekä allekirjoituksessa käytetystä algoritmista (JWT.io [n.d.]).

Tietosisältöosa puolestaan voi sisältää mitä vain sovelluksen vaatimaa tietoa. Yleensä tietosisältöosaan on sisällytetty tietoa käyttöoikeustietueen omistajasta, kuten esimerkiksi tämän sähköpostiosoite. Tietosisältöosa voi sisältää myös tietoa käyttöoikeustietueen luomisajankohdasta, tai sen vanhenemisajankohdasta, jonka jälkeen käyttöikeustietue menettää merkityksensä. Tärkeää on huomata, että tietosisältö ei ole salattua ja on täysin luettavissa käyttöoikeustietueesta. (JWT.io [n.d.].)

JWT:n viimeinen osa, eli allekirjoitusosa, on muodostettu enkryptaamalla otsakeosassa määritetyllä kryptografisella algoritmilla otsakeosasta, tietosisältöosasta ja salaisesta avaimesta koostuva merkkijono yhdeksi pitkäksi enkryptatuksi merkkijonoksi (JWT.io [n.d.]). Salainen avain on vain JWT:n luojan, esimerkiksi autentikointipalvelimen, tiedossa.

$$Luo(otsake, tietosis., avain) = enkryptaa(otsake + tietosis. + avain)$$

Toiminta-ajatukseltaan JWT käyttöoikeustietue perustuu siis siihen, että sen tietosisältöä ei voida muuttaa. Mikäli tietosisältö muuttuu allekirjoituksen jälkeen, ei otsakkeen ja muokatun tietosisällön enkryptaus johda enää alkuperäiseen allekirjoitukseen.

$$Varmenna(jwt, avain) = enkryptaa(jwt.otsake + jwt.tietosis. + avain) == jwt$$

Tarkoitus on, että käyttäjän autentikoituaan palvelin luo JWT-käyttöoikeustietueen, joka on voimassa vain tietyn ajan, ja jonka vain palvelin pystyy varmentamaan salaista avainta käyttäen. Käyttäjä liittää palvelimelta saamansa JWT:n kaikkiin autorisointia vaativiin pyyntöihin.

JWT on erinomainen autorisointimekanismi REST-ohjelmointirajapinnoille, jotka edellyttävät tilattomuutta. JWT on kevyt lähettää pyyntöjen mukana rajapinnalle ja siihen voidaan sisällyttää pyynnön käsittelyn edellyttämää käyttäjäkohtaista tietoa.

3 SUUNNITTELUPERIAATTEET JA YLEISARKKITEHTUURI

Etävalvontaratkaisun prototyyppi suunniteltiin tavoitteiden mukaisesti priorisoiden skaalautuvuus, joustavuus ja jatkettavuus. Skaalautuvuuden näkökulmasta ratkaisun tavoitteena oli tukea useita eri valvontalaitteita ja niiden tuottamasta valvontatiedosta kiinnostuneita käyttäjiä samanaikaisesti. Joustavuuden näkökulmasta puolestaan ratkaisun tavoitteena on mahdollistaa sen komponenttien sijoittaminen joustavasti eri alustoille ja mahdollistaa niiden itsenäinen kehitys, siirto ja mahdollisesti jopa vaihtaminen. Esimerkiksi käyttöliittymä voidaan vaihtaa toiseen vaikuttamatta ratkaisun muihin osiin. Joustavuuden ja jatkettavuuden mahdollistamiseksi myös ratkaisun palvelimelle toteutettiin julkinen REST-ohjelmointirajapinta, joka mahdollistaa integroinnin toisiin ratkaisuihin, kuten esimerkiksi etävalvontajärjestelmän tuottamien kuvien jatkokäsittelyyn konenäön avulla toisessa sovelluksessa. API-arkkitehtuuri mahdollistaa myös vaivattoman käyttöliittymien lisäyksen tai vaihdon. Prototyypissä toteutettiin vain selainsovellus, mutta jatkossa on mahdollista kehittää esimerkiksi mobiilisovellus, joka käyttää samaa API:a.

Ratkaisun arkkitehtuurista voidaan vaihtaa mikä tahansa osa muuttamatta muiden osien sisäistä toimintaa ollenkaan tai korkeintaan vain hyvin vähän. Kameralaite voidaan vaihtaa toiseen ja sen lisäksi ratkaisuun voidaan liittää muitakin sensoreita, mikä edellyttää vain olemassa olevan rajapinnan laajennusta uudentyyppisten pyyntöjen käsittelyn mahdollistamiseksi. Tarvittaessa myös prototyyppivaiheessa valittu GCP:n Cloud Storage -tiedostovarastoratkaisu voidaan vaihtaa mihin tahansa vastaavaan pilvipohjaiseen tiedostovarastoon.

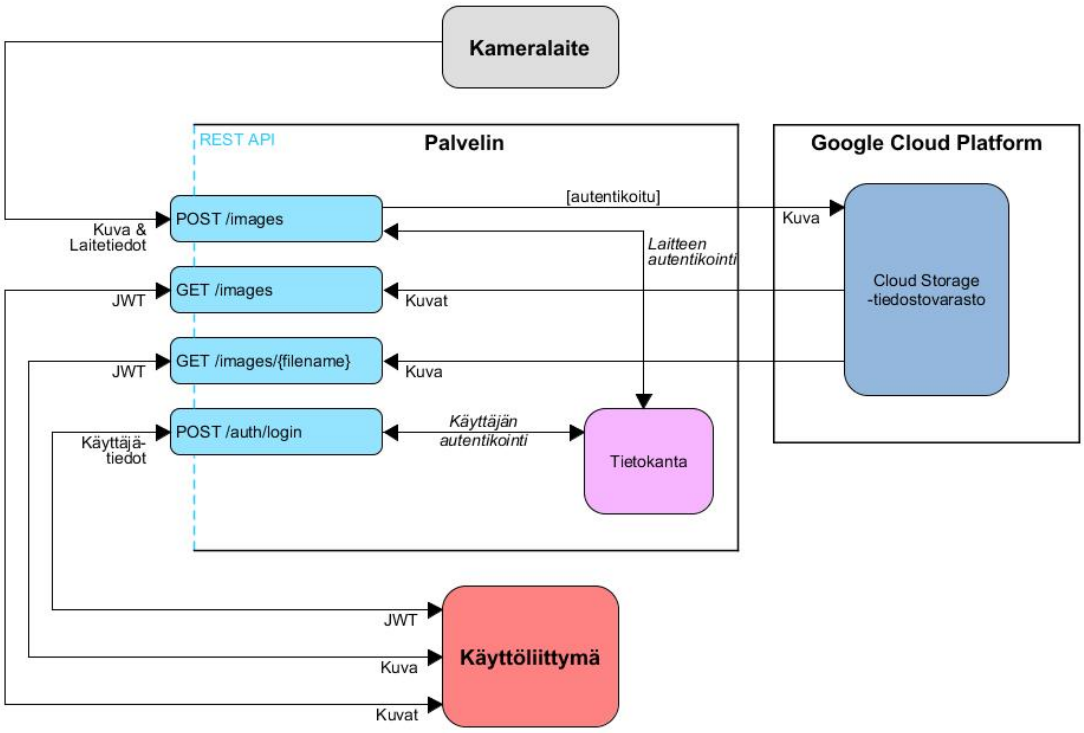
Etävalvontaratkaisun REST-ohjelmointirajapinnan ollessa julkinen, on valtaosa sen resursseista asetettu vaatimaan autentikoitua pyynnön lähettäjää. Autentikointivaatimus koskee sekä kameralaitteita että käyttäjiä. Julkinen ohjelmointirajapinta vaati myös sen CORS-käytäntöjen muokkaamista siten, että resursseja voi pyytää mikä tahansa asiakas. Autentikointiin vaaditut käyttäjätiedot ja ohjelmointirajapintaan lähetetyt valvontakuvat ovat arkaluontoista tietoa. Tästä syystä kaikki etävalvontaratkaisun tiedonsiirto käyttää HTTPS-protokollaa. Työhön valitut PaaS-muotoiset alustat tarjoavat oletusarvoisesti

suojatut HTTPS-yhteydet, joten HTTPS:n manuaaliselta käyttöönotolta vältyttiin. Tämä oli merkittävä tekijä käytettyjen PaaS-palveluntarjoajien valinnassa. Etävalvonta-ratkaisun tiedostovaraston tarjoaa Googlen pilvipalvelu ja REST-ohjelmointirajapinnan palvelimen alustan Heroku-palveluntarjoaja. Valittuja työkaluja esitellään tulevissa luvuissa enemmän.

Toiminnallisesti etävalvontaratkaisu on suunniteltu käytettäväksi ei-reaaliaikaiseen visuaaliseen valvontaan. Tämä tarkoittaa sitä, että kameralaite ei välitä videokuvaa, vaan valokuvia määrätyin intervallein. Intervalli voidaan määrittää valvontakohteen mukaan. Tässä opinnäytetyössä esimerkkivalvontakohteena käytetään kesämökin vesimittaria. Tämänkaltainen valvontakohde ei vaadi kovin reaaliaikaista valvontaa. Kuva kerran vuorokaudessa on riittävä kuvanottotiheys, jotta voidaan havaita mahdolliset vuodot tai vain suorittaa edellisellä mökkireissulla unohtunut mittarinluku.

Suunniteltu ratkaisu toimii niin, että kameralaite asetetaan valvomaan valvottavaa kohdetta, jonka jälkeen se lähettää omalla tunnisteellaan varustettuja valvontakuvia palvelimen REST-ohjelmointirajapintaan, joka varmentaa laitteen tunnisteen ja tallentaa kuvan aikaleiman kanssa tiedostovarastoon. Käyttäjä, eli laitteen omistaja on yhdistetty kameralaitteeseen relaatiotietokannassa. Käyttäjän kirjautuessa sisään REST-ohjelmointirajapinnan kautta, annetaan hänelle JWT-käyttöoikeustietue. JWT:n pyyntöihin liittämällä käyttäjä pääsee käsiksi omien kameralaitteidensa kuviin ohjelmointirajapinnan kautta.

Prototyyppivaiheessa toteutettiin vain välttämättömimmät toiminnot etävalvonta-ratkaisun toiminnan havainnollistamiseski ja testaamiseksi. Tästä syystä ohjelmointirajapinta ei esimerkiksi tue uuden käyttäjän rekisteröitymistä. Prototyyppivaiheessa toteutettu ratkaisukokonaisuus on esittetty kuvassa 4.



Kuva 4. Etävalvontaratkaisun osat ja tiedonkulku.

4 KÄYTETYT TYÖKALUT JA RATKAISUT

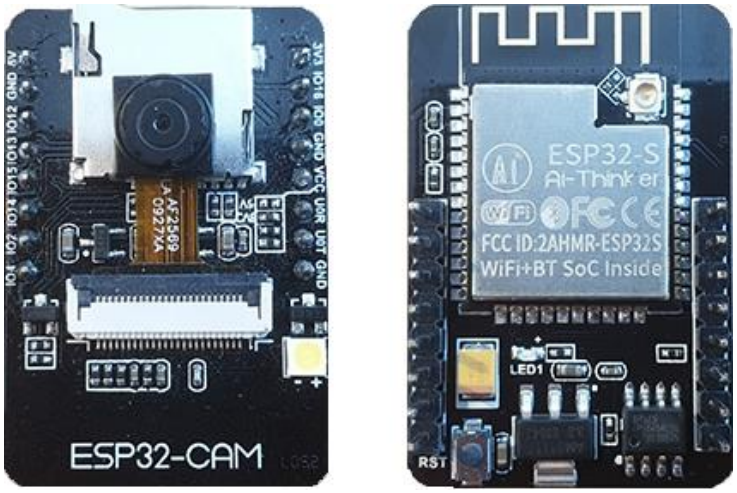
ESP32-järjestelmäpiiri

Käyttövaatimusten perusteella valvontalaitteen ensisijaisiksi teknisiksi vaatimuksiksi muodostuivat WiFi-tuki, kamerakomponentti, sekä pieni virrankulutus.

Vaatimuksiin sopivaksi vaihtoehdoksi valikoitui ESP32-CAM järjestelmäpiiri, joka on edullinen, ja ominaisuuksiltaan jopa kattavampi kuin etävalvontaratkaisun kannalta olisi tarpeen. ESP32 on kiinalaisen Espressif Systemsin kehittämä järjestelmäpiiri, josta löytyy etävalvontaratkaisuun sopiva CAM-versio. Työssä käytetty ESP32-CAM (kuva 5) on kiinalaisen valmistajan, AI-Thinkerin, versio.

ESP32-CAM:ssa on 32-bittinen prosessori, 512 kt sisäistä SRAM-muistia ja 4 Mt ulkoista PSRAM-muistia, OV2640-kameramoduuli, useita unitiloja, WiFi-radio, eri verkkotoimintatiloja, tuki laiteohjelmiston etäpäivitykselle, sekä muistikorttipaikka (AI-Thinker [n.d.]).

Merkittävä tekijä järjestelmäpiirin valinnassa oli myös hyvän dokumentaation olemassaolo. Espressif Systemsin mikro-ohjaimet ja järjestelmäpiirit ovat vasta lähivuosina kasvattaneet suosiotaan, joten tarjolla on pitkään ollut vain kiinan kielisiä manuaaleja tai niistä käännösohjelmalla tehtyjä huonoja käännöksiä. Onneksi valitulle järjestelmäpiirille löytyi jo erittäin hyvät englannin kieliset lähteet.



Kuva 5. ESP32-CAM.

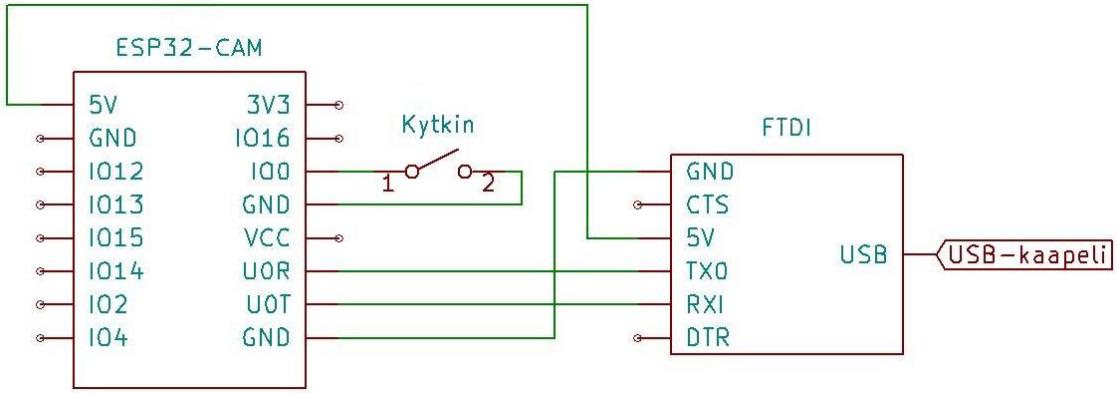
ESP-IDF-sovelluskehys

ESP32-CAM:in kehitykseen käytettiin Espressif Systemsin omaa ESP-IDF (Espressif IoT Development Framework) -sovelluskehystä, joka koostuu ohjelmakoodikirjastosta, sen ohjelmointirajapinnasta ja valmiista komentosarjoista (Espressif Docs [n.d.]).

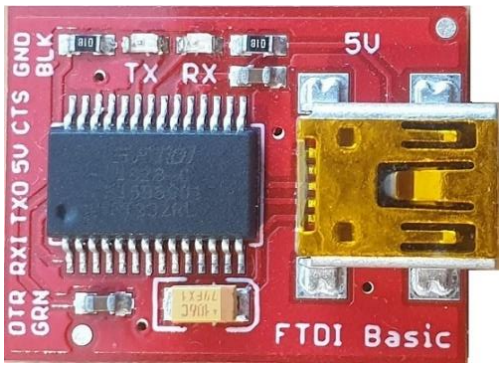
ESP-IDF:n asennukseen löytyi kattava ja ajantasainen ohjeistus Espressif Systemsin verkkosivuilta, eikä ohjeen toistoa nähdä tarpeelliseksi tämän työn kannalta. Myös ESP-IDF:n API:lle löytyi erittäin hyvä dokumentaatio Espressif Systemsin verkkosivuilta, jonka avulla entuudestaan vieraan teknologian kanssa pääsi nopeasti tuottavalle tasolle.

ESP-IDF:llä laiteohjelmiston kehitysprosessi lähdekoodista laiteohjelmistoksi koostuu kehittäjän näkökulmasta kolmesta vaiheesta. Ensin kirjoitetaan C-lähdekoodi, jonka jälkeen se käännetään ristiinkääntäjällä ESP32-CAM:n prosessoriarkkitehtuurille sopivaksi. Kääntäminen tapahtuu ESP-IDF:llä vaivattomasti ajamalla Python-komentosarja *idf.py* argumentilla ”build”. Käännetyn laiteohjelmiston lataaminen ESP32-CAM:lle tapahtuu saman Python-komentosarjan avulla, mutta ennen sen ajoa,täytyy ESP32-CAM olla yhdistettynä kehityskoneeseen USB-portin kautta, mutta koska ESP32-CAM:ssa ei ole USB-porttia, tulee välissä käyttää FTDI:tä, joka muuttaa USB-signaalin ESP32-CAM:lle sopivaksi. Tämän jälkeen voidaan komentosarja ajaa. Tällä kertaa argumentilla ”flash”, joka kirjoittaa laiteohjelmiston ESP32-CAM:lle. Laiteohjelmiston lataus ESP32-CAM:lle vaatii laitteen olevan kehitystilassa, joka

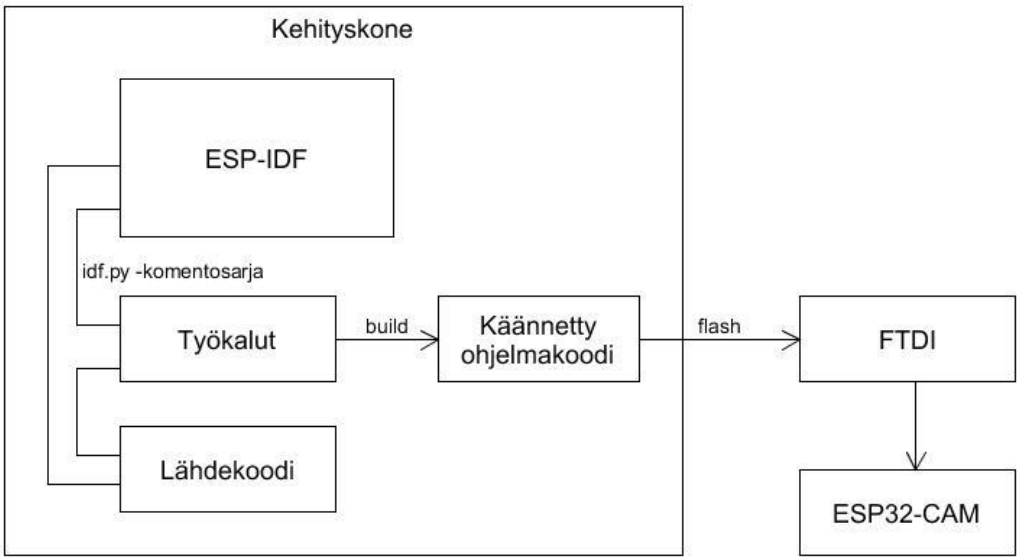
tapahtuu käynnistämällä laite IO0 kytkettynä viereiseen GND-napaan. Kameralaitteen normaalia ajoa varten tulee se käynnistää ilman tätä napayhteyttä. Kehityksen mahdollistava ESP32-CAM:n ja FTDI:n välinen kytkentä on esitetty kuvassa 6 ja käytetty FTDI kuvassa 7. Kuva 8 esittää kaikki kehitystyön osat sekä kehityksen kulun.



Kuva 6. FTDI:n ja ESP-CAM:in välinen kytkentä.



Kuva 7. Käytetty FTDI.



Kuva 8. ESP-IDF Kehityksen kulku.

Node.js-sovelluskehys

Node.js on JavaScript-tulkki, joka on rakennettu Google Chrome -selaimen JavaScript-tulkiksi kehitetyn V8:n päälle (Node.js [n.d.]). Käytännössä Node.js tuo aikaisemmin vain selainympäristöön tarkoitetun JavaScript-komentosarjakielen suoritus-mahdollisuuden selainympäristön ulkopuolelle. JavaScript on pitkään ollut suosittu työkalu verkkosivujen kehityksessä ja näin ollen hyvin alan kehittäjien hallussa. Sen käytön mahdollistaminen myös palvelinohjelmointiin yhdenmukaistaa selainsovellus-kehitysprosesseja, kun samaa teknologiaa voidaan käyttää sekä asiakkaan että palvelimen päässä.

Tässä opinnäytetyössä toteutetun REST-ohjelmointirajapinnan palvelin on toteutettu käyttäen Node.js-sovelluskehystä ja kirjastoja express, bcryptjs, ja jsonwebtoken.

PostgreSQL-tietokanta

PostgreSQL on yli 30 vuotta kehitetty SQL:ään pohjautuva ja sitä laajentava relaatio-tietokannan hallintajärjestelmä, joka on yksi suosituimmista relaatiotietokanta-järjestelmistä tänä päivänä (PostgreSQL [n.d.]a). PostgreSQL:n suosion ja pitkän

historian ansiosta sille on hyvä tuki monella eri palveluntarjoajalla, joka mahdollistaa tietokannan ylläpidon muualla kuin paikallisella palvelimella. Esimerkiksi merkittävimmät pilvialustat Google Cloud Platform, Amazon Web Services, Azure sekä Heroku tarjoavat PostgreSQL-tietokantaratkaisuja (PostgreSQL [n.d.]b). Tässä opinnäytetyössä luotu prototyyppi käyttää PostgreSQL-tietokantaa, joka on Heroku-pilvialustalla.

Cloud Storage -tiedostovarasto

Google Cloud Platformin PaaS-muotoinen tiedostovarastoratkaisu tarjoaa hyödyllisiä ominaisuuksia työssä toteutetun etävalvontaratkaisun näkökulmasta. Tiedostovarastoon tallennetut tiedostot ovat automaattisesti enkryptattuja ja näin ollen suojattuja. Lisäksi tiedostovarastoratkaisu tarjoaa mahdollisuuden asettaa automatisoituja toimintoja, kuten tiedostojen varmuuskopiointia toiseen palvelinkeskukseen toimintavarmuuden parantamiseksi. (Google Cloud [n.d.].)

Cloud Storage -tiedostovarastoa käytetään valvontakuvien säilytykseen etävalvonta-ratkaisussa.

Angular-sovelluskehys

Angular on Googlen kehittämä sovelluskehys SPA-arkkitehtuurin mukaisten selainsovellusten kehitykseen (Angular.io [n.d.]). Angularin kehityksessä käytetty ohjelmointi- tai komentosarjakieli on ensisijaisesti TypeScript, joka laajentaa perinteistä JavaScript-komentosarjakieltä lisäämällä kieleen vahvan tyypityksen ohjelmointi-virheiden vähentämiseksi. TypeScriptin käyttö ei ole pakollista, ja Angular-sovelluskehys kääntää kirjoitetun TypeScriptin selaimen ymmärtämäksi JavaScriptiksi ohjelman käännösvaiheessa joka tapauksessa.

Modernin SPA-arkkitehtuurimallin mukaisesti Angular tuo sovelluskehitykseen modularisuutta monikäyttöisten moduulien ja komponenttien muodossa (Angular.io [n.d.]).

Heroku-alustapalvelu

Heroku on PaaS-muotoista palvelua tarjoava pilvipalvelu, jossa palvelinsovelluksia voidaan ylläpitää vaivattomasti ja tiettyyn käyttörajaan asti ilmaiseksi (Heroku [n.d.]). Heroku tukee Node.js pohjaisia palvelimia, sekä tarjoaa PostgreSQL-tietokannan, mistä syystä se on erinomainen valinta etävalvontaratkaisun alustaksi ainakin prototyyppi-vaiheessa. Lisäksi alustalla ajetut palvelimet ovat oletusarvoisesti HTTPS:ää käyttäviä.

5 TEKNINEN TOTEUTUS

5.1 Kameralaite

Etävalvontaratkaisun keskeisin osa on itse kameralaite, joka välittää kuvia valvotusta kohteesta ennaltamäärätyin intervallein. Päätös intervallein otettavista kuvista perustuu vaatimuksiin, joita laitteelle asetettiin suunnitteluvaiheessa. Esimerkkikäyttökohteen, kesämökin vesimittarin, etävalvonnan vaatimuksia määriteltäessä todettiin, että etävalvonta voi olla ei-reaaliaikaista, sen tulee olla edullista ja mahdollisimman ekologista, jotta etävalvontaratkaisu ja erillisen valvontalaitteen käyttöönotto on kannattavaa. Toimivaa vanhaa analogista vesimittaria ei pelkän etälukumahdollisuuden vuoksi haluttu vaihtaa uuteen ja kalliiseen ratkaisuun.

Etävalvontajärjestelmän kameralaite on toteutettu niin, että käynnistyessään se käynnistyy joko internetyhteydettömänä tukiasemana (engl. access point) tai WiFi-asemana (engl. WiFi station) eli asiakkaana, joka yhdistyy internetiin ja pystyy kommunikoimaan palvelimien kanssa. Valintaan tukiasemaksi tai WiFi-asemaksi käynnistymisestä vaikuttaa se, onko halutun tukiaseman SSID ja salasana tallennettu kameralaitteeseen.

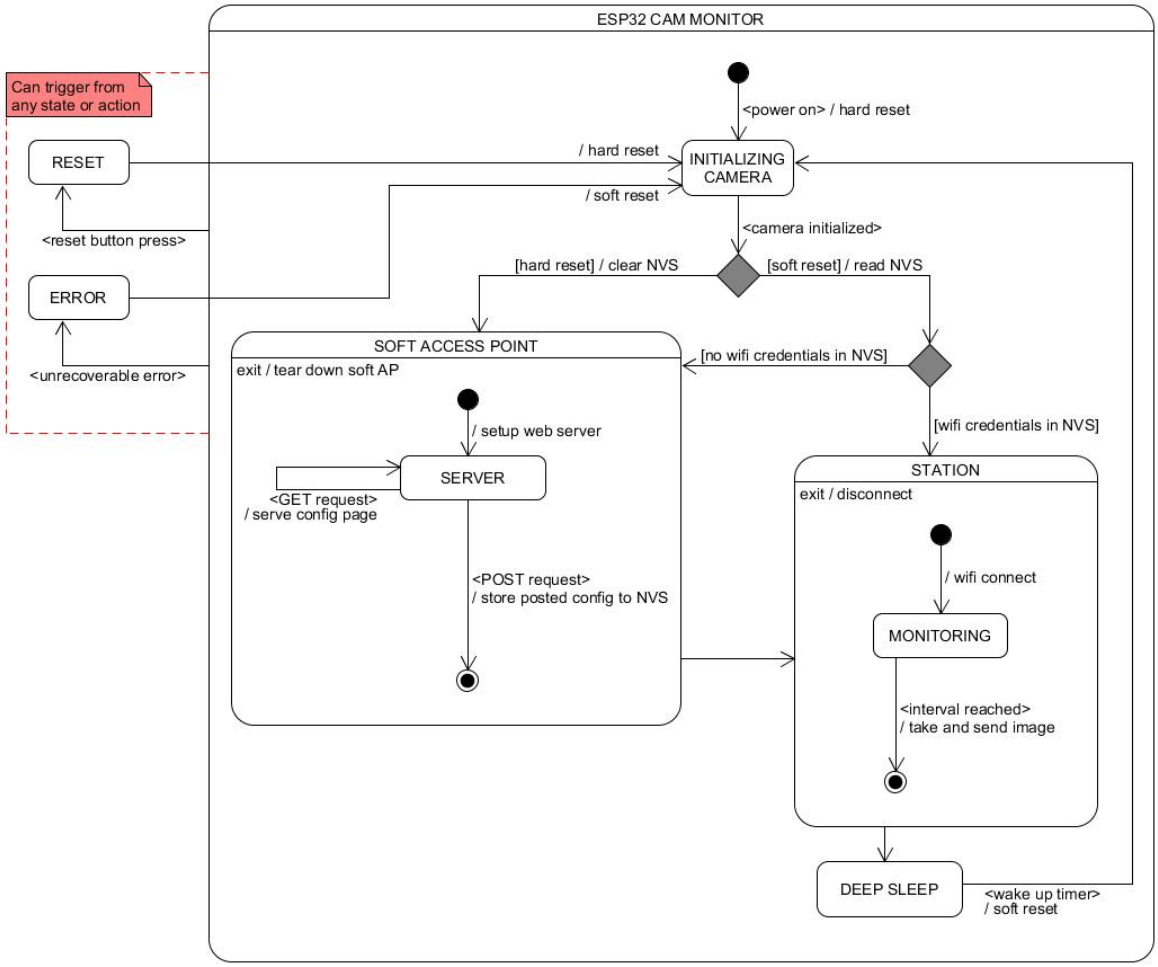
Kameralaitteen käynnistyessä tukiasemaksi käynnistää se omaan verkkoonsa palvelinprosessin, joka vastaa HTTP GET -pyyntöihin lähettämällä konfigurointi HTML-sivun, jossa halutun tukiaseman SSID:n ja salasanan voi syöttää lomakkeeseen. Lomakkeen voi HTML-sivulta lähettää painikkeella aktivoidun HTTP POST -pyynnön avulla takaisin kameralaitteen palvelimelle, joka tallentaa lomakkeella syötetyt tiedot haihtumattomaan muistiin (engl. Non-Volatile Storage). Tämän jälkeen laite vaihtaa tilansa WiFi-asemaksi ja yhdistyy haluttuun tukiasemaan tallennetun SSID:n ja salasanan avulla.

Tilanteessa, jossa tukiasemaan yhdistymiseen vaaditut tiedot löytyvät kameralaitteen käynnistyessä muistista, luetaan SSID ja salasana sieltä, jolloin kameralaite käynnistyy suoraan WiFi-asemaksi.

Prototyyppivaiheessa tukiasematilassa toimivan kameralaitteen käynnistämä palvelin ei ole suojattu HTTPS:llä, mutta ESP32 tukee myös vaihtoehtoa suojatusta yhteydestä. HTTP-liikenne tapahtuu kuitenkin vain yksityisessä verkossa, joten tätä ei vielä prototyyppivaiheessa katsottu tarpeelliseksi toteuttaa. Suojatun yhteyden lisäksi ESP32 tukee myös flash-muistin enkryptausta, jolloin SSID ja salasana voidaan suojata myös laitteelle tallennettuina. Muistin enkryptausta ei myöskään prototyyppivaiheessa toteutettu, koska riskiä laitteen päätymisestä vääriin käsiin ei nähty kovin suurena. Tilanne on toinen, jos prototyypin kehitystä jatketaan ja etävalvontaratkaisu tuotteistetaan. Tällöin turvallisuusnäkökulmaa tulee painottaa eri tavalla. Turvallisuusnäkökulma on kuitenkin otettu huomioon ja ESP32:n turvallisuus-ominaisuuksien tuki oli yksi syy sen valikoitumisessa kameralaitteen alustaksi.

Mikäli kameralaitteesta katkaistaan virta tai se käynnistetään uudelleen reset-painikkeen painalluksella, tyhjennetään haihtumaton muisti, jolloin laite käynnistyy jälleen tukiasemaksi, koska SSID:tä ja salasanaa ei löydy enää muistista.

WiFi-asemana toimiessaan kameralaite ottaa kuvan esimääritetyin aikavälein, lähettää otetun kuvan määrättyyn palvelimen rajapintaan ja vaipuu syvään unitilaan (deep sleep) virrankulutuksen minimoimiseksi. Unitilasta kameralaite herää, kun on aika ottaa ja lähettää seuraava etävalvontakuva palvelimelle. Lähetyksen jälkeen kierto alkaa alusta ja laite menee takaisin uneen. Syvässä unitilassa kameralaite kuluttaa virtaa 10–100 μA, 20–40 mA normaaliajossa ja WiFi-lähetyksessä n. 200–300 mA. Erot ovat huomattavia. Kameralaitteen toiminta on esitetty kuvassa 9.



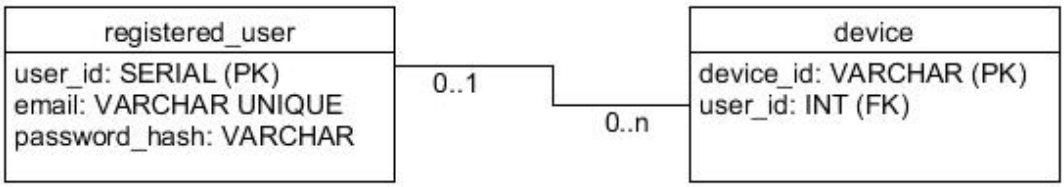
Kuva 9. Kameralaitteen toiminta.

5.2 Palvelinpuoli

Palvelin ja sen ohjelmointirajapinta yhdistävät kehitetyn etävalvontaratkaisun kaikki osat. Palvelimen kanssa kommunikoivat niin kameralaitteet kuin valvontakuvista kiinnostuneet käyttäjät. Palvelimen ohjelmointirajapinta on toteutettu modernilla REST-arkkitehtuurilla, jonka ansiosta rajapinta on helppo ja selkeä käyttöinen sekä laajennettavissa. Prototyyppivaiheessa palvelimelle ja sen rajapinnalle toteutettiin perusominaisuudet, joiden avulla kokonaisratkaisun toimivuutta pystyttiin testaamaan ja havainnollistamaan. Käytännön testaamisen mahdollistamiseksi palvelin on prototyyppivaiheessa sijoitettu Herokuun, joka on PaaS-ratkaisuna palvelimien ylläpitoa tarjoava yritys.

Palvelinpuoli koostuu Node.js-kehyksellä toteutetusta palvelimesta, PostgreSQL-tietokannasta sekä Google Cloud Platformin PaaS-muotoisesta tiedosto-varastoratkaisusta, Cloud Storagesta. Node.js-palvelin toimii REST-ohjelmointi-rajapintana, jonka kautta kaikki etävalvontaratkaisun kommunikaatio tapahtuu. Tietokannan rooli on toimia rekisteröityneiden käyttäjien ja tunnettujen kameralaitteiden tallennuspaikkana, jota vasten ohjelmointirajapinnan asiakkaat autentikoidaan. Cloud Storagea käytetään valvontakuvien säilytykseen, josta palvelin voi niitä hakea autentikoituaan ja autorisoituaan asiakkaan pyynnön.

PostgreSQL-tietokantaa käytetään palvelimella käyttäjä- ja laitetietojen pitkäaikais-säilytykseen. Prototyyppivaiheen tietokanta on hyvin suppea, mutta riittävä käyttäjien ja laitteiden autentikointiin. Tietokannan rakenne on esitetty kuvassa 10.



Kuva 10. Tietokannan taulut.

Tietoturvasyistä tietokantoihin ei milloinkaan tallenneta salasanoja sellaisenaan tekstimuodossa, vaan tiivisteenä (engl. hash). Tiiviste, eli hajautusarvo, muodostetaan yksisuuntaisella hajautusfunktiolla. Hajautusfunktiot ovat kryptografisia funktioita, jotka samalla syötteellä tuottavat aina saman muotoisen ja mittaisen tuloksen. Yksisuuntaisuus puolestaan tarkoittaa sitä, että tuloksesta on lähes mahdotonta johtaa alkuperäistä syötettä. (Buchanan 2017, 87–92.)

Yksisuuntaisten hajautusfunktioiden avulla käyttäjän rekisteröityessään syöttämä salasana voidaan tiivistää ja tiiviste tallentaa tietokantaan. Käyttäjän sisäänkirjautuessa salasanalle toistetaan sama tiivistystoimenpide, josta syntynyttä tiivistettä verrataan tietokantaan tallennettuun. Tiivisteiden täsmätessä käyttäjä on autentikoitu ja tälle voidaan luoda JWT.

Cloud Storage -tiedostovarastossa säilytetään etävalvontakameran lähettämiä kuvia. PaaS -muotoinen tiedostovarastoratkaisu tarjoaa hyödyllisiä ominaisuuksia toeteutetun etävalvontaratkaisun näkökulmasta. Tiedostovarastoon tallennetut tiedostot ovat automaattisesti enkryptattuja ja näin ollen suojattuja. Tiedostovaraston fyysinen sijainti on myös kehittäjän valittavissa ja prototyyppivaiheessa tiedostovarasto sijoitettiin Googlen Suomessa sijaitsevaan Haminan palvelinkeskukseen, jolloin tiedostohaun viive Suomessa ja lähialueilla on minimoitu. Lisäksi tiedostovarastoratkaisu tarjoaa mahdollisuuden asettaa automatisoituja toimintoja, kuten tiedostojen varmuuskopiointia toiseen palvelinkeskukseen toimintavarmuuden parantamiseksi. Automatisoituja toimintoja on todella paljon eri vaihtoehtoja, jonka takia PaaS-muotoinen tiedostovarastointi on etävalvontaratkaisun kannalta erinomainen. Prototyyppivaiheessa automatisoiduista toiminnoista ei kuitenkaan otettu käyttöön kuin ajastettu tiedostojen poisto, jonka avulla yli viikon vanhat valvontakuvat poistuvat tiedostovarastosta automaattisesti.

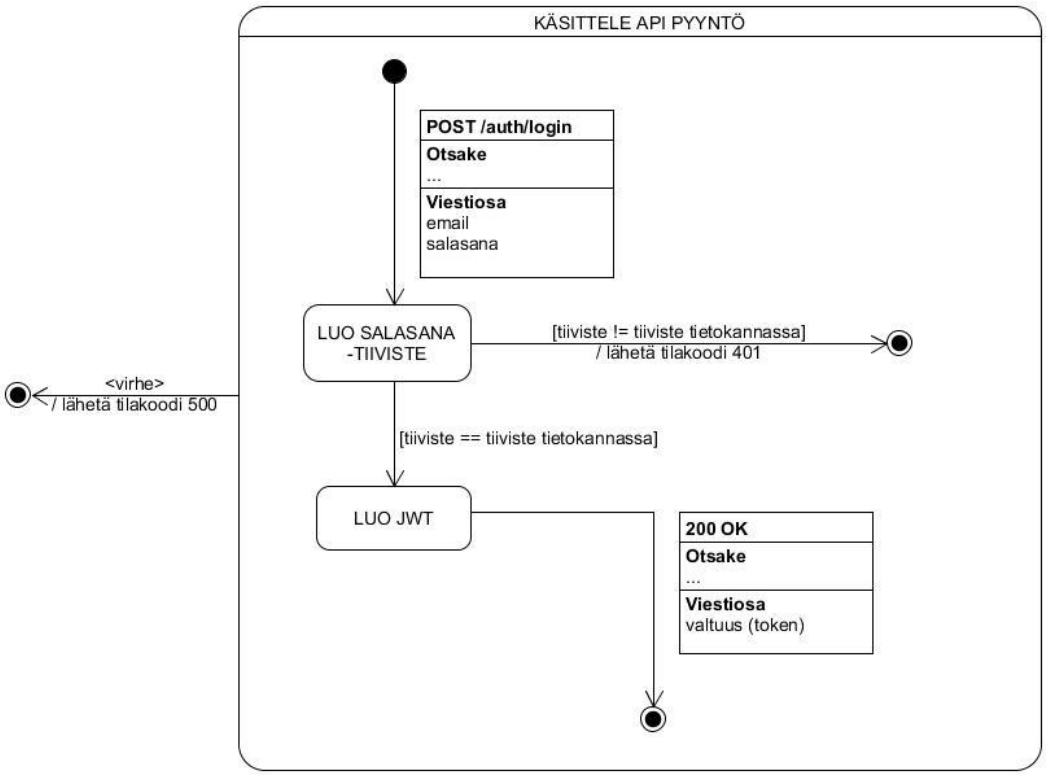
Kameralaitteen lähettämät valvontakuvat tallennetaan tiedostovarastoon laitetunnisteen ja aikaleiman kanssa, jolloin ne ovat ajoitettavissa ja sidottavissa tiettyyn laitteeseen, sekä laitteen kautta tiettyyn käyttäjään.

Cloud Storage -tiedostovarastossa tiedostot säilytetään enkryptattuna, eikä niitä pääse käsittelemään kuin autorisoitu GCP-käyttäjä. Tämän vuoksi Google tarjoaa Cloud Storage API:ssaan keinon luoda allekirjoitetun URL:n (engl. signed url), jonka avulla tietty tiedosto voidaan julkaista kertakäyttöisen URL:n taakse määrätyksi ajaksi. Määrätyn aikarajan umpeuduttua allekirjoitettu URL ei enää toimi, eikä tiedostoon pääse käsiksi. Tätä ominaisuutta käytetään etävalvontaratkaisun REST-ohjelmointirajapinnan valvontakuvien haussa esitystä varten.

Toteutettu REST-ohjelmointirajapinta

Palvelimen REST-ohjelmointirajapinta on etävalvontaratkaisun osien kommunikaation yhdistävä kanava. Prototyyppivaiheessa rajapinnalle on toteutettu neljä eri funktiota, joista kolme kohdistuu kameralaitteen kuviin ja yksi käyttäjän autentikointiin.

Ohjelmointirajapinnan URL /auth/login on tarkoitettu käyttäjän autentikoimiseen. Autentikointi tapahtuu lähettämällä tähän URL:iin HTTP POST -pyyntö, jonka viestiosassa on JSON-muodossa käyttäjän rekisteröimä sähköpostiosoite ja salasana. Palvelin tiivistää salasanan ja vertaa tiivistettä tietokannasta löytyvään käyttäjän salasanan tiivisteeseen. Tiivisteiden ollessa samat, palvelin luo JWT:n, johon se sisällyttää käyttäjän sähköpostiosoitteen, sekä JWT:n vanhentumisajankohdan. Lopuksi palvelin lähettää asiakkaalle vastauksen tilakoodilla 200 ja sisällyttää viestiosaan luodun JWT:n. Edellä kuvattu on tilanne, jossa kaikki tapahtuu odotetulla tavalla. Tilanteessa, jossa jotain menee pieleen, esimerkiksi rajapintaan lähetetään HTTP POST -pyyntö ilman sähköpostiosoitetta ja salasanaa, ei JWT:tä koskaan luoda ja palvelin lähettää asiakkaalle asianmukaisen tilakoodin vastauksessaan. Sisäänkirjautumispyynnön käsittely palvelimella on esitetty kuvassa 11.



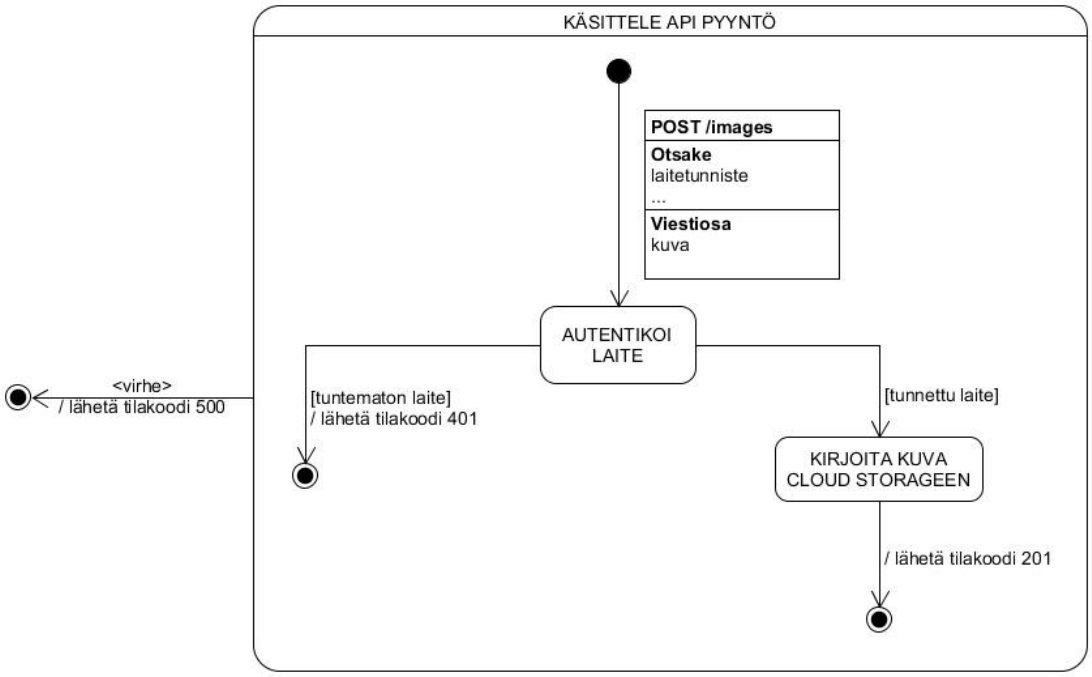
Kuva 11. REST-ohjelmointirajapinnan kautta sisäänkirjautuminen.

REST-rajapinnan kaikki /images URL:iin kohdistetut pyynnöt vaativat autorisoinnin, joten pyyntöihin on sisällytettävä sisäänkirjautumamlla luotu JWT. Tämä tapahtuu

HTTP-pyynnön ”authorization”-otsakkeen avulla, johon JWT sijoitetaan ennen pyynnön lähettämistä.

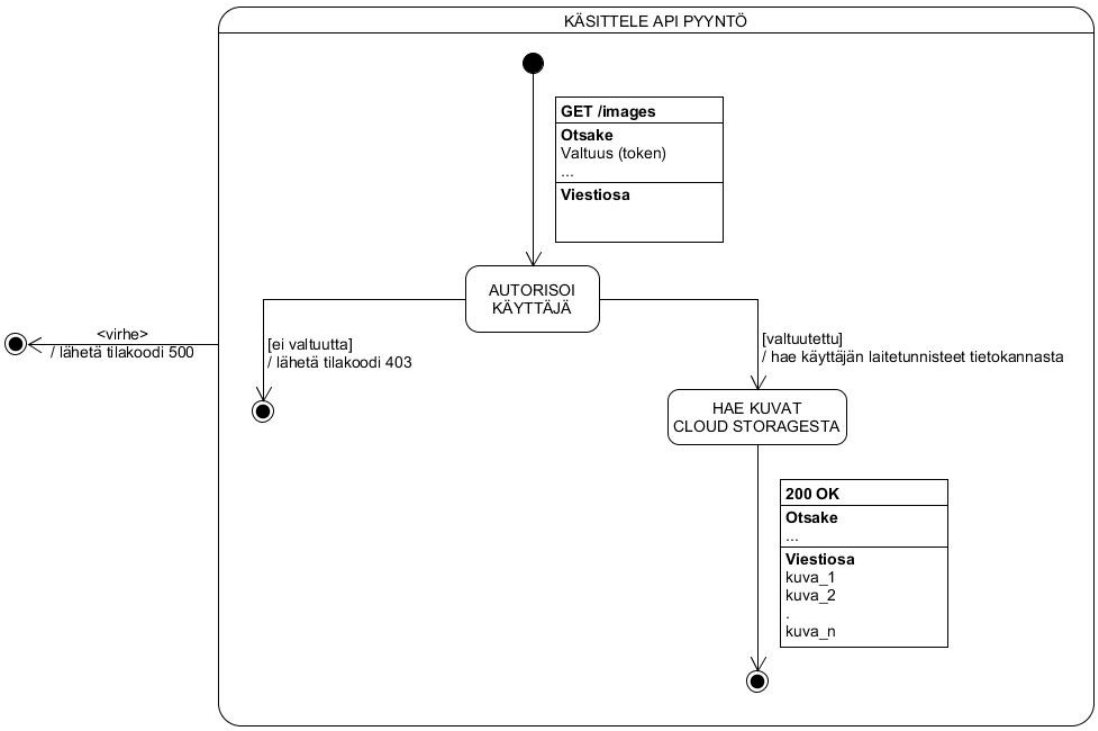
Autorisoituja pyyntöjä /images URL:iin voidaan kohdistaa sekä HTTP GET -metodilla että HTTP POST -metodilla. GET-metodilla rajapinnasta haetaan kaikki JWT:n sisältämän sähköpostiosoitteen identifioiman käyttäjän kuvat. POST-metodilla puolestaan autorisoitu kameralaite pystyy lähettämään valvontakuvia Cloud Storage -tiedostovarastoon tallennettavaksi.

Kameralaitteen autorisointi perustuu laitteen löytymiseen tietokannasta. Kameralaitteen lähettäessä valvontakuvan ohjelmointirajapintaan, sisällyttää se oman tunnisteensa HTTP-pyynnön otsakkeeseen, josta palvelin sen lukee pyyntöä käsitellessään ja tarkistaa sen löytymisen tietokannasta. Autorisointiprosessi on esitetty kuvassa 12. Valitulla autorisointitavalla on heikkoutensa, mutta se todettiin prototyyppivaiheessa riittäväksi. Kameralaitteen autorisointia ja sen haasteita ohjelmointirajapinnassa on pohdittu enemmän palvelinpuolen havainnot- ja jatkokehitysosiossa.



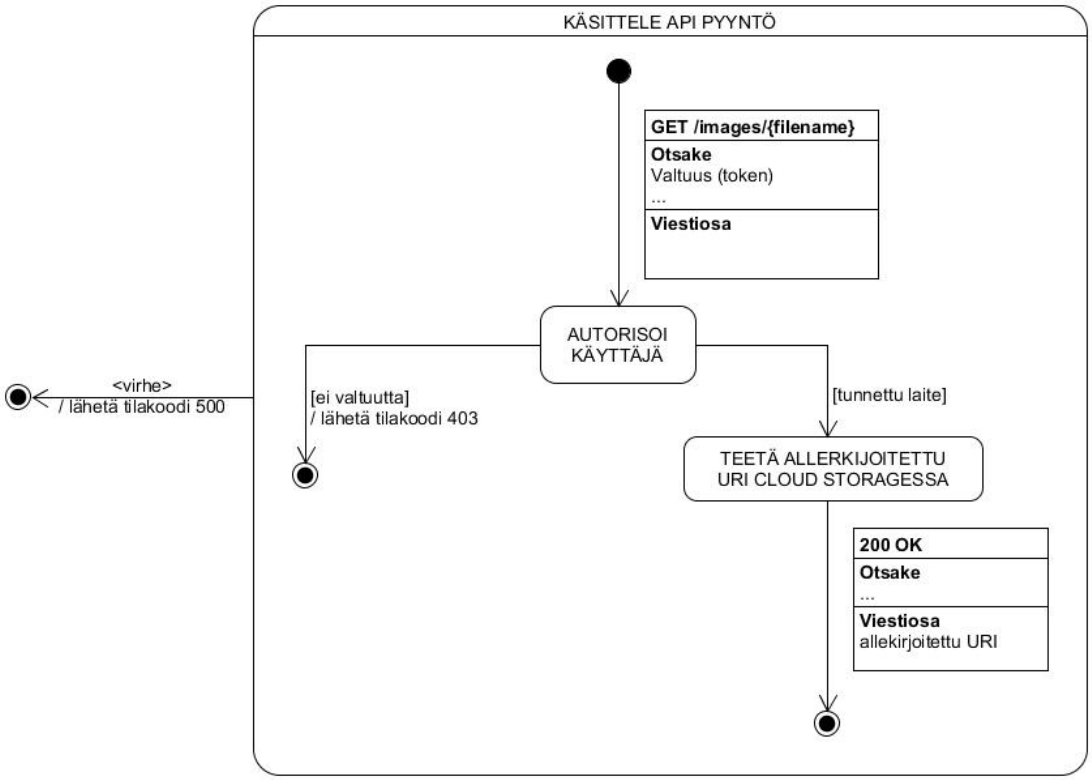
Kuva 12. Valvontakuvan lähetys REST-ohjelmointirajapintaan.

Käyttäjän autorisointi puolestaan tapahtuu modernilla ja REST-arkkitehtuurin mukaisella tavalla JWT:tä käyttäen. Palvelimen tarvitsee vain varmentaa pyynnön otsakkeessa oleva JWT. JWT:n varmennuksen onnistuttua palvelin tarkistaa tietokannasta, mitkä kameralaitteet kuuluvat JWT:hen sisällytetyn sähköpostiosoitteen omistajalle. Tämän jälkeen palvelin hakee kaikkien käyttäjän kameralaitteiden kuvat Cloud Storage -tiedostovarastosta ja lähettää niiden nimet JSON-muodossa vastauksessaan asiakkaalle. Ohjelmointirajapinnan kautta toteutettu kuvien haku on esitetty kuvassa 13.



Kuva 13. Käyttäjän valvontakuvien haku REST-ohjelmointirajapinnan kautta.

Viimeinen prototyyppivaiheessa toteutettu ohjelmointirajapintametodi kohdistuu yksittäiseen kuvaresurssiin ja sen tarkoitus on palauttaa asiakkaalle allekirjoitettu URL, jonka avulla yksittäiseen kuvaan Cloud Storage -tiedostovarastossa pääsee käsiksi. Tämä prosessi on kuvattu kuvassa 14.



Kuva 14. Näytettävän kuvan URL-osoitteen haku REST-ohjelmointirajapinnasta.

CORS toteutus

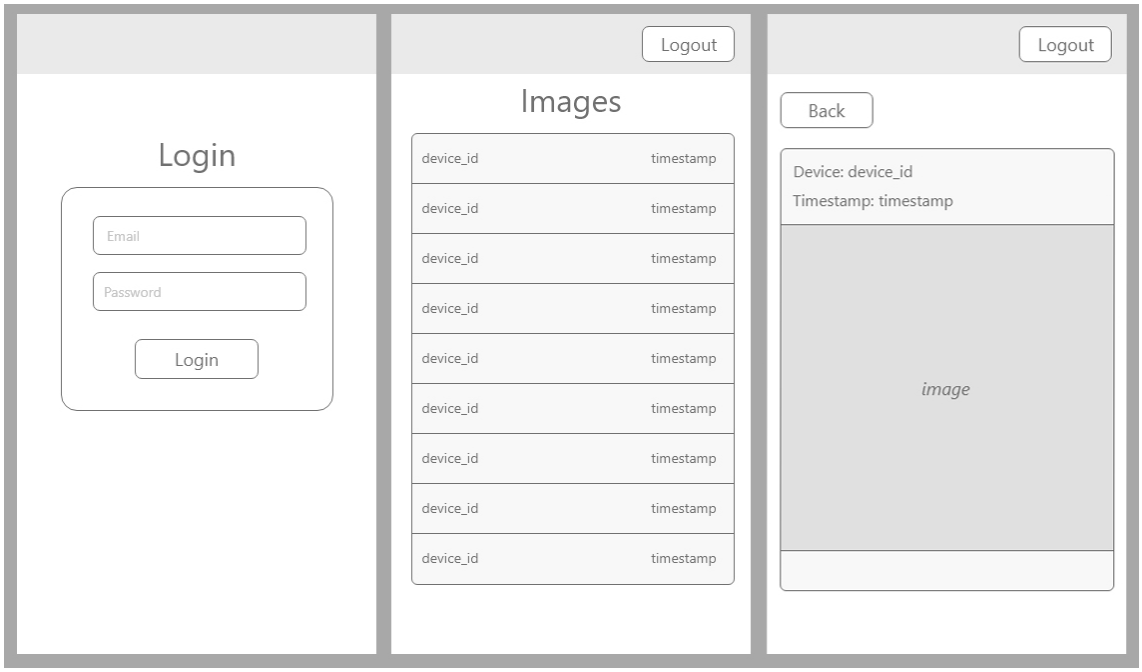
REST-ohjelmointirajapinnan tarkoitus oli olla julkinen, joten pyynnöt sen resursseihin tuli mahdollistaa CORS-otsakkeilla. Palvelin on määritelty lisäämään kaikkiin vastauksiinsa otsakkeet *Access-Control-Allow-Origin* arvolla ”*”, *Access-Control-Allow-Headers* arvolla ”*”, sekä *Access-Control-Allow-Methods* arvolla ”OPTIONS, GET, POST, HEAD”.

5.3 Asiakaspuoli

Etävalvontaratkaisun prototyypissä luotiin hyvin yksinkertainen selainpohjainen käyttöliittymä, joka koostuu kolmesta näkymästä: sisäänkirjautuminen, valvontakuvat, ja valvontakuvan tarkastelu. Käyttöliittymän tarkoitus on havainnollistaa etävalvonta-

ratkaisun helppokäyttöisyyttä ja hyödyllisyyttä käytännössä, sekä testata kehitetyn REST API:n toimivuutta.

Käyttöliittymän näkymät toteutettiin suunniteltujen rautalankamallien pohjalta käyttäen Angular JavaScript -kehystä. Rautalankamallit (kuva 15) suunniteltiin Adobe XD -työkalulla, joka on käyttöliittymä- ja käyttäjäkokemussuunnitteluohjelma. Rautalankamallit toteutettiin ainoastaan mobiilikoossa, koska käyttöliittymä on hyvin yksinkertainen eikä sen asettelu muutu eri päätelaitteilla.

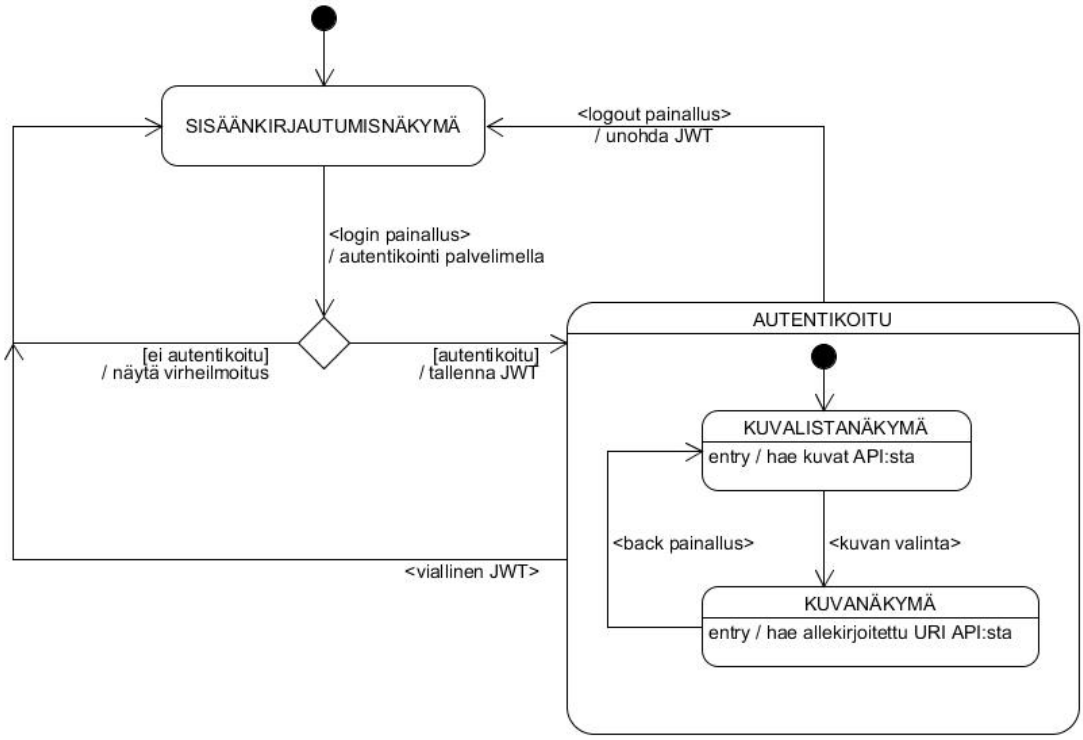


Kuva 15. Käyttöliittymän sunnitteluvaiheen rautalankamallit.

Toiminnaltaan käyttöliittymä on hyvin yksinkertainen. Ensimmäinen näkymä, johon käyttäjä sivulle tullessaan laskeutuu, on sisäänkirjautumissivu. Kirjautumissivulta käyttäjätiedot lähetetään palvelimen rajapintafunktiolle URL-osoitteessa /auth/login, joka vastaa käyttäjän autentikoinnin onnistumisenmukaisella tilakoodilla sekä onnistuneen autentikoinnin tapauksessa JWT:llä. JWT:n saatuaan käyttöliittymä ohjautuu automaattisesti seuraavaan näkymään eli etävalvontakuvien listaukseen. Välittömästi käyttäjän näkymälle ohjattuaan selain lähettää HTTP GET -pyynnön palvelimen rajapintafunktiolle URL-osoitteessa /images, joka varmentaa pyyntöön liitetyn JWT:n, hakee siihen sisällytetyn sähköpostiosoitteen omistajalle rekisteröityjen kameralaitteiden kuvat sekä listaa ne näkymään. Käyttäjän valitessa kuvan valvontakuvalistasta sitä

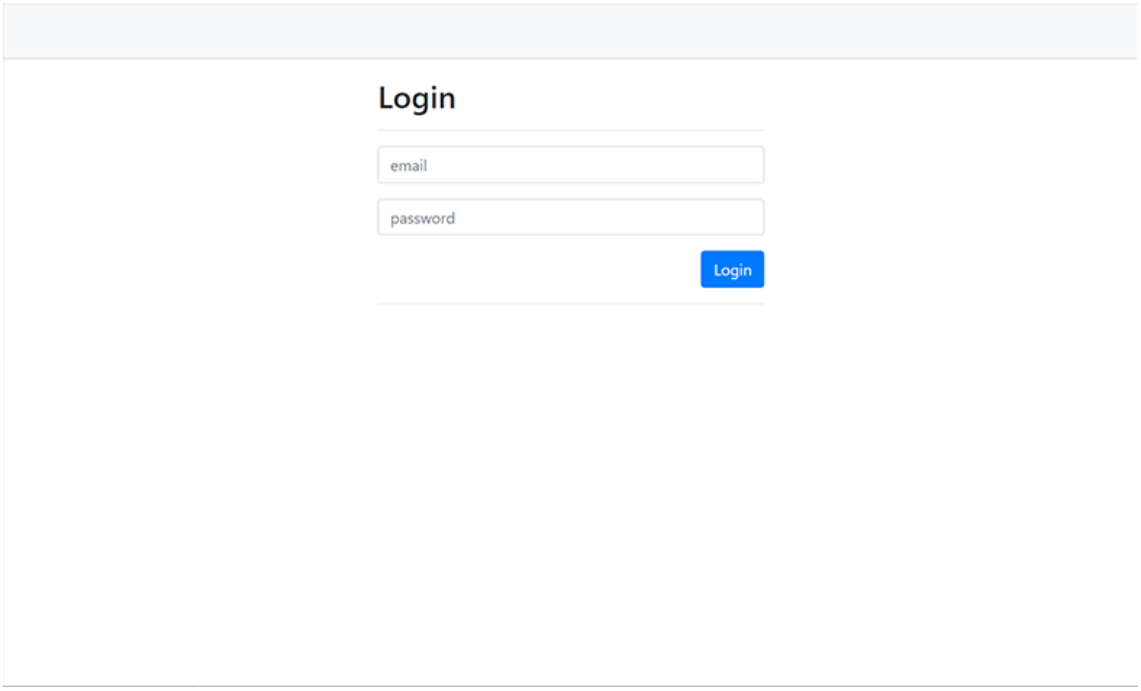
klikkaamalla, ohjaa selain käyttäjän valvontakuvan tarkastelunäkymään, jossa kuva viimein näytetään. Jälleen näkymään laskeuduttaessa, lähettää selain pyynnön rajapintafunktiolle URL-osoitteessa /images/*klikatun_kuvan_nimi*, johon palvelin vastaa lähettämällä kuvan allekirjoitetun URL-osoitteen mikäli pyyntöön sisällytetyn JWT:n varmennus onnistui. Palvelimen tarjoama allekirjoitettu URL sijoitetaan vastauksen saapumisen jälkeen suoraan näkymän HTML:n img-tunnisteen src-attribuuttiin, jolloin kuva tulee näkyviin. Käyttäjän ei siis tarvitse nähdä eikä käyttää allekirjoitettua URLia itse vaan käyttöliittymä hoitaa sen tämän puolesta.

Kaikissa JWT:n varmennustilanteissa, joissa varmennus epäonnistuu, käyttöliittymä palauttaa käyttäjän sisäänkirjautumisnäkymään. JWT:n olemassaolo tarkistetaan myös aina näkymää vaihtaessa. Käyttöliittymän näkymäsiirtymät ja toimintalogiikka on esitetty kuvassa 16.

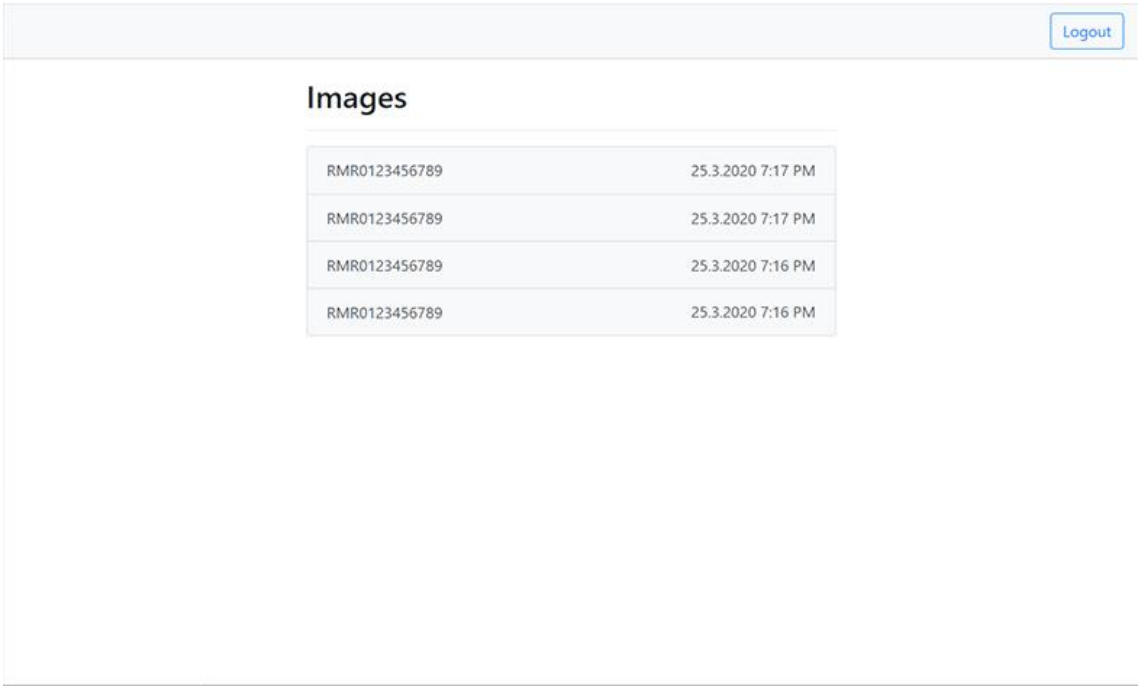


Kuva 16. Käyttöliittymän näkymäsiirtymät.

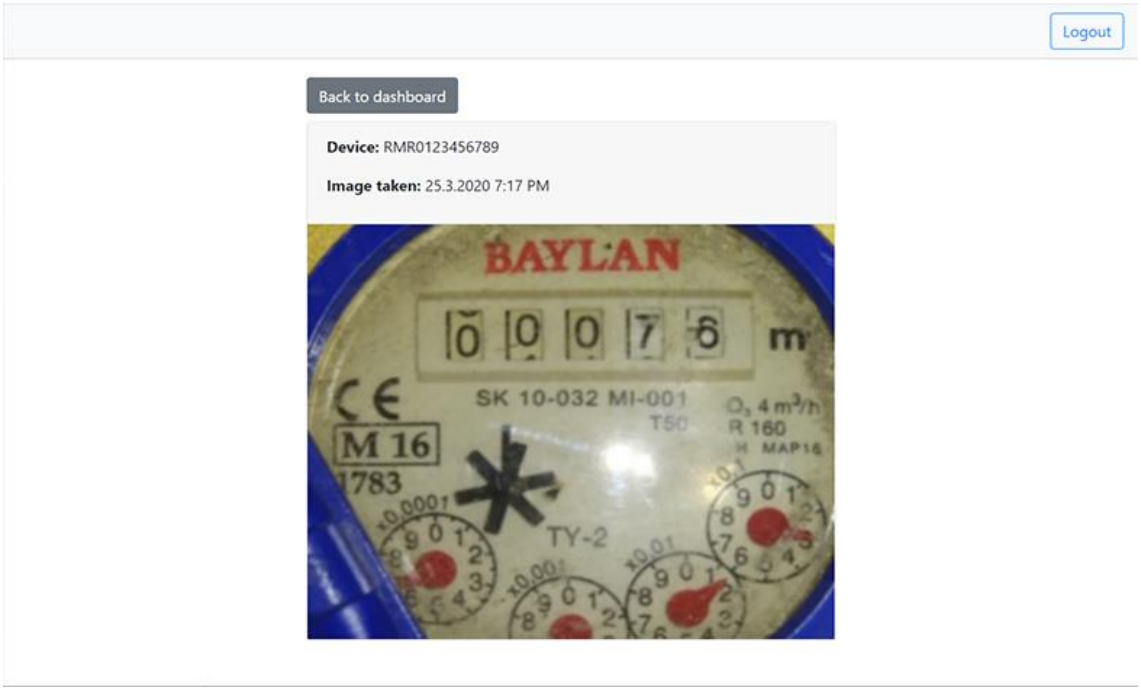
Käyttöliittymän näkymät ja vesimittaria valvovan kameralaitteen valvontakuvien tarkastelu käyttöliittymän kautta on esitetty kuvissa 17, 18 ja 19.



Kuva 17. Kuvakaappaus sisäänkirjautumisnäkymästä selaimessa.



Kuva 18. Kuvakaappaus toteutetusta kuvalistanäkymästä selaimessa.



Kuva 19. Kuvakaappaus toteutetusta valvontakuvan tarkastelu -näkymästä selaimessa.

6 HAVAINNOT JA JATKOKEHITYS

6.1 Kameralaite

Kehitetty kameralaite on prototyyppivaiheessa, kuten muutkin työn osat, joten lähtökohtaisesti sitä tullaan jatkokehittämään prototyypin kehityksessä nousseiden havaintojen tukemana. Merkittävin havainto, joka prototyypin kehityksen aikana tehtiin, oli kuvan lähetyksen ajoittaiset epäonnistumiset tai korruptoituneiden kuvien lähetys. WiFi-yhteyden ollessa heikko lähetys saattaa kestää liian kauan, jolloin palvelin aikakatkaisee pyynnön, jolloin kuvan lähetys epäonnistuu joko kokonaan tai vain osa siitä pääsee perille. Tätä pyritään jatkossa parantamaan korvaamalla ESP32:n integroitu antenni ulkoisella antennilla sekä tutkimalla eri kuvaresoluutioiden vaikutusta toimintavarmuuteen. Ulkoisen antennin vaikutusta virrankulutukseen tulee myös jatkossa arvioida.

Jatkokehityksen kannalta itsestäänselviä asioita toteutettavaksi ovat integroitu salama, jotta etävalvontaa voidaan suorittaa myös pimeissä olosuhteissa, sekä kotelo, joka mahdollistaa monipuoliset käyttöolosuhteet. Käyttöolosuhteiden monipuolistamiseksi tulee tutkia myös langattomuuden mahdollisuutta ja sen ympäristövaikutuksia. Työn tavoitteena oli luoda ekologisuutta edistävä ratkaisu, jota ei pidä riskeerata esimerkiksi kertakäyttöisiin pattereihin pohjautuvalla virtaratkaisulla. Käyttökohteesta riippuen akku- tai aurinkovoimalla toimivia ratkaisuja tulee jatkokehitysvaiheessa pohtia.

Ekologisuuden lisäksi joustavuus ja jatkettavuus olivat tämän opinnäytetyön perustavista tavoitteista. Joustavuuden ja jatkettavuuden edistämiseksi tulisi jatkossa tarjota kameralaitteen käynnistämän palvelimen jakaman konfigurointisivun tarjota enemmän muutettavia asetuksia käyttäjälle nykyisten SSID:n ja salasanan lisäksi. Jatkossa sivulta tulisi voida konfiguroida ainakin kuvanottointervalli, jolloin käyttäjä voisi itse päättää, kuinka usein etävalvontakuvan haluaa.

Mikäli kameralaitteita on tarkoitus saattaa suuremmalle käyttäjäkunnalle, tulee niiden tukea FOTA-päivityksiä (engl. Firmware Over the Air), HTTPS:ää oletuksena sekä enkryptata kaikki mahdollinen muisti. Nämä ovat asioita, joihin jatkokehityksessä tulee

erityisesti keskittyä, koska esineiden internet on edelleen monin paikoin heikosti suojattu, haavoittuvainen ja siksi paljon hyväksikäytetty.

Kameralaitteelle ei prototyyppivaiheessa toteutettu minkäänlaista koteloa. Jatkokehitysvaiheessa koteloa suunnitellessa tulee ottaa huomioon vaatimukset joustavuudesta ja ekologisuudesta. Kotelon tulee olla käyttökelpoinen monissa käyttökohteissa ja helposti asennettavissa, jolloin kyseeseen tulee vesitiiviys ja monipuoliset kiinnitys- ja jalustamahdollisuudet. Ekologisuuden näkökulmasta ensisijaisena ratkaisuna tulisi tutkia 3D-tulostusta kierrätysmuovia käyttäen.

6.2 Palvelinpuoli

Kokonaisuutena arvioiden REST-muotoinen ohjelmointirajapintatoteutus todettiin prototyypin pohjalta erittäin toimivaksi ja onnistuneeksi ratkaisuvalinnaksi. Ratkaisu täyttää asetetut tavoitteet joustavuuden ja jatkettavuuden osalta. Rajapinta on täysin erillinen ja muista etävalvontaratkaisun osista riippumaton ja sen päälle voidaan rakentaa monia erilaisia käyttöliittymiä ja jatkoratkaisuja. Yhtenä jatkoratkaisutavoitteena on luoda erillinen sovellus, joka käyttää tässä työssä toteutettua rajapintaa valvontakuvien hakuun jatkokäsittelyä varten tekoälyn ja konenäön avulla. Konenäön avulla pyrittäisiin lukemaan vesimittarilukema kuvasta digitaaliseen muotoon. Tämä on vain yksi jatkokehitysidea valokuvamuotoisen etävalvonnan tarjotessa lukemattomia integrointimahdollisuuksia.

Prototyypin pohjalta kehitettävää havaittiin kameralaitteen autentikoinnin ja autorisoinnin osalta. Tämänhetkinen malli, jossa kameralaite valvontakuvan lähettäessään lähettää myös oman laitetunnisteensa tarkistettavaksi tietokantaa vasten, ei ole kestävä ratkaisu jatkon kannalta. Tämä malli kyllä varmistaa sen, että vain tunnetut laitteet voivat tallentaa valvontakuvia, mutta mikään ei varmenna lähettäjän alkuperää, jolloin kuka tahansa voi tekeytyä lähettäväksi kameralaitteeksi, jos vain saa haltuunsa kameralaitteen kiinteän laitetunnisteen. Käytännössä tämä voi johtaa tilanteeseen, jossa kolmas osapuoli voi täyttää käyttäjän valvontakuvat omilla kuvillaan. Jatkokehitysvaiheessa tämä tulee ratkaista sekä kameralaite- että palvelinratkaisua parantamalla. Mahdollisesti lisäämällä kameralaitteen konfigurointisivulle käyttäjän

sisäänkirjautumistoiminto. Kameralaitteen kirjautuminen voisi pohjautua myös JWT:hen.

REST-arkkitehtuurin mukaista välimuistitusta prototyypissä toteutettu palvelinratkaisu ei vielä tue. Jatkokehityksessä sekin tulisi toteuttaa. Käyttäjän hakiessa kameralaitteidensa valvontakuvat rajapinnasta, tallennettaisiin ne välimuistiin, jolloin niitä ei tarvitsisi jatkuvasti hakea uudelleen Cloud Storage -tiedostovarastosta asti heikentäen suorituskykyä ja palvelimen vastausnopeutta.

6.3 Asiakaspuoli

Selainsovellusprototyyppi nosti esiin SPA-arkkitehtuurin haasteen näkymien suojauksesta ei-autorisoiduilta käyttäjiltä. SPA-arkkitehtuuria noudattavissa sovelluksissa toimintalogiikka on kaikki asiakkaan päässä, jolloin selainsovelluksen käyttäjä pääsee sivuston ohjelmakoodiin käsiksi ja sitä muuttamalla näkymiin, jotka on tarkoitettu vain autorisoiduille käyttäjille. Tämä on yleinen haaste sovelluksissa, joissa toiminnallisuus on asiakkaan puolella. Havainto ei kuitenkaan edellytä toimenpiteitä, koska näkymät itsessään eivät sisällä arkaluontoista tietoa, vaan kaikki tieto haetaan niille REST-ohjelmointirajapinnan kautta. Näin ollen selainsovelluksen väärinkäyttö johtaa korkeintaan tyhjille näkymille.

Asiakaspuolen jatkokehitys tulee sisältämään mobiilisovelluksen kehityksen selain-sovelluksen rinnalle, jolloin etävalvonta helpottuu. Lisäksi sekä selainsovelluksen että tulevan mobiilisovelluksen toimintoja tulee jatkokehittää ja lisätä. Työssä toteutettiin vain toiminnallisuuden testaamiseksi ja havainnollistamiseksi pakolliset toiminnot, mutta käyttäjämukavuutta tulee lisätä esimerkiksi mahdollisuudella järjestää kuvia laitekohtaisesti, sekä aikajärjestykseen. Kuvien lataaminen ja poistaminen ovat myös käyttäjien usein odottamia itsestäänselviä ominaisuuksia, joita prototyyppiversio ei tue.

Asiakassovelluksia on tarkoitus voida tulevaisuudessa luoda erilaisia eri valvontakohteisiin ja käyttötarkoituksiin. Esimerkiksi valvontakuvia vesimittareista on tarkoitus käsitellä palvelimen päässä tekoälyn avulla ja lukea mittarilukema konenäöllä, jolloin asiakassovellukseen voitaisiin luoda kuvaajia kulutuksesta sekä näyttää hälytyksiä havaituista vuodoista.

6.4 Kokonaisratkaisu

Kokonaisvaltaisesti tarkasteltuna etävalvontaratkaisun prototyyppi oli onnistunut ja tuotti toivotusti jatkokehityksen kannalta tärkeitä havaintoja. REST-ohjelmointirajapintaratkaisu osoittautui toimivaksi valinnaksi joustavan ja jatkettavan ratkaisun toteuttamiseksi.

Lopullisia kustannuksia etävalvontaratkaisulle ei vielä prototyypin pohjalta pystytty tarkasti määrittämään, koska käytettyjen PaaS-ratkaisuiden hinnoittelu perustuu resurssien käyttömäärään, joka puolestaan riippuu käyttäjien ja ohjelmointirajapintaa käyttävien sovellusten määrästä. Lisäksi prototyyppivaiheen PaaS-ratkaisut eivät välttämättä ole lopullisia, joten kustannusrakenne saattaa niiden osalta muuttua merkittävästikin jatkokehitysvaiheessa. Prototyypin kustannukset jäivät kuitenkin niin pieniksi, että edullisuustavoitte voidaan tässä vaiheessa katsoa saavutetuksi. Prototyyppi käyttää niin vähän PaaS-ratkaisuiden resursseja että niiden käyttö pysyi laskutettavan käyttörajan alapuolella, jolloin ainoat kustannukset syntyivät kameralaitteen hankinta- ja virrankulutuskustannuksista, jotka jäivät alle 15 €:n. Nämä kustannukset voidaan katsoa edullisiksi, vaikka niille ei täydellistä vertailukohdetta olekaan, vastaavan ratkaisun olemassa olon puutteen vuoksi. Kustannuksia voidaan kuitenkin verrata ja suhteuttaa esimerkkikäyttökohteena toimineen vesimittarin olemassa olevaan etävalvonta-ratkaisuun, Leaklookiin. Leaklookin vesimittarin etäluvun mahdollistava lisälaite maksaa 199 € (Leaklook.io [n.d.]). Tähän verrattuna toteutettu etävalvontaratkaisu on hyvin edullinen.

7 YHTEENVETO

Opinnäytetyön tavoite oli luoda etävalvontaratkaisun prototyyppi, joka tukee useaa käyttäjää ja kameralaitetta, on sekä ekologinen, edullinen että joustava. Työssä onnistuttiin luomaan tavoitteen mukainen prototyyppi, jonka pohjalta saatiin arvokasta tietoa jatkokehityksen kannalta. Kaikki ratkaisun osat saatiin toteutettua suunnitellusti ja toimimaan keskenään toivotulla tavalla kokonaisvaltaisena järjestelmänä. Etävalvontaratkaisua voidaan tavoitteenmukaisesti soveltaa eri käyttökohteisiin ja käyttää ohjelmointirajapinnan kautta muissa sovelluksissa. Joustava ratkaisu lisää tavoitteen mukaisesti ekologisuutta lisäämällä kulutuksenseurannan mahdollisuuksia ja mahdollistamalla valvontalaitteen käytön monipuolisesti eriliaisissa valvontakohteissa. Edullisuuden tavoitekin katsotaan saavutetuksi, vaikka loppukäyttäjän kustannuksia ei vielä prototyypin pohjalta pystyttykään tarkasti määrittämään.

LÄHTEET

AI-Thinker [n.d.]. ESP32-CAM Product Specification. Viitattu 22.3.2020 https://loboris.eu/ESP32/ESP32-CAM%20Product%20Specification.pdf

Angular.io [n.d.]. Introduction to Anguar Concepts. Viitattu 23.3.2020 https://angular.io/guide/architecture

Auth0 [n.d.]. Authentication and Authorization. Auth0. Viitattu 21.3.2020 https://auth0.com/docs/authorization/concepts/authz-and-authn

Buchanan, W. 2017. Cryptography. Gistrup: River Publishers

Conrad, E.; Misenar, S. & Feldman, J. 2016. Eleventh Hour CISSP® Study Guide. 3. Yhdysvallat, MA Cambridge: Syngress.

Espressif Docs [n.d.]. ESP-IDF Get Started. Viitattu 22.3.2020 https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html

Fielding, R. & Reschke, J. 2014a. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, IETF RFC 7230. Viitattu 15.3.2020 https://tools.ietf.org/html/rfc7230.

Fielding, R. & Reschke, J. 2014b. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Contentand Routing. IETF RFC 7231. Viitattu 15.3.2020 https://tools.ietf.org/html/rfc7231.

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. Väitöskirja. Information and Computer Science. Irvine: University of California. Viitattu 16.3.2020 https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

Fielding, R. 2008. REST APIs Must be Hypertext-driven. Viitattu 22.3.2020 https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

Google Cloud [n.d.]. Google Cloud Storage Product Overview. Viitattu 22.3.2020 https://cloud.google.com/storage

Heroku [n.d.]. What is Heroku?. Viitattu 23.3.2020 https://www.heroku.com/what

Jones, M.; Bradley, J. & Sakimura, N. 2015. JSON Web Token (JWT). IETF RFC7519. Viitattu 21.3.2020 https://tools.ietf.org/html/rfc7519

JWT.io [n.d.]. Introduction to JSON Web Tokens. Viitattu 22.3.2020 https://jwt.io/introduction/

van Kesteren, A. 2014. Cross-Origin Resource Sharing. W3C. Viitattu 21.3.2020 https://www.w3.org/TR/cors/

Kiviniemi, A. 22.10.2019. Jukalle kävi hurja kämmi - postissa saapui nelinumeroinen vesilasku: ”Mietin jo, onko kotini kohta homeessa”. Iltalehti. Viitattu 25.3.2020

Leaklook.io [n.d.]. LeakLook-palvelu omakotitaloon. Viitattu 29.3.2020 https://leaklook.io/tuote/leaklook-palvelu/

MDN Contributors [n.d.]a. Mozilla Developer Network Documentation, HTTP Messages. Viitattu 15.3.2020 https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

MDN Contributors [n.d.]b. Mozilla Developer Network Documentation, Client-Server Overview. Viitattu 15.3.2020 https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview

Node.js [n.d]. Node.js homepage. Viitattu 22.3.2020 https://nodejs.org/en/

PostgreSQL [n.d.]a. PostgreSQL About page. Viitattu 22.3.2020 https://www.postgresql.org/about/

PostgreSQL [n.d.]b. PostgreSQL Hosting Providers Europe. Viitattu 22.3.2020 https://www.postgresql.org/support/professional_hosting/europe/

Puder, A.; Römer, K. & Pilhofer, F. 2006. Distributed Systems Architecture – A Middleware Approach. San Francisco: Morgan Kaufmann Publishers.

Rescorla, E. 2000. HTTP Over TLS. Network Working Group RFC 2818. Viitattu 25.3.2020 https://tools.ietf.org/html/rfc2818

Rescorla, E. 2018. “The Transport Layer Security (TLS) Protocol Version 1.3”. IETF RFC 8446. Viitattu 25.3.2020 https://tools.ietf.org/html/rfc8446

Suomen Turvatuote [n.d.]. Langattomat WiFi-valvontakamerat. Viitattu 30.3.2020 https://suomenturvatuote.fi/category/24/langattomat-wifi-valvontakamerat