Tampereen ammattikorkeakoulu

Selainpohjaisen kuvastegano-grafiatyökalun toteuttaminen

Roope Lindström

OPINNÄYTETYÖ
Huhtikuu 2020

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

LINDSTRÖM, ROOPE:
Selainpohjaisen kuvasteganografiatyökalun toteuttaminen

Opinnäytetyö 30 sivua
Huhtikuu 2020

Työn tarkoituksena oli toteuttaa selainpohjainen sovellus, jonka avulla voidaan piilottaa kuvatiedostoon teksti- tai kuvamuotoinen viesti tai paljastaa viesti kuvasta. Tätä kutsutaan steganografiaksi, joka on kryptografian eli salakirjoitustekniikan muoto. Kuvasteganografiaa käytetään muun muassa digitaalisen materiaalin vesileimaamiseen.

Piilotettava data koodataan alkuperäisen kuvatiedoston pikseleiden värikanavien vähiten merkitseviin bitteihin, jolloin lopputuloksena tuotetusta kuvasta on vaikea paljaalla silmällä huomata eroa alkuperäiseen kuvaan. Kuvatiedostojen muokkaustoiminnot toteutettiin Python-ohjelmointikielellä, johon pohjautuvaa Flask-sovelluskehystä on hyödynnetty sovelluksen REST-rajapinnan toteuttamisessa. Sovelluksen verkkokäyttöliittymä toteutettiin JavaScript-ohjelmointikieleen pohjautuvan React-käyttöliittymäkirjaston avulla, ja käyttöliittymän kautta voidaan lähettää palvelimelle kuvatiedostoja ja piilotettavaa dataa sekä vastaanottaa palvelimelta muokkaustoimintojen tuloksena tuotettuja kuvatiedostoja.

Työn tuloksena on yksinkertainen selainkäyttöliittymä ja palvelintoteutus, jonka REST-rajapintaa voidaan hyödyntää myös käyttöliittymästä irrallisena komponenttina. Sovelluksen avulla tuotetuista kuvista nähdään myös, miten toiminnoissa käytettyjen vähiten merkitsevien bittien määrä vaikuttaa lopputuloksen kuvanlaatuun.

Toteutettua sovellusta voidaan käyttää esimerkkinä kryptografian ja steganografian mahdollisuuksista. Työssä käytetyt menetelmät ovat yksinkertaisia, eikä esimerkiksi steganografisten kuvien analysointityökaluilta suojautumiseen ole otettu kantaa. Huomiota on kuitenkin kiinnitetty sovelluksen laajennettavuuteen, joka on mahdollista toteuttaa esimerkiksi parantamalla kuvatiedostojen muokkaustoiminnoissa käytettyjä algoritmeja.

Asiasanat: steganografia, kryptografia, python, javascript, react

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

LINDSTRÖM, ROOPE:
Implementation of a Browser-Based Image Steganography Tool

Bachelor's thesis 30 pages
April 2020

The purpose of this thesis was to implement a browser-based image steganography tool, which could be used to conceal a text or image file within another image file or reveal a message from an image. Steganography is a form of digital cryptography, which can be used to watermark material such as text, videos and images.

The data to be concealed is encoded pixel by pixel in the least significant bits of the original image file, which results in an image file that is difficult to distinguish from the original image with the naked eye. The image processing operations were implemented in the Python programming language, which is also the language in which the Flask micro web framework is written in. Flask was used to implement the REST API of the application, and the API serves as the bridge between the server and the user interface, which was implemented in React.

The result of this thesis is a simple web application and a server implementation, which can be used as its own separate component. The generated images can be used to demonstrate how the number of least significant bits used to encode the data affects the output image quality.

The implemented application can be used as an example of the possibilities of cryptography and steganography. The methods presented in this thesis are simple and the generated images aren’t proofed for steganographic image analysis tools. However, attention has been paid to the extensibility of the application, which is possible to implement by improving the encoding and decoding algorithms used in the image processing operations, for example.

Key words: steganography, cryptography, python, javascript, react

SISÄLLYS

LYHENTEET JA TERMIT

Base64	Binääritiedostojen tekstimuotoinen koodaustapa
C	Yleiskäyttöinen proseduraalinen ohjelmointikieli
C++	Yleiskäyttöinen korkean tason ohjelmointikieli
CSS	Cascading Style Sheets Kuvauskielisten dokumenttien tyylimäärittelykieli
DCT	Discrete cosine transform Diskreetti kosinimuunnos, häviöllinen pakkausalgoritmi
DOM	Document Object Model HTML- ja XML-dokumenttien käyttämä puumalli
Git	Hajautettu versionhallintajärjestelmä
HTML	Hypertext Markup Language Selaimessa esitettävien dokumenttien kuvauskieli
HTTP	Hypertext Transfer Protocol Internetin hyödyntämä siirtoprotokolla
Kryptoanalyysi	Salaus- tai salakirjoitustekniikoiden murtaminen
Kryptografia	Salaus- tai salakirjoitustekniikka
LSB	Least significant bit Vähiten merkitsevä bitti
PNG	Portable Network Graphics Kuvatiedostomuoto, joka tukee häviötöntä pakkausta
POST	HTTP-protokollan määrittämä metodi datan lähettämiseen palvelimelle
REST	Representational state transfer – Verkkosovellusten ohjelmointirajapintojen arkkitehtuurimalli
RGB	Red, Green, Blue Tietokonenäyttöjen ja televisioiden hyödyntämä väritila
Unicode	Maailmanlaajuinen merkistöstandardi
UTF-8	Unicode Transformation Format 8-bit Unicoden merkistökoodaus
WSGI	Web Server Gateway Interface Python-ohjelmointikielen verkkopalvelinstandardi
XML	Extensible Markup Language Datan säilömiseen ja siirtämiseen käytetty kuvauskieli

1 JOHDANTO

Tässä opinnäytetyössä toteutetaan selainpohjainen sovellus, jonka avulla voidaan piilottaa kuvatiedostoon teksti- tai kuvamuotoinen viesti tai paljastaa viesti kuvasta. Tätä kutsutaan steganografiaksi, joka on kryptografian eli salakirjoitustekniikan muoto. Steganografia on laaja termi, jolla voidaan tarkoittaa minkä tahansa viestin tai datan piilottamista toiseen tai samaan dataformaattiin. Tässä työssä keskitytään kuvasteganografiaan, jota käytetään muun muassa digitaalisen materiaalin vesileimaamiseen.

Toteutettava sovellus koostuu Python-ohjelmointikieleen pohjautuvalla Flask-ohjelmistokehyksellä toteutetusta REST-rajapinnasta sekä JavaScript-ohjelmointikieleen pohjautuvalla React-käyttöliittymäkirjastolla toteutetusta verkkokäyttöliittymästä. Käyttöliittymän kautta voidaan lähettää palvelimelle kuvatiedostoja ja piilotettavaa dataa sekä vastaanottaa palvelimelta muokkaustoimintojen tuloksena tuotettuja kuvatiedostoja.

Datan piilottamisessa ja paljastamisessa käytetään yksinkertaisia menetelmiä, jotka perustuvat kuvatiedostojen pikselien värikanavien vähiten merkitsevien bittien hyödyntämiseen. Työssä tarkastellaan myös menetelmissä käytettyjen vähiten merkitsevien bittien määrän vaikutusta lopputuloksen kuvanlaatuun, eli siihen, kuinka paljon tuotettu kuva eroaa alkuperäisestä kuvasta paljaalla silmällä tarkasteltuna.

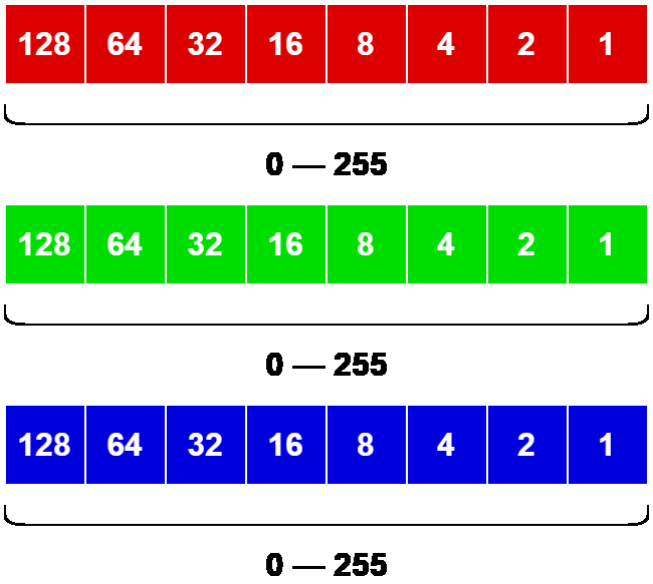
2 STEGANOGRAFIAN PERIAATTEITA

Steganografialla tarkoitetaan viestin, kuvan tai tiedoston piilottamista toiseen vastaavaan tiedotusvälineeseen (Merriam-Webster n.d.). Se on kryptografian muoto, jota käytetään esimerkiksi tekijänoikeuden alaisen kuva-, video- tai tekstimateriaalin digitaaliseen vesileimaamiseen (Wayner 2009, 324).

2.1 LSB-tekniikka

Yksinkertainen tapa piilottaa viesti kuvatiedostoon on hyödyntää kohinaa, joka esiintyy etenkin valokuvissa kirkkauden ja värin satunnaisena vaihteluna. Värikuvatiedostoa voidaan esittää kaksiulotteisena pikselimatriisina, jonka jokainen pikseli sisältää 24-bittisen esityksen kuvan väriarvoista. (Wayner 2009, 9; Farooque & Rohankar 2013.)

24-bittinen RGB-väri koostuu punaisesta, vihreästä ja sinisestä kanavasta (kuvio 1), eli jokainen kanava on kooltaan 8 bittiä. Yhden kanavan väriarvo voi tällöin vaihdella välillä 0 ja 255 (Mstafa & Bach 2013). Kuvion oikean reunan bittejä kutsutaan vähiten merkitseviksi biteiksi (englanniksi least significant bits, LSB), sillä niiden muuttaminen muuttaa lopullista väriarvoa vähiten (taulukko 1). Jos salattavan viestin koodaamiseen käytetään vain yhtä tai kahta vähiten merkitsevää bittiä, muokatusta kuvasta on usein vaikea huomata paljaalla silmällä eroa alkuperäiseen kuvaan, varsinkin jos alkuperäinen kuva ei ole tarkasteluhetkellä saatavilla (Computerphile, 2015).



KUVIO 1. 24-bittisen RGB-värin kanavat

TAULUKKO 1. Binäärilukujen desimaaliesityksiä

Binääriesitys	Desimaaliesitys
1111 1111	255
1111 1110	254
0111 1111	127

Kuvatiedostoon voidaan tallentaa arbitraarista binääridataa, esimerkiksi UTF-8-koodattua tekstiä. UTF-8 on koodaustapa, jolla voidaan esittää Unicode-merkistön merkit 1–4 tavulla riippuen merkin Unicode-kokonaislukuarvosta (Kuhn 2009; UTF-8 2019).

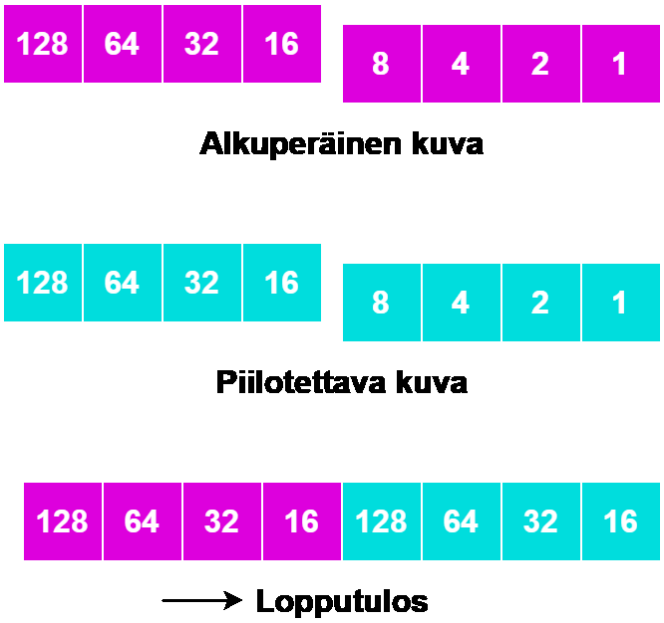
Jos viestin tallentamiseen käytetään yhtä vähiten merkitsevää bittiä jokaisella RGB24-kuvatiedoston värikanavalla ja kuvan leveys ja korkeus ovat 512 pikseliä, voidaan tiedostoon tallentaa 98 kilotavua dataa. Määrä vastaa siis lähes sataa tuhatta merkkiä, jos jokainen merkki on koodattu yhdellä tavulla.

2.2 Kuvatiedoston piilottaminen toiseen kuvatiedostoon

Kun kuvatiedostoon piilotetaan toinen kuvatiedosto, piilotettava kuva on usein alkuperäistä kuvaa pienempi tai korkeintaan samankokoinen, jotta alkuperäisen kuvan koodaamiseen jää vähintään puolet käytettävissä olevista biteistä. Kuvan

piilottamisessa voidaan myös käyttää esimerkiksi yksiväristä kuvaa, jolloin kuvan pikseleistä tallennetaan vain yksi keskimääräinen kirkkausarvo kolmen väriarvon sijaan. Tyypillinen tapa yhdistää kaksi kuvatiedostoa on tallentaa alkuperäisen kuvan vähiten merkitseviin bitteihin piilotettavan kuvan eniten merkitsevät bitit (kuvio 2). (Solak & Altinişik 2018.)

Jos alkuperäisen kuvan värikanavien biteistä käytetään neljää eniten merkitsevää bittiä alkuperäisen kuvan koodaamiseen, piilotettava kuva voidaan koodata värikanavien neljään vähiten merkitsevään bittiin, jolloin lopputuloksessa molemmista kuvista on jäljellä neljä merkitsevää bittiä. Piilotettu kuva voidaan paljastaa yksinkertaisesti siirtämällä lopputuloksena syntyneen kuvan neljä vähiten merkitsevää bittiä neljän eniten merkitsevän bitin paikalle.



KUVIO 2. Kuvatiedoston piilottaminen toiseen kuvatiedostoon

3 SOVELLUSTEKNIIKAT

Toteutettava sovellus koostuu selainkäyttöliittymästä ja palvelintoteutuksesta, jota selain hyödyntää REST-rajapinnan kautta. Kaikki toteutettu ohjelmakoodi on säilötty Git-etärepositorioon varmuuskopioinnin ja versiohistorian säilyttämisen vuoksi. Sovelluksen selainkäyttöliittymä on toteutettu JavaScript-ohjelmointikieleen pohjautuvalla React-ohjelmistokehyksellä ja palvelintoteutus Python-ohjelmointikieleen pohjautuvalla Flask-ohjelmistokehyksellä. Sovelluksen kehittämiseen käytetään Microsoftin Visual Studio Code -lähdekoodieditoria, joka mahdollistaa esimerkiksi kehittämistä helpottavien työkalujen käytön. Sovelluskehityksessä käytetään lähdekoodin staattiseen analysointiin tarkoitettuja työkaluja, jotka helpottavat yhdenmukaisen lähdekoodin kirjoittamista ja estävät mahdollisten virheiden esiintymistä lähdekoodissa. JavaScript-koodissa tähän käytetään Eslint-työkalua ja Python-koodissa Pylint-työkalua.

3.1 JavaScript-ohjelmointikieli

JavaScript on dynaamisesti tyypitetty, tulkattava tai ajonaikaisesti käännetty (englanniksi just-in-time, JIT) ohjelmointikieli, jota käytetään verkkoselainten pääasiallisena skriptikielenä. JavaScript on prototyyppipohjainen ohjelmointikieli, joka tukee useita ohjelmointiparadigmoja, esimerkiksi proseduraalista ohjelmointia, olio-ohjelmointia ja funktionaalista ohjelmointia. (Mozilla Web Docs 2020a.)

3.1.1 ECMAScript-standardi

JavaScript pohjautuu ECMAScript-standardiin, joka määrittelee muun muassa ohjelmointikielen tietotyypit, operaattorit, syntaksin ja lausekkeet (Ecma International 2019). ECMAScript-standardista julkaistaan vuosittain uusi versio, jonka sisältämät uudet ominaisuudet valitaan Ecma TC39 -komitean laatiman standardointiprosessin mukaisesti (Ecma TC39 n.d.; taulukko 2).

TAULUKKO 2. TC39-standardointiprosessin tasot

	Taso	Laatuvaatimukset
1	Proposal (ehdotus)	-
2	Draft (luonnos)	Semantiikka, syntaksi ja rajapinnat toteutettu pääosin
3	Candidate (ehdokas)	Semantiikka, syntaksi ja rajapinnat toteutettu
4	Finished (valmis)	Kaikki toteutetut muutokset integroitu standardiin

ES.Next on dynaaminen nimi, jolla viitataan seuraavaan julkaistavaan ECMAScript-versioon. ES.Next sisältää edellä mainittuja ehdotuksia uusista ominaisuuksista. (Mozilla Web Docs 2020b).

Varsinainen JavaScript-toteutus riippuu ajoympäristöstä ja on usein toteutettu ajoympäristölle natiivilla ohjelmointikielellä, esimerkiksi C++:lla. Esimerkkejä JavaScript-toteutuksista ovat Googlen avoimen lähdekoodin V8-moottori sekä Mozillan tuotteissa käytetty SpiderMonkey-ajoympäristö (Mozilla Web Docs 2020a; V8 2020).

3.1.2 React-kirjasto

React on Facebookin kehittämä JavaScript-kirjasto, joka on tarkoitettu erityisesti verkkokäyttöliittymien toteuttamiseen. Reactin näkymät ovat deklaratiivisia, joten niiden tilan muuttuminen aiheuttaa selaimen DOM-rakenteen päivittymisen. Näkymien tila on sidottu komponentteihin, joita käytetään React-käyttöliittymien rakennuspalikoina. Komponenteista voidaan muodostaa monitasoinen puurakenne, joka muistuttaa HTML-tiedoston rakennetta. Reactin komponenttilogiikka kirjoitetaan JavaScriptillä, mikä mahdollistaa komponenttien tilan muuttamisen ja säilömisen DOM-rakenteen ulkopuolella sekä ulkoisten JavaScript-kirjastojen hyödyntämisen komponenttien toteuttamisessa. (Facebook 2020d.)

Komponentin päätehtävä on vastaanottaa props-parametrejä ja palauttaa niiden perusteella renderöitävä puurakenne. Reactin props-mekanismi mahdollistaa parametrien välittämisen puuhierarkian ylemmiltä komponenteilta alemmille kom-

ponenteille ja muodostaa dynaamisen React-käyttöliittymän perustan. React tukee komponenttien kirjoittamista JSX-syntaksilla, joka on XML-kuvauskielen kaltainen JavaScript-lisäosa (kuva 1). JSX helpottaa React-elementtien kirjoittamista, sillä se vaatii vähemmän kirjoitettavia merkkejä kuin puhtaalla JavaScriptillä kirjoitetut React-komponentit sekä mahdollistaa hyödyllisten varoitusten ja virheilmoitusten näyttämisen sovelluskehityksen aikana. Lisäksi virhetilanteita voi ehkäistä erillisellä PropTypes-kirjastolla, jota käytetään React-komponenttien props-parametrien ajonaikaiseen validoimiseen ja tyyppien tarkastamiseen. (Facebook 2020c.)

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

KUVA 1. JSX-syntaksi (Kuva: Facebook 2020)

React-sovellus voidaan toteuttaa yhden sivun sovelluksena (englanniksi single-page application, SPA), eli sen tarvitsemat resurssit ladataan yhdellä HTML-sivulla. Tällöin sivun käyttäminen ei aiheuta sen latautumista uudelleen ja sisältöä voidaan ladata ja esittää sivulla dynaamisesti. (Facebook 2020b.)

Create React App (myöhemmin CRA) on Facebookin kehittämä komentorivityökalu, joka helpottaa ja nopeuttaa React-sovelluksen kehitysympäristön käyttöönottoa, sovelluskehitystä, testaamista ja paketointia tuotantokäyttöä varten. CRA sisältää muun muassa JSX-esiprosessorin, tuen uudemmille ECMAScript-versioille sekä reaaliaikaisen kehityspalvelimen, jonka avulla sovellusta voi kehittää lataamatta sivua uudelleen manuaalisesti. (Facebook 2020a.)

3.1.3 Node.js-ajoympäristö

Node.js (myöhemmin Node) on asynkroninen JavaScript-ajoympäristö, jonka avulla JavaScript-koodia voidaan ajaa palvelinsovelluksessa. Node on suunniteltu skaalautuvien verkkosovellusten toteuttamiseen ja se mahdollistaa useiden verkkoyhteyksien käsittelemisen rinnakkaisesti (kuva 2). (OpenJS Foundation 2020.)

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

KUVA 2. Node.js-sovellus (Kuva: OpenJS Foundation 2020)

npm on ohjelmistorekisteri, jota käytetään pääasiassa Node- ja JavaScript-pohjaisten sovellusten paketinhallintaratkaisuna. npm koostuu verkkosivustonsa lisäksi JavaScript-ohjelmistorekisteristä, jonka julkinen tietokanta sisältää rekisterissä julkaistut paketit metatietoineen, sekä komentorivityökalusta, jota käytetään pakettien hallintaan ja asentamiseen kehitysympäristössä. Sovelluksen riippuvuuksien hallintaan käytetään package.json-tiedostoa, joka mahdollistaa riippuvuuksien versiotietojen tallentamisen versionhallintajärjestelmiin. (npm n.d.)

3.2 Python-ohjelmointikieli

Python on tulkattava korkean tason ohjelmointikieli, jonka ominaisuuksia ovat muun muassa dynaaminen tyypitys, olioparadigma ja luokat sekä virheiden käsittelymekanismi. Python on suosittu ohjelmointikieli aloittelevien ohjelmoijien keskuudessa sen selkeän syntaksin vuoksi, minkä lisäksi sitä käytetään laajasti esimerkiksi tilastojen visualisoinnissa sekä koneoppimisratkaisuissa. Pythonia on mahdollista laajentaa C- tai C++-koodilla ja se sisältää ohjelmointirajapintoja käyttöjärjestelmäkutsuille Linux-, macOS- ja Windows-alustoilla. (The Python Software Foundation 2020.)

3.2.1 Flask-ohjelmistokehys

Flask on WSGI-standardin mukainen Python-ohjelmistokehys, jonka avulla voidaan toteuttaa verkkosovelluksia ja -rajapintoja (Armin Ronacher 2019; kuva 3). Flask on mikro-ohjelmistokehys, mikä tarkoittaa, että ohjelmistokehyksen ydin on mahdollisimman yksinkertainen, mutta samalla mahdollisimman laajennettava. Käytännössä tämä antaa kehittäjälle enemmän vapauksia muiden ohjelmistoriippuvuuksien suhteen kuin monoliittisissa ohjelmistokehyksissä. Flask ei siis sisällä esimerkiksi tietokanta-abstraktioita tai verkkolomakevalidointia, mutta tukee näiden integroimista sovellukseen ulkoisten kirjastojen avulla. (Pallets 2019.)

```
from flask import Flask, escape, request

app = Flask(__name__)

@app.route('/')
def hello():
    name = request.args.get("name", "World")
    return f'Hello, {escape(name)}!'
```

KUVA 3. Flask-sovellus (Kuva: Armin Ronacher 2019)

3.2.2 Imageio-kirjasto

Imageio on Python-kirjasto, joka tarjoaa rajapinnan muun muassa kuva- ja videodatan lukemiseen ja kirjoittamiseen. Imageio tukee useita tiedostomuotoja ja on helposti laajennettavissa lisäosien avulla. (Almar Klein 2020.)

3.2.3 NumPy-kirjasto

NumPy on Python-kirjasto, jota käytetään pääasiassa tieteellisen tietojenkäsittelyn sovellusratkaisuissa. NumPy sisältää tuen muun muassa moniulotteisille taulukko-objekteille ja niiden iteroinnille, C- ja C++-integraatioille sekä satunnaislukujen generoimiselle, ja sitä voidaan hyödyntää esimerkiksi moniulotteisen datan tehokkaassa käsittelyssä. (NumPy 2020.)

3.2.4 pip-paketinhallintajärjestelmä

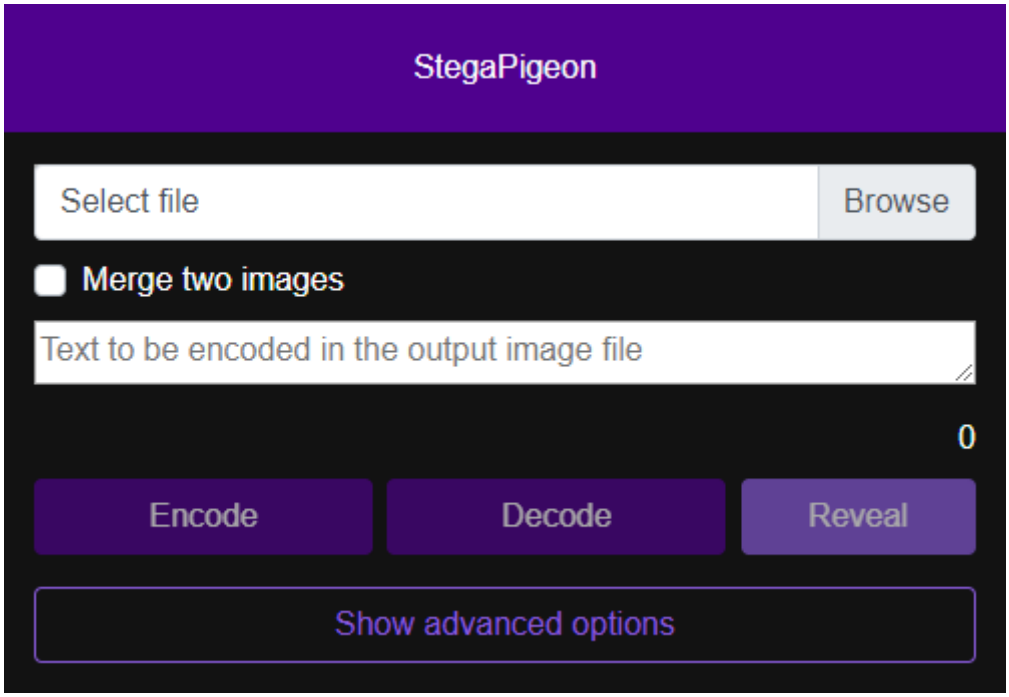
pip on Pythonin paketinhallintajärjestelmä ja komentorivityökalu, jonka avulla voidaan asentaa Python-sovellusten riippuvuuksia esimerkiksi PyPI-repositoriosta (Python Package Index). pip tukee pakettien asentamista tekstitiedoston perusteella, mikä mahdollistaa riippuvuuksien versiotietojen tallentamisen versionhallintajärjestelmiin. (PyPA 2020.)

3.3 Sass-tyylimäärittelykieli

Sass on tyylimäärittelykieli, joka voidaan kääntää CSS-kielelle. Sass tukee kaikkien CSS-versioiden mukaisia CSS-määrittelyjä, minkä lisäksi se sisältää lisäominaisuuksia, esimerkiksi muuttujat, käyttäjän määrittämät funktiot ja listat. Monet tyylikirjastot, kuten Bootstrap, sisältävät Sass-määrittelyjä, mikä mahdollistaa niiden räätälöimisen kehittäjän tarpeisiin. (Sass 2019.)

4 SOVELLUKSEN KÄYTTÖLIITTYMÄ

Sovellus on toteutettu yksinkertaisena verkkokäyttöliittymänä React-käyttöliittymäkirjaston ja Bootstrap-tyylikirjaston avulla (kuva 4). Sovelluksen päätoiminnot ovat tekstin ja kuvatiedoston piilottaminen kuvatiedostoon sekä näiden vastaavat purkamistoiminnot. Käyttöliittymä on todettu toimivaksi Google Chrome- ja Mozilla Firefox -verkkoselaimilla.



KUVA 4. Sovelluksen verkkokäyttöliittymä

4.1 Tekstin piilottaminen

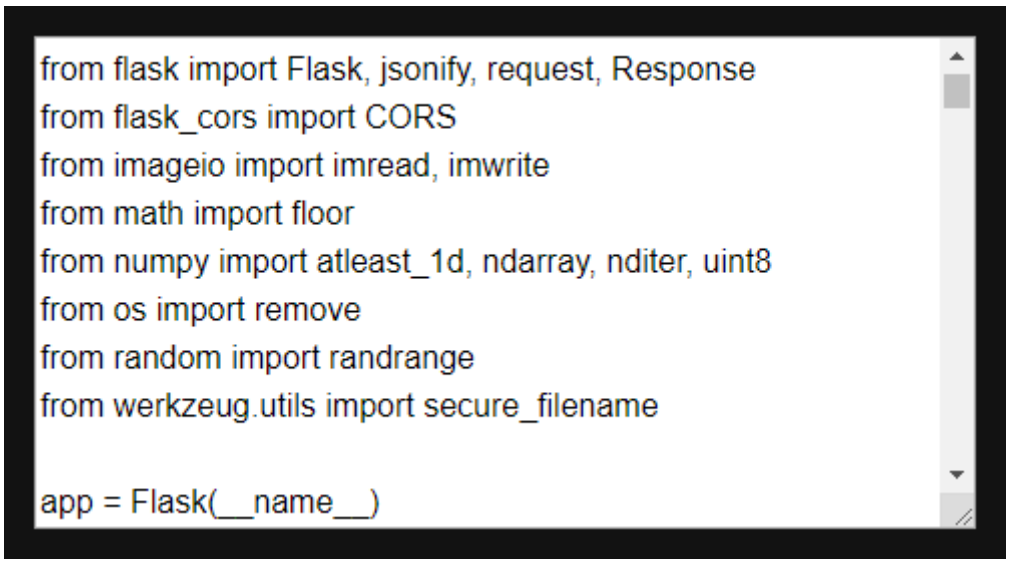
Tekstiä piilotettaessa valitaan kuva ja kirjoitetaan tai liitetään piilotettava teksti käyttöliittymässä näkyvään tekstikenttään. Käytetään esimerkkinä kuvaa, jonka leveys ja korkeus ovat 512 pikseliä (kuva 5), ja tekstinä sovelluksen palvelintoteutuksen lähdekoodia, jonka pituus on noin 10 000 merkkiä (noin 370 koodiriviä). Piilotettava data koodataan (englanniksi encode) oletuksena lopputuloksena tuotetun kuvan pikselien värikanavien yhteen vähiten merkitsevään bittiin (kuva 6). Tässä tapauksessa tuotetun kuvan pikseleistä noin 7 % on muuttunut väriarvoiltaan alkuperäiseen kuvaan verrattuna.

KUVA 5. Alkuperäinen kuva (Kuva: Marco Paulo Prado 2020)

KUVA 6. Kuva, johon on piilotettu tekstiä

4.2 Tekstin paljastaminen

Tekstiä luettaessa kuvasta valitaan kuva käyttöliittymässä ja käytetään purkamistoimintoa (englanniksi decode). Purkamisoperaatiossa on käytettävä samaa määrää vähiten merkitseviä bittejä kuin piilotusoperaatiossa, jotta piilotettu data voidaan lukea kuvasta oikein. Operaation tuotoksena luettu teksti esitetään käyttöliittymässä erillisessä tekstikentässä (kuva 7).



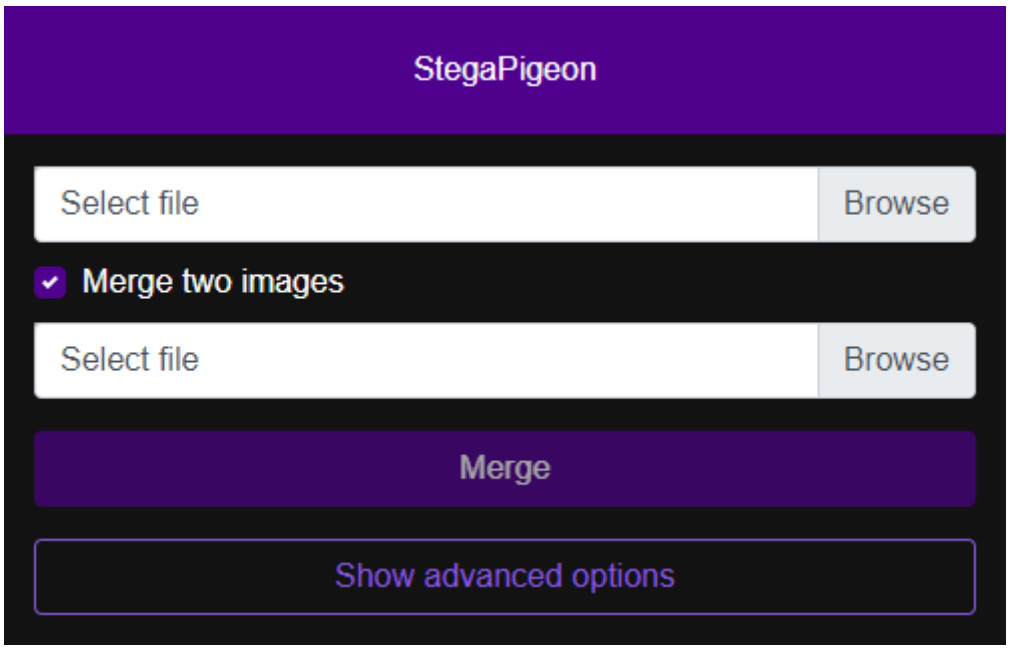
KUVA 7. Kuvasta paljastettu teksti

4.3 Kuvatiedoston piilottaminen

Kuvatiedostoa piilotettaessa toiseen kuvatiedostoon valitaan käyttöliittymässä kaksi kuvaa, joista ensimmäinen on peitekuva ja toinen piilotettava kuva (kuva 8). Käytetään esimerkkinä kahta samankokoista kuvaa, joiden leveys ja korkeus ovat 512 pikseliä (kuva 9; kuva 10).

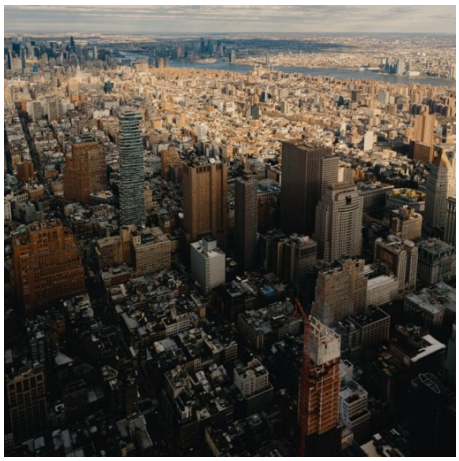
Kuvien yhdistämisoperaatiossa (englanniksi merge) piilotettavan kuvan koodaamiseen käytetään oletuksena neljää vähiten merkitsevää bittiä pikselien värikanavia kohden, jolloin sekä peitekuvan että piilotettavan kuvan väriarvojen koodaamiseen käytetään sama määrä vähiten merkitseviä bittejä. Korkeammilla LSB-määrillä peitekuvan koodaamiseen jää vähemmän käytettäviä bittejä kuin piilotettavan kuvan koodaamiseen, mikä näkyy lopputuloksen näkyvän kuvanlaadun heikkenemisenä.

Lopputuloksen kuvanlaatu (kuva 11) on heikompi kuin alkuperäisessä peitekuvassa ja kuvien ero on huomattavissa paljaalla silmällä, jos lopputulosta voidaan verrata alkuperäiseen kuvaan. Jos piilotettavan kuvan koko olisi esimerkiksi puolet peitekuvan koosta tai piilotettava kuva koodattaisiin harmaasävyisenä, kuva voitaisiin piilottaa käyttämällä pienempää määrää vähiten merkitseviä bittejä pikselien värikanavia kohden.

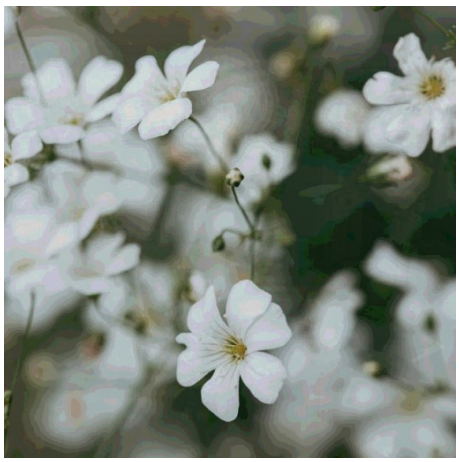


KUVA 8. Kuvatiedoston piilottamisnäkymä

KUVA 9. Peitekuva (Kuva: Katarzyna Korobczuk 2020)



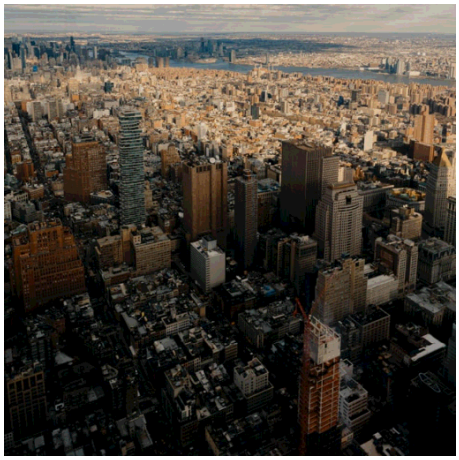
KUVA 10. Piilotettava kuva (Kuva: Trae Gould 2020)



KUVA 11. Kuva, johon on piilotettu toinen kuva

4.4 Kuvatiedoston paljastaminen

Kuvatiedoston paljastaminen (englanniksi reveal) toimii vastaavasti kuin tekstin paljastaminen. Käytetään esimerkkinä edellisessä luvussa tuotettua kuvaa. Mikäli käytettävä määrä vähiten merkitseviä bittejä on asetettu vastaamaan kuvan piilottamisessa käytettyä LSB-asetusta, voidaan peitekuvaan piilotettu kuva lukea oikein (kuva 12).



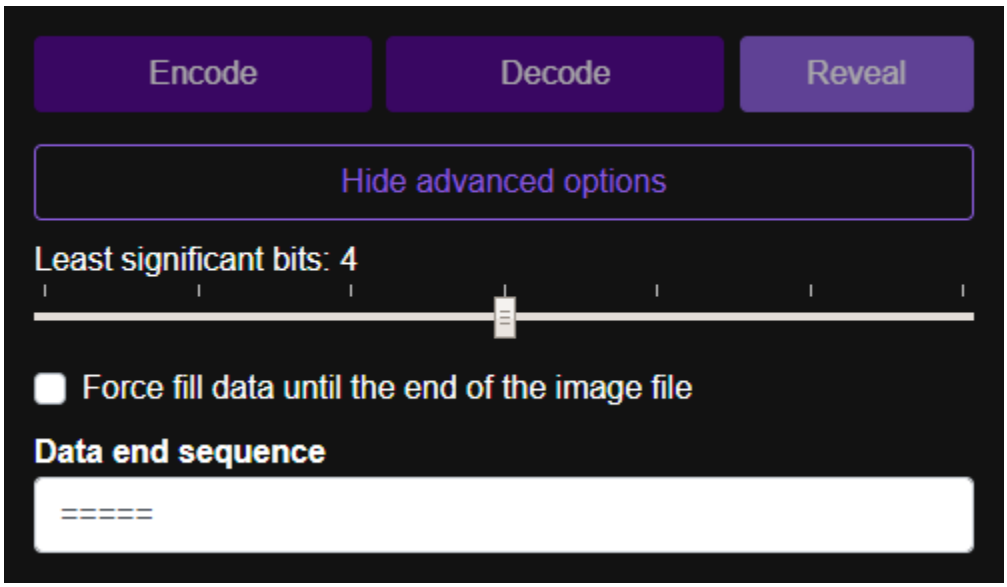
KUVA 12. Paljastettu kuva

4.5 Sovelluksen asetukset

Sovelluksen käyttöliittymässä on mahdollista määrittää muokkausoperaatioissa käytettäviä asetuksia (kuva 13). Näistä tärkein on koodauksessa ja purkamisessa käytettyjen vähiten merkitsevien bittien määrä, joka vaikuttaa lopputuloksen kuvanlaatuun (kuva 14, vasemmalla yläkulmassa alkuperäinen kuva, oikealla alakulmassa käytössä 7 vähiten merkitsevää bittiä pikselien värikanavia kohden).

Toinen asetus mahdollistaa kuvan täyttämisen kokonaan tilanteissa, joissa piilotettavaa dataa ei ole kuvatiedoston loppuun asti. Tämä on hyödyllistä silloin, kun muokatun ja alkuperäisen kuvan välisen eron näkee helposti paljaalla silmällä (kuva 15, kuusi vähiten merkitsevää bittiä pikselien värikanavia kohden).

Kolmas asetus määrittelee merkkijonon, joka lisätään piilotettavan tekstin perään ja jonka mukaan piilotetun datan loppu päätellään purkamisoperaatiossa. Tämä mahdollistaa tekstimuotoisen datan koodaamisen ja purkamisen myös niissä tilanteissa, joissa oletuksena käytetty datan päättymistä ilmaiseva merkkijono sisältyy piilotettavaan dataan itseensä.



KUVA 13. Sovelluksen asetusvalikko

KUVA 14. Vähiten merkitsevien bittien määrän vaikutus kuvanlaatuun (Kuva: Gerson Cifuentes 2020)

KUVA 15. Muokatun ja alkuperäisen kuvan raja (Kuva: Wells Chan 2020)

5 SOVELLUKSEN PALVELINTOTEUTUS

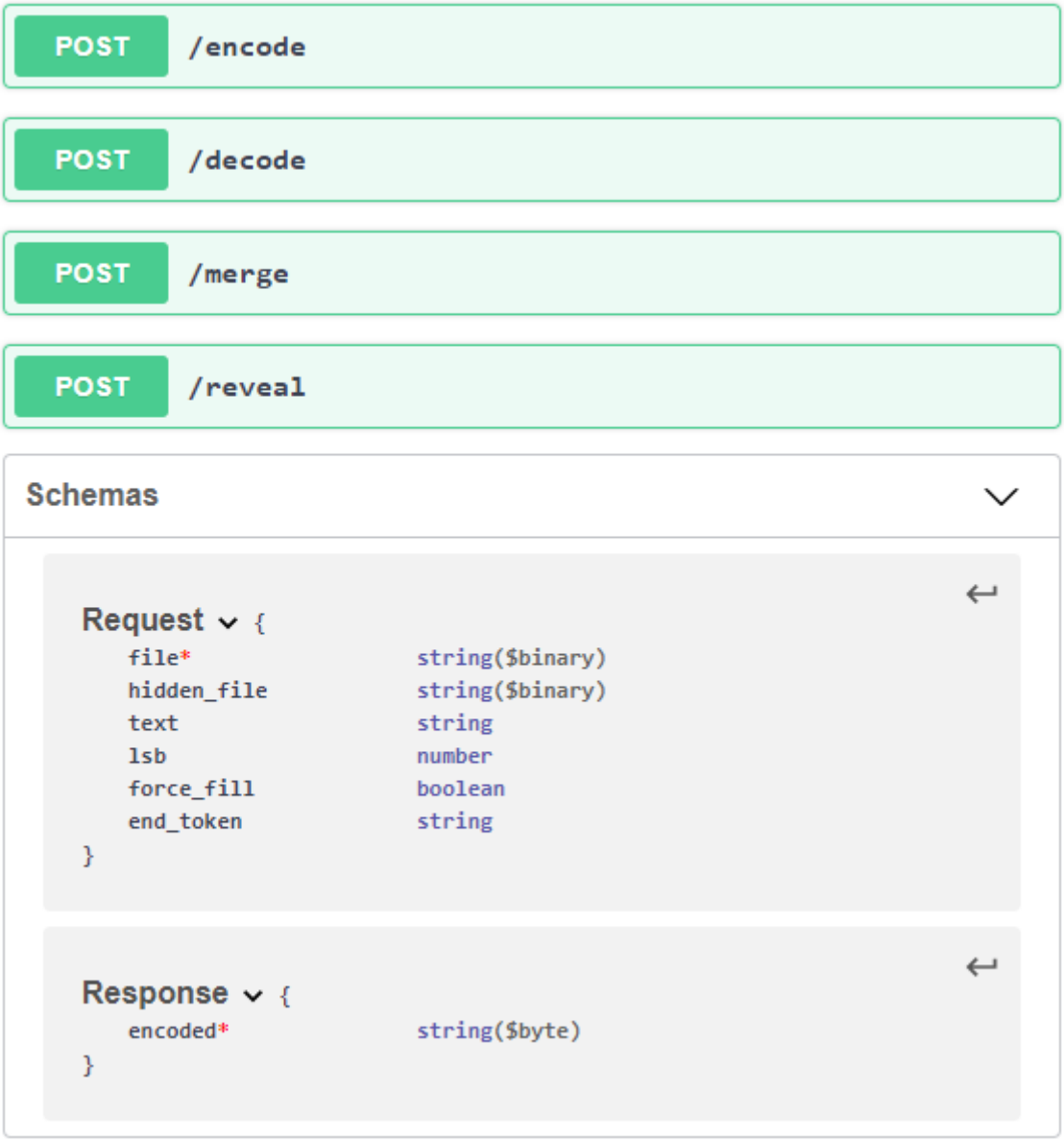
Sovelluksen palvelintoteutus on toteutettu Python-pohjaisella Flask-ohjelmistokehyksellä, joka mahdollistaa yksinkertaisen REST-rajapinnan luomisen sovelluksen tarjoamille toiminnoille. Palvelinsovelluksen ajoympäristönä on käytetty kehityksen aikana Windows-käyttöjärjestelmää, mutta Pythonin alustariippumattomuus mahdollistaisi palvelimen ajamisen myös esimerkiksi Linux- tai macOS-käyttöjärjestelmissä.

5.1 REST-rajapinta

Palvelinsovelluksen REST-rajapinta koostuu neljästä osoitepäätteestä (englanniksi endpoint), jotka on kuvattu erillisessä OpenAPI-määrittelyssä (kuvio 3). Rajapinta sisältää vain POST-tyyppisiä päätteitä, sillä palvelinsovellus ei tallenna vastaanottamaansa dataa tietokantaan, vaan poistaa kaikki HTTP-pyyntöihin liittyvät tiedostot niiden käsittelyn jälkeen.

Palvelin vastaanottaa HTTP-pyynnöt lomakemuotoisena datana (multipart/form-data), mikä mahdollistaa usean datatyypin lähettämisen samassa pyynnössä. Lähetettävien kuva- ja tekstitiedostojen lisäksi palvelimelle voidaan siis lähettää myös käyttöliittymässä valitut parametrit, kuten vähiten merkitsevien bittien määrä tai kuvan datalla täyttämiseen liittyvä valinta.

Palvelimen suorittaman toiminnon jälkeen lopputuloksena tuotettu binäärimuotoinen kuvatiedosto koodataan Base64-muotoiseksi merkkijonoksi, mikä mahdollistaa kuvatiedoston välittämisen HTTP-pyynnön vastauksena ilman kuvan tallentamista palvelimelle tai tietokantaan. Pyynnön jälkeen kuvatiedosto voidaan siis poistaa palvelimelta välittömästi, mikä säästää palvelimen resursseja sekä levytilaa.



KUVIO 3. OpenAPI-määrittely

Flask-sovelluksessa osoitepäätteet ja sallitut HTTP-metodit määritellään Pythonin decorator-syntaksilla, mikä mahdollistaa reittien toteuttamisen yksinkertaisesti Python-funktioiden avulla (kuva 16). HTTP-pyyntöihin liittyvää dataa voidaan käsitellä Flaskin globaalin request-objektin kautta ja pyynnön vastaus palautetaan reittifunktion paluuarvona. Virhetilanteissa palvelimelta palautetaan virheviesti, joka voidaan käsitellä ja esittää käyttöliittymässä.

```
@app.route('/decode', methods=['POST'])
def decode_route():
    """
    POST decode
    """
    try:
        decoded = decode(image=upload(request)[0],
            lsb=get_lsb(request), end_token=get_end_token(request))
    except TypeError as e:
        print(repr(e))
        return Response(str(e), status=400)
    except ValueError as e:
        print(repr(e))
        return Response(str(e), status=400)

    return jsonify({ 'decoded': decoded })
```

KUVA 16. Flask-reititys

5.2 Datan koodaaminen kuvatiedostoon

Kuvatiedostot luetaan palvelimella Imageio-kirjaston avulla, mikä mahdollistaa kuvien iteroimisen NumPy-kirjaston moniulotteisella iteraattorilla (kuva 17). Pikseleitä käsitellään kolmialkioisina listoina, joiden alkiot vastaavat pikseleiden värikanavien numeroarvoja. Jokaisen värikanavan numeroarvosta korvataan 1–7 vähiten merkitsevää bittiä riippuen HTTP-pyynnössä lähetetystä LSB-parametristä.

Koodattavaa binääridataa seurataan dataindeksillä, joka vastaa seuraavan koodattavan bitin järjestysnumeroa. Indeksiin lisätään siis vähiten merkitsevien bittien määrä jokaisen värikanavan käsittelyn jälkeen. Data koodataan suoraan alkuperäisen kuvan päälle, joten iterointi voidaan lopettaa, kun kaikki data on koodattu kuvatiedostoon. Jos kuva halutaan täyttää datalla ja koodattava data loppuu ennen alkuperäistä kuvaa, dataindeksi asetetaan koodattavan datan alkuun ja data koodataan kuvaan niin monta kertaa, että kuva saadaan täytettyä. Tästä ei ole haittaa koodauksen purkamisessa, sillä purkaminen lopetetaan jo ensimmäisen datan loppumista merkitsevän merkkijonon kohdalla.

```
110     for i in nditer(image):
111         pixel.append(i)
112 
113         if (len(pixel) == image.shape[2]):
114             for n in range(3):
115                 offset = data_index + lsb
116                 clamped = clamp(offset, 0, data_len - 1)
117 
118                 pixel[n] = int(to_bin(pixel[n])[:-lsb]
119                     + binary_data[data_index : clamped], 2)
120 
121                 data_index += lsb
122 
123                 if offset != clamped:
124                     done = True
125 
126                     if force_fill:
127                         data_index = 0
128 
129             image[y][x] = pixel
130             pixel = []
131             x += 1
132 
133             if (x == image.shape[1]):
134                 x = 0
135                 y += 1
136 
137             if done and not force_fill:
138                 break
139 
140     imwrite(filename, image.astype(uint8))
```

KUVA 17. Datan koodaaminen

5.3 Koodauksen purkaminen

Koodausta purkaessa kuvatiedosto iteroidaan vastaavasti kuin datan koodausvaiheessa (kuva 18). Data luetaan pikseleiden värikanavista vähiten merkitsevien bittien määrän perusteella ja säilötään binääridatamuuttujaan. Tekstimuotoisen datan tapauksessa binääridata jaetaan tavuihin, eli kahdeksan bitin kokoisiin lohkoihin, joista jokainen muutetaan tavua vastaavaksi merkiksi. Kuvatiedostoa paljastaessa vaihdetaan vähiten ja eniten merkitsevien bittien paikat keskenään käyttämällä bittien peittämistä (englanniksi bitmask) ja siirtämistä (englanniksi bit shift).

```
163     for i in nditer(image):
164         pixel.append(i)
165 
166         if (len(pixel) == image.shape[2]):
167             for n in range(3):
168                 binary_data += to_bin(pixel[n])[-lsb:]
169 
170             pixel = []
171             x += 1
172 
173             if (x == image.shape[1]):
174                 x = 0
175                 y += 1
176 
177     all_bytes = [binary_data[i: i + 8]
178         for i in range(0, len(binary_data), 8)]
179 
180     decoded_data = ''
181 
182     for byte in all_bytes:
183         if len(byte) < 8:
184             break
185 
186         decoded_data += chr(int(byte, 2))
187 
188         if decoded_data[-len(end_token):] == end_token:
189             decoded_data = decoded_data[:-len(end_token)]
190             break
191 
192     return decoded_data
```

KUVA 18. Koodauksen purkaminen

5.4 Rajoitukset

Palvelin tukee ainoastaan PNG-tyyppisiä kuvatiedostoja, sillä edellä mainitut tekniikat eivät toimi pakatuille kuvatiedostotyypeille, eikä palvelinsovellukseen ole toteutettu kuvatiedostojen tiedostotyypin muuntamistoimintoa. Lisäksi yksittäisen kuvatiedoston koko on rajoitettu kymmeneen megatavuun, jotta verkon yli ei siirrettäisi liikaa dataa kerralla, eikä kuvatiedostojen iteroimiseen ja käsittelemiseen kuluisi liikaa aikaa.

6 POHDINTA

Opinnäytetyön tuloksena on yksinkertainen ja toimiva selainkäyttöliittymä sekä palvelintoteutus, jonka REST-rajapintaa voidaan hyödyntää myös käyttöliittymästä irrallisena komponenttina. Sovelluksen avulla tuotetuista kuvista nähdään myös, miten toiminnoissa käytettyjen merkitsevien bittien määrä vaikuttaa lopputuloksen kuvanlaatuun; esimerkiksi yhtä tai kahta vähiten merkitsevää bittiä käytettäessä lopputulosta on vaikea erottaa alkuperäisestä kuvasta paljaalla silmällä.

Toteutettua sovellusta voidaan käyttää esimerkkinä kryptografian ja steganografian mahdollisuuksista. Työssä käytetyt menetelmät ovat kuitenkin yksinkertaisia, joten esimerkiksi steganografisten kuvien analysointityökaluilta suojautumiseen ei ole otettu kantaa. Kryptoanalyysi on salakirjoitustekniikoiden murtamiseen keskittynyt tieteenala, joka tutkii esimerkiksi salausalgoritmien luotettavuutta. Steganalyysillä tarkoitetaan steganografisten kuvien ja muiden viestiformaattien, joihin voidaan piilottaa dataa, analysointia.

Työssä käytetyt menetelmät toimivat vain kuvaformaateille, jotka on pakattu häviöttömästi. Huomiota on kuitenkin kiinnitetty sovelluksen laajennettavuuteen, joka on mahdollista toteuttaa esimerkiksi parantamalla muokkaustoiminnoissa käytettyjä algoritmeja. Esimerkiksi JPEG-kuvissa voitaisiin hyödyntää diskreettiä kosinimuunnosta (DCT), jonka myötä piilotettua dataa olisi huomattavasti vaikeampi havaita analysointityökaluilla verrattuna tässä työssä käytettyyn LSB-tekniikkaan.

Toteutettavassa sovelluksessa ei ole mahdollista hyödyntää salausalgoritmeja piilotettavan datan koodaus- ja purkamisvaiheessa, ellei sovelluksen käyttäjä salaa dataa erillään sovelluksesta. Datan salaaminen ei lisää suojaa analysointityökaluilta, mutta se suojaa itse dataa tilanteissa, joissa ulkopuolinen taho saa piilotettavan datan haltuunsa. Steganografia perustuu kuitenkin usein siihen, että viestintäväline, johon viesti on piilotettu, on saatavilla, mutta löytääkseen piilotetun viestin täytyy tietää, että viesti on ylipäätään olemassa.

LÄHTEET

Almar Klein. 2020. imageio - Python library for reading and writing image data. Luettu 21.3.2020. https://imageio.github.io/

Armin Ronacher. 2019. Flask. Päivitetty 8.7.2019. Luettu 21.3.2020. https://palletsprojects.com/p/flask/

Computerphile. 2015. Secrets Hidden in Images (Steganography). YouTube-video. Julkaistu 4.8.2015. Viitattu 28.2.2020. https://www.youtube.com/watch?v=TWEXCYQKyDc

Ecma International. 2019. ECMAScript® 2019 Language Specification. ECMA-262, 10th edition, June 2019. Luettu 20.3.2020. https://www.ecma-international.org/ecma-262/10.0

Ecma TC39. n.d. The TC39 Process. Luettu 20.3.2020. https://tc39.es/process-document/

Facebook. 2020a. Create React App. Luettu 20.3.2020. https://create-react-app.dev/

Facebook. 2020b. Glossary of React Terms. Luettu 20.3.2020. https://reactjs.org/docs/glossary.html

Facebook. 2020c. Introducing JSX. Luettu 20.3.2020. https://reactjs.org/docs/introducing-jsx.html

Facebook. 2020d. React. Luettu 20.3.2020. https://reactjs.org/

Farooque, M. A., Rohankar, J. S. 2013. Survey on various noises and techniques for denoising the color image. International Journal of Application or Innovation in Engineering & Management 2 (11), 217.

Kuhn, M. 2009. UTF-8 and Unicode FAQ. Päivitetty 11.5.2009. Luettu 29.2.2020. https://www.cl.cam.ac.uk/~mgk25/unicode.html

Merriam-Webster. n.d. Steganography. Luettu 22.2.2020. https://www.merriam-webster.com/dictionary/steganography

Mozilla Web Docs. 2020a. About JavaScript. Päivitetty 4.3.2020. Luettu 14.3.2020. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

Mozilla Web Docs. 2020b. JavaScript language resources. Päivitetty 4.3.2020. Luettu 20.3.2020. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources

Mstafa, R. J., Bach, C. 2013. Information Hiding in Images Using Steganography Techniques. 2013 ASEE Northeast Section Conference. Norwich University.

NumPy. 2020. NumPy. Luettu 21.3.2020. https://numpy.org/

npm. n.d. About npm. Luettu 21.3.2020. https://docs.npmjs.com/about-npm/

OpenJS Foundation. About Node.js®. Päivitetty 12.1.2020. Luettu 20.3.2020. https://nodejs.org/en/about/

Pallets. 2019. Foreword. Päivitetty 6.5.2019. Luettu 21.3.2020. https://flask.palletsprojects.com/en/1.1.x/foreword/

PyPA. Pip – The Python Package Installer. 2020. Päivitetty 19.2.2020 Luettu 21.3.2020. https://pip.pypa.io/en/stable/

The Python Software Foundation. 2020. General Python FAQ. Päivitetty 21.3.2020. Luettu 20.3.2020. https://docs.python.org/3/faq/general.html

Sass. 2019. Documentation. Päivitetty 5.9.2019. Luettu 21.3.2020. https://sass-lang.com/documentation

Solak, S., Altinişik, U. 2018. The Least Significant Two-bit Substitution Algorithm for Image Steganography. International Journal of Computer (IJC) 31 (1), 150–156.

UTF-8. 2019. UTF-8 and Unicode Standards. Päivitetty 7.2.2019. Luettu 29.2.2020. http://www.utf-8.com

V8. 2020. V8 JavaScript engine. Päivitetty 2.3.2020. Luettu 20.3.2020. https://v8.dev/

Wayner, P. 2009. Disappearing cryptography: Information hiding: Steganography & watermarking. 3. painos. Burlington, Massachusetts: Morgan Kaufmann Publishers.