TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Saamuel Starke 160492IAIB

Integreeritav ja kohandatava disainiga kaardimakse lahendus

Bakalaureusetöö

Juhendaja: Tarvo Treier
MSc

Tallinn 2022

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Saamuel Starke

01.05.2022

Annotatsioon

Käesoleva töö eesmärgiks oli integreeritava kaardimakse lahenduse loomine, mille väljanägemist saab muuta.

Töös kirjeldatakse kaardimaksete töötlemise protsessi ning selleks vajalikke nõudeid ja kohustusi. Autor analüüsis erinevaid meetodeid, millega on võimalik saavutada kohandatav stiliseerimine kasutajaliidesele. Töö tulemusena valmis kaks rakendust, mille koostöös on võimalik pakkuda *Payment Card Industry Data Security Standard*'i nõuetele vastavat kaardimakse lahenduse kasutajakogemust.

Lõputöö tulemus on kliendirakendus, mille saab võtta aluseks tulevateks lisaarendusteks. Maksevormi tervikliku lahenduse pakkumiseks on vaja integreerida API päringud, lisada kaardiandmete kliendipoolne krüpteerimine ja 3DS kontrolli teostamine.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 6 peatükki, 8 joonist.

Abstract

Embedded Card Payment Solution with Modifiable Design

The goal of this thesis is to build an embedded card payment experience, which look and feel can be modified.

In this thesis, the author gives a brief overview of card payment processing, its requirements, and obligations. Different methods are being analysed, which could be used to accomplish customisable user experience by the integrator. The realisation of this thesis is based on two frontend applications, both are used in collaboration to offer a Payment Card Industry Data Security Standard compliant card payment experience.

The end product could be further expanded by making use of payment APIs, Client-Side Encryption, and 3D Secure support to offer a fully-fledged card payment experience to merchants.

The thesis is in Estonian and contains 28 pages of text, 6 chapters, 8 figures.

Lühendite ja mõistete sõnastik

3DS	*Three Dimensional Secure,* turvaliste internetiostude programm
AES	*Advanced Encryption Standard*, täiustatud krüpteerimisstandard
API	*Application Programming Interface*, rakendusliides
BEM	*Block Element Modifier,* blokk element modifikaator nimetamise meetod
BIN	*Bank Identification Number,* panga identifitseerimisnumber
CDE	*Card Data Environment,* kaardiandmete keskkond
CDN	*Content Delivery* Network, sisutarne võrk
CSE	*Client Side Encryption,* kliendipoolne krüpteerimine
CSS	*Cascading Style Sheets,* veebilehe kujundamiseks kasutatav märgistuskeel
CVC	*Card Verification Code*, kaardi turvakood
EMVCo	*Europay, Mastercard, and Visa Companies,* kaardiskeemide poolt loodud standard
ES6	*ECMAScript 6,* JavaScripti standard
global namespace	Globaalne nimeväli
HTML	*HyperText Markup Language,* hüperteksti märgistuskeel
iFrame	HTML teek
JS	*JavaScript,* JavaScript programmeerimiskeel
Library	Kollektsioon ressurssidest, mida arvutiprogrammid kasutavad
NPM	*Node Package Manager,* Node,js paketi haldur
PAN	*Permanent Account Number,* peamine kontonumber
PCI DSS	*Payment Card Industry Data Security Standard*, kaardimakse tööstusharu andmeturvalisuse standard
PSD2	*Revised Payment Services Directive,* muudetud makseteenuste direktiiv
RSA	*Rivest–Shamir–Adleman,* avaliku võtmesüsteemiga krüpteerimise algoritm
SCSS	*CSS preprocessor,* CSS-i genereerimiseks mõeldud süntaktiline programm

SDK	*Software Development Kit,* tarkvaraarenduskomplekt
TLS	*Transport Layer Security, t*ranspordikihi turbeprotokoll
URL	*Uniform Resource Locator*, üldine infoallika asukohamääraja

Sisukord

Jooniste loetelu

1 Sissejuhatus

Veebimaksete teostamiseks on palju erinevaid viise, ülemaailmselt enim populariseeritud on krediit- ja deebetkaardi maksed. Internetist maksete vastuvõtmiseks kasutavad kaupmehed üldjuhul maksevahendajaid (*payment facilitator*). Maksevahendamise mudel arenes välja maksetele spetsialiseerunud ettevõtete poolt, et vähendada keerukust internetimaksete kasutusele võtmisel ja pakkuda maksete teostamise teenust neile, kellele see varem kättesaadav ei olnud. [1] Barjäär makseid vastu võtta on langenud ning üha enam e-poode ja ärisid saab endale lubada ülemaailmset maksete teostamist, mis aitab ehitada globaalseid ärisid ning ettevõtted saavad keskenduda oma põhitegevusaladele. Maksevahendajad hoolitsevad paljude vajalike protsesside eest maksete läbiviimisel: kaupmeeste registreerimine maksete teostajatena ja sellega seonduvad protsessid, makseliidestused, tagasimaksed, pettuste tuvastamine ja raporteerimine, suhtlemine kaardi skeemidega, vastavus regulatsioonidele erinevates regioonides, maksud ja paljud teised ülesanded. Kogu see keerukus on varjatud makse küsija ja maksja jaoks. Liidestuse teostamiseks pakuvad maksevahendajad kaupmeestele erinevaid tööriistu. Selleks võib olla näiteks JavaScripti *library* ja mobiili SDK.

1.1 Ülesande püstitus

Autor teostab uurimust ettevõttele, kes tegeleb maksete vahendamisega, kuid on praeguseni seda pakkunud kasutades ettevõtte enda süsteemides majutatud lahendusi. Vaadeldaval ettevõttel ja selle hallataval maksesüsteemil on potentsiaalne huvi laiendada maksete teostamist otse läbi kaupmeeste süsteemide. Seega on vaja makselahendust, mida on võimalik paigaldada kaupmeeste süsteemidesse.

Töö eesmärgiks on luua tehniline lahendus kaupmeeste süsteemidesse integreeritavale kaardimakse vormile, mis vastaks PCI DSS nõuetele ja oleks kohandatava disainiga. Töö eesmärkide saavutamiseks tuleb täita järgnevaid nõudeid ja ülesandeid:

- Ettevõtted, kes hoiustavad ja käitlevad krediit- ja deebetkaardi andmeid peavad vastama nõuetele, mis on määratud *Payment Card Industry Data*

Security Standard (PCI DSS) poolt, loodav lahendus peab neid nõudeid täitma.

- Analüüsida ja arendada meetod, mis võimaldaks kolmandal osapoolel integreeritava liidese disaini muuta.
- Arendada kaardiandmete kogumiseks ja valideerimiseks vajalikud sisendväljad
- Makselahendus on integreeritav valmislahendus ehk karbitoode, mille protsess järgib mustrit: paigalda, seadista, kasuta.

Oodatav tulemus on JavaScripti *library*, mida on võimalik integreerida kaupmeeste süsteemis majutatud veebilehtedele. Töö peamine osa keskendub kaardimaksetele, kuid põhiline kontseptsioon kehtib ka teiste maksemeetodite puhul. Oluline on siinkohal märkida, et vaadeldaval maksesüsteemil puuduvad töö kirjutamise hetkel API-*d,* mida realiseeritav rakendus saaks kasutada.

1.2 Ülesehitus

Bakalaureusetöö teises peatükis antakse ülevaade kaardimaksete teostamise taustast, protsessidest ja vajalikest nõuetest.

Kolmandas peatükis analüüsitakse erinevaid meetodeid, millega on võimalik muuta makseliidese väljanägemist vastavalt kaupmehe vajadustele. Analüüsi tulemina valitakse meetod, mille kriteeriumideks on eeldatava arenduse kogumaht ja integreerimise kasutajakogemus.

Neljandas peatükis käsitletakse lahenduse realiseerimist. Antakse ülevaade kasutatud tehnoloogiatest, kahest valminud rakendusest ning tehtud otsustest.

Viiendas peatükis analüüsitakse tehtud töö eesmärkide täitmist, loodud tulemuse positiivseid ja negatiivseid külgi ning tuuakse välja võimalikud edasiarendused.

Viimases peatükis antakse ülevaade tehtud tööst.

2 Ülevaade kaardimaksete teostamisest

Antud peatükis kirjeldatakse kaardimaksete teostamiseks vajalikku taustinfot, nõudeid ja maksetöötlemise komponentide ja institutsioonide vahelist koostööd.

2.1 Peamine kontonumber

Peamine kontonumber ehk PAN viitab 14 - 16 või isegi kuni 19 numbrilisele unikaalsele identifikaatorile, mida tuntakse ka kaardinumbrina. Enamasti on PAN numbrid trükitud maksekaardi esiküljele. Selle numbriga on võimalik viidata kindlale kontole, et teada saada rohkem informatsiooni selle omaniku kohta: nimi, kontojääk, limiidid. Kuid seda mitte alati, näiteks deebetkaartide puhul ei viita see number mingile kindlale seotud kontole. [2]

Tavaliselt genereeritakse PAN numbrid konto või kaardi loomisel. PAN numbrid on kasutusel ka ettemakstud-, kinke- või kliendikaartidel.

2.2 Panga identifitseerimisnumber

BIN ehk panga identifitseerimisnumber on PAN numbri esimese 6 või 8 numbri pikkune identifikaator. Ajalooliselt on olnud BIN numbri pikkuseks 6 ühikut, kuid kuna üha enam finantsasutusi soovivad anda välja oma maksekaarte, siis lõpuks tekib vältimatu olukord – numbrid saavad otsa. Selleks pikendati BIN numbrit 8 numbrini.

BIN-i esimest numbrit tuntakse ka *Major Industry Identifier*, mille järgi on võimalik määrata kaardi väljaandja ettevõte ehk skeem. Lihtsustatult American Express kaardid algavad numbriga 3, Visa kaardid numbriga 4 ja Mastercard poolt väljaantud kaardid algavad numbriga 5 [3].

Kogu BIN numbrist on võimalik välja lugeda kaardi väljastanud finantsasutuse nimi, aadress, riik, valuuta, kaardi tüüp (krediit/deebet) ja kaardi omanik (eraisik/ettevõtte) ning veel teisigi parameetreid.

2.3 Kaardi skeemid

Kaardi skeem on keskne maksevõrk, mis kasutab maksekaarte, et töödelda makseid. Skeemide peamiseks ülesandeks on maksete haldus, kogu arvelduprotsess kuniks maksed on lõpule viidud. Maksete teostamiseks järgivad skeemid kindlaid reegleid ja protseduure. [4] Skeemideks on erinevad ettevõtted nagu Visa, Mastercard, UnionPay, American Express ja teised väiksemad kohalikud skeemid. Kuigi erinevaid skeeme on mitmeid võib üldjoontes täheldada, et sellel turul valitseb tugevalt Visa ja Mastercardi duopol.

Skeemidega teevad lähedast koostööd kaartide väljastajad, kelleks on erinevad finantsasutused ja pangad.

2.4 Makse liikumine

Kaardimakse teostamisel on neli osapoolt:

- Kaardiomanik – Ostja, kelle kaart on väljastatud panga või finantsasutuse poolt.
- Kaardiväljastaja – Väljastab skeemide krediit- ja deebetkaarte eraisikust ja ettevõttetest klientidele. Väljastajateks on pangad või muud kvalifitseeritud finantsasutused.
- Kaupmees – Jaemüüja või ettevõtte, kes osutab teenuseid või müüb kaupa kaardiomanikule.
- Makse vastuvõtja – Institutsioon, mis pakub kaupmehele lepingulist teenust, et töödelda makseid kaupmehe nimel. Vastutab selle eest, et kaardiomaniku kontolt krediteeritud raha jõuaks kaupmehe kontole.

Kuigi makse vahendamisel on neli peamist osapoolt, siis tegelikult hõlmab kogu protsess palju teisi vahendajaid, kes tagavad mingi kindla osa selles protsessis. [4] Kuna kaardimakse elutsükkel hõlmab väga paljusid osapooli ning kõik soovivad teenida tehtud töö eest kasumit, siis on kaardimaksetel üsnagi kõrged vahendustasud, mille üldjuhul maksab kinni kaupmees.

2.5 PCI DSS

Et salvestada ja käidelda kaardi andmeid, on vaja teha seda turvaliselt. Selleks on loodud PCI DSS standard ja seda haldav organisatsioon *Payment Card Industry Security Standards Council,* kelle ülesandeks on reguleerida kaardiandmete käitlemist ja jälgida nende regulatsioonide täitmist. Seda kõike selleks, et vältida kaardiandmetega hooletut ümberkäimist, mis võib tuua kaasa lekkeid ja pettusi. PCI DSS sätestab palju erinevaid nõudeid, mida on vaja järgida. Need hõlmavad nii tehnoloogiat kui ka ettevõtte füüsilisi protseduure. Ettevõtted, kes soovivad maksekaardi andmeid töödelda ja salvestada peavad taotlema PCI DSS sertifikaati. Selle saamiseks tuleb süsteemid ja protseduurid viia vastavusse PCI DSS nõuetega. Sertifikaadi omamisel tuleb lisaks ka iga aasta läbida audit, kus kontrollitakse süsteemide ja protsesside tegelikku vastavust nõuetele. [5]

2.6 Card Data Environment (CDE)

Ettevõtted, kes on saanud PCI DSS sertifikaadi võivad töödelda kaardiandmeid. Seda tehakse spetsiaalselt loodud kaardiandmete keskkonnas (CDE). Tehnoloogilises mõistes on selleks loodud eraldi süsteem, mis on lahus ettevõtte teistest süsteemidest. Selle keskkonna serverid jooksevad eraldi riistvara peal, täites rangeid füüsilisi ja virtuaalseid turvanõuded. Kogu CDE võrk on isoleeritud ja kaitstud tulemüüriga. On tähtis, et kaardiandmed ei lekiks sellest süsteemist väljapoole. CDE sisene tehniline disain ja lahendus täitmaks ettemääratud nõudeid on suuresti iga ettevõtte enda otsustada. [5]

Maksetöötlemise süsteemi kaardiandmete keskkond, millele autor rakendust loob, järgib lihtsaid printsiipe. Kõik selles keskkonnas toimuvad tegevused peavad käitlema kaardiandmeid, kõik üleliigne, milleks ei ole kaardiandmete lugemine, toimub väljaspool seda keskkonda. Selline mõtteviis aitab hoida süsteemi puhtana, mis omakorda aitab vältida süsteemi kasvamisega tekkivaid vigu, mis võiksid ohustada andmete turvalist hoidmist.

2.7 Andmete krüpteerimine

PCI DSS määrab ära, et andmete turvaliseks hoiustamiseks ja transportimiseks peavad kaardiandmed olema krüpteeritud. CDE sees hoiustakse andmeid krüpteerituna. Vaadeldavas maksesüsteemis dekrüpteeritud andmeid loeb ainult arvuti. Inimese silmad

ei näe kunagi kaardinumbreid ühestki vaadeldava ettevõtte süsteemi kasutajaliidesest ega andmebaasist. Väljaspool kaardiandmete keskkonda andmeid salvestada ja hoiustada ei tohi ning seda ka krüpteeritud kujul.

2.7.1 Andmete krüpteerimine andmebaasis

CDE andmebaasides hoiustatakse andmeid krüpteerituna. Andmeid dekrüpteeritakse ainult siis, kui selleks on vajadus. Dekrüpteerimata kujul ei salvestata andmeid kunagi. PCI DSS määrab ära, et krüpteerimiseks tuleb kasutada tugevaid krüptograafilisi algoritme [5]. Näiteks AES – 128 bitti või kõrgem, RSA 2048 bitti või kõrgem.

Krüptovõtmeid vahetatakse regulaarselt ja see käib *dual control* põhimõttel. See tähendab, et üksikisik ei saa kunagi võtmeid vaadata ega vahetada. Selleks on vajalik kahe või enama isiku koostöö.

2.7.2 Andmete krüpteerimine transiidis

Vaadeldava maksesüsteemi kogu andmeedastus, nii CDE sees kui ka üle avaliku interneti, toimub kasutades TLS (*Transport Layer Security*) versiooni 1.2 või 1.3. Selleks kasutatakse Cloudflare teenuseid.

Lisaks on võimalik andmeid transpordiks krüpteerida kasutades *Client Side Encryption'it* (CSE). See tähendab, et enne transporti krüpteeritakse saadetavad andmed avaliku võtmega. Kui andmed jõuavad sihtkohta, ehk CDE *backend* teenustesse, siis seal dekrüpteeritakse saabunud andmed privaatvõtmega, ning seejärel uuesti krüpteeritakse andmed talletamiseks kasutades meetodeid mainitud punktis 2.7.1.

Kui kaardiandmed liiguvad ainult vaadeldava süsteemi sees, siis TLS ja CSE kombineeritult kasutamine pole otseselt vajalik, sisuliselt toimub andmete topelt krüpteerimine, mis lisab küll ekstra turvakihi, kuid CSE kasutamisega kaasneb ka palju tehnilist kompleksust. Kliendipoolne krüpteerimine tuleks implementeerida kliendirakenduses ja dekrüpteerimine *backend* teenustes. Aja jooksul on vaja vahetada võtmeid ning uuendada kasutatud krüptograafia standardeid.

2.8 Autoriseerimine

Kaardimakse töötlemiseks esmane samm on autoriseerimine. Kaupmees saadab päringu makse vastuvõtjale, kes omakorda saadab päringu edasi kaardi väljastajale. Kaardi

väljastaja kontrollib, kas kontol on piisavalt raha, et tehingut lubada. Kui on võimalik tehingut teostada, siis vormistatakse broneering maksja kontolt, mis tähendab, et maksja kontojääk väheneb. [6] Sel hetkel veel raha päriselt omanikku ei vaheta. Edukas autoriseerimine on lubadus, et raha on kontol olemas, see on broneeritud ja jõuab kõigi eelduste kohaselt kaupmehe kontole. Raha liikumine toimub *capture* faasis, mis on raha maksja kontolt kaupmehe kontole liikumise õiguslikult siduv protsess [7].

Autoriseerimine võib ka ebaõnnestuda, enamasti kas finantsilisel või tehnilisel põhjusel. Ebaõnnestumisel saadab makse vastuvõtja üldjuhul ka ebaõnnestumise koodi või sõnumi. Ühed enim levinumad põhjused, miks autoriseerimine ebaõnnestub on ebapiisavad vahendid, valed, aegunud kaardiandmed või pettuse kahtlustus [8].

2.9 Turvaliste internetiostude programm

Kaardiandmete lekkimine või füüsiliste kaartide kadumine, ning nende andmete väärkasutus pettusteks, on suur probleem. Turvaliste internetiostude programm tuntud ka *Three Dimensional Secure* (3DS) on turvalisuse protokoll, mis loodi, et kaitsta kaardiomanikke, kaupmehi, kaardi väljastajaid ja makse töötlejaid pettuste eest [9]. Seda saavutatakse toetudes rohkematele teguritele kui ainult kaardiandmete teadmine. 3DS on osa autoriseerimise protsessist. Ilma eduka 3DS kontrolli läbimiseta ei ole võimalik makset autoriseerida. Siinkohal on oluline märkida, et 3DS küsimine ei ole kohustuslik igas regioonis ja olukorras ning sõltub, kas maksesooritaja pank ja kaupmees on liitunud 3DS programmiga [10].

2.9.1 3DS1

Esimene versioon *3D Secure 1*, mis arendati 1990ndate lõpus ja 2000 aastate alguses Visa Inc initsiatiivil. Iga skeem on võtnud kasutusele selle protokolli ja loonud oma brändi tud implementatsiooni: Visa on *Verified by Visa*, Mastercard on *Mastercard SecureCode*, ja American Express on *SafeKey.* [11], [12]

Esimese generatsiooni peamiseks lahenduseks oli ümbersuunamine panga lehele, kus pidi sisestama parooli ja PIN koodi. Sellega kaasnes palju ostukorvidest loobumisi ja maksete vastuvõtumäär langes. 3DS1 üheks suureks probleemiks on kasutajakogemus. See aegunud tehnoloogia on eriti problemaatiline mobiilseadmetes, kus tekivad kuvamise ja

ühilduvuse probleemid, lisaks on see protsess maksjale väga aeglane ja kohati ebavajalik. [11]

2.9.2 3DS2

Uus standard *3D Secure 2* loodi 2015. aastal EMVCo poolt ja on nüüd edutatud kui tugeva autentimise meetod täiendatud makseteenuste direktiivis (PSD2). Peamise eesmärgina pidi 3DS2 lahendama ära kohutava kasutajakogemuse probleemi. Lisaks oli see vajalik, et olla vastavuses PSD2 direktiivist tulenevate Euroopa nõuetega. [11].

3DS2 suureks erinevuseks on dünaamiline maksekogemus. Kaupmehed peavad saatma lisainformatsiooni kliendi kohta, mille analüüsi tulemusena võivad pangad pakkuda *frictionless* kogemust. See tähendab, et maksja kasutajakogemus on võrdeline maksega, mis ei kasuta 3DS autentimist. Kui 3DS1 puhul oli kasutajakogemus alati brauseris, siis nüüd on võimalus mobiilsetele maksetele pakkuda *native* ehk rakendusesisest kogemust, mis enamasti tähendab panga mobiilirakendusest makse kinnitamist. See loob võimaluse kasutada makse kinnitamiseks biomeetriat. [13]

3 Meetodid ja nende rakendamise võrdlus

Antud peatükis analüüsitakse loodava rakenduse vajadusi ja meetodeid, millega on võimalik kohandatavat stiili rakendada.

3.1 Rakenduse vajadused

Loodav integreeritav kaardimakse lahendus peab koguma kaardiandmete töötlemiseks vajalike andmeid: kaardinumber, aegumiskuupäev ja turvakood. Kaardiandmete sisestus maksjale peab olema loogiline ja arusaadav. On oluline, et maksja mõistaks, millises formaadis on andmed nõutud ja eksimuste korral kuvatakse selgitavad veateated. Lisaks peab lahendus toetama veebibrauserites kasutatavat automaatse täitmise funktsionaalsust, mis tagab kasutajamugavuse.

Arendajatele, kes hakkavad pakutavat lahendust kaupmeeste veebilehtedele integreerima, on oluline, et makseliidese installeerimine oleks lihtne. See tähendab, et ei erineks igapäevatöös teadaolevatest tavadest ja edaspidine haldus ei oleks ajakulukas. Makselahendust paigaldades on seda võimalik kohe kasutama hakata. Maksevorm peab olema eellaetud neutraalse teema ja stiilidega. Kui on soov, siis peab saama vormi välimust muuta vastavalt oma maitsele või veebilehe disainikeelele. Muuta peab saama kõike alates kirjastiilist, värvidest kuni suuruste ja kumerusteni.

Et kaardiandmeid töödelda peavad kaupmehed olema PCI DSS sertifitseeritud. Kui kaupmehe süsteeme läbivad kaardiandmed, siis tuleb kogu kaardiandmeid haldav süsteem viia vastavusse PCI DSS nõuetele. Enamike kaupmeeste süsteemid ei ole selliste nõuetega vastavuses ning nad ei saa kaardiandmeid oma süsteemides hoida ja töödelda. See tähendab, et on ülimalt oluline, et loodavas lahenduses ei läbiks kaardiandmed kaupmehe süsteeme, vaid liiguksid ja püsiksid ainult maksetöötlust teostava ettevõtte hallatavas süsteemides. Sellisel juhul on kaupmehed võimelised kaardimakseid vastu võtma ilma, et nende süsteemid oleksid PCI DSS standarditele vastavad. Sertifikaadi saamiseks peavad kaupmehed täitma PCI DSS enesehindamise küsimustiku, millega

kinnitavad, et nende süsteemides ei töödelda kaardiandmeid ja neil puudub ligipääs kaardiandmetele.

3.2 Kohandatav stiil

Väga olulise tähtsusega selle rakenduse juures on kohandatav maksevormi stiil. Oluline on eripära, et kuna rakendus on kui installeeritav valmislahendus, siis arendajatel, kes seda integreerivad ei ole kontrolli rakenduse koodi üle ega saa muuta äriloogika käitumist. Ei ole võimalik lisada juurde plokke, nuppe või lisavälju, kuid siiski peab olema võimalik muuta olemasolevate komponentide stiili vastavalt soovidele. Autor analüüsis turul olevaid lahendusi ning autorile teadaolevalt on kolm erinevat võimalust, millega teostada kohandatavat stiili.

3.2.1 CSS JavaScriptis

Üks võimalus on kasutada JavaScripti, et luua deklaratiivselt stiilid, mida arendajad saavad konfiguratsioonina kaasa anda. See meetod on tuntud kui ka *CSS in JS.*

See eeldaks makselahenduse rakenduses raamistike kasutamist, mis sellist funktsionaalsust toetaks. Ühed populaarsemad selleks on Emotion.js, JSS ja Styled Components. [14]

```
const styles = {
  '@global': {
    a: {
      textDecoration: 'underline'
    }
  },
  button: {
    fontSize: 12,
    '&:hover': {
      background: 'blue'
    }
  }
};
```

Joonis 1. Näide meetodist CSS JavaScriptis, kasutades JSS raamistiku

Selle meetodi boonuseks on CSS ühe suure nõrkuse, *global namespace,* elimineerimine. See tähendab, et kõik deklareeritavad stiilid on terve rakenduse ulatuses kättesaadavad. Kui muuta HTML nupu elemendi stiili, siis muutub kõigi rakenduses kasutavate nuppude elementide stiil. Samuti kehtib see klassi nimedele ja kõigile muudele selektoritele, mida CSS toetab. *Global namespace* pole iseenesest probleem, vaid ongi tahtlik lahendus CSSi loojate poolt, et vältida olukorda, kus elemendile tuleb korduvalt määrata samu stiile, näiteks kirjatüüpi ehk fonti, mis üldjuhul on globaalne igas rakenduses. Kuid selline globaalne lähenemine hakkab rakenduste kasvades üha rohkem probleeme valmistama. Suurtes veebirakendustes on tuhandeid HTML elemente ja on üsna tavaline, et ühel hetkel läbi juhuse kirjutatakse üle elemendi stiil, kuna selektor, millega viidatakse elemendile hakkab korduma või lihtsalt stiilid lekivad.

Miinuseks on kasutajakogemus kaupmehele. Paljud arendajad ei ole kokku puutunud *CSS in JS* metoodikaga, seega peaksid nad seda õppima, et kirjutada õiget süntaksit. Samuti erineb erinevate *CSS in JS* raamistike süntaks, mis tähendab, et isegi kui ollakse tuttavad selle metoodikaga, on siiski vaja süveneda kasutatava raamistiku süntaksilistesse eripäradesse. Kui kaupmehel ei ole otsest ligipääsetavust koodile, mis defineerib HTML elemendid ja struktuuri, võib tekkida vigu ja olukordi, mida on raske mõista ja lahendada.

3.2.2 Stiliseerimine kasutades API-t

Nagu eelneva meetodi puhul tuleb ka selle meetodi korral kohandatud stiilid kaasa anda konfiguratsioonis. Suur erinevus on selles, et kui eelnevas peatükis kasutati *CSS in JS* raamistiku ja süntaksit, siis nüüd tuleb stiilid lisada defineeritud API spetsifikatsiooni järgi. Kaupmehe jaoks kasutuskogemuse metoodilises mõttes pole suurt erinevust, tuleb lugeda spetsifikatsiooni ja selle järgi toimida.
Kuna kogu API on defineeritud maksevahendaja poolt, siis on võimalik pakkuda paremat tuge veateadete näol. Võimalus on defineerida tüübifail, mida TypeScriptil põhinevad rakendused saaksid kasutada, ennetades tüübi erinevusest tulenevaid vigu juba kompileerimise ajal.

Sellist lähenemist kasutab Stripe Elements UI. Stiilid antakse kaasa Stripe poolt defineeritud API kaudu, mis järgib CSS süntaksi [15].

```
const appearance = {
  theme: 'stripe',

  variables: {
    colorPrimary: '#0570df',
    colorBackground: '#ffffff',
    colorText: '#6d3f5b,
    colorDanger: '#df1f41',
    fontFamily: 'Ideal Sans, system-ui, sans-serif',
    spacingUnit: '8px',
    borderRadius: '2px',
  }
};
const elements = stripe.elements({clientSecret, appearance});
```

Joonis 2. Näide stiliseerimisest kasutades API meetodit Stripe Elements UI näitel

Selle meetodi miinuseks on väga suur esialgne investeering arendusele. Kuna on vaja defineerida ja ehitada kogu stiliseerimiseks mõeldud API ja seda toetav dokumentatsioon.

Plussiks on hea kasutajakogemus arendajatele, kui API spetsifikatsioon on hästi defineeritud ja võimalik tüübi tugi TypeScriptile. Selle meetodi puhul on ka lihtsam uuendada ja avastada versiooni uuendustest tekkivaid vigu. Enamasti põhjustavad seda lõhkuvad muudatused, mis tähendab, et varasemalt kasutatud funktsioonid või rakenduse osad on eemaldatud või ümber defineeritud ning versiooni uuendust tehes rakendus ei tööta või töötab ebakorrektselt.

3.2.3 Stiilid CSS ülekirjutamisega

Selle meetodiga defineeritakse stiilid CSS failis, kasutades CSS süntaksi. Kaupmehele tähendab see juba teada-tuntud süntaksi kirjutamist, mis peaks lihtsustama integratsiooni tegemist.

Selle meetodi mõte on sektsioonis 3.2.1 mainitud *global namespace* omaduse ära kasutamine. Kaupmehele antakse defineeritud CSS klasside nimekiri, mida on võimalik muuta ja üle kirjutada enda rakenduse stiili failis.

Eeliseks on lihtsus, kaupmees saab kasutada talle sobivaid ja tuntuid tööriistu. CSS-i või mõnda muud sobivat CSS laienduse raamistiku nagu SCSS või LESS. Lisaks selle meetodi esialgseks arendamiseks kuluv investeering on väiksem, kuna ei pea ehitama stiliseerimiseks vajaliku API-t, vaid saab kasutada ära CSS-i omadusi stiilide muutmiseks.

Miinusteks on juba 3.2.1 punktis väljatoodud ohud, mis kaasnevad *global namespace* omadusega. Siinkohal võib välja tuua, et CSS stiilide konflikti vältimist aitab hästi ära hoida defineeritud ja organiseeritud klassi nimede struktuur. Hea tava on selleks järgida BEM (*Block Element Modifier*) meetodit.

3.3 Kaardiandmete liikumine

Selleks, et kaardiandmed ei läbiks kaupmehe süsteeme tuleb kaardiandmeid koguvad vormid hoida makseteenuse osutaja süsteemides. Selleks on kaks võimalust:

- Ümbersuunamine maksesüsteemi lehele ehk *redirect* – Selle meetodi miinus on kasutajakogemus. Ostukorvis olles suunatakse ümber makse teostamiseks eraldi lehele ja pärast makse sooritamist toimub tagasisuunamine kaupmehe lehele. Sarnast mustrit kasutavad väga paljud teised maksemeetodid, näiteks pangalingi kaudu maksmine.

- *iFrame* veebitehnoloogia –See on sisuliselt teise veebilehe kuvamine enda lehe sees. Selle tulemusena on võimalik pakkuda väga loomulikku kasutajakogemust. Enamasti kasutajad ei märka, et tegelikult kasutatakse hoopis teiselt lehelt kuvatud rakendust.

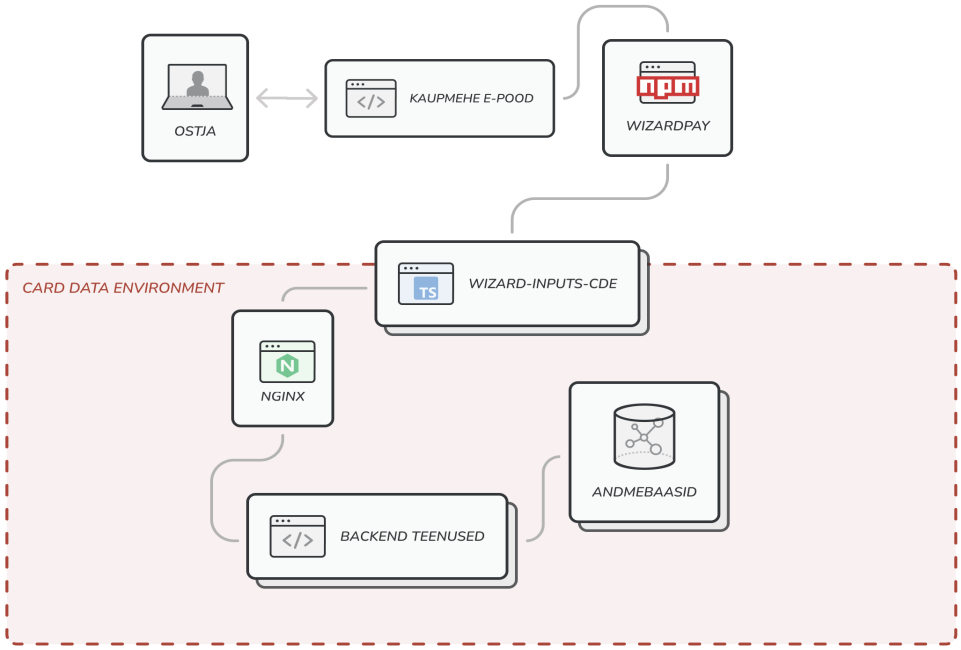
Kuna töö eesmärk on luua sisseehitatud makselahendus, siis esimene variant ei ole sobilik ja tuleb kasutada *iFrame* tehnoloogiat.

4 Rakenduse realisatsioon

Analüüsist lähtuvalt tuleb ehitada kaks rakendust. Üks rakendus keskendub kaardiandmete kogumisele ja teine rakendus serveerib kogu makselehte. Autor valis kohandatava stiilide lahenduseks CSS muutmise meetodi, sest see pakub arendajale tuttavat kasutajakogemust ja aeg rakenduse arendusest kuni turule jõudmiseni on lühem kui meetodi 3.2.2 puhul. Seda põhjusel, et stiliseerimiseks saab kasutada CSS-i võimalusi ning ei pea stiliseerimiseks spetsiaalset API lahendust defineerima ja ehitama.

Kaardiandmeid koguv vorm peab olema lahus kogu ülejäänud rakendusest, seega loodi eraldi repositoorium wizard-inputs-cde. (Joonis 3) See rakendus on serveeritud CDE keskkonnas ja on sisend kaardiandmete info liikumisele. Rakendus on privaatne ja selle lähtekood ei ole avalik.

Library rakenduse jaoks loodi eraldi repositoorium wizardpay. Selle rakenduse lähtekood võib olla avalik, kuid see ei ole ilmtingimata vajalik. *Library* on allalaetav Node Package Managerist (NPM) või CDN (*Content Delivery Network*) kaudu. Wizardpay rakenduse eesmärgiks on wizard-inputs-cde'st tulevate kaardivormi väljade kokkupanemine ühtseks kaardivormiks ning rakendada konfigureeritavaid stiile. Integratsioon kaupmehe süsteemidesse käib selle rakenduse kaudu (vt Joonis 3).



Joonis 3. Rakenduse ülevaade

4.1 Eelnõuded

Enne makselehe kuvamist peaks kaupmees tegema päringu, et luua maksesessioon, millega määratakse ära küsitava makse suurus, valuuta ja muud kaupmehega seonduvad andmed. Kuna vaadeldavas maksesüsteemis puuduvad veel taolised *API-d* sessiooni loomiseks ning sessiooni detailide pärimiseks, siis selle töö raames anname rakendusele vajalikud sisendparameetrid kaasa staatiliselt.

4.2 Kasutatud raamistikud ja pakid

Mõlemad loodud rakendused kasutavad tuntud raamistikku React. Valik sai tehtud tõdemusena, et vaadeldava ettevõtte suurem osa frontend rakendustest on kirjutatud React'is, ning hea tava on seda jätkata.

4.2.1 React.js

React on raamistik kasutajaliideste loomiseks. See võimaldab tükeldada rakenduse osad komponentideks, mida on võimalik taaskasutada. React on deklaratiivne, iga oleku kohta

on võimalik rakenduses teha vaateid ning oleku muutusel uuendab React efektiivselt ainult neid komponente, kus olek muutub. See väldib terve lehe uuendamist. React on avatud lähtekoodiga ja see on loodud Facebooki poolt. [16]

4.2.2 TypeScript

TypeScript on tugevalt tüübitud programmeerimiskeel, mis on ehitatud JavaScripti peale. See lisab ekstra süntaksi, millega on võimalik JavaScripti koodi tüüpida. TypeScripti kood konverteeritakse alati JavaScriptiks. [17]

4.2.3 Rollup

Rollup on mooduli kokkupakkija. Töö põhimõte on väga sarnane Webpackile, mis võtab ja kompileerib individuaalsed JavaScripti moodulid kokku üheksainsaks allikaks, mida veebibrauserid oskavad lugeda. Kuigi Webpack ja Rollup suudavad täita sama ülesannet, siis tavaliselt kasutatakse Webpacki rakenduste jaoks ja Rollupi *library* jaoks.[18]

4.2.4 ESLint

JavaScripti ja Typecripti koodi analüsaator. See aitab hoida koodi struktureeritud ja järgida häid tavasid kogu projekti ulatuses.

4.2.5 Storybook

Storybook on arendajatele mõeldud tööriist, millega on võimalik arendamisel isoleerida komponente või vaateid, et kiirendada arendusprotsessi.

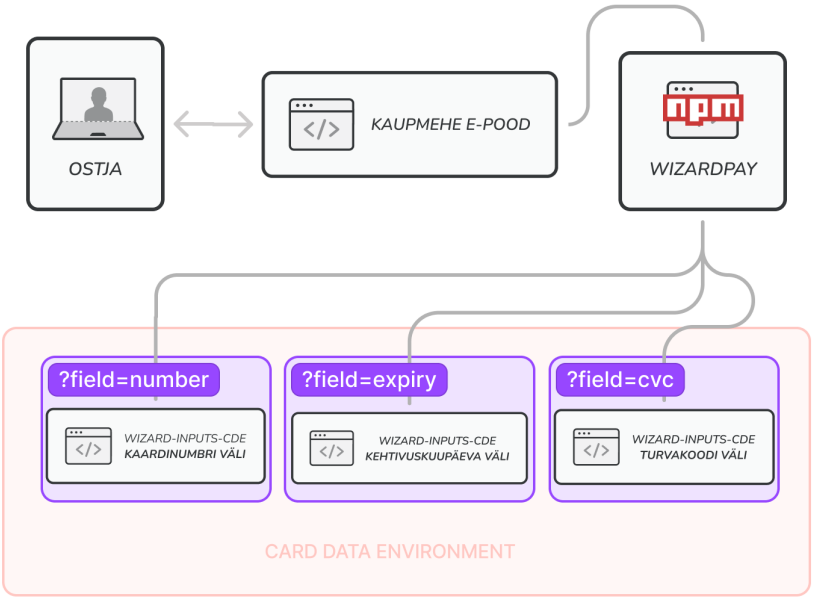
4.3 Kaardiandmete vorm (CDE rakendus)

Rakenduse ülesanne on serveerida sisendvälju, koguda kaardiandmeid ja teostada kontrolli andmete formaadi üle. Joonisel 3 on see rakendus kujutatud wizard-inputs-cde nimega.

Rakendus kasutab *query* parameetri osa URL-ist. Selleks kasutatakse query-string pakki, mis aitab URL-ist lihtsasti filtreerida vajaliku parameetri. Antud rakenduses on kokkuleppeliselt selleks parameetriks `field`.

Kui rakenduse hostiks on `https://wizard-cde.com`, siis kaardinumbri välja kuvamiseks tuleb kasutada URL-i `https://wizard-cde.com?field=number`.

(Joonis 4.) `Field` parameetri väärtused, mille puhul kuvatakse vastav vorm on: `number`, `expiry` ja `cvc`.



Joonis 4. Kaardiandmete sisendväljade kuvamine *iFrame*'s

Kõigil kuvatavatel sisendväljadel on määratud stiilid, mis eemaldavad äärejooned ning taust on muudetud läbipaistvaks. Joonisel 5. on vasakul kuvatud tavaline vaikimisi stiilidega sisendvälja HTML element ja paremal on `https://wizard-cde.com?field=number` kuvatav, eemaldatud stiilidega, sisendvälja element.

Joonis 5. Sisendväljade võrdlus

4.3.1 Väljade kontroll

Kõigil väljadel on kontrollid eeldatavale formaadile. Allpool on välja toodud kontrollid, mida rakendatakse. Kõik kontrollid on koondatud `validations.ts` faili.

- Kaardinumber on kohustuslik väli, kus tohivad esineda ainult numbrid pikkusega 16 kuni 19 tähemärki.
- Kehtivuskuupäev on kohustuslik väli, mis peab olema sisestatud MM YY formaadis, lubatud on ainult numbrid. Kehtivuskuupäev ei tohi olla rohkem kui kaks kuud minevikus, lugemist alustatakse tänasest kuust.
- Turvakood ehk CVC ei ole kohustuslik väli, kus tohivad esineda ainult numbrid, turvakoodi pikkus on 3 tähemärki.

Iga mitteläbiva kontrolli puhul tagastatakse veakood vastava informeeriva kliendipoolse sõnumiga.

Lisaks formaadi kontrollile rakendatakse kaardinumbrile ka kontrollsumma. Kaardinumbritel kasutatakse Luhni algoritmi, mida tuntakse ka kui mod 10 kontrolli. Luhni kontrolli saab teostada, siis kui kaardinumber on lõpuni sisestatud. Kui kaardinumber ei läbi Luhni kontrolli, siis väljastatakse vastav veateade.

4.3.2 Skeemi tuvastamine

Kaardinumbrit sisestades on võimalik tuvastada kaardi väljastanud skeem. Seda on võimalik teha kahte moodi, kasutades regulaaravaldisi või BIN andmebaasi.

Skeemi ja kaardi tuvastamine on väga oluline äriloogikalisest seisukohast, kui teostatakse optimeerimisi, et hoida kokku kuludelt või kiirendada makse sooritamist. Selleks on vaja täpselt teada, mis kaardiga on tegu. Selle jaoks regulaaravaldise kasutamine on rangelt mittesoovitatav ja seda väga lihtsatel põhjustel. Iga päev väljastatakse uusi kaarte ja väljastatud BIN-ide vahemikud muutuvad tihti. Isegi kui esialgu tundub, et on võimalik regulaaravaldistega kaarte tuvastada, siis selle haldamine ja vigade parandamine muutub väga kiiresti võimatuks. Sellisteks puhkudeks tuleks kasutada andmebaasi. Internetis on saadaval nii tasuta andmebaase kui ka tasulisi teenuseid.

Kasutajaliidese visualiseerimise seisukohalt saab regulaaravaldisi üsna edukalt kasutada. See ei ole kõige täpsem, kuid see eest ei pea tegema päringuid andmebaasi. Enamasti pole tulemuse saamiseks vaja sisestada BIN-i oma täies ulatuses, vaid juba paari esimese numbri sisestamisega saab tulemuse. Autori loodud rakendus kasutab samuti regulaaravaldisi, et määrata kaardi väljastanud skeemi. Tuvastatud skeemi logo kuvatakse kaardi sisestusvälja paremas ääres.

4.3.3 Autoriseerimine

Kui maksja on sisestanud oma kaardiandmed ja vajutab "maksa" nuppu, siis kaardiandmed transporditakse CDE *backend*'i. Algatatakse autoriseerimise protsess ja kaardiandmed salvestatakse. Antud rakenduses ei realiseeritud autoriseerimist kahel põhjusel, esiteks puuduvad selleks vajalikud *API endpoint*'id ja teiseks tulenevast keerukusest ning nõuetest nii kliendirakendusele kui ka *backend*'ile. Probleemi ja selle lahendusi on analüüsitud järgnevas alapeatükis.

4.3.4 Autoriseerimise murekohad

Autoriseerimise päringu tegemisel *backend* teenustesse peab saadetava päringuga kaasas olema kõik eelnevalt kogutud kaardiandmed. PCI DSS nõuetele vastavalt tuleb autoriseerimise päring teha rakendusest wizard-pay-cde. Sel juhul tekib probleem, kuna kaardivormi iga sisendväli on eraldi *iFrame* sees, siis meil on kokku kolm rakenduse sessiooni. On tekkinud olukord, kus iga rakendus teab kolmandikku informatsioonist, kuid autoriseerimise päringu tegemiseks on vaja teada kogu tervikut.

- Kolme wizard-input-cde rakenduse sessiooni vahel on teoreetiliselt võimalik informatsiooni vahetada, kuid seda saab teha ainult kasutades brauseri kohalikku salvestusruumi (*local storage*). See aga tähendab, et info liigub rakendusest väljapoole, ehk talletatakse maksja arvutis ja rikutakse PCI DSS reegleid. Seega antud lahendus ei ole sobilik.

- Informatsiooni saab kokku koguda wizardpay rakenduses ja teha sealt päring, kuid see tähendab, et kaardi info on kaupmehe süsteemis, mis ei ole aktsepteeritav. Kui kasutada sama metoodikat ja lisada kliendipoolne krüpteerimine (CSE), siis see lahendus toimiks.

- On ka võimalus teha kolm erinevat päringut autoriseerimiseks, mis tähendab, et iga väli teeb päringu ja saadab kolmandiku infost. Tehtavate päringute arv on kolm korda suurem ja tõenäosus tehnilisteks vigadeks või viideteks kasvab. Ekstra keerukuse lisab asjaolu, et vaadeldava maksesüsteemi PCI DSS sertifikaadi tase ei luba salvestada CVC koodi. Sel juhul on kriitilise tähtsusega, et kaardinumber ja kehtivuskuupäev jõuaks backendi enne kui CVC. Kuigi teoreetiliselt see lahendus toimiks, siis praktikas on see halb.

Parim ja ka ainuke variant on kasutada kliendipoolset krüpteerimist. PCI DSS juhistest võib välja lugeda, et kaardiandmeid ei tohi hoiustada ka krüpteeritult kujul väljaspool CDE, kuid andmete transiit on lubatud avalikes võrkudes, kui andmed on krüpteeritud ja avalik võrk ei tea ega oma juurdepääsu krüpteerimise võtmele. [5]

4.3.5 3D Secure implementatsioon

Autoriseerimise protsessi osaks on ka 3DS kontroll. Antud rakenduses seda ei implementeeritud, sest 3DS on tugevalt seotud autoriseerimisega ja implementatsioon on spetsiifiline maksesüsteemi eripärasustele ning vajaks API-de olemasolu. Tehnilise lahendusena ei vaja 3DS kontrolli teostamine midagi uudset, 3DS aken kuvatakse kasutades *iFrame* ja suhtlus kuvatud lehega toimub sarnaselt nagu kirjeldatakse punktis 4.4.2.

4.4 Makseliides

Selle rakenduse ülesandeks on kogu maksevormi kasutajaliidese kokkupanemine ja 4.3 osas kirjeldatud CDE rakenduses loodud väljade kuvamine, kasutades *iFrame*. Joonisel 3 on see rakendus kujutatud wizardpay nimega.

4.4.1 Inline frame

iFrame ehk *inline frame* on HTML teek. See teek defineerib kastikujulise ala HTML dokumendi sees, mille abil on võimalik kuvada eraldiseisvat dokumenti selle lehe osana.

iFrame teegile on kaks alternatiivi HTML5 standardis. Nendeks on *embed* ja *object* teek. Neid kasutatakse üldjuhul staatilise sisu kuvamiseks, nagu pildid, pdf või svg failid. *iFrame* kasutatakse kui on vaja kuvada veebilehti.[19]

iFrame teegil on ka mitmeid atribuute, millega on võimalik määrata suurust, stiili ja muid metaandmeid. `src` atribuudiga on võimalik määrata, millist lehte kuvada. On kahte tüüpi `src` URL-e, absoluutne ja suhteline. Absoluutse URL-i abil saab kuvada teisi veebilehti, suhtelise URL-iga on võimalik kuvada sama veebilehe domeenil olevat mingit teist lehekülge. Käesolevas töös kasutatakse absoluutseid URL-e, sest kuvatavad rakendused on majutatud teisel domeenil.

4.4.2 *iFrame* ja infovahetus

Lehte, mis kuvab *iFrame* kutsutakse kui *parent*, ehk vanemaks ja lehte, mis on kuvatud *iFrame* kaudu kutsutakse kui *child* ehk lapseks. Vanema ja lapse vahel on võimalik vahetada ka infot. Selleks kasutatakse `window.postMessage()` funktsionaalsust. See meetod loob võimaluse turvaliseks kommunikatsiooniks rakenduse väliste domeenidega, kasutades mõlema lehe `window` objekte. [20]

PostMessage funktsioonile on võimalik anda kolm sisendparameetrit:

- message – andmed, mida soovitakse saata.
- targetOrigin – sõnumi adressaat, võib märkida ka *, kuid sel juhul saavad sõnumit lugeda kõik, ja see on tungivalt mittesoovitav.
- transfer – ei ole kohustuslik väli, sellega on võimalik lisada lisaandmeid, mis antakse üle saajale. [20]

Selleks, et saadetavat sõnumit kätte saada tuleb saaja rakenduses luua kuulaja. Seda saab teha kasutades `window.addEventListener` funktsionaalsust.

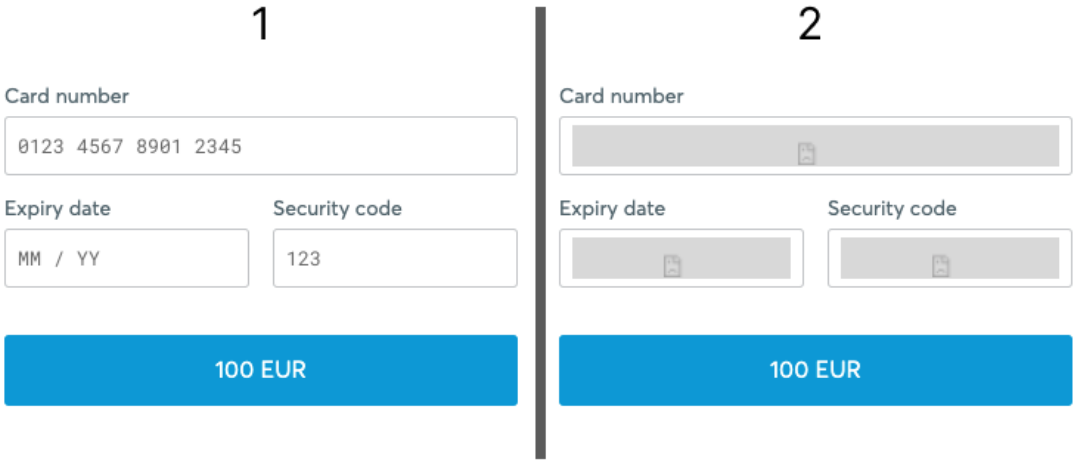
Töö käigus loodi kahe rakenduse vaheline suhtlusprotokoll, mis koosneb `action` parameeterist ja andmeobjektist. `Action` parameetri järgi oskab sõnumisaaja sõnumit filtreerida ja sellele vastavalt vajalike protsesse käivitada.

4.4.3 Sisendväljad

Sisendväljad tulevad wizard-pay-cde rakendusest, mida on kirjeldatud punktis 4.3. ja kujutatud joonisel 4 ja 5. Nagu mainitud punktis 4.3 on nendelt sisendväljadelt eemaldatud kõik stiilid ja muudetud läbipaistvaks. See on vajalik selleks, et wizardpay rakenduses oleks võimalik neid kasutada ja need seguneksid kasutajaliidesele määrtatud stiiliga.

Stiliseerimise meetodiks on CSS ülekirjutamine, kuna sisendväljad tulevad eraldi rakendusest, siis neid stiliseerida CSS reeglite ülekirjutamisega ei saa. Selle lahendamiseks loome wizardpay rakenduses *iFrame* poolt kuvatud sisendväljade ümber vajalikud visuaalsed elemendid, piirjooned, ja *input label* ehk sisendi sildid, mis loovad visuaalse mulje nagu tegemist oleks sisendväljale määratud stiilidega.

Joonisel 6 on kuvatud kaks vormi. Vasakpoolsel vormil on kuvatud mõlema rakenduse koostöös terviklik kasutajaliides. Paremal pool on kuvatud ainult wizardpay rakendus, *iFrame*'st tulevate sisendväljadeta, mida on kujutatud hallide kastidena.



Joonis 6. Võrdlus tervikliku vormi ja sisendväljadeta vormi vahel

Selline tehniline lahendus lahendab ära eelnevalt mainitud probleemi, kus *iFrame* sees olevat CSS ei ole võimalik üle kirjutada.

Klikkides sisendväljale, muutub see aktiivseks, aktiivset olekut tuleb eristada ka visuaalselt. Kuna wizardpay rakenduses ei ole tegelikult sisendvälju, on vaid neid emuleerivad stiilid, siis on vaja tuvastada, millal on sisendväli aktiveeritud, et muuta stiili aktiivsele olekule omaseks. Selleks kasutamegi punktis 4.4.2 kirjeldatud sõnumivahetust.

Olles tuvastanud sisendis aktiivse oleku saadetakse info wizard-cde-input rakendusest wizardpay rakendusse, mille tulemusena saame muuta elemendi klassinime aktiivseks, et kuvada aktiivse oleku visuaal. Samasugune põhimõte kehtib ka kasutajapoolse veateate kuvamisel, wizard-cde-input rakendus tuvastab sisendväljas ilmnenud vea ja see info saadetakse wizardpay rakendusse koos vastava veateate koodi ja tõlgitud sõnumiga, mille

tulemusena muudetakse sisendvälja stiilid veateatele kohaseks ning kuvatakse veateade wizardpay rakenduses. Võrdlus veaoleku ja mitteaktiivse sisendi vahel toodud joonisel 7.

Joonis 7. Näide mitteaktiivsest ja veaoleku sisendväljast

4.4.4 Stiliseerimine

Kõik wizardpay stiilid järgivad BEM (*Block Element Modifier*) metoodikat. BEM toetab taaskasutatavate komponentide ehitamist ja suurendab koodi jagamist. [21] BEM ei ole oma olemuselt midagi muud kui konventsioon. BEM-i järgides on võimalik stiilid eriti kompaktseks saada, kasutades CSS laiendeid nagu SCSS, mis süntaktiliselt lubavad klassinimede treppimist, kuid see ei ole ilmtingimata vajalik. Autor kasutas wizardpay rakenduses CSS ilma laienditeta. Lisa 2 on väljatoodud kõik CSS klasside nimetused.

4.4.5 Pakkimine ja avaldamine

Rakenduse NPM registris avaldamise jaoks on vaja kogu kood üheks JavaScripti mooduliks kompileerida. Selleks on kasutatud Rollup pakki. Rollup kasutab uut standardiseeritud formaati JavaScripti moodulite jaoks, mis lisati ES6 (*ECMAScript6*) uuendusega [18].

Rollup konfiguratsioon on defineeritud `rollup.config.js` failis. Lisaks baaskonfiguratsioonile on kasutatud erinevaid pluginaid. Neist eraldi tasub välja tuua kaks: postcss ja postcssUrl. Neid kasutatakse, et lisada CSS tugi ja genereerida `dist` kausta kombineeritud *`wizardpay.css`* fail. Kuna eelaetud teema kasutab ka fonte, siis tuleb ka need pakki kaasa panna. Selleks loob kompliaator `_assets` kausta, kuhu koondatakse kokku failid, mille faililaiend vastab Regex-ile `"/\.(png|jpe?g|gif|webp|svg|woff2?)$/"` ning muudetakse CSS failis olevad URL-i *pathid,* ehk viidad ressursi asukohani.

Tehtud töö osana avaldati wizardpay *library* NPM registrisse nimega @saamuelstarke/wizardpay. Kuna autor avaldas selle privaatselt bakalaureusetöö tegemise jaoks, siis ei ole see avalikkusele kättesaadav.

5 Järeldused

Töö tulemusena valmis kaks rakendust. Üks rakendus on kliendipoolne integreeritav makseliidese *library* ja teine on kaardiandmete sisendvälju kuvav abirakendus. Makseliides on kohandatavate stiilidega, mida on võimalik muuta kasutades CSS-i.

5.1 Rakenduse paigaldus

Testimaks loodud valmislahenduse integreerimist kaupmehe vaatepunktist, loodi uus rakendus store-app. Laeti alla @saamuelstarke/wizardpay pakk NPM registrist rakendusse. Imporditi kaardivälja komponent ja initsialiseeriti vajalike parameetritega. Järgnevalt loodi CSS fail stiilide muutmiseks ning selle põhjaks kopeeriti klassinimede mall (Lisa 2). Seejärel muudeti kaardivormi väljanägemist lisades CSS reegleid (Lisa 3). Tulemus on kujutatud joonisel 8. Autori hinnangul järgib toote paigaldus eesmärgiks seatud mustrit: paigalda, seadista, kasuta.

Joonis 8. Muudetud stiilidega kaardivorm

5.2 Stiilide kohandamine

Töö tulemusena valminud stiliseerimise osa rakenduses töötab vastavalt seatud eesmärkidele. Stiilide muutmine on lihtne ja loogiline, sest kasutab CCS-i, mis on üldjuhul *frontend* arenduses tavapärane praktika. Stiliseerimise ainuke eripära on sisendväljad, kuna need on kuvatud *iFrame*'s, siis peab teksti värvi ja suuruse muutmiseks konfiguratsiooni edasi andma API kaudu. Rakenduses ei realiseeritud fondi muutmist sisendväljades, sest implementatsioon tekitaks potentsiaalseid turvaohte. Tegemist on CDE-s paikneva rakendusega, mis tähendab, et fondi edastamine CDN kaudu või mõne teise meetodiga vajab potentsiaalse ründevektori tõttu erilist tähelepanu.

5.3 Autoriseerimine ja CSE

Töös kasutatud stiliseerimise meetodi tõttu on kaardiandmed jaotatud kolme erineva CDE rakenduse seansi vahel ning selle tulemusena on autoriseerimine raskendatud ja vajab lisaarendust kliendipoolse krüpteerimist teostava funktsionaalsuse näol. Seda funktsionaalsust kasutades on võimalik koguda kokku kaardiandmed ja need PCI DSS nõuetele vastaval viisil *backendi* toimetada, et autoriseerimist alustada.

5.4 PCI DSS nõuetele vastavus

Loodud rakendus on mõeldud vaadeldava maksesüsteemi osana, mis on PCI DSS sertifitseeritud. Lisaarenduste tegemisel ei ole vajalik sertifikaati uuesti taotleda, juhul kui muutused märkimisväärselt ei mõjuta CDE arhitektuuri ja turvalisust. [5] Loodud lahendus ei vaja PCI DSS sertifitseerimise auditit, sest kasutab samu metoodikaid ja tõekspidamisi kaardiandmete turvalisuse hoidmiseks, mis maksesüsteemi praegused kliendirakendused. Sellest tulenevalt võib loodud rakendusi lugeda olemasoleva süsteemi osaks, mis on PCI DSS nõuetele vastavad.

5.5 Tulevik

Valminud lahendust on lisaarenduste teostamisel võimalik edukalt kasutada, et pakkuda makseteostamise teenust kaupmeestele nende hallatavast süsteemist. Vajalikud lisaarendused kaupmehele tervikliku teenuse pakkumiseks:

- Kliendipoolne krüpteerimise funktsionaalsus

- Autoriseerimine ja 3DS implementatsioon
- *API* integratsioonid, töö kirjutamise hetkel vajalikud API-d puudusid
- Reacti sõltuvuse eemaldamine wizardpay rakenduses, et tagada töökindlus kõikides JavaScripti rakendustes

5.5.1 Alternatiivne tulevik

Rakendust ehitades selgusid valitud meetodi nõrgad küljed, mida autor eelnevas analüüsis ei osanud ette näha. Autori hinnangul on pikas perspektiivis parem valik punktis 3.2.2 pakutud lahendus – stiliseerimine kasutades API-t. See vajab küll suurema mahulist arendustööd, et ehitada stiliseerimist toetavad API-d, kuid lihtsustaks teatud aspekte võrreldes töös kasutatud metoodikaga:

- API lahendus pakuks paremat tuge, kui lõhkuvad muudatused puudutavad stiliseerimist
- Puudub vajadus kliendipoolseks krüpteerimiseks
- Kogu makseliides on selle meetodiga ühe *iFrame* sees. Kolme *iFrame* kuvamine tõstab oluliselt tõenäosust potentsiaalseteks tõrgeteks

Saavutamaks täielik kontroll stiilide üle, on antud meetodi puhul vaja ära lahendada turvaliselt fontide edastamine API kaudu. Sama probleem eksisteerib ka teostatud lahenduses, kuid see on vähem olulisem, sest mõjutab ainult sisendväljade fonti, mitte kogu maksevormi fonti.

6 Kokkuvõte

Käesoleva töö eesmärk oli integreeritava kaardimakse lahenduse loomine, mille väljanägemist saab kaupmees vastavalt oma vajadustele muuta. Valminud lahendus koosneb kahest loodud rakendusest.

Eesmärgi realiseerimiseks analüüsiti vajadusi ja nõudeid kasutajaliidesele ning erinevaid meetodeid kohandatava stiliseerimise saavutamiseks. Autor valis lahenduse, milleks osutus stiliseerimine CSS-i ülekirjutamisega. Valiku kriteeriumiteks oli arendusmaht funktsionaalsuse realiseerimiseks ja integreerimise kasutajakogemus kaupmehele.

Maksevormi stiliseerimine vastab püstitatud nõuetele ja eesmärkidele. Lihtsuse loob CSS-i kasutamine ja *BEM* metoodikat järgiv struktureeritud klassinimede mall.

Rakenduse arendusel oli vaja suurt tähelepanu pöörata PCI DSS standardist tulenevate kaardiandmete turvalise töötlemise nõuetele, mis mõjutas oluliselt rakenduse loomisel tehtavaid valikud ja võimalusi.

Töö eesmärgid said täidetud, maksevorm on ehitatud pidades silmas PCI DSS nõudeid, on muudetava stiiliga ja implementeeritud viisil, mis võimaldab kasutada toodet valmislahendusena. Maksevormi tervikliku lahenduse pakkumiseks kaupmehele jääb autoriseerimispäringuks vajalike kaardiandmete kogumine, mis vajab kliendipoolset krüpteerimist ja 3DS kontrolli teostamist. Antud implementatsioon väljub konkreetse töö skoobist ja on teostatav töö edasiarenduses.

Kasutatud kirjandus

[1] „Payfacs: A guide to payment facilitation - Stripe“ https://stripe.com/enee/guides/payfacs [Kasutatud 23.04.2022].

[2] „Primary Account Number (PAN)“, Investopedia. https://www.investopedia.com/terms/p/primary-account-number-pan.asp [Kasutatud 22.04.2022].

[3] „BIN List & Range“. https://www.bincodes.com/bin-list/ [Kasutatud 1.05.2022].

[4] „What is a Card Scheme and how do they work? | EBANX“. https://business.ebanx.com/en/resources/payments-explained/credit-card-schemes [Kasutatud 22.04.2022].

[5] „Payment Card Industry Data Security Standard“. https://www.pcisecuritystandards.org/documents/PCI-DSS-v4_0.pdf?agreement=true lk 26-28, lk 102-111. [Kasutatud 24.04.2022].

[6] BigCommerce, „How Credit Card Authorization Works“, BigCommerce. https://www.bigcommerce.com/ecommerce-answers/credit-card-authorization/ [Kasutatud 30.04.2022].

[7] „What Does ‘Credit Card Capture’ Mean?“ https://www.tidalcommerce.com/learn/credit-card-capture [Kasutatud 26.04.2022].

[8] „Stripe: Optimizing authorization rates: How to reduce network declines“. https://stripe.com/en-ee/guides/optimizing-authorization-rates [Kasutatud 26.04.2022].

[9] „What is 3D Secure (3DS)? | Payments Explained | EBANX“. https://business.ebanx.com/en/resources/payments-explained/3d-secure [Kasutatud 26.04.2022].

[10] „3-D Secure“, SEB klienditoe veeb. https://support.ecommerce.sebgroup.com/et/merchant-support/3-d-secure/ [Kasutatud 30.04.2022].

[11] „PSD2, SCA, and 3DS2 Explained“. https://www.spreedly.com/blog/psd2-sca-3ds2-basics [Kasutatud 25.04.2022].

[12] „3D Secure 1“. https://www.gpayments.com/about/3d-secure/ [Kasutatud 27.04.2022].

[13] „3D Secure 2.0: What is it and how does it work?“ https://3dsecure2.com/ [Kasutatud 27.04.2022].

[14] „A Thorough Analysis of CSS-in-JS“, CSS-Tricks, 26. mai 2021. https://css-tricks.com/a-thorough-analysis-of-css-in-js/ [Kasutatud 24.04.2022]

[15] „Elements Appearance API“. https://stripe.com/docs/stripe-js/appearance-api [Kasutatud 28.04.2022].

[16] „React – A JavaScript library for building user interfaces“. https://reactjs.org/ [Kasutatud 27.04.2022].

[17] „JavaScript With Syntax For Types.“ https://www.typescriptlang.org/[Kasutatud 27.04.2022].

[18] „rollup.js“. https://rollupjs.org/guide/en/ [Kasutatud 27.04.2022].

[19] „From object to iframe — other embedding technologies - Learn web development | MDN“. https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Other_embedding_technologies [Kasutatud 29.04.2022].

[20] „Window.postMessage() - Web APIs | MDN“. https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage [Kasutatud 29.04.2022].

[21] V. S. contributors Vladimir Starkov, „BEM — Block Element Modifier“. http://getbem.com/ [Kasutatud 30.04.2022].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Saamuel Starke

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose ”Integreeritav ja kohandatava disainiga kaardimakse lahendus”, mille juhendaja on Tarvo Treier
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

08.04.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – CSS klasside mall

```
.wizardpay-checkout {

}

.wizardpay-checkout__field {

}

.wizardpay-checkout__field--focused {

}

.wizardpay-checkout__field--error {

}

.wizardpay-checkout__label {

}

.wizardpay-checkout__label--focused {

}

.wizardpay-checkout__label--error {

}

.wizardpay-checkout__error--text {

}

.wizardpay-checkout__pay-button {

}

.wizardpay-checkout__card-number-field {

}

.wizardpay-checkout__expiry-date-field {

}

.wizardpay-checkout__security-code-field {

}
```

Lisa 3 – stiliseeritud CSS klassid

```
.wizardpay-checkout {
    font-family: W95FA;
}

.wizardpay-checkout__field {
    background-color: #ffffff;
    border-radius: 0;
    transition: background 0.15s ease, border 0.15s ease, box-shadow 0.15s
ease, color 0.15s ease;
    box-shadow: inset -1px -1px #ffffff, inset 1px 1px #0a0a0a, inset -2px -
2px #dfdfdf, inset 2px 2px #808080;
}

.wizardpay-checkout__field--focused {
    border: 2px solid #0081ba;
}

.wizardpay-checkout__field--error {
    border: 2px solid #cf2929;
}

.wizardpay-checkout__label {
    color: #fff;
}

.wizardpay-checkout__label--focused {
    color: #fff;
}

.wizardpay-checkout__label--error {
    color: #cf2929;
}

.wizardpay-checkout__error--text {
    color: #fff;
    display: block;
    padding-top: 8px;
}

.wizardpay-checkout__pay-button {
    border-radius: 0;
    background-color: #A7A9AD;
    transition: background 0.15s ease, border 0.15s ease, box-shadow 0.15s
ease, color 0.15s ease;
    box-shadow: inset -1px -1px #ffffff, inset 1px 1px #0a0a0a, inset -2px -
2px #dfdfdf, inset 2px 2px #808080;
}
```