Sustavi za naplatu troškova u web okruženju

Gal, Leon

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

SUSTAVI ZA NAPLATU TROŠKOVA U WEB OKRUŽENJU

Diplomski rad

Leon Gal

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Leon Gal
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1283R, 07.10.2022.
JMBAG:	0165081555
Mentor:	doc. dr. sc. Krešimir Romić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Hrvoje Leventić
Član Povjerenstva 1:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 2:	Robert Šojo, univ. mag. ing. comp.
Naslov diplomskog rada:	Sustavi za naplatu troškova u web okruženju
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Zadatak rada je opisati sustave za provedbu plaćanja u današnjim web okruženjima uz kratki povijesni pregled evolucije ovakvih sustava. Dati pregled trenutno dostupnih rješenja na tržištu i usporediti ih na više razina (lakoća integracije, cijena, sigurnost i sl.). Opisati tehničku osnovu ovakvih sustava što uključuje enkripcijske standarde, procese autorizacije i integraciju s financijskim institucijama. Na jednom ili više primjera programskog koda pokazati integraciju sustava za naplatu troškova u web aplikaciji. Predstaviti buduće trendove i inovacije u ovom području.
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	19.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	27.9.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	27.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 27.09.2024.

Ime i prezime Pristupnika:	Leon Gal
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1283R, 07.10.2022.
Turnitin podudaranje [%]:	9

Ovom izjavom izjavljujem da je rad pod nazivom**: Sustavi za naplatu troškova u web okruženju**

izrađen pod vodstvom mentora doc. dr. sc. Krešimir Romić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

U posljednjih nekoliko desetljeća došlo je do revolucije u načinu na koji obavljamo plaćanja. Tradicionalne metode plaćanja postupno su zamijenjene elektroničkim plaćanjima, a s njima su se pojavili i novi igrači na tržištu - pružatelji usluga internet plaćanja. Oni su omogućili jednostavnije, brže i sigurnije online transakcije, transformirajući tako trgovinu i financijske usluge. Kako bi poduzeća mogla učinkovito prihvaćati online plaćanja, potrebna im je pouzdana i sigurna infrastruktura. Ovaj rad će opisati sam postupak nastanka i dolazak do ideje online plaćanja, uvesti čitaoca u pojmove koji se koriste, te prikazati proces integracije 3 vodeća pružatelja usluga u ovoj domeni. Zatim slijedi usporedba implementiranih rješenja sa smjernicama za potencijalno korištenje navedenih sustava.

1.1. Zadatak diplomskog rada

Opisati sustave za provedbu plaćanja u današnjim web okruženjima uz kratki povijesni pregled evolucije ovakvih sustava. Dati pregled trenutno dostupnih rješenja na tržištu i usporediti ih na više razina (lakoća integracije, cijena, sigurnost i sl.). Opisati tehničku osnovu ovakvih sustava što uključuje enkripcijske standarde, procese autorizacije i integraciju s financijskim institucijama. Na jednom ili više primjera programskog koda pokazati integraciju sustava za naplatu troškova u web aplikaciji. Predstaviti buduće trendove i inovacije u ovom području.

2. PREGLED PODRUČJA TEME

Kompanija koja je razvila temelj za sigurno plaćanje internetom je *First Virtual Holdings* osnovana 1990-ih u Kaliforniji [1]. Svi današnji sustavi u nekoj mjeri koriste protokole koje je ta tvrtka razvila te se ona smatra pionir u području internetskog plaćanja. Prema [2] sustavi za naplatu se dijele na elektroničke (POS terminali, debitne/kreditne kartice, *e-cash*...) i internetske sustave naplate (*Cyber cache*, neto računi, virtualni posjedi). Danas je ova podjela usko povezana s obzirom da većina elektroničkih pružatelja usluge pruža i internetske usluge naplate. Primjerice, neki od pružatelja usluge naplate POS uređajima su PayPal-ov Zettle, Stripe, Authorize.Net te Moneris, no upravo te kompanije pružaju i procesore za obradu internetskih plaćanja. Ovakvi sustavi za naplatu troškova se većinom koriste u privatnim sektorima, dok se kod javnih/državnih pokušava stvoriti neovisnost o nekom pružatelju usluge pa se većinom koriste interni sustavi naplate ili se zasebno implementiraju. Primjer jednog javnog sustava za naplatu javnih davanja poput poreza i kazni putem platnih kartica ili elektroničkih transfera je sustav E-Pristojbe. Uspostavljen je odlukom Vlade Republike Hrvatske te omogućuje naplatu javnih davanja, novčanih kazni i upravnih pristojbi putem platnih kartica [3]. Plaćanje se obavlja kartičnim putem unutar e-usluga koje su dostupne putem sustava e-Građani ili na šalterima različitih institucija putem POS uređaja [4]. Nadalje, online načini plaćanja se ostvaruju i u mobilnim okruženjima. Razne Android aplikacije na pametnim telefonima poput Ngpay, Paytm pružaju online uslugu plaćanja. Posebno popularan način plaćanja u zadnje vrijeme je mobilna aplikacija KEKS Pay čiji je vlasnik Erste Banka [5]. Omogućava prebacivanje novca kontaktima bez naknada za transakcije neovisno o banci koju koriste, plaćanje parkinga, računa, plaćanja u trgovinama, ugostiteljskim objektima i drugo. Osim toga, u izvoru [2] spominju se i drugi načini u kojima klijenti mogu koristiti svoje telefone za plaćanja. Korištenjem mobilnog interneta, korisnici mogu prenijeti PIN broj, poslati SMS poruku ili koristiti WAP (engl. *Wireless Application Protocol*) za plaćanje putem interneta.

Od sličnih radova, 2002. godine napisan je članak na temu usporedbe elektroničkih sustava naplate [6] u kojem su detaljno opisani Ecash, Mondex i Visa Cash sustavi elektroničke naplate.

3. POVIJESNI RAZVOJ PLAĆANJA

U ranim fazama ljudske povijesti trgovina je tipično bila karakterizirana različitim razmjenama vrijednosti. Rane aktivnosti plaćanja vezane uz trgovinu obavljale su se korištenjem sustava koji se naziva trampa ili robna razmjena (engl. *barter/bartering*). To je postupak koji se sastoji od razmjene između dvije strane, bez načina plaćanja, te kao takav ne uključuje novac, nego se za robu ili uslugu naplaćivalo drugom robom ili uslugom [7]. Primjerice, jedna strana proizvodi mlijeko dok druga proizvodi mliječne proizvode, trampom postižu dogovor da, strana koja proizvodi mlijeko opskrbljuje stranu koja proizvodi mliječne proizvode, a strana koja proizvodi mliječne proizvode nudi svoje proizvode u zamjenu. Trampa je imala svojih mana jer često nije bilo jednostavno zamijeniti robu izravno za neku drugu robu (ne treba svakome ono što mi nudimo te smo tako ograničeni na manji spektar osoba za trgovanje) te roba ima kratak rok trajanja brzo propada i gubi na vrijednosti. Vrlo brzo se razvila potreba za univerzalnijim načinom plaćanja neovisno o robi ili usluzi koja se zahtjeva te potreba za štednjom. Tako je počelo trgovanje materijalima koji ne propadaju tako lagano, odnosno, plemenitim metalima: dukatima, srebrnjacima, zlatnicima te drugim dugovječnim sirovinama koje su zapravo preteče današnjeg novca. Ideja je da vrijednost imaju u svojoj težini te samim time i pokriće koje predstavljaju. To je način trgovanja koji se još uvijek koristi, no umjesto sirovina se koristi propisani novac određene države ili unije država. Otvaranjem financijskih institucija te banki započelo je centralizirano skladištenje financija te sigurnija štednja. Uvođenjem sigurnijeg skladištenja novca banke uvode i naknade za korištenje njihovih usluga. Banke su uvele čekove koji su bili prva zamjena za novac, točnije dokaz o posjedovanju istog. Konkretno u Republici Hrvatskoj čekovi su gotovo izumrli te jedina banka koja zaprima i naplaćuje inozemne čekove je Hrvatska poštanska banka [8]. Prve ideje plaćanja kreditnim karticama pojavljuju se 1950-tih u Americi, no malo tko je bio u mogućnosti uspostaviti financijski sustav u kojem je karticu koju je izdala banka treće strane prihvaćao veći broj trgovaca. Problem je bio što korisnici nisu htjeli prihvatiti karticu koju bi malo trgovaca prihvatilo, a s druge strane trgovci nisu htjeli prihvatiti karticu koju je koristio mali broj korisnika. Uz sve to, obrada transakcija se odvijala ručno te se proces obrade sastojao iz više koraka:

1. Nakon kupnje trgovac uzima otisak kartice pomoću mehaničkog uređaja (klizač, engl. *Imprinter*) (Slika 3.1.) u svrhu preslikavanja podataka kartice.

2. Kupac zatim potpisuje prodajni slip.

3. Na kraju dana, trgovac šalje akumulirane prodajne slipove banci na obradu.

4. Banka ručno pregledava svaku transakciju i provjerava kreditni limit kupca i valjanost kupovine.

5. Ako je sve u redu transakcija bude odobrena i sredstva se prenesu s kreditnog računa kupca.

Sl. 3.1. Uređaj za otiskivanje kartica, klizač

Kao što je vidljivo iz prethodno navedenih koraka kreditne kratice koje bi trebalo olakšati i ubrzati proces plaćanja su zapravo bile sporije te zahtijevale više rada nego korištenje tradicionalnog novca. Velika promjena koja je dotaknula svijet plaćanja bio je sustav plaćanja BASE 1 [9].

Bank America System Engineering 1 (BASE 1) je prvi elektronički autorizacijski sustav za plaćanje kreditnom karticom. Razvila ga je *Bank of America* kao dio VisaNet sustava 1973. godine. Uvođenjem ovog sustava uklonjeni su gotovo svi prethodno nabrojani koraci. Nakon kartične transakcije detalji kartice se elektronički šalju centralnom računalu za autorizaciju. Sustav je tada, gotovo u stvarnom vremenu, provjerio valjanost kartice, pokriće na kartici te odobrio ili odbio transakciju. Upravo ovaj revolucionarni sustav je potaknuo velik broj prodavača i poduzeća na uvrštavanje kartičnih sustava naplate u svoja poslovanja.

Koncept plaćanja za robu i usluge je oduvijek bio prisutan te se kao takav nastavlja razvijati i dan danas. Vrlo brz razvitak poluvodičke tehnologije te proces digitalizacije uvelike su izmijenili načine na koji se trguje, a pogotovo na koje će se tek trgovati. Upravo zbog tog su magnetske trake na karticama zamijenjene EMV (Europy, Mastercard, Visa) čipovima koji pružaju puno veću sigurnost te podršku za NFC (engl. *Near-field Communication*) komunikaciju. Sve većim napretkom pametnih uređaja i telefona pristup NFC tehnologiji omogućen je gotovo uvijek. Sama

činjenica da danas gotovo svi mobiteli, pametni satovi, narukvice, prijenosna računala i dr. imaju podršku za NFC tehnologiju, tako da se rijetko može dogoditi da se zaboravi novčanik ili kartica i da nema načina kako platiti. Drugim riječima danas sve postaje novčanik i elektronički način plaćanja. Sve više se značaja daje sustavima internet naplate (*Payment getaway*) pa je upravo zbog tog danas vrlo zastupljena i internetska trgovina (engl. *e-Commerce*) koja sve više zastupa elektroničke načine plaćanja. Svatko je danas u mogućnosti otvoriti račun na bilo kojem od desetaka sustava za naplatu te tako postati trgovac, odnosno biti u mogućnosti primati sredstva od kupaca. Štoviše, svi uređaju koji komuniciraju na mreži su usmjereni na ravnopravnost ostalih sudionika u plaćanju, nebitno bili oni prodavači ili kupci, te se integriraju s metodama plaćanja kako bi pružili jednostavnost i neovisnost o mediju plaćanja. Više nije bitno koristi li se za kupovinu tradicionalni novac, kartica ili pametni toster. Na taj način tehnologija otvara novu mogućnost plaćanja koja je puno jednostavnija od korištenja novca te neovisna o tome gdje se kupac nalazio u svijetu, što će polako dovesti do izbacivanja novca iz uporabe.

4. MODERNI SUSTAVI NAPLATE

Osnovna definicija plaćanja je da je plaćanje zamjena novčane vrijednosti za primitak robe ili usluga [10]. U takvom kontekstu se tada internetska transakcija može definirati kao ona u kojoj se novčana vrijednost prenosi elektronički ili digitalno između dva subjekta kao kompenzacija za robu ili uslugu. Pod subjekt se misli na banke, tvrtke, vladu pa čak i obične potrošače. Modernih sustava naplate mogu se podijeliti na elektroničke i web sustave naplate. Iako se oba sustava zasnivaju na elektroničkom plaćanju, web sustavi su otišli korak dalje, omogućili plaćanje s bilo koje lokacija samo putem internetskog pristupa te su kao takvi drugačije tehničke prirode. S obzirom da je u ovom radu naglasak na web sustave naplate, elektronički sustavi su opisani samo kao osnova za razumijevanje web sustava.

4.1. Elektronički sustavi naplate

Elektronički sustavi naplate (engl. *e-Payment Systems*) u daljem tekstu e-plaćanja, su načini obavljanja transakcija ili plaćanja računa putem interneta ili putem elektroničkog medija, bez upotrebe fizičkih čekova ili gotovine. Veliku su popularnost dobili krajem 20. stoljeća jer uvelike olakšavaju trgovanje. Najviše u kontekstu obveza prema dobavljačima, gdje su e-plaćanja pozitivna za obje strane, a načini kako to postižu nabrojani su u nastavku.

Prednosti e-plaćanja:

1. Smanjeni troškovi - Došli smo do stupnja razvoja tehnologije kad je vrijednost/cijena elektroničkog zapisa dokumenta u odnosu na vrijednost papira zanemarivo malena. Ako se uzme u obzir prosječna veličina digitalnog računa koja ne prelazi par megabajta i količina memorije koju svatko ima samo na mobilnim uređajima vrlo je lako zaključiti da svakodnevno “nosimo” ekvivalent pohrane od preko 100,000 papira ili preko 500kg papira. Prema tome što više plaćanja tvrtka može obraditi elektronski, manje će potrošiti na papir i poštarinu, zajedno s vremenom potrebnim za ispis, potpisivanje, slanje dokumenta i čekova. Štoviše, sama digitalizacija plaćanja može smanjiti troškove obrade plaćanja za do 80%.

2. Poboljšani odnosi s dobavljačima - Tvrtke mogu poboljšati odnose s dobavljačima omogućavanjem bržih i sigurnijih plaćanja koja sadrže mnogo više informacija radi lakšeg usklađivanja podataka o transakcijama.

3. Povećavana vidljivost – Sustavi e-plaćanja pružaju bolju vidljivost statusa plaćanja, povijesti transakcija, financijske metrike i točne revizijske tragove[1] . Dodatno, smanjuju troškove i vjerojatnost pogrešaka pri unosu podataka.

4. Poboljšana sigurnost - Elektronička plaćanja sama su po sebi sigurnija od papirnatih čekova, a posebne metode poput virtualnih kartica nude još veću zaštitu od prijevara. Povrh toga, sustavi e-plaćanja su iznimno sigurni jer uključuju dodatne značajke i kontrole koje pomažu osigurati proces plaćanja poput enkripcije podataka.

Najpopularnije metode elektroničkog plaćanja uključuju kreditne kartice, debitne kartice, virtualne kartice i ACH (engl. *Automated clearing house*) (izravni depozit, izravno terećenje i elektronički čekovi). Općenito govoreći, transakcije e-plaćanja mogu se kategorizirati u tri segmenta: maloprodajno e-plaćanje, korporativno e-plaćanje i veleprodajno e-plaćanje.

U segmentu maloprodajnog e-plaćanja transakcije se sastoje od dva aktera, potrošača (*consumer*) i poduzeća (*bussines*). Općenito, postoje tri vrste maloprodajnih transakcija: od potrošača do poduzeća (C2B), od poduzeća do potrošača (B2C) i potrošač potrošaču (C2C) (ili *peer-to-peer* P2P). C2B transakcije najčešći su oblik plaćanja, bilo da se radi o potrošačima koji kupuju dobra i usluga ili o pojedincima koji plaćaju porez državi. Državni subjekti se mogu promatrati kao poduzeća u svrhu ove analize, te su također uključeni u ovu kategoriju maloprodajnog e-plaćanja. B2C transakcija pokriva isplatu nadnica ili plaća koje šalju poslodavci svojim zaposlenicima ili EFT-a (engl. *Electronic fond transfer*) kao što je povrat novčane vrijednosti od poduzeća do potrošača. Još jedan oblik B2C transakcije koji postaje prilično popularan je isplata države građanima poput prijenosa mirovina putem digitalnih sredstava. Treća klasifikacija maloprodajnog e-plaćanja, C2C transakcije, odnosi se na plaćanja između pojedinaca koja proizlaze iz internetske aukcije ili prijenosa sredstava između pojedinaca.

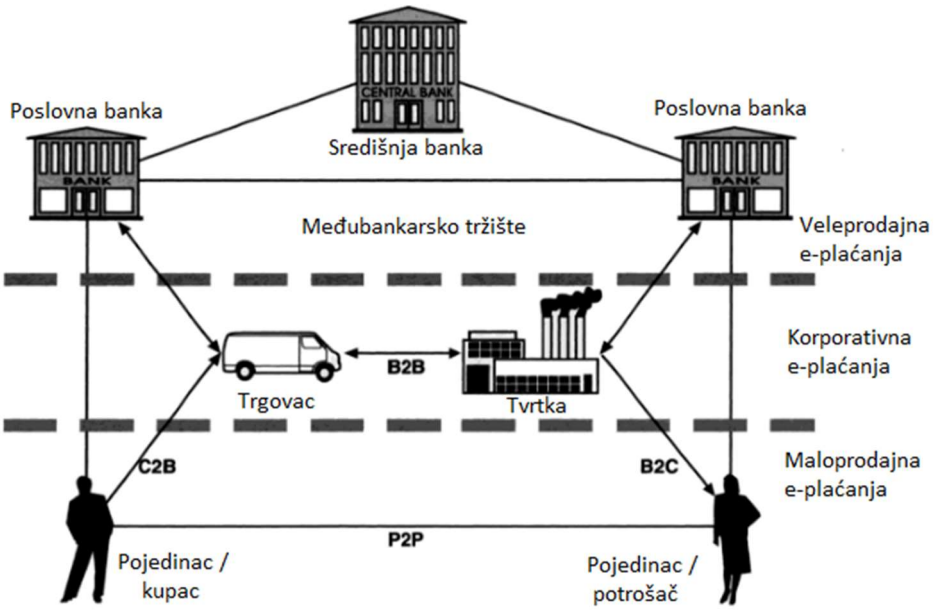
Sljedeći segment elektroničkog plaćanja je korporativno e-plaćanje. Akteri su poduzeća (*bussines*), banke (*bank*) i vlada (*goverment*). Definirano je kao plaćanje za robu i usluge između poduzeća ili korporativnih subjekata. Vrste korporativnog e-plaćanja su: poduzeća do poduzeća (B2B) u svrhe nabave proizvoda, banke do poduzeća ili poduzeća do banke (B2B), poduzeća do vlade (B2G) u svrhu plaćanja poreza ili državnih pristojbi te vlade do poduzeća (G2B) kod povrata

[1] Prikaz dokumentiranog tijeka financijskih i drugih transakcija od njihovog početka do završetka, s ciljem omogućavanja rekonstrukcije svih pojedinačnih aktivnosti i njihova odobrenja.

novca i poreza. Također, postoje sve veća prihvaćanja e-plaćanja između vladinih agencija koje su nazvane G2G transakcije.

Treći segment e-plaćanja je veleprodajno e-plaćanje te se ono definira kao plaćanje između banaka i središnje banke. Ovo je segment plaćanja koji je pod konstantnim opterećenjem te se najveća količina novca prebacuje ovim putem. Zbog konstantnog povećanja opterećenja ovakvih sustava, brzine obrade transakcija, sigurnosti i jednostavnosti korištenja, infrastruktura između banaka se konstantno transformira uspostavljanjem novih burzi, automatiziranih klirinških kuća (ACH) (engl. *Automated clearing house*), sustava za bruto poravnanje u stvarnom vremenu (RTGS) (engl. *Real-Time Gross Settlement*), sustava za izravnu obradu informacija (STP) (engl. *Straight-through processing*) te sustava za kontinuirano naseljavanje (CLS) (engl. *Continuous Linked Settlement*). Veleprodajno bankarstvo, financijski sektor i sektor vrijednosnih papira je pod konstantnom transformacijom.

Na slici (Slika 4.1.) prikazan je kompletan opseg transakcija e-plaćanja (platni promet) odnosno potencijalne permutacije i kombinacije entiteta koji mogu biti uključeni u transakcije. Slika je vrlo jasan pokazatelj širine elektroničke transakcije između različitih identiteta i segmenata.



Sl. 4.1. Opseg platnog prometa (autor: Mistura Mohammed Usman)

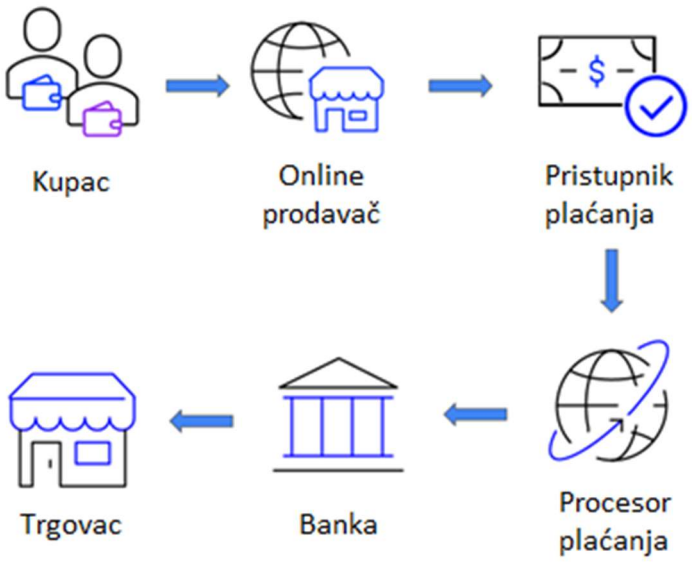
4.2. Web sustavi naplate

Danas su web sustavi naplate troškova usko integrirani u svaki aspekt modernog života te plaćanje "internetom" postaje svakodnevna praksa sve većem broju ljudi. Osnovna jedinica plaćanja u web

okruženjima je sustav za naplatu troškova ili skraćeno IPG (engl. *Internet Payment Geteway*). IPG je naziv za platformu koja služi za autorizaciju online financijskih transakcija koje se odvijaju u realnom vremenu. Dakle, riječ je o integriranom softverskom rješenju koje automatizira proces naplate proizvoda, odnosno usluga, između kupca (naručitelja), web shopa (trgovine) i banke. Kada se radi o internetskoj trgovini IPG ispunjava identičnu svrhu kao i POS uređaji u fizičkim trgovinama. On posreduje između prodavača i banke kako bi omogućili realizaciju sigurne naplate. Ugrubo, proces naplate se odvija tako da *payment gateway* zaprima podatke s kreditnih kartica. Zatim ih prosljeđuje prema banci koja zatim autorizira ili odbija transakciju. Informacija o odobrenju, odnosno odbijanu transakcije povratno se, posredstvom *payment gateway* sustava, povratno dostavlja platitelju i banci, odnosno kartičnoj tvrtki.

4.2.1. Obrada transakcije

Osnovni dio IPG sustava čini *payment processor* koji je odgovoran za upravljanje tehničkim vezama između pristupnika plaćanja, banke prihvatitelja i banke izdavatelja. U prijevodu on je zadužen za obradu transakcije te premještanje novca s korisnikove banke u banku poduzeća. Obrada transakcije se izvršava u smjeru prikazanom na slici (Slika 4.2.). Korisnik inicira naplatu odabirom metode plaćanja te klikom na gumb za plaćanje. Tada sustav prodavača šalje podatke o transakciji *payment getaway* sustavu koji ih enkriptira i prosljeđuje procesoru na obradu. Procesor tada autorizira transakciju, kontaktira banku kupca i prodavača kako bi izvršio niz provjera: provjerava valjanost kartice kupca, datum isteka, ima li pokrića na računu za traženu uplatu, ID i valjanost prodavača, povijest transakcija (sumnjive radnje, pranje novca), dostupnost mreže te usklađenost s regulativama plaćanja. Ukoliko je sve u redu procesor prihvaća zahtjev te kontaktira banku kupca za umanjenje sredstva i banku prodavača za povećanje sredstava minus naknade transakcije. Nakon uspješne transakcije procesor prodavaču daje do znanja da je plaćanje uspješno provedeno, te prodavač označava narudžbu kao zaprimljenu.



Sl. 4.2. Tok obrade transakcija (autor: Mark Stiltner)

4.2.2. Regulative usklađenosti plaćanja

Kako bi sve radilo u skladu sa zakonom i sigurnosnim mjerama propisane su stroge regulative usklađenosti plaćanja (engl. *Payments Compliance*) koje povećavaju sigurnost i standardiziraju proces plaćanja. One se odnose na različita pravila, propise i zakone koji postavljaju i upravljaju najboljom praksom plaćanja. To uključuje niz standarda za različite aktivnosti plaćanja, uključujući olakšavanje transakcija, pohranjivanje podataka o plaćanju i praćenje prijevara. Najvažnije regulative u modernom internetskom plaćanju su:

1. PCI DSS – Kreiran je 2004. godine od strane pet glavnih tvrtki kreditnim karticama (Visa, Mastercard, Discover, JCB i American Express). PCI DSS (engl. *Payment Card Industry Data Security Standard*) je široko prihvaćeni skup politika i postupaka namijenjenih optimizaciji sigurnosti transakcija kreditnim, debitnim i gotovinskim karticama te zaštiti vlasnika kartica od zlouporabe njihovih osobnih podataka. Ova regulativa je osmišljena kako bi spriječila "curenja" osjetljivih podataka i smanjila rizik prijevare organizacija koje obrađuju podatke o karticama.

   Ovom regulativom, trgovci su podijeljeni u 4 razine ovisno o godišnjem broju transakcija koji odrade IPG-om ili fizičkim putem preko POS uređaja. Cilj ovakve podjele je da, što je veći broj obrađenih transakcija godišnje, mjere sigurnosti koje moraju zadovoljavati budu strože i zahtijevaju da trgovac bude tim ozbiljniji.

Godine 2022. izašla je trenutno najnovija verzija ove regulative PCI DSS v4.0 s podrškom za višestruku provjeru autentičnosti, zahtjevima za lozinku, obnovljenim standardima e-trgovine te s naglaskom na fleksibilnost i prilagodbu modernim metodama plaćanja.

2. PSD2 – Na snagu je stupila 2019. godine od Europske Unije. PSD2 (engl. *Payment Services Directive 2.0*) je regulativa koja definira standarde za stvaranje jedinstvenog i učinkovitijeg tržišta plaćanja u regiji. Cilj regulative je postići 4 glavna cilja:
    a. Doprinijeti integriranijem i učinkovitijem europskom tržištu plaćanja. Iako je svojstvena za EU svaka organizacija koja obavlja prekogranično poslovanje unutar regije također mora poštivati ovu regulativu.
    b. Dodatno izjednačiti uvjete za pružatelje usluga plaćanja uključivanjem novih trgovaca.
    c. Učiniti plaćanja jednostavnijim i sigurnijim.
    d. Poboljšati zaštitu europskih potrošača i poduzeća.
3. 3DS2 & SCA – 3DS2 (3D Secure 2.0) je višefaktorski protokol autentifikacije koji zahtijeva PCI DSS, a SCA (engl. *Strong Customer Authentication*) je zahtjev za provjeru identiteta PSD2. I 3DS2 i SCA imaju za cilj poboljšati i pojednostaviti proces autentifikacije korisnika, no 3DS2 nastoji pružiti učinkovitije i efikasnije rješenje za SCA.
4. KYC – KYC (engl. *know your customer*) u doslovnom prijevodu bi značilo upoznajte svog kupca/klijenta, što se odnosi na postupke koje pružatelji usluga plaćanja koriste da bi učinili upravo to, razumjeli tko su njihovi klijenti, čime se bave njihovi poslovi i prirodu rizika koji predstavljaju. Sastoji se od niza smjernica i propisa koje financijske usluge zahtijevaju za provjeru identiteta pojedinaca. KYC verifikacije su sve više u elektroničkom obliku (eKYC) te ih karakteriziraju provjere poput verifikacija putem osobne iskaznice, verifikacija lica, verifikacija kućanskih računa kao potvrde o boravištu, biometrijske verifikacije, itd. Nastavci na klasični KYC (ukoliko je korisnik druga tvrtka ili trgovac) su: KYCC (engl. *know your customer's customer)* te KYB (engl. *know your buisiness*).
5. AML – Regulativa koja se zajedno s KYC bori protiv pranja novca (engl. *Anty-money laundering*) te se odnosi na skup propisa i postupaka koje provode financijske institucije i obrađivači plaćanja radi otkrivanja, sprječavanja i prijavljivanja aktivnosti povezanih s pranjem novca. Pranje novca uključuje prikrivanje podrijetla nezakonito stečenih sredstava prikazivanjem legitimnih. Kriminalci iskorištavaju financijski sustav uvođenjem nezakonitih sredstava u legitimne kanale, čime im se omogućuje korištenje opranog novca bez izazivanja sumnje. AML i KYC propisi posebno su važni u plaćanjima kako bi se osigurala transparentnost, odgovornost i sigurnost. Pružatelji usluga plaćanja, uključujući

banke, operatere prijenosa novca i platforme za digitalno plaćanje, podliježu strogim obvezama AML-a i KYC-a kako bi spriječili da njihove usluge iskoriste kriminalci.

6. GDPR – Opća uredba o zaštiti podataka (engl. *General Data Protection Regulation*) stupila je na snagu 2018. godine te određuje koja su prava pojedinaca, a u skladu s tim i koje su obveze subjekata koji obrađuju osobne podatke poput voditelja obrade odnosno izvršitelja obrade. Odnosi se na sve subjekte koji obrađuju podatke građana EU, te propisuje da se podaci moraju obrađivati na zakonit, pošten i transparentan način, a pojedincima se mora dati pravo na pristup, ispravak, brisanje i prijenos svojih osobnih podataka.

4.2.3. Dostupni sustavi web naplate

Danas postoje stotine IPG sustava te je vrlo bitno odabrati onaj koji će pružati potrebnu podršku i robusnost za poduzeće koje se želi priključiti. Kod odabira treba imati na umu faktore poput: potencijalnog rasta poduzeća, sigurnosti transakcija, lakoće implementacije i održivosti.

Web sustavi naplate koji su trenutno vrlo popularni na ovim područjima:

1. CorvusPay – CorvusPay je hrvatska, vrlo pouzdana usluga web naplate koja posluje na području cijele EU te je podržana u gotovo svim balkanskim zemljama (BiH, Srbija, Kosovo, Makedonija). Nude razne kanale i metode plaćanja, lakoću integracije te konfiguraciju po potrebama poduzeća. Uz to pružaju i dodatne module za *in app* transakcije i usluge poput dodatnih prodajnih mjesta. IPG broji preko 3,500 aktivnih trgovaca. Cjenik je vrlo jednostavan, licenca za uslugu je 30€ mjesečno uz naknade po transakciji od 0.5%.
2. Monri – Monri je relativno nov hrvatski pružatelj usluga plaćanja (PSP[2]) osnovan 2023. koji je nastao objedinjenjem Remaris-a, Gastrobit-a i WSpay-a te je time postigao sva rješenja za plaćanje na jednom mjestu. Omogućuje spajanje prodajnog mjesta online shopa sa jednom ili više kartičarskih kuća i omogućuje naplatu korištenjem različitih kartičnih brandova te podršku za Aircash, Google Pay, IPS, KEKS Pay, PayPal i mnoge druge. Broje više od 10,000 trgovaca iako su cijene više nego kod konkurencije, mjesečna naknada korištenja iznosi 36€ te naknada po transakciji također 0.5%.

Vrste globalno popularnih i aktualnih web sustava naplate su:

[2] Engl. *Payment Service Provider* – Tvrtka koja poduzećima omogućava prihvaćanje elektroničkih plaćanja

1. 2Checkout – Platforma specijalno orijentirana za poduzeća, tj. poslovne korisnike koja pruža isključivo online bazirane kartične transakcije. S implementacijske strane je vrlo lagan za korištenje s obzirom na API koji ima podršku za veliku većinu programskih jezika. Nudi različite načine plaćanja te čak i podršku za PayPal transakcije. Cjenik se sastoji od 4 plana ovisno o potrebama, nema mjesečne naknadu ali zato malo više uzima prilikom transakcija. Minimalna cijena se kreće od 3.5% plus 0.30€ naknade po uspješnoj transakciji.
2. Stripe - Stripe je irsko-američki pružatelj usluga plaćanja koja trgovcima omogućuje prihvaćanje kreditnih i debitnih kartica ili drugih metoda plaćanja. Njihovo rješenje za obradu plaćanja najprikladnije je za tvrtke koje ostvaruju većinu svoje prodaje online, s obzirom da je većina njihovih značajki usmjerena upravo web naplati. Milijuni poduzeća aktivno koriste Stripe te ga to čini jednim od popularnijih IPG sustava. Minimalna naknada za korištenje iznosi 1.5% plus 0.25€ po transakciji za sve Europske kartice.
3. Paypal - PayPal jedan je od najpopularnijih i prihvaćenijih online sustava naplate na svijetu. Ima više od 325 milijuna računa, dostupan je u preko 200 zemalja te podržava 25 valuta diljem svijeta. Paypal je orijentiran i za poduzeća ali i za osobnu upotrebu. Omogućuje vrlo jednostavan besplatan API koji omogućava laku integraciju. Kod kupovine kao korisnik ne postoje naknade, jedino kod prodavanja proizvoda kao trgovac. Tada naknade za web plaćanja iznose 2.59% plus 0.39€ po transakciji.
4. Square - američki financijski servis razvijen 2009. kojemu su primarni cilj mala do srednja poduzeća. Prvi su razvili mogućnost plaćanja preko 3.5mm utora na mobilnim uređajima (engl. *Square Reader*). Koriste samo američki dolar kao valutu te se sve ostale valute pretvaraju u dolare prilikom kupnje/prodaje, naravno, uz naknade. Nemaju mjesečnu pretplatu nego samo naknade za transakcije koje iznose 2.9% plus 0.15€ po transakciji. Pružaju vrlo detaljan besplatan API koji je podržan od strane mnogo programskih okvira.

Prema [11], na uzorku od više od milijun internetskih stranica 2018, godine, PayPal je bio daleko najveći globalni lider u sustavima web naplate. Poredak na tržištu te godine prikazan je tablicom 4.1.

Tablica 4.1. Tržišni udio svjetskih sustava naplate (Izvor: Datanyze, 2018)

No.	Sustav naplate	Broj domena	Tržišni udio
1	PayPal	778,385	72.99%
2	Stripe	113,132	10.61%
3	Square	20,644	1.94%
4	Authorize.net	18,811	1.76%
5	Amazon Pay	17,716	1.66%
6	Klarna	16,623	1.56%
7	CCBill	11,923	1.12%
8	Braintree	11,735	1.10%
9	Google Checkout	10,654	1%
10	WorldPay	5,125	0.48%

5. TEHNIČKA OSNOVA TAKVIH SUSTAVA

Sustavi plaćanja su jedni od najsloženijih i najvažnijih softverskih sustava zbog velike potrebe za sigurnosti, dostupnosti, velikog broja korisnika i nerijetkih hakerskih napada. Cilj ovog poglavlja nije ići u „*State of the art*“[3] dubinu problema već opis tehničke podloge koja sačinjava svaki sustav moderne web naplate. Radi lakšeg razumijevanja razmatrat će se sustav plaćanja koji nije potpuno funkcionalan, nema implementirane sve sigurnosne mjere te nije usklađen s regulativama u modernom plaćanju opisane u poglavlju 4.2.2, no dovoljan je za objasniti tehnička svojstva. Takav jednostavni sustav se sastoji od 5 aktera koji međusobno razmjenjuju podatke s ciljem uspješnog provođenja transakcije. Radi se o **distribuiranom** sustavu koji koristi heterogene komunikacijske protokole za razmjenu informacija od kojih su neki dobro poznati poput REST (engl. *REpresentational State Transfer*) ili SOAP (engl. *Simple Object Access Protocol*), do nekih manje popularnih poput ISO 20022 i ISO 15022.

Prvi akter je klijentska aplikacija, u koju korisnik unosi podatke za plaćanje. Na primjeru pseudokoda 5.1. vidljiv je prikaz tvrdo kodiranih podataka *podaciKartica* i *podaciPlacanja* koje bi se u realnom okruženju dobile s frontenda neke web aplikacije, mobilne aplikacije ili čak preko POS uređaja. *PodaciPlacanja* sadrži sve bitne podatke za provođenje plaćanja te predstavlja zahtjev (engl. *request*) u ovom sustavu a njegovi sastavni dijelovi su:

- platitelj - sadrži sve osnovne podatke o fizičkoj ili pravnoj osobi koja inicira plaćanje poput naziva (ime i prezime), adrese, načina plaćanja, ibana kartice...
- primatelj - sadrži podatke o primatelju poput naziva primatelja i ibana.
- transakcija - definira valutu i iznos koji se planira provesti transakcijom.
- token - niz znakova koji predstavljaju informacije o kartičnim podacima platitelja

PodaciKartica sadrže sve osjetljive podatke o kartici koji se koriste za plaćanje i vrlo je bitno da ne idu po mreži u formatu koji je lako čitljiv. U tu svrhu se vrši proces tokenizacije [12]. Sama tokenizacija se odvija u servisu za tokenizaciju (engl. *Tokenization service provider*). Osjetljivi kartični podaci se šalju HTTPS protokolom, najčešće tokenizacijski servisi koriste neki vid REST API-ja preko kojeg se omogućava samo ovlaštenim stranama pristup tokenizaciji i detokenizaciji. Naravno, sama tokenizacija podrazumjeva slanje osjetljivih podatka nekom provjerenom servisu

[3] Stanje tehnike - razina stanja tehničkih mogućnosti u danome trenutku koja se odnosi na proizvode, procese i usluge

kojem se vjeruje u toj mjeri da se radije podaci šalju tom servisu na tokenizaciju nego da se koristi bazično kriptiranje HTTPS protokola. Važno je napomenuti da tokenizacija nije kriptiranje ili hashiranje te se sam token ne može povezati s kartičnim podacima izvan sustava za tokeniziranje. Kad servisu stigne zahtjev ne obradu s kartičnim podacima, on ga sprema u bazu te baza generira određeni ključ s kojim može pristupiti tim podacima, nešto poput hash tablice u programskim jezicima. Isto tako i u suprotnom smjeru, stigne li zahtjev za detokenizaciju s određenim tokenom, servis vrati ono što se nalazi u bazi (*hash* tablici) pod tim tokenom, no naravno tek ako je druga strana dokazala svoju legitimnost i ima prava na tražene podatke.

Linija ***Kod***

```
1:      podaciKartica = {
2:        broj_kartice: 4780135462178981,
3:        cvv: 556,
4:        datum_isteka: 05/28
5:      }
6:      token = tokenizeService.tokenizeCardDetails(podaciKartica);
7:      podaciPlacanja = {
8:        platitelj: {
9:            naziv: "Ivan Horvat",
10:           nacin_placanja: "kratica",
11:           iban: "HR9124840086111874136",
12:           adresa_platitelja: "Zagrebacka 4, 3100, Osijek, Hrvatska"
13:       }
14:     primatelj: {
15:           naziv: "E-payment d.o.o",
16:           iban: "HR4624840081755513281"
17:       }
18:     transakcija: {
19:           iznos: 100.00,
20:           valuta: "EUR"
21:       }
22:     token: token
23:     }
24:
26:     getaway.submit(podaciPlacanja);
```

Sl. 5.1. Primjer koda za klijentsku aplikaciju

Nakon što je aplikacija dobila token u daljnjem procesu će isključivo njega koristiti za slanje *getaway-u*, zajedno s ostatkom podataka o plaćanju. Funkcija *getaway.sumbit()* šalje zahtjev IPG sustavu za obradu transakcije i povratno očekuje poruku (status kod) o uspješnoj transakciji ili pogrešci, tu staje funkcionalnost klijentske aplikacije.

Drugi akter predstavlja IPG ili samo *getaway*. S obzirom da je sustav obrade tj. procesor odvojen od samog sustava upravljanja u ovom primjeru koda je također razdvojen na dvije cjeline. Vidljivo je iz koda 5.2. da *getaway* vrši samo upravljanje zadacima koje procesor mora obraditi. Naravno, u stvarnim okruženjima *getaway* ima više funkcija nego na ovom primjeru, neke od njih su autorizacija i sigurnost, višezadaćnost (engl. *multitasking*), istovremeno izvršavanje (*engl.*

Linija Kod

```
1:    function submit(podaciPlacanja) {
2:      processor.authorize(podaciPlacanja);
3:      processor.reduceBalance(podaciPlacanja.platitelj, podaciPlacanja.iznos
4:      processor.addBalance(podaciPlacanja.primatelj, podaciPlacanja.iznos);
5:      return Success("Plaćanje uspješno provedeno!");
6:    }
```

Sl. 5.2. Primjer koda za getaway

concurrency) te raspoređivanje opterećenja (*engl. load balancing*). Nakon poziva svake od procesorskih funkcija procesor bi vratio poruku o grešci te bi se odgovor (*engl. response*) vratio korisniku, ali ako sve prođe u redu, *getaway* vraća uspješnu poruku o plaćanju.

Treći akter je prethodno spomenuti procesor čija je funkcija detaljnije objašnjena u poglavlju 4.2.1. Ukratko procesor obrađuje sve što mu upravljački dio sustava zada. Najčešće to ide redom gdje se prvo vrši autorizacija točnosti kartičnih podataka i iznosa na računu, zatim se umanjuje iznos s platiteljevog računa i povećava iznos s primateljevog. Procesor prilikom autorizacije detokenizira kartične podatke pozivanjem servisa za tokeniziranje, odrađuje niz provjera valjanosti nad kartičnim podacima dok je u pseudokodu 5.3. pokazana samo provjera broja kartice pomoću *Luhn-ovog checksum* algoritma [13].

Linija Kod

```
1:    function authorize(podaciPlacanja) {
2:      podaciKartica = tokenizeService.getCardDetails(podaciPlacanja.token);
4:      if(!LuhnChecksum.validate(podaciPlacanja.broj_kartice)) {
5:        return Error("Neispravan broj kartice");
6:      }
7:      buyerBankService.validateCardDetails(podaciKartica);
        availableBalance =
9:    buyerBankService.getAvailableBalance(podaciPlacanja.platitelj.iban);
10:     if(availableBalance < podaciPlacanja.transakcija.iznos) {
11:       return Error("Nedovoljan iznos za uspješno plaćanje");
12:     }
13:     sellerBankService.checkSellerValidity(podaciPlacanja.primatelj);
15:   }
17:   function reduceBalance(platitelj, iznos) {
18:     buyerBankService.reduceBalance(platitelj, iznos);
20:   }
22:   function addBalance(primatelj, iznos) {
23:     sellerBankService.addBalance(primatelj, iznos);
24:   }
```

Sl. 5.3. Primjer koda za payment processor

Zatim odrađuje komunikaciju s bankom platitelja kako bi provjerio detalje o kartičnim podacima, dohvatio podatke o iznosu na računu platitelja. Potom vrši provjeru ima li platitelj dovoljno sredstava za traženu transakciju. Između ostalog, u jednom trenutku prije autorizacije provjerava i validnost (engl. *Validity*) samog računa primatelja za slučaj da se radi o prevarantima. Procesor također pruža i funkcije za povećanje i smanjenje iznosa na računu gdje u stvarnosti posao, u vidu zahtjeva, delegira na konkretnu banku koja potom ažurira sredstva na računu. Važno je napomenuti da se svaka implementacija procesora za obradu transakcija razlikuje te je ovo samo primjer strukture i toka podataka koji treba uvažiti s dozom opreza ukoliko se planira implementirati ovakav sustav.

Posljednja dva aktera su servisi banki platitelja i primatelja. One pružaju funkcije koje procesor obrade koristi kako bi provjerio ispravnost kartičnih podataka, provjerio iznos na računu te povećao ili umanjio iznose s istih (Slika 5.4.).

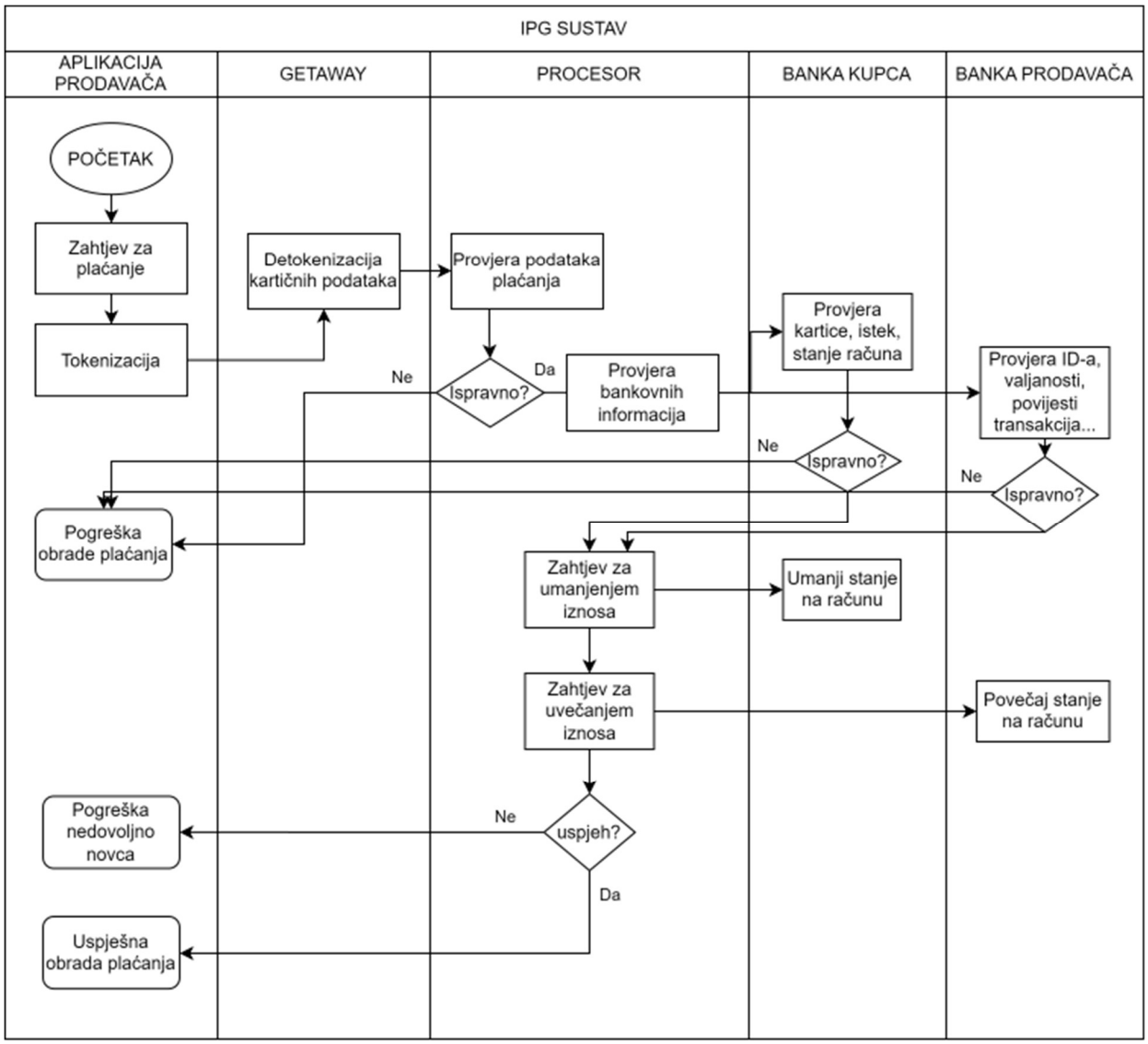
Linija Kod

```
1:    function validateCardDetails(podaciKartica) {
2:      if(!database.existsCardNumber(podaciKartica.broj_kartice)) {
3:        Error("Kartica se ne nalazi u sustavu");
4:      }
5:      if(!database.checkCVV(podaciKartica.cvv)) {
6:        Error("Kontrolni broj kartice nije ispravan");
7:      }
8:      if(podaciKartica.datum_isteka < Date.now()) {
9:        return Error("Kartica je istekla");
10:     }
11:   }
12:   function reduceBalance(platitelj, iznos) {
13:     availableBalance = getAvailableBalance(platitelj.iban);
14:     database.setAvailableBalance(platitelj.iban, availableBalance - iznos)
15:   }
16:   function addBalance(primatelj, iznos) {
17:     availableBalance = getAvailableBalance(primatelj.iban);
23:     database.setAvailableBalance(primatelj.iban, availableBalance + iznos)
24:   }
26:   function getAvailableBalance(iban) {
27:     database.getAvailableBalance(iban);
38:   }
39:   function checkSellerValidity(primatelj) {
        //check transactionHistory, IP address checks, multiple submits,
40:   unusual requests, credentials, email adresses etc...
41:   }
```

Sl. 5.4. Primjer koda za bankarski servis

Konkretno, banka platitelja kod provjere ispravnosti kartice provjerava u bazi postoji li traženi broj kartice, odgovara li CVV kontrolni broj (*engl. Card Verification Value*) i dali je kartica datumski validna, odnosno da nije istekla. Funkcije *addBalance()* i *removeBalance()* također pristupaju bazi i postavljaju iznos na računu na onaj argumentom proslijeđen. Funkcija *checkSellerValidity()* provjerava ima li osoba koja prima sredstva na račun zabrinjavajuću povijest transakcija, postoji li mnogo uplata s različitih IP adresa u kratkom vremenu i sve ostale metode zaštite korisnika koji uplaćuje novac poput: pranja novca, raznih prevara, pokušaja iskorištavanja rupa u sustavu (engl. *exploits*) i dr.

Na dijagramu staze za plivanje (engl. *Swimlane diagram*) na slici 5.5. moguće je vidjeti grafički prikaz povezanosti ovakvog sustava sa svim ranije objašnjenim komponentama.



Sl. 5.5. Prikaz strukture IPG sustava

6. INTEGRACIJA SUSTAVA ZA NAPLATU

Integracija sustava za naplatu kompleksan je proces te, krenuti od nule u implementaciju istog zahtijeva niz procedura i koraka kako bi, prvenstveno zakonski bili u mogućnosti vršiti takvu uslugu. Neki od koraka koji se trebaju razmotriti prilikom implementacije vlastitog IPG sustava bi obuhvaćali:

- Izradu trgovačkog računa (engl. *merchant account*) u banci. To je tip računa koji omogućava poduzećima da prihvaćaju i procesiraju elektroničke i kartične transakcije
- Dopuštenje financijskih institucija ili banaka koje je postavljeno nizom regulativa usklađenosti plaćanja i standarda za sigurnu obradu transakcija (PCI DSS) koji su detaljnije objašnjeni u poglavlju 4.2.2.
- Izradu ili kupovinu sigurnog softvera za procesiranje plaćanja (IPG) koji bi obrađivao elektroničke transakcije, vršio zaštitu korisničkih podataka te vršio komunikaciju s bankama i financijskim ustanovama. Softver bi morao biti robustan, siguran i u skladu s regulativama.
- Postavljanje vlastitog ili zakup servera koji bi bio poslužitelj za IPG servis, na kojem bi se pokretao prethodno navedeni softver za procesiranje plaćanja.
- Izradu korisničke aplikacija (web aplikacije) te sučelja intuitivnog i jednostavnog za upotrebu od strane korisnika. Osigurati podršku za veći broj kartica i načina plaćanja te različite valute u slučaju internacionalnog trgovanja.
- Razmatranje infrastrukture za rješavanje problema s korisničkom službom koji se odnose na plaćanja, kao što su povrati novca, sporovi i kartični povrati sredstava[4] (engl. *chargeback*) [14]. Uspostava jasnih politika i postupaka za te scenarije ključna je za održavanje povjerenja i pouzdanosti.
- Provedbu mehanizama za otkrivanje i sprečavanje prijevara radi zaštite od neovlaštenih transakcija i mogućih povreda sigurnosti.

[4] *Chargeback* pokreće potrošač kako bi osporio troškove koji terete njegovu karticu. Najčešće se koristi kada potrošač nije zaprimio proizvod ili uslugu, nije zaprimio u ugovorenom roku ili nije zadovoljan pruženom uslugom ili kupljenom robom.

- Konstantno unapređivanje sustava kroz ažuriranja i nadogradnje, ostajanje u toku s izmjenama regulativa i tehnologija kako bi IPG ostao usklađen i konkurentan te kako bi sve funkcioniralo ispravno i sigurno.

Neki od prethodno nabrojanih koraka pružaju prepreke ovom radu u implementaciji IPG-a „od nule“ kako s pravnih/zakonskih tako i s financijskih gledišta. Primjerice, otvaranje *merchant* računa podrazumijeva plaćanje mjesečnih naknada izdavačkoj banci, certifikati i regulative se plaćaju tijelima za kontroliranje, održavanja servera i prometa na mreži i drugo. Vrlo brzo se dolazi do vrlo velike brojke koju nije lako točno odrediti, no prema [15] gruba procjena cijene za izradu minimalno održivog proizvoda (engl. *Minimum Viable Product*) iznosi između 200 i 250 tisuća dolara.

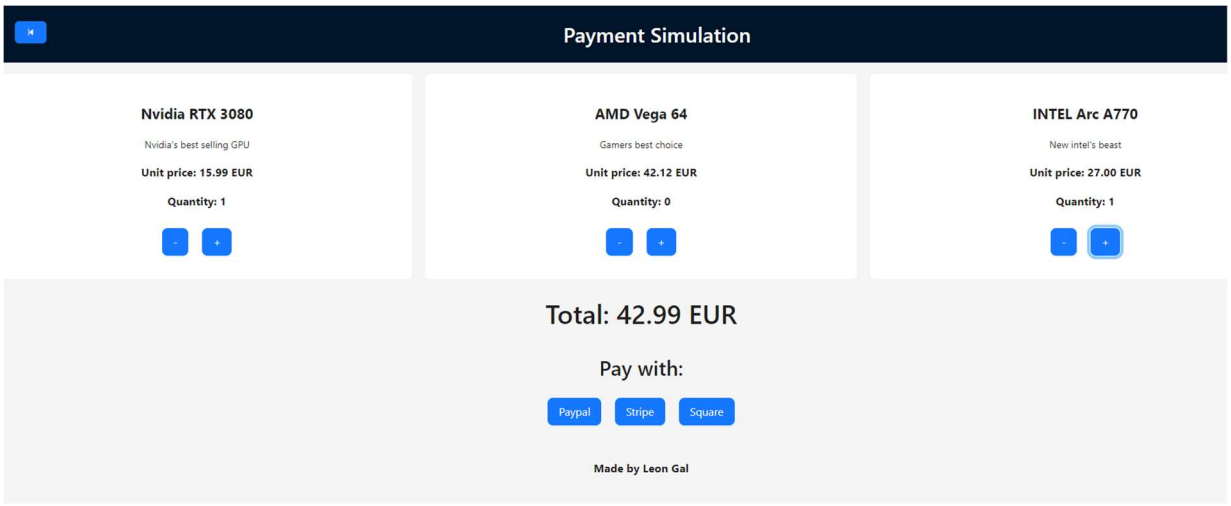
Implementacija sustava za naplatu vrši se na način da se izradi programski kod koji koristi postojeće IPG API-je (engl. *Application Programming Interface*), na njih se spaja te preko njih vrši autorizaciju i obradu transakcija. Implantirana su 3 API-ja od 3 različita IPG davatelja usluga (engl. *provider*). Korišteni su API-ji svjetski poznatih sustava naplate, to su Paypal, Stripe i Square. Detaljnije pojedinost i o njima su navedene u poglavlju 4.2.3. Odabrani su prvenstveno iz razloga što svaki od njih nudi niz vrlo robusnih i besplatnih API-ja koji pružaju niz funkcionalnosti, poput kreiranja naloga (engl. *Invoicing*), praćenja narudžbi (engl. *Shipment Tracking*), uspostavu token načina plaćanja (engl. *Payment Method Tokens*) i dr. Koriste se njihovi ponuđeni SDK-ovi (engl. *Software Development Kit*) koji će olakšati samu sintaksu pisanja zahtjeva (engl. *request*) na *backend*-u. Odabrana je implementacija naplate na način da se implementira funkcionalnost poveznice za plaćanje (engl. *Payment Link*). Poveznica za plaćanje funkcionira na način da, nakon što korisnik klikne na gumb za plaćanje, prodavačev softver kreira narudžbu (engl. *Order*) sa proizvodima (engl. *Item*) te s njom šalje zahtjev IPG API-ju koji mu vraća poveznicu na koju treba preusmjeriti korisnika i na kojoj korisnik može obaviti plaćanje. Po završetku plaćanja API preusmjerava korisnika na željenu putanju koja se definira u zahtjevu uz narudžbu.

Od korištenih tehnologija, za komunikaciju s API-jima, donosno za *backend* dio projekta se koristit programski jezik Java i alat za upravljanje projektima Maven. Sam *backend* koristi programski okvir (engl. *framework*) Spring Boot koji se vrti na Apache Tomcat web serveru. Uređivač (engl. *editor*) za *backend* dio korišten je IDE (engl. *Integrated Development Enviroment*) Spring Tools Suite 4 (STS4). Java je izbor zbog svoje pouzdanosti, široke primjene u razvoju poslovnih aplikacija i otvorenosti koda (engl. *Open source*), kao i web server te korišteni uređivač.

Frontend dio projekta je razvijen koristeći React JS, popularnu JavaScript biblioteku za izgradnju korisničkog sučelja, odnosno *frontend*-a [16], zajedno s Ant Design, dizajnerskim okvirom koji nudi bogat set komponenti. Kao razvojno okruženje za *frontend* odabran je Visual Studio Code uređivač koji se ističe svojom besplatnosti, brzinom i integracijom s brojnim dodacima koji olakšavaju razvoj.

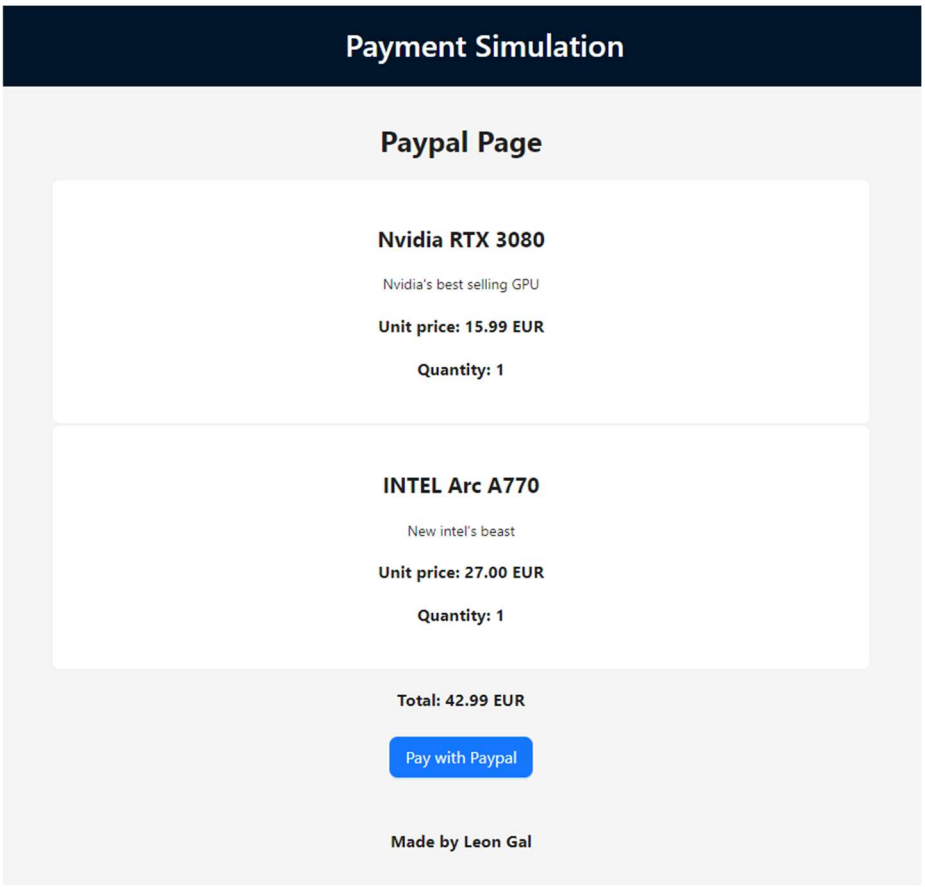
6.1. Arhitektura sustava i modeli

Ulazna točka u sustav je *frontend* preko kojeg klijent šalje zahtjeve. Odabiranje stavki sa slike 6.1. dobiva se izračunata ukupna cijena u eurima te se odabire jedan od gumbova za nastavak procedure plaćanja s odabranim pružateljem IPG usluge.



Sl. 6.1. Izgled stranice za odabir proizvoda

Potom se prelazi na stanicu za pregled detalja „računa“ i potvrđivanje za nastavak s naplatom. Nakon klika na gumb „*Pay with PayPal/Stripe/Square*“, slika 6.2., pokreće se obrada na *backend*-u koja generira poveznicu na koju se korisnika preusmjerava kako bi unio kartične podatke i proveo plaćanje. Poveznica na koju *backend* preusmjerava korisnika je specifična za svaki IPG te će biti detaljnije objašnjena svaka u narednim poglavljima.



Sl. 6.2. Izgled pregleda detalja računa

Implementacija je podijeljena na način da kontroler prima zahtjeve s *frontend*-a te ih prosljeđuje servisu koji komunicira s API-jima, vraća kontroleru odgovore, koje zatim kontroler vraća korisniku i po potrebi preusmjerava na potrebne stanice. Arhitektura koja se koristi u projektu je *Controller-Service-Repository* (CSR) koja je vrlo popularna u Spring Boot aplikacijama, no u ovom slučaju se ne koristi baza podataka pa nije potreban repozitorij dio. Ovakvim pristupom se vrši razdvajanje odgovornosti na viši sloj, kontroler, koji brine o izlaganju funkcionalnosti i REST pozivima te na niži sloj, servis, koji brine o poslovnoj logici i konkretnoj implementaciji.

S obzirom da se ne koristi baza, modeli sustava su zapravo DTO-ovi (engl. D*ata Transfer Object*) te se koriste kod zaprimanja zahtjeva u kontroleru i obrade unutar servisa. Jedina dva modela koja se koriste su narudžba (*Order*) i stavka (*Item*). Izgled klasa modela je prikazan na slici 6.3.

Order klasa se koristi za skladištenje podataka o narudžbi poput namjere (*intent*), načina plaćanja (*payment_method*), valute (*currency*), opisa plaćanja (*description*) i poruke za platitelja (*note_to_payer*). *Order* klasa također sadrži listu *Item*-a od kojih svaki sadrži podatke o pojedinoj stavci za koju se planira odraditi transakcija. Ti su podaci, naziv stavke (*name*), količina (*quantity*),

jedinična cijena (*price*) te opcionalni opis stavke (*description*). Metoda *getTotalPrice()* računa ukupnu cijenu cijele narudžbe, a metoda *getTotalAmount()* računa cijenu uzevši u obzir i količinu stavki.

```
@Data
public class Order implements Serializable {

	@NotNull
	private String intent;
	@NotNull
	private String paymentMethod;
	@NotNull
	private String currency;
	private String description;
	private List<Item> items;
	private String noteToPayer;


	public Double getTotalPrice() {
	  Double total = 0.0;
	  for (Item item : items) {
	       total += item.getTotalAmount();
	  }
	  return total;
	}
}
```

```
@Data
public class Item implements Serializable {

	private String name;
	private String description;
	@NotNull
	private Integer quantity;
	@NotNull
	private Double price;

	public Double getTotalAmount() {
		return price*quantity;
	}
}
```

Sl. 6.3. Modeli korišteni u sustavu

Sami koraci integracije su vrlo slični za svakog pojedinog pružatelja IPG usluge te se ugrubo proces integracije i povezivanja osobne aplikacije može svesti na ovih 6 koraka:

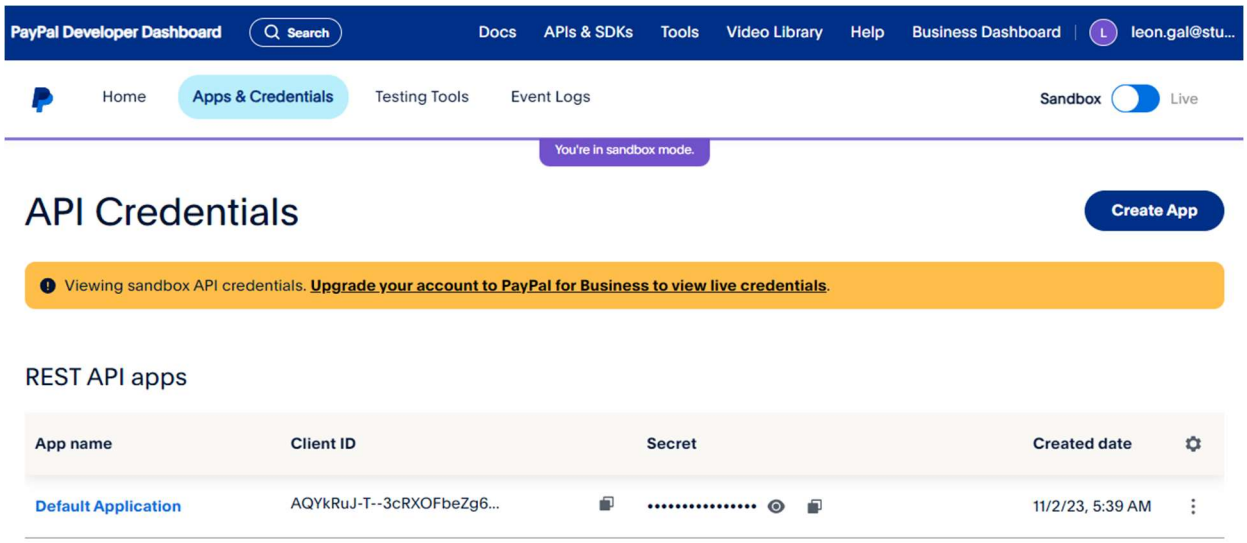
1. Registracija na *provider* stranici
2. Generiranje javnih/privatnih ključeva
3. Uvođenje biblioteka *provider*-a u projekt
4. Postavljanje vjerodajnica na server
5. Implementacija servisa za komunikaciju s API-jima *provider*-a
6. Testiranje sustava nekim od dostupnih alata

Svaki korak je detaljnije opisan u narednim poglavljima.

6.2. Integracija s PayPal API-jem

Za integraciju s PayPal sustavom plaćanja potrebno je izraditi razvojni račun (engl. *developer account*) na službenoj PayPal stranici za razvojne programere [17]. Prilikom izrade računa automatski se dodjeljuje i inicijalna aplikacija unutar koje se izrađuje testni trgovački račun (engl. *sandbox merchant account*) te vjerodajnice potrebne za programsko povezivanje s API-jem.

Naknadno je potrebno izraditi i osobni testni račun preko kojeg se vrše uplate u svrhu testiranja. Izgled stanice za developere na kojoj se mogu vidjeti i izraditi vjerodajnice prikazan je na slici 6.3.



Sl. 6.3. Izgled stranice s vjerodajnicama

Za prelazak na produkcijski (engl. *live*) način rada, potrebno je nadograditi račun unošenjem osobnih informacija i potvrditi račun elektroničkom poštom.

6.2.1. Uvođenje ovisnosti u projekt i konfiguracija

Najprije je potrebno uvesti SDK u projekt. Ovisnosti (engl. *dependencies*) projekta se dodaju u *pom.xml* datoteku, a s obzirom da se koristi Maven za upravljanje paketima sa službene stranice Maven Repozitorija [18] se dohvaćaju ovisnosti o željenim SDK-ovima, konkretno u ovom poglavlju Paypal API SDK čija je deklaracija prikazana primjerom XML-a 6.4.

Linija Kod

```
1:    <dependency>
2:        <groupId>com.paypal.sdk</groupId>
4:        <artifactId>rest-api-sdk</artifactId>
5:        <version>1.14.0</version>
6:    </dependency>
```

Sl. 6.4. Primjer XML koda za dodavanje Paypal ovisnosti u projekt

Kako bi SDK dobio vjerodajnice o računu na koji se šalju zahtjevi potrebno je definirati kontekst. Kontekst se sastoji od 3 svojstva (engl. *property*) koja se zbog sigurnosti, lakše izmjenjivosti i alata za verzioniranje čuvaju u zasebnoj datoteci naziva *application.properties*. Praksa je da se u toj datoteci čuvaju rijetko promjenjive konstante aplikacije kojima se potom može pristupati u bilo kojem dijelu koda. Kao što je vidljivo na slici 6.5. za postavljanje konteksta potrebna su 3 podatka:

- *mode*: način rada aplikacije, može biti *sandbox* ili *production*
- *client_id*: Koristi se za identificiranje aplikacije, potreban za uspješno povezivanje na klijentov račun.
- *client_secret*: Provjerava autentičnost ID-a klijenta.

Ovi podaci se čuvaju u navedenoj vanjskoj datoteci te se na servise za verzioniranje koda objavljuju testni/lažni (engl. *mock*) podaci jer originalni moraju ostati tajni. Prednost ovog načina je što su testni i produkcijski načini vrlo lako izmjenjivi, zamjenom *properties* datoteke.

```
#paypal
paypal.mode=sandbox
paypal.client.id=AQYkRuJ-T--3cRXOFbeZg6E-V-btFACipoWyjg_6oKcY9KNpuzGvhwTlwnA1Oljb77
YcKxa4GhJzgD86
paypal.client.secret=EPKIRjzqv4jZeUAXLFRkSmMu4VxZ_t8vwO1xJKMpj8Pcf90PplGLSZ8cHdd21B5
ZTZKgO0h7x68yM510
```

Sl. 6.5. Paypal testni podaci unutar *application.properties* datoteke

Zatim konfiguracijska klasa na slici 6.6. povlači podatke iz *properties* datoteke direktno u konstruktor koji potom kreira objekt *APIContext* tipa preko kojeg SDK samostalno brine o slanju informacija iz konteksta API-ju prilikom svakog zahtjeva.

```
@Configuration
public class PaypalConfig {

	@Value("${paypal.client.id}")
	private String clientId;
	@Value("${paypal.client.secret}")
	private String clientSecret;
	@Value("${paypal.mode}")
	private String mode;

	@Bean
	public Map<String, String> paypalSdkConfig() {
		Map<String, String> configMap = new HashMap<>();
		configMap.put("mode", mode);
		return configMap;
	}

	@Bean
	public OAuthTokenCredential oAuthTokenCredential() {
		return new OAuthTokenCredential(clientId, clientSecret, paypalSdkConfig());
	}

	@Bean
	public APIContext apiContext() throws PayPalRESTException {
		APIContext context = new APIContext(oAuthTokenCredential().getAccessToken());
		context.setConfigurationMap(paypalSdkConfig());
		return context;
	}
}
```

Sl. 6.6. Paypal konfiguracijska klasa

Od verzije 1.8.0 predefinirani (engl. *deafault*) konstruktor te još par implementacija konstruktora klase *ApiContext* su postali zastarjeli (engl. *depricated*) [19]. Iako ih je moguće koristiti u zastarjelom načinu implementacije koji je prikazan na prethodnoj slici, predlaže se refaktoriranje (engl. *refactoring*) koda na način prikazan na slici 6.7. Naravno, važno je napomenuti da je sustav testiran i radi ispravno u oba slučaja te da zastarijevanje koda u kontekstu ovog rada nije važno koliko je bitno u pravom, poslovnom okruženju gdje se vrši produkcija, prevencija sigurnosnih propusta, dizanje verzija SDK-a i dugoročno održavanje sustava.

```
@Configuration
public class PaypalConfig {

	@Value("${paypal.client.id}")
	private String clientId;
	@Value("${paypal.client.secret}")
	private String clientSecret;
	@Value("${paypal.mode}")
	private String mode;

	@Bean
	public APIContext apiContext() {
		return new APIContext(clientId, clientSecret, mode);
	}
}
```

Sl. 6.7. Prikaz refaktoriranog aktualnog koda

6.2.2. Kontroler i servis

Paypal kontroler se sastoji od 3 metode raspoređene na 3 rute. Sve rute počinju s */api/paypal* prefiksom kako bi se funkcijski odijelile od ostalih *provider*-a. Rute na metodama prikazane slikom 6.8. su:

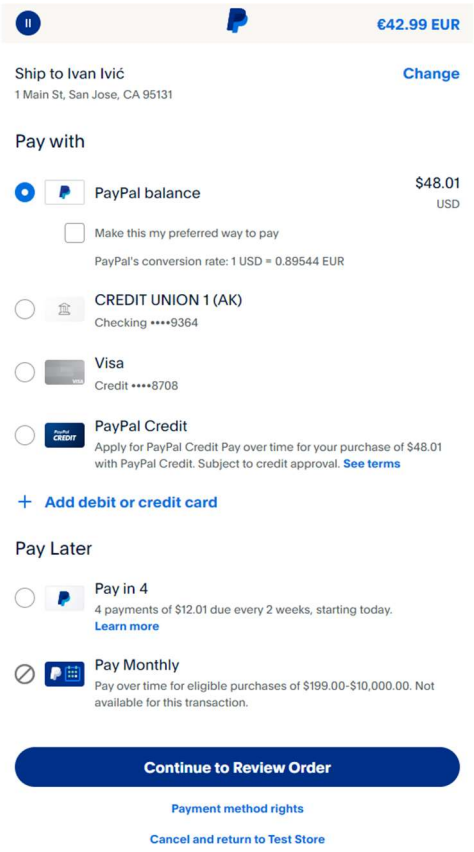
1. */pay* – Glavna POST ruta koja kreira i obrađuje PayPal plaćanje, te po uspješnom završetku vraća poveznicu na platnu stranicu.
2. */pay/success* – GET ruta koja izvršava ili dovršava PayPal plaćanje koje je platitelj odobrio, po završetku preusmjerava na *frontend* stranicu aplikacije.
3. */pay/cancel* – GET ruta koja se poziva ukoliko korisnik odustane od plaćanja, preusmjerava natrag na *frontend* stanicu aplikacije.

```
@Slf4j
@RestController
@RequiredArgsConstructor
@RequestMapping("api/paypal")
public class PaypalController {

  private final PaypalService service;

  @Value("${base.url}")
  private String baseUrl;
  @Value("${front.url}")
  private String frontUrl;
  public static final String SUCCESS_URL = "paypal/pay/success";
  public static final String CANCEL_URL = "paypal/pay/cancel";

  @PostMapping("/pay")
  public ResponseEntity<String> payment(@RequestBody @Validated Order order) {
    try {
      Payment payment = service.createPayment(order, baseUrl + CANCEL_URL, baseUrl +
SUCCESS_URL);
      log.info(payment.toJSON());
      return ResponseEntity.ok(payment.toJSON());
    } catch (PayPalRESTException e) {
      e.printStackTrace();
      return ResponseEntity.internalServerError().build();
    }
  }

  @GetMapping(value = "pay/cancel")
  public void cancelPay(HttpServletResponse resp) {
    resp.setHeader(HttpHeaders.LOCATION, frontUrl);
    resp.setStatus(HttpStatus.FOUND.value());
  }

  @GetMapping(value = "pay/success")
  public void successPay(HttpServletResponse resp, @RequestParam("paymentId")
       String paymentId, @RequestParam("PayerID") String payerId) {
    try {
      Payment payment = service.executePayment(paymentId, payerId);
      log.info(payment.toJSON());
      if (payment.getState().equals("approved")) {
        resp.setHeader(HttpHeaders.LOCATION, frontUrl + "?paymentId=" + paymentId);
        resp.setStatus(HttpStatus.FOUND.value());
      }
    } catch (PayPalRESTException e) {
      log.error(e.getMessage());
    }
  }
}
```

Sl. 6.8. Paypal kontroler

Tok podataka funkcionira tako da se nakon klijentovog zahtjeva prvo okine ruta */pay* s *Order* zahtjevom, ona kreira nalog za plaćanje te se korisnika preusmjerava na PayPal-ovu stranicu za plaćanje. Klijent mora imati PayPal račun kako bi mogao nastaviti s plaćanjem, jer je potreban login na stranicu. Potom klijent odabire način plaćanja, uređuje podatke za dostavu i informacije o kartici te potvrđuje plaćanje. PayPal potom nakon obrade plaćanja okida rutu */pay/success* koja korisnika preusmjerava natrag na stranicu s porukom o uspješnom plaćanju. U bilo kojem trenutku

korisnik može odustati od plaćanja te se poziva ruta */pay/cancel* koja vraća korisnika na početnu stranicu ali bez ikakve poruke. Izgled PayPal-ove platne stranice prikazan je na slici 6.9.



Sl. 6.9. Izgled Paypal stranice za naplatu

Vidljivo je na slici da je dolazna valuta u EUR, upravo zato što testna trgovina koristi eure i šalje valutu u eurima, no PayPal prima kao valutu plaćanja američke dolare. Paypal vrši konverziju na valutu trgovca, u ovom slučaju testni *bussiness account* ima US lokaciju i inicijalnu valutu USD.

Prilikom kreiranja *request*-a poveznice za plaćanje, SDK odrađuje samostalno mapiranje objekata u sam *request*. Ovakav način omogućava lako čitljivu, lako izmjenjivu programsku sintaksu prikazanu na slici 6.10. koja ne uključuje manipulaciju čistim *request*-om.

```
public Payment createPayment(Order order, String cancelUrl, String successUrl) throws
PayPalRESTException {
	Amount amount = new Amount();
	amount.setCurrency(order.getCurrency());
	Double total = BigDecimal.valueOf(order.getTotalPrice()).setScale(2,
RoundingMode.HALF_UP).doubleValue();
	amount.setTotal(String.format(Locale.US, "%.2f", total));

	Transaction transaction = new Transaction();
	transaction.setDescription(order.getDescription());
	transaction.setAmount(amount);

	List<Item> items = new ArrayList<>();
	for (com.example.demo.models.Item item : order.getItems()) {
		items.add(new Item(item.getName(), item.getQuantity().toString(),
item.getPrice().toString(), order.getCurrency()));
	}
	ItemList itemList = new ItemList();
	itemList.setItems(items);
	transaction.setItemList(itemList);

	List<Transaction> transactions = new ArrayList<>();
	transactions.add(transaction);

	Payer payer = new Payer();
	payer.setPaymentMethod(order.getPaymentMethod());

	Payment payment = new Payment();
	payment.setIntent(order.getIntent());
	payment.setPayer(payer);
	payment.setNoteToPayer(order.getNoteToPayer());
	payment.setTransactions(transactions);
	RedirectUrls redirectUrls = new RedirectUrls();
	redirectUrls.setCancelUrl(cancelUrl);
	redirectUrls.setReturnUrl(successUrl);
	payment.setRedirectUrls(redirectUrls);

	return payment.create(apiContext);
}
```

Sl. 6.10. Metoda za kreiranje poveznice za plaćanje

Primjer čistog (engl. *raw*) *request*-a koji se šalje IPG API-ju vidljiv je na slici 6.11. Svi ostali parametri *request*-a su intuitivni ili prethodno objašnjeni u poglavlju 6.1.

Linija Kod

```
1:    {
2:      "intent": "sale",
3:      "payer": {
4:        "payment_method": "paypal"
5:      },
6:      "transactions": [
7:        {
8:          "amount": {
9:            "currency": "EUR",
10:           "total": "15.99"
11:         },
12:         "description": "Payment for items",
13:         "item_list": {
14:           "items": [
15:             {
16:               "name": "Nvidia RTX 3080",
17:               "quantity": "1",
23:               "price": "15.99",
24:               "currency": "EUR"
26:             }
27:           ]
38:         }
39:       }
40:     ],
41:     "note_to_payer": "Contact us for any questions on your order.",
42:     "redirect_urls": {
43:       "return_url": "http://localhost:8080/api/paypal/pay/success",
44:       "cancel_url": "http://localhost:8080/api/paypal/pay/cancel"
45:     }
46:   }
```

Sl. 6.11. Primjer *requesta* za kreiranje poveznice za plaćanje

Nakon što API obradi *request* vraća *response* u kojem se nalazi poveznica za odlazak na platnu stranicu, zajedno s ostalim podacima o plaćanju prikazanim na slici 6.12. Poveznica se izvlači iz polja *links* sa slike, koje je polje HATEOAS poveznica (engl. *Hypermedia as the Engine of Application State*) koje omogućavaju preusmjeravanje čak i ako se dogode neke dinamičke promjene.

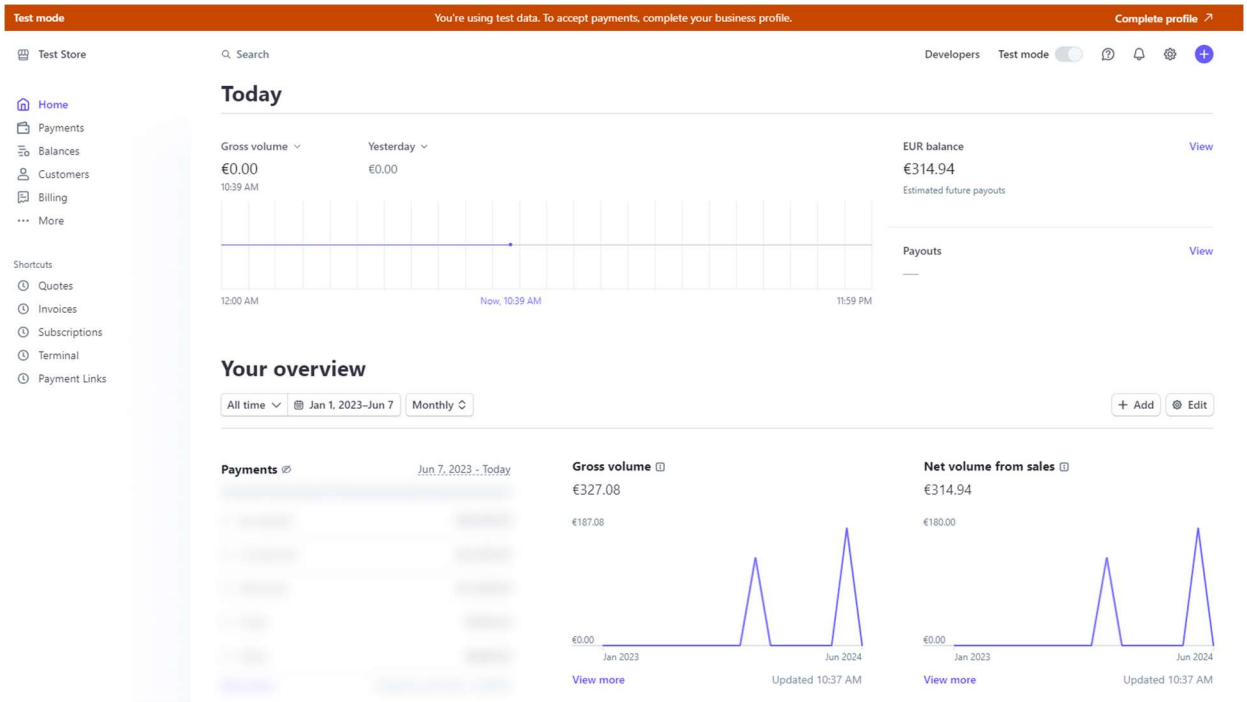
Linija Kod

```
1:  {
2:    "id": "PAYID-MZOZWKY51945052PG352734S",
3:    "intent": "sale",
4:    "state": "created",
5:    "payer": {
6:      "payment_method": "paypal"
7:    },
8:    "transactions": [
9:      {
10:       "amount": {
11:         "total": "15.99",
12:         "currency": "EUR"
13:       },
14:       "description": "Payment for items",
15:       "item_list": {
16:         "items": [
17:           {
23:             "name": "Nvidia RTX 3080",
24:             "price": "15.99",
26:             "currency": "EUR",
27:             "quantity": 1
38:           }
39:         ]
40:       },
41:       "related_resources": "xxxxxx"
42:     }
43:   ],
44:   "note_to_payer": "Contact us for any questions on your order.",
45:   "create_time": "2024-06-03T10:30:02Z",
46:   "links": [
47:     {
        "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAYID-
48: MZOZWKY51945052PG352734S",
49:       "rel": "self",
50:       "method": "GET"
51:     },
52:     {
        "href": "https://www.sandbox.paypal.com/cgi-
53: bin/webscr?cmd=_express-checkout&token=EC-1GF31834BD997933F",
54:       "rel": "approval_url",
55:       "method": "REDIRECT"
56:     }
57:   ]
58: }
```

Sl. 6.12. Primjer *response*-a kreiranja poveznice za plaćanje

6.3. Integracija sa Stripe API-jem

Prilikom integracije sa Stripe API-jem, slično kao i kod PayPal-a, također je potrebno stvoriti korisnički račun na službenoj stranici [20] prilikom kojeg se dobiva testna trgovina i testne vjerodajnice pomoću kojih se vrši spajanje vlastitog servera na njihov API. Za prihvaćanje stvarnih plaćanja, odnosno prelazak na produkcijski način rada, potrebno je dodati informacije o poslovanju, potvrditi identitet vlasnika i povezati bankovne podatke. Početna stranica nadzorne ploče developere (engl. *developer dashboard*) u testnom načinu rada na kojoj se nalazi pregled osnovnih statistika profita i plaćanja prikazana je na slici 6.13.



Sl. 6.13. Izgled Stripe nadzorne ploče

Gotovo identično kao kod PayPal SDK-a ovisnost ovog SDK-a se uvodi u projekt tako da se doda u *pom.xml* datoteku. Navedeni SDK je službeni podržani SDK za Javu naziva Stripe-Java [21]. Deklaracija ovisnosti je prikazana na slici 6.14.

Linija Kod

```
1:    <dependency>
2:      <groupId>com.stripe</groupId>
4:      <artifactId>stripe-java</artifactId>
5:      <version>24.0.0</version>
6:    </dependency>
```

Sl. 6.14. Primjer XML koda za dodavanje Stripe ovisnosti u projekt

Sa strane konfiguracije, Stripe SDK je nešto jednostavniji jer ne zahtjeva klasu koja bi predstavljala konfiguraciju poput PayPal-a gdje je sve bilo obgrljeno u nekakav kontekst iz kojega su se vadile vjerodajnice. Također, za ispravan rad, potreban mu je samo tajni ključ (engl. *sercret key*) koji se skladišti u *application.properties* datoteci na serveru. Stripe je pojednostavio pisanje *request*-a na način da uz *request* jednostavno prosljeđuje tajni ključ API-ju i na taj način vrši autentifikaciju.

Izrada poveznice za plaćanje je također jednostavnija jer njena izrada zahtjeva samo jednu rutu u kontroleru na koju se prosljeđuju podaci o nalogu i stavkama u prethodno spomenutom objektu *Order*. Primjer kontrolera prikazan je na slici 6.15. gdje je vidljivo da nakon izrade poveznice ju kontroler samo prosljeđuje *frontendu* te se ostatak plaćanja odvija na Stripe-ovoj strani što pojednostavljuje sam kod.

```
@Slf4j
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/stripe")
public class StripeController {

	private final StripeService stripeService;

	@PostMapping("/payment")
	public ResponseEntity<String> createPaymentLink(@RequestBody Order order) throws
StripeException {
		Session paymentLink = stripeService.createPaymentLink(order);
		log.info(paymentLink.toJson());
		return ResponseEntity.ok(paymentLink.getUrl());
	}
}
```

Sl. 6.15. Stripe kontroler za kreiranje poveznice plaćanja

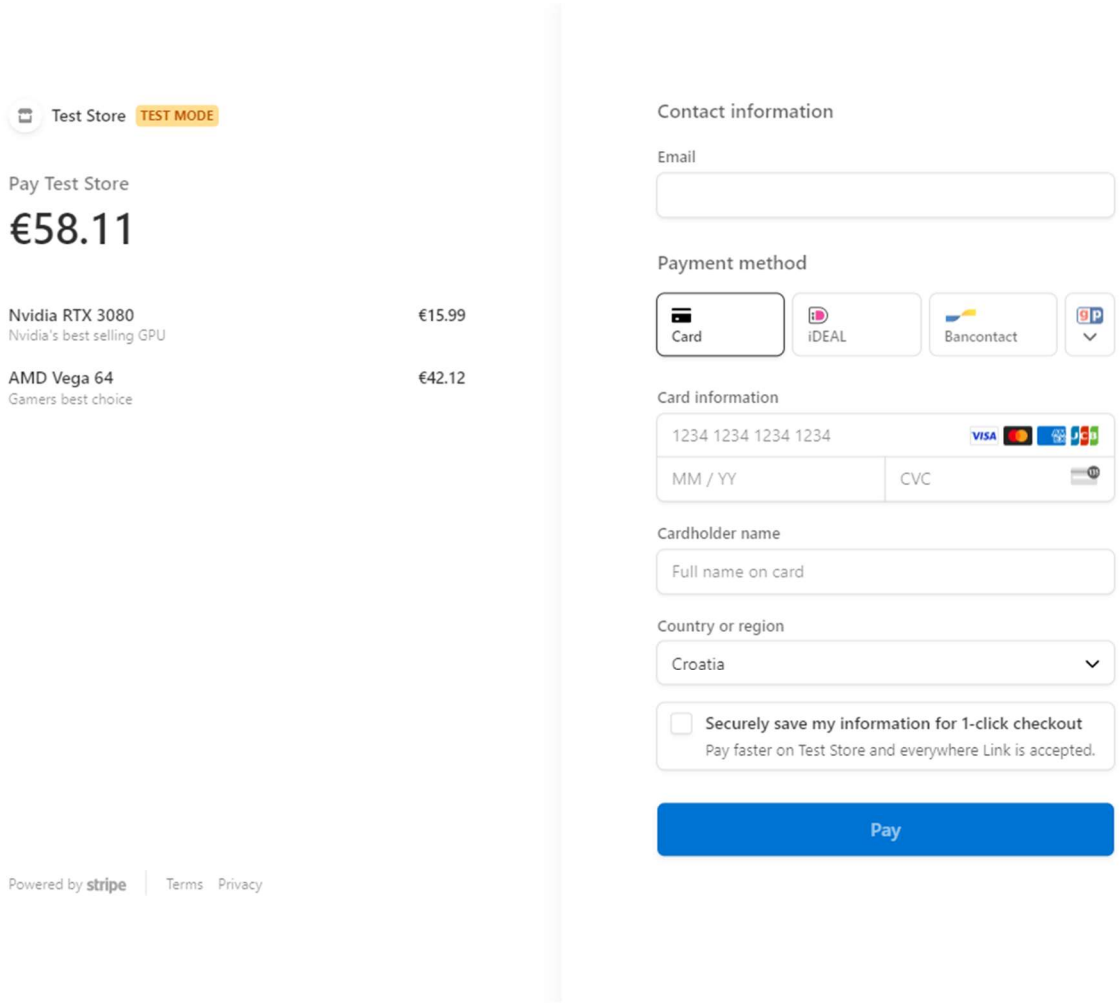
Dio koda koji kreira *request* sa svim potrebnim parametrima i vrši komunikaciju s API-jem se nalazi u *StripeService* servisu, točnije metoda *createPaymentLink()* prikazana slikom 6.16.

```
@Service
public class StripeService {

  @Value("${stripe.secret.key}")
  private String apiKey;
  @Value("${front.url}")
  private String frontUrl;

  public Session createPaymentLink(Order order) throws StripeException {
        Stripe.apiKey = apiKey;
        List<Object> lineItems = new ArrayList<>();
        for (Item item : order.getItems()) {
               // Create product
               Map<String, Object> productParams = new HashMap<>();
               productParams.put("name", item.getName());
               productParams.put("description", item.getDescription());

               Product product = Product.create(productParams);
               // Create price
               PriceCreateParams priceParams = PriceCreateParams.builder()
                      .setProduct(product.getId()).setCurrency("eur")
                      .setUnitAmount(Double.valueOf(item.getPrice() * 100)
                      .longValue()).build();
               Price price = Price.create(priceParams);

               // Create line item
               Map<String, Object> lineItem = new HashMap<>();
               lineItem.put("price", price.getId());
               lineItem.put("quantity", item.getQuantity());
               lineItems.add(lineItem);
        }
        Map<String, Object> params = new HashMap<>();
        params.put("line_items", lineItems);
        params.put("mode", "payment");
        params.put("success_url", frontUrl + "?session_id={CHECKOUT_SESSION_ID}");
        return Session.create(params);
  }
}
```

Sl. 6.16. Metoda u servisu za kreiranje poveznice plaćanja

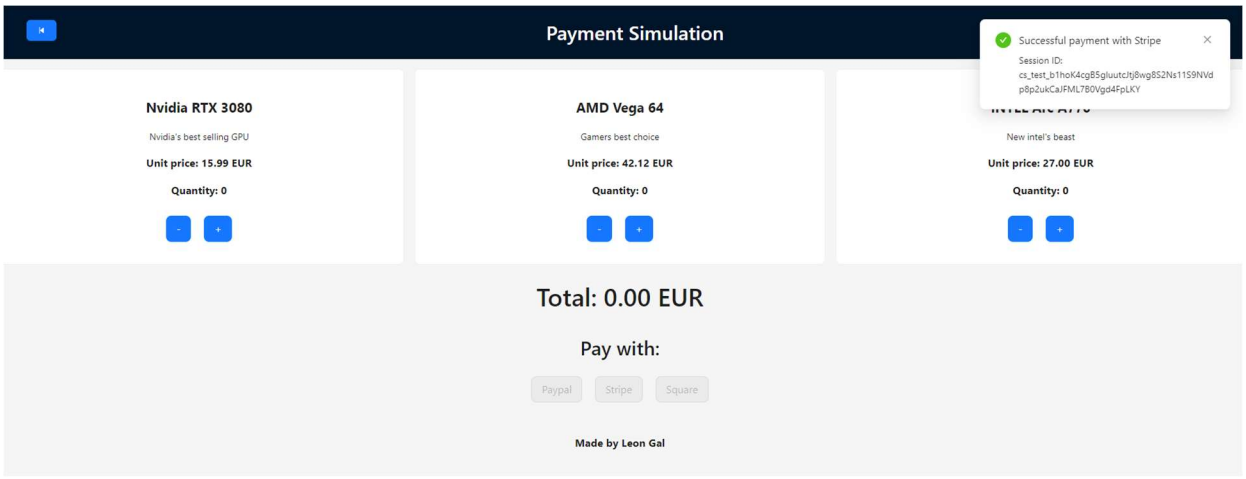
Ubrizgavanje ovisnosti privatnog ključa se vrši na prethodno spomenuti jednostavniji način, samo navođenjem linije 29. gdje se automatski ključ šalje prilikom sljedećeg *request*-a. Nakon tog slijedi postavljanje stavki, cijena i količina u *request*. Ovaj SDK funkcionira na način da se svi parametri postavljanju unutar *Hash* mapa hijerarhijski, onako kako bi trebali biti u *request*-u. *Mode* parametar označava da se želi kreirati regularno plaćanje koje sadržava link za plaćanje i obavlja se plaćanjem na linku ili pozivanjem API-ja s ID-jem tog plaćanja koji signalizira da je plaćanje provedeno. Parametar *success_url* predstavlja putanju na koju će se preusmjeriti korisnika nakon uspješno obavljenog plaćanja (početna stranica aplikacije).

Nakon što se u *response*-u dobije poveznica ona se šalje korisniku kako bi popunio informacije za uspješno plaćanje. Izgled stanice za naplatu prikazan je na slici 6.17. gdje je na lijevoj strani vidljiv detaljan prikaz svake stavke, cijene te ukupne cijene. Na desnoj strani su polja za popunjavanje s kartičnim podacima ili odabirom nekog od podržanih načina plaćanja.



Sl. 6.17. Izgled Stripe stranice za naplatu

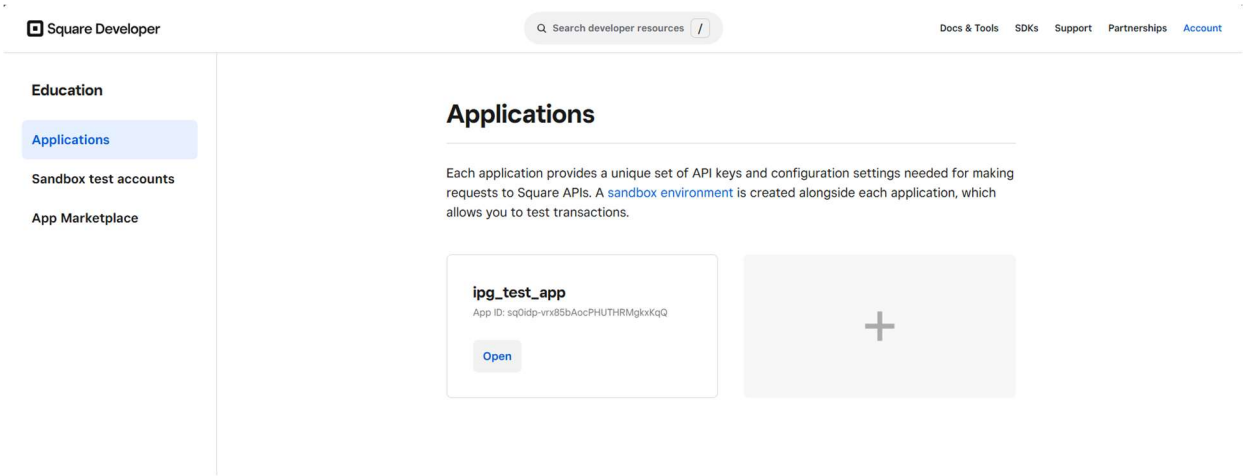
Kako je ovo testno okruženje, Stripe pruža i testne kartične podatke koje omogućuju razne brendove kartica, simuliranje uspješnog plaćanja, odbijene kartice, nedovoljno pokrića i mnogo drugih rubnih bitnih slučajeva. Najčešće korištena testna kartica za uspješnu provedbu plaćanja je Visa čiji broj treba biti 4242424242424242, CVC bilo koji troznamenkasti broj i datum isteka bilo koji datum u budućnosti. Po završetku plaćanja korisnik biva preusmjeren natrag na aplikaciju s porukom o uspješnom plaćanju prikazanu na slici 6.18.



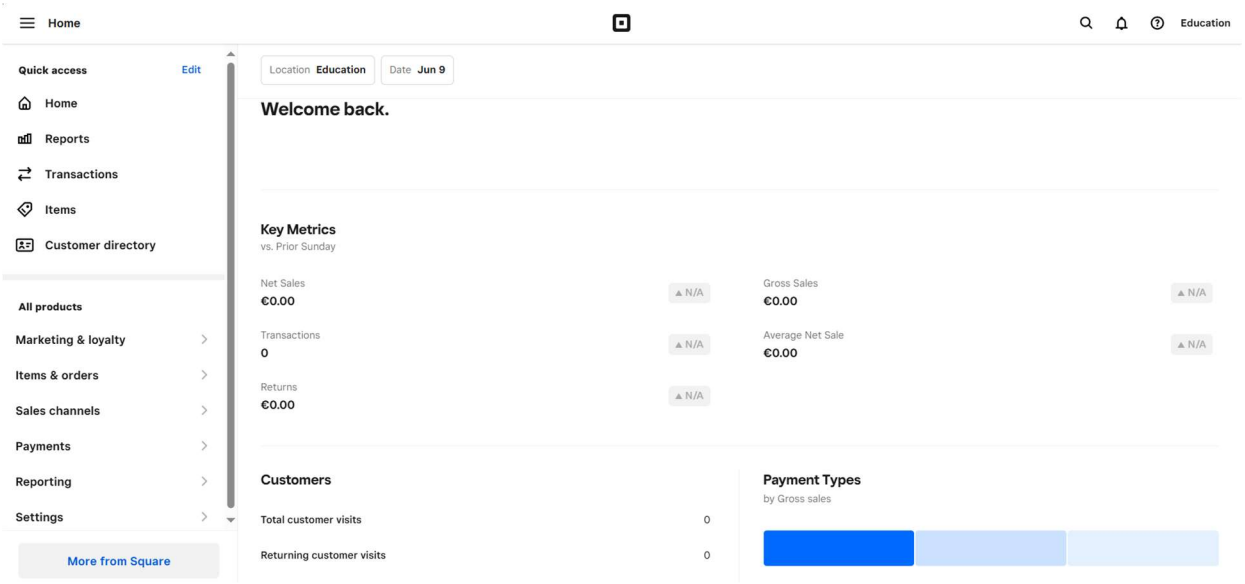
Sl. 6.18. Poruka o uspješnoj provedbi plaćanja

6.4. Integracija sa Square API-jem

Za uspješnu integraciju sa Square API-jem, kao i kod prethodna dva *provider*-a potrebno se registrirati i napraviti korisnički račun na službenoj web stranici [22]. Kreirani račun se dijeli na razvojni dio i prodavački dio. Razvojni dio računa prikazan na slici 6.19. koristi se za generiranje vjerodajnica i povezivanje vlastitog servera sa njihovim API-jem, postavljanje testnih računa te pregledavanje dokumentacije. S druge strane prodavački dio prikazan slikom 6.20. koristi za pregled transakcija, kreiranje predefiniranih naloga i stavki, prikaz statistike poslovanja, generiranja izvještaja i dr.



Sl. 6.19. Izgled stanice za razvojne programere



Sl. 6.20. Izgled prodavačke stranice

Za povezivanje sa Square-ovim API-jem koristi se službeni SDK za Java programski jezik čije je uključivanje u projekt prikazano na slici 6.21.

Linija Kod

```
1:    <dependency>
2:    <groupId>com.squareup</groupId>
4:      <artifactId>square</artifactId>
5:      <version>35.0.0.20231115</version>
6:    </dependency>
```

Sl. 6.21. XML kod za dodavanje Square SDK-a u projekt

Kontroler prikazan na slici 6.22. je strukturiran da samo delegira posao na servis klasu s *Order* objektom koji dobije u *request*-u.

```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/square")
public class SquareController {

  private final SquareService squareService;

  @PostMapping
  public ResponseEntity<Object> createPaymentLink(@RequestBody @Validated Order orderRequest)
  {
       try {
              return squareService.createPaymentLink(orderRequest);
       } catch (ApiException | IOException e) {
              e.printStackTrace();
              return ResponseEntity.internalServerError().build();
       }
  }
}
```

Sl. 6.22. Square kontroler za kreiranje poveznice plaćanja

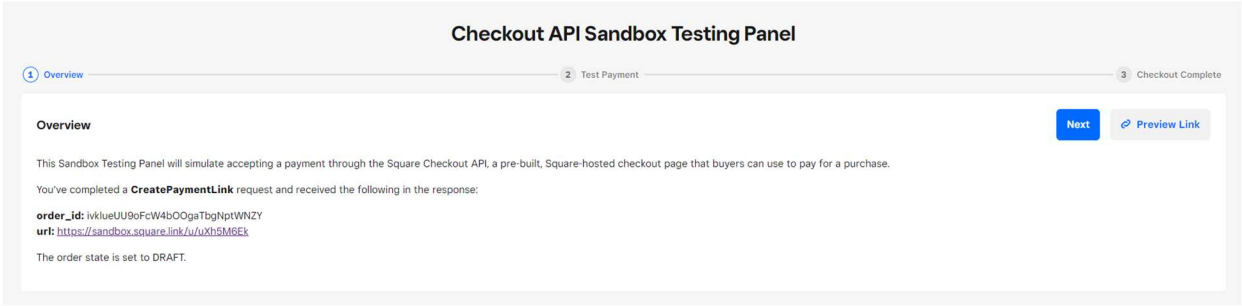
U servisu (slika 6.23.), metoda *createPaymentLink()* zatim odrađuje mapiranje *request*-a u klase koje SDK pretvara u *request* koji šalje API-ju. Prilikom kreiranja *request*-a za API potrebno je pružiti *access token*, koji nije ništa drugo nego tajni ključ, te *locationID*, odnosno identifikator lokacije koji se nalazi u *application.properties* datoteci a moguće ga je pronaći i kreirati u postavkama korisničkog računa.

Identifikator lokacije pojednostavljuje preglednost naplate u složenim i velikim poslovanjima gdje pristiže velik broj uplata s jedinica koji su geografski udaljene, primjerice online trgovački lanac u više država. Kada kupac odradi plaćanje ono se odmah povezuje s relevantnom trgovinom ili fizičkom lokacijom što postiže bolju organizaciju, točno praćenje i jednostavnije otklanjanje pogreški. Konkretno na korištenom računu je postavljena lokacija Osijek.

```
@Service
@Slf4j
@RequiredArgsConstructor
public class SquareService {

  @Value("${square.access.token}")
  private String squareAccessToken;
  @Value("${square.location.id}")
  private String locationId;
  @Value("${front.url}")
  private String frontUrl;

  public ResponseEntity<Object> createPaymentLink(Order orderRequest) throws ApiException,
  IOException {
        SquareClient client = new SquareClient.Builder()
               .environment(Environment.SANDBOX).accessToken(squareAccessToken).build();

        LinkedList<OrderLineItem> lineItems = new LinkedList<>();
        for (Item item : orderRequest.getItems()) {
               Money basePriceMoney = new Money.Builder().amount(item.getPrice().longValue())
                              .currency(orderRequest.getCurrency()).build();

               OrderLineItem orderLineItem = new OrderLineItem.Builder(item.getQuantity()
               .toString()).name(item.getName())
               .note(item.getDescription())
               .itemType(!orderRequest.getIntent().equals("sale") ? "GIFT_CARD" : "ITEM")
               .basePriceMoney(basePriceMoney).build();
               lineItems.add(orderLineItem);
        }

        com.squareup.square.models.Order order =
               new com.squareup.square.models.Order.Builder(locationId)
               .lineItems(lineItems).build();

        CheckoutOptions checkoutOptions = new CheckoutOptions.Builder()
               .redirectUrl(frontUrl).build();

        CreatePaymentLinkRequest body = new CreatePaymentLinkRequest.Builder()
               .description(orderRequest.getDescription())
               .paymentNote(orderRequest.getNoteToPayer()).order(order)
               .checkoutOptions(checkoutOptions).build();

        CreatePaymentLinkResponse paymentLink = client.getCheckoutApi()
               .createPaymentLink(body);
        SquareClient.shutdown();

        if (paymentLink.getErrors() == null || paymentLink.getErrors().isEmpty()) {
               log.info(paymentLink.toString());
               return ResponseEntity.ok(paymentLink.getPaymentLink());
        }
        log.error(paymentLink.getErrors().stream()
               .map(error -> error.getCode() + " - " + error.getDetail()).toString());
               return ResponseEntity.internalServerError().body(paymentLink.getErrors());
        }
}
```

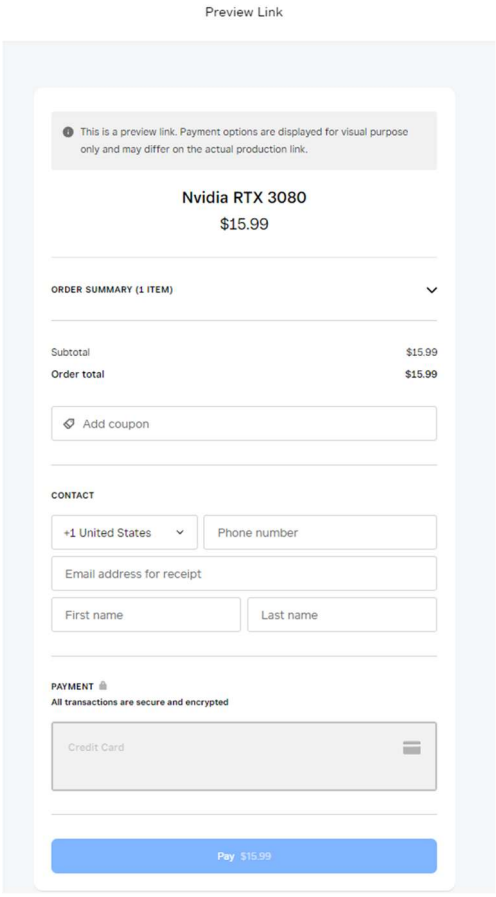
Sl. 6.23. Square servis za kreiranje poveznice plaćanja

Prilikom odabira načina plaćanja sa Square-om poveznica za plaćanje vodi na Square-ovu ploču za testiranje (engl. *Checkout API Sandbox Testing Panel*) prikazanu na slici 6.24.



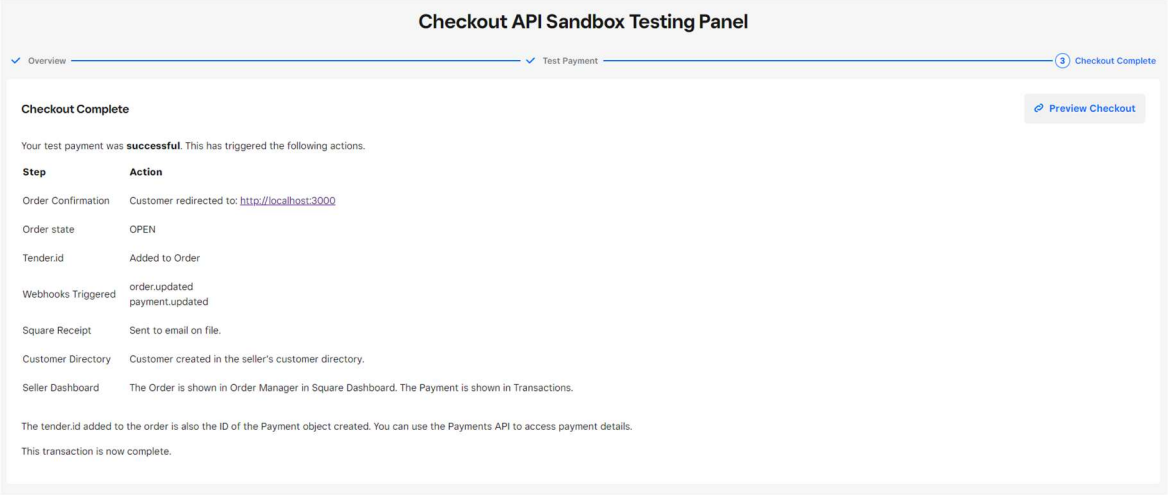
Sl. 6.24. Ploča za testiranje plaćanja

Za razliku od drugih *provider*-a Square je specifičan po tom što ne pruža testne kartice ili druge alate za testiranje nego jednostavno ima posebnu stranicu koja omogućuje samo pregled poveznice (*preview link* gumb) koju bi korisnik odmah vidio te ispunio kartičnim podacima. Stranica za plaćanje je prikazana na slici 6.25.



Sl. 6.25. Izgled Square stranice za naplatu

Ukoliko se želi simulirati plaćanje potrebno je na ploči za testiranje kliknuti na *Test Payment* gumb, koji potom odradi plaćanje i prikaže detalje provedenog plaćanja i tok radnji koje bi ta akcija pokrenula da je riječ u produkcijskom okruženju 6.26.



Sl. 6.26. Detalji nakon provedenog plaćanja

7. USPOREDBA SUSTAVA ZA NAPLATU

Što se tiče alata za testiranje, sva 3 *providera* nude *sandbox* okruženja. PayPal i Stripe su vrlo slični po pristupu samog testiranja te nude testne kartice. Laganu prednost ima PayPal jer pruža veću fleksibilnosti testiranja generatorom testnih kartica za svaku državu i brand kartice, dok kod Stripe-a postoji samo lista predefiniranih testnih kartica. Oba *provider*-a podržavaju simulaciju različitih scenarija (uspješno, odbijeno, prevara, istekla kartica, povrat novca, nedovoljno pokrića...). PayPal ima definirane konstante koje se unose u ime vlasnika kartice kako bi se aktivirao taj scenarij s tom karticom, primjerice unosom *CCREJECT-EC* postižemo pogrešku o isteku kartice dok Stripe jednostavno ima listu kartica za svaki od scenarija. Također prednost PayPal-a u samom testiranju je mogućnost otvaranja proizvoljnog broja korisničkih testnih računa s kojih je moguće obavljati kupnju na prodavačevoj stranici. Tu je najveća prednost preglednost cijelog sustava, jer se u svaki testni račun može ući preko testne PayPal stranice te vidjeti povijest plaćanja i dodavati novčana sredstva, dok se s druge strane na prodavačevom testnom računu mogu vidjeti uplate s korisničkih testnih računa.

Square je išao drugom filozofijom u testiranju svojih plaćanja. Kao što je rečeno u poglavlju 6.4 prilikom kreiranja poveznice plaćanja, koristi *sandbox* testni panel preko kojeg se simulira korisnikovo ispunjavanje forme te uspješno plaćanje kreiranog naloga. Također omogućuje i pregled izgleda forme za plaćanje, ali ju nije moguće popuniti. Ovakav pristup je manje fleksibilan od prethodna dva *providera* te po završetku plaćanja nema mogućnost preusmjeravanja korisnika natrag na trgovinu. Nadalje, mana ovog pristupa se očituje u ne mogućnosti testiranja ranije spomenutih scenarija plaćanja jer jedina opcija koja se nalazi na panelu je uspješno plaćanje tako da nije moguće u testnom okruženju pokriti druge osnovne scenarije (odbijeno, prevara, istekla kartica, povrat novca, nedovoljno pokrića...).

S gledišta lakoće integracije, svaki *provider* pruža API podjednake programerske složenosti no integraciju dodatno olakšava činjenica da su korišteni SDK-ovi koji uvelike olakšavaju kreiranje *requesta* te lako čitljivu sintaksu koda. Ipak, PayPal za uspješnu integraciju, kao što je prikazano u poglavlju 6.2, zahtjeva kompleksniju konfiguraciju korištenjem ApiContext *bean*-a. Sva 3 *providera* imaju velike mogućnosti skalabilnosti i integracije, od niza programskih jezika koji

podržavaju SDK-ove (Java, C#, PHP; Ruby...) preko velikog i robusnog API-ja do *Webhook*[5]-ova koji uvelike olakšavaju razvoj interaktivne web trgovine.

PayPal uz spomenuta *sandbox* okruženja, pruža i napredne alate sustava poput online *Webhook* simulatora te centraliziranog pregleda e-pošte sa svih *sandbox* računa. Stripe s druge strane pruža niz Postman *request*-ova koji se mogu preuzeti i koristiti za testiranje ponašanja samog API-ja bez potrebe implementacije *backend*-a te online sučelje naredbenog retka (CLI). Slično nešto ima i Square, točnije online API *explorer* koji također omogućava kreiranje zahtjeva bez potrebe za vlastitim serverom no puno primitivnije i s manje mogućnosti.

Ključna stvar koju svi *provideri* imaju je mogućnost pregledavanja zapisnika događaja (engl. *event logs*) te povijesti API poziva. Na taj način je programerima uvelike olakšano praćenje sustava u stvarnom vremenu, otklanjanje pogrešaka, pregled revizijskih tragova, analiziranje sustava što se odražava na bolju sigurnost sustava.

Usporednim istraživanjem svojstava svakog *providera* u tablici 7.1. su zabilježeni različiti faktori usporedbe koji prikazuju prednosti i razlike u korištenju jednog *providera* u odnosu na neki drugi. Razmatrani faktori su: cijena usluge, podržane valute i države, vrste kartica, naknada za povrat te sadrži li trgovački račun usluga koju nude.

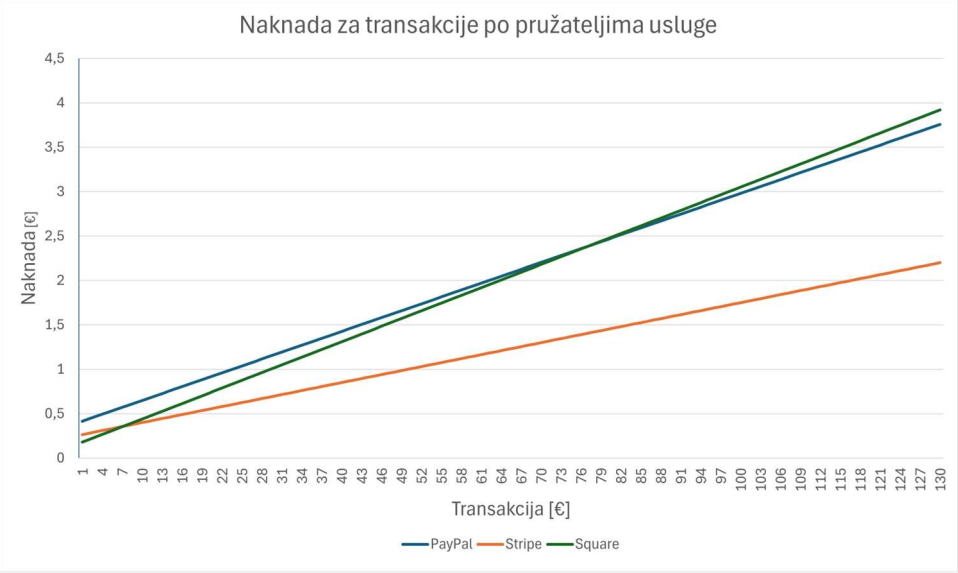
Paypal je u tablici podijeljen na dvije vrste usluge standardna (globalno dostupna), korištena u ovom radu, te PayPal Payments Pro koja je dostupna samo u Americi, Kanadi i Velikoj Britaniji. Pro verzija pruža potpuno prilagodljiv sustav plaćanja koji omogućuje prihvaćanje svih glavnih kreditnih i debitnih kartica PayPal-a i PayPal Credit-a, potpuno prilagođavanje naplatne stranice za kupce. Dobiju se i sve mogućnosti standardne usluge plus mogućnost prihvaćanja plaćanja putem telefona, faksa i e-poštom (s virtualnim terminalom). Globalno dostupna verzija sličnih funkcionalnosti naziva se PayPal Payflow Pro.

[5] Vrsta HTTPS zahtjeva od PayPala do krajnje točke na vašem poslužitelju kad god se dogodi odgovarajuća vrsta događaja, suprotnog smjera od klasičnih API poziva. Posebno su korisni za primanje obavijesti o događajima koji su pokrenuti od strane PayPal-a.

Tablica 7.1. Usporedba sustava za naplatu

Sustav naplate	Trgovački račun	Cijena	Naknada za povrat	Valute	Države	Vrsta kartice
PayPal Standard	Da	0€ mjesečno, 2.59%+0.39€ po transakciji	16€	25	203	9
PayPal Payments Pro	Da	0€ mjesečno, 2.89%+0.39€ po transakciji	16€	23	3	9
Stripe	Da	0€ mjesečno, 1.5%+0.25€ po transakciji	15€	100+	25	6
Square	Ne	0€ mjesečno, 2.9%+0.15€ po transakciji	Nema	27	8	6

Jedini *provider* koji ne nudi trgovački račun uz usluge plaćanja je Square, što znači da je potrebno negdje drugdje imati račun na koji bi sjedale transakcije. Vidljivo je da *provideri* nemaju mjesečnu naknadu za uslugu ali imaju određene cijene naknada po transakcijama. PayPal se čini najskuplji, Stripe ima nešto niže naknade dok je Square u sredini s naknadama. Kako bi se točnije prikazali troškovi prilikom transakcija, potrebno je uzeti barem 2 krajnja slučaja, primjerice transakcija od 1€ i transakcija od 100€. Ovim postupkom je moguće točno kvantificirati koji je *provider* jeftiniji za koju vrstu usluge, jeftinije transakcije ili skuplje. Nadalje, ako se uzme još više vrijednosti iz tog raspona dobiva se graf sa slike 7.1. koji detaljno prikazuje odnos svakog *providera* u odnosu na cijenu transakcije. Iz grafa je vidljivo da je PayPal za manje transakcije najskuplji a Square najisplativiji, dok je za veće transakcije Stripe daleko najisplativiji te je Square tada vodeći u iznosu naknade.



Sl. 7.1. Prikaz transkacijskih naknada po *providerima*

Nadalje, naknada za povrat sredstava na zahtjev kupca se svugdje naplaćuje gotovo podjednako osim kod Square-a koji ju nema. Broj podržanih valuta je najveći kod Stripe-a dok je PayPal podržan kao usluga u najviše država, čak preko 200. PayPal podržava 9 vrsta kartica dok ostala dva *providera* podrže 6.

Sa strane responzivnosti te brzine obrade transakcija provedena su 2 mjerenja vremena:

1. Od trenutka slanja zahtjeva do dobivanja poveznice za plaćanje
2. Od trenutka kad se klikne na gumb plati pa do statusa uspješne obrade

Mjerenja su provedena na Google Chrome pregledniku korištenjem ugrađenog mrežnog alata za mjerenje vremena paketa (*F12 -> Network -> Timing*). Provedeno je po 10 mjerenja za svakog *providera* te su rezultati prikazani tablicama 7.2. i 7.3.

Tablica 7.2. Vrijeme generiranja poveznice za plaćanje

No.	PayPal [s]	Stripe [s]	Square [s]
1	1,57	2,52	2,00
2	0,76	2,19	1,54
3	0,71	2,71	1,20
4	1,02	2,66	1,33
5	1,07	2,01	1,03
6	1,21	2,51	1,38
7	0,96	1,55	1,36
8	0,90	2,56	1,10
9	0,86	2,43	1,18
10	0,92	1,51	1,04
Prosjek [s]	0,998	2,265	1,316

Iako su mjerenja rađena na istom uređaju s identičnom internetskom konekcijom vidljiva su odstupanja u odzivima kod svih *providera* zbog niza faktora poput zagušenja IPG procesora i količine trenutnog prometa. Zbog tog razloga je prikazan prosjek svih mjerenja iz kojeg je vidljivo da je u brzini kreiranja naloga i vraćanja poveznice natrag serveru najbrži PayPal dok je Stripe daleko najsporiji.

Tablica 7.3. Vrijeme obrade uspješnog plaćanja

No.	PayPal [s]	Stripe [s]	Square [s]
1	1,25	1,55	1,35
2	1,08	1,55	1,58
3	1,02	1,56	1,27
4	1,34	1,54	1,27
5	1,32	1,91	1,32
6	1,32	1,75	1,27
7	1,14	1,66	1,27
8	1,15	1,68	1,63
9	1,44	1,68	1,55
10	1,22	1,72	1,21
Prosjek [s]	1,228	1,660	1,372

S druge strane, prilikom obrade plaćanja te vraćanja statusa o uspješnom plaćanju poredak je identičan, PayPal je također najbrži a Stripe je najsporiji u obradi.

Važno je napomenuti da u oba slučaja, posebno u drugom slučaju mjerenja kad *provideri* „obrađuju“ zahtjev za plaćanje, rezultati vrlo vjerojatno nisu mjerodavni s onim vremenima koja bi se dala izmjeriti na produkcijskim okruženjima, obzirom da se sve izvršava na testnim okruženjima s testnim podacima i vrlo vjerojatno se dosta provjera i komunikacija ne izvršava zbog uštede resursa nego se samo lažiraju (engl. *mock*).

8. ZAKLJUČAK

Cilj rada je bio opisati sustave za provedbu plaćanja u današnjim web okruženjima, opisati tehničku osnovu IPG sustava, pružiti implementaciju tri pružatelja usluge i zatim ih usporediti na više razina.

Najvažnija stvar kod sustava naplate jest sigurnost. Ona se pokušava podići na što višu razinu raznim zakonima i normama koje IPG sustavi moraju zadovoljiti. Implementacija kvalitetnog IPG sustava je vrlo mukotrpan posao koji zahtjeva velika ulaganja te kolektiv iskusnih ljudi. Upravo iz tog razloga je preporuka da sustavi koji nisu u državnom vlasništvu te ne prate stroge regulative zbog jednostavnosti i isplativosti koriste neke od postojećih pružatelja IPG usluge.

PayPal pruža najviše funkcionalnosti, najrobusniji API za veliki spektar usluga, najbrža vremena obrade zahtjeva te je u većini slučajeva skuplji od Stripe-a. Najbolja je ponuda za manje internacionalne prodavače s obzirom na podršku velikog broja država. Stripe je idealan za online trgovine koje ne žele velik izbor integracija i mogućnosti nego što jednostavnije odraditi prodaju. Square pruža najbolje opcije za plaćanja uživo te maloprodajne transakcije.

Bitno je naglasiti da su ovim istraživanjem obuhvaćena samo navedena 3 pružatelja IPG usluga, dok ih na tržištu ima mnogo više. To ne znači da su odabrani pružatelji najbolji i najisplativiji za svakoga. Naprotiv, evidentno je da svaki pružatelj IPG usluge nudi neku konkurentsku prednost za specifičnu vrstu korisnika te za regiju u kojoj se ti korisnici nalaze.

LITERATURA

[1] „First Virtual Holdings, Inc. (A)“ [online]. Dostupno na: https://hbsp.harvard.edu/product/98E007-PDF-ENG. [Pristupljeno: 7.9.2024.].

[2] M., Masihuddin, B., Khan, M., Mattoo, R., Olanrewaju, „A Survey on E-Payment Systems: Elements, Adoption, Architecture, Challenges and Security Concepts“, *Indian J. Sci. Technol.*, sv. 10, str. 1–19, lip. 2017.

[3] „SUSTAV ZA NAPLATU JAVNIH DAVANJA REPUBLIKE HRVATSKE“ [online]. Dostupno na: https://njdprez.fina.hr/. [Pristupljeno: 4.8.2024.].

[4] „e-Pristojbe“ [online]. Dostupno na: https://rdd.gov.hr/projekti-i-eu-projekti/eu-projekti/e-pristojbe/1589. [Pristupljeno: 4.8.2024.].

[5] „KEKS Pay - aplikacija za brzo slanje i primanje novca - bez naknada“ [online]. Dostupno na: http://www.kekspay.hr/. [Pristupljeno: 7.9.2024.].

[6] „An analysis and comparison of different types of electronic payment systems“ [online]. Dostupno na: https://ieeexplore.ieee.org/abstract/document/952002. [Pristupljeno: 11.9.2024.].

[7] „Barter (or Bartering) Definition, Uses, and Example“ [online]. Dostupno na: https://www.investopedia.com/terms/b/barter.asp. [Pristupljeno: 24.5.2024.].

[8] „HPB - Zaprimanje i naplata inozemnih čekova“ [online]. Dostupno na: https://www.hpb.hr/hr/zaprimanje-i-naplata-inozemnih-cekova/232. [Pristupljeno: 24.5.2024.].

[9] „Base I: What It Means, in The History of Credit Card Payments“ [online]. Dostupno na: https://www.investopedia.com/terms/b/base-i.asp. [Pristupljeno: 24.5.2024.].

[10] M., Tan, *E-payment: The Digital Exchange*. NUS Press, 2004.

[11] M., Radonic, „Payment processing in web-based environments - Benchmark of the World’s Leading Payment Processors“, 2018.

[12] Z. C., Nxumalo, P., Tarwireyi, M. O., Adigun, „Towards privacy with tokenization as a service“, u *2014 IEEE 6th International Conference on Adaptive Science & Technology (ICAST)*, str. 1–6, 2014.

[13] „Luhn Algorithm - Credit Card Number Checker - Online Generator“ [online]. Dostupno na: https://www.dcode.fr/luhn-algorithm. [Pristupljeno: 24.5.2024.].

[14] „CHARGEBACK - povrat novca za plaćanja obavljena karticama - Hrvatska Udruga za Zaštitu Potrošača“ [online], 18-kol-2023. .

[15] „How to Build Your Own Payment Gateway - Softjourn“ [online]. Dostupno na: https://softjourn.com/insights/how-to-build-your-own-payment-gateway. [Pristupljeno: 25.5.2024.].

[16] „React“ [online]. Dostupno na: https://react.dev/. [Pristupljeno: 25.5.2024.].

[17] „PayPal Developer“ [online]. Dostupno na: https://developer.paypal.com/home. [Pristupljeno: 9.6.2024.].

[18] „Maven Repository: Search/Browse/Explore“ [online]. Dostupno na: https://mvnrepository.com/. [Pristupljeno: 28.5.2024.].

[19] „APIContext (rest-api-sdk 1.8.0 API)“ [online]. Dostupno na: https://javadoc.io/doc/com.paypal.sdk/rest-api-sdk/1.8.0/com/paypal/base/rest/APIContext.html. [Pristupljeno: 28.5.2024.].

[20] „Stripe | Financial Infrastructure to Grow Your Revenue“ [online]. Dostupno na: https://stripe.com/en-hr. [Pristupljeno: 8.6.2024.].

[21] „stripe/stripe-java“. Stripe, 07-lip-2024.

[22] „Power your entire business“ [online]. Dostupno na: https://squareup.com/us/en?optimizely-snippet-injection-enabled=true. [Pristupljeno: 9.6.2024.].

SAŽETAK

U ovom radu je prvo objašnjen povijesni pregled plaćanja i trgovanja koji je postepenim razvojem tehnologije doveo do današnjih metoda elektroničke naplate te naplate u web okruženjima. Opisana je tehnička osnova web sustava naplate u vidu pseudokoda te su definirane regulative usklađenosti prilikom obrade plaćanja. Odrađena je integracija sustava za naplatu korištenjem 3 svjetski poznata pružatelja usluge IPG-a te je u završnom poglavlju izvršena usporedba implementiranih sustava naplate.

Ključne riječi: integracija IPG sustava, Java, PayPal, Stripe, sustavi naplate, Square, usporedba naplate, web naplata

ABSTRACT

Payment gateways in web environment

This paper explains the historical overview of payments and trading, which, with the development of technology, led to today's methods of electronic payments and payments in web environments. The technical basis of the web payment system in the form of pseudocode has been writen, and the payments compliance for payment processing have been defined. The integration of payment system was done using 3 mainstream IPG service providers. The comparison of the implemented payment systems was made in the final chapter.

Keywords: IPG integration, Java, PayPal, payment systems, payments comparison, Stripe, Square, web payments