UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Blaž Parkelj

Analiza in primerjava programskih in poslovnih modelov aplikacij v obliki storitev

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01852/2012
Datum: 05.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **BLAŽ PARKELJ**

Naslov: **ANALIZA IN PRIMERJAVA PROGRAMSKIH IN POSLOVNIH MODELOV APLIKACIJ V OBLIKI STORITEV**

ANALYSIS AND COMPARISON OF APPLICATION AND BUSINESS MODELS FOR SOFTWARE AS A SERVICE

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Proučite koncept programske opreme kot storitve in opredelite glavne lastnosti. Predstavite programski model, ki vključuje večnajemniški model, upravljanje dostopa, integracijo, arhitekturo kompozitnih aplikacij, metapodatkovne storitve ter nadzorovanje in merjenje. analizirajte poslovne modele ter dogovore o nivoju storitev in njegove značilnosti. Opišite vpliv na IT oddelek in izdelajte analizo TCO (Total Cost of Ownership). Implementirajte vzorčno večnajemniško aplikacijo v oblaku v obliki SaaS.

Mentor:

prof. dr. Matjaž B. Jurič

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Blaž Parkelj,

z vpisno številko 63060255,

sem avtor/-ica diplomskega dela z naslovom:

Analiza in primerjava programskih in poslovnih modelov aplikacij v obliki storitev

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  prof. dr. Matjaža Branka Juriča

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Zahvala

Iskrena zahvala gre prijateljem in sorodnikom za vso podporo in potrpežljivost v času pisanja diplomske naloge. Prav tako se zahvaljujem mentorju prof. dr. Matjažu Branku Juriču za nasvete in usmerjanje pri izdelavi diplomske naloge.

Kazalo vsebine

Seznam uporabljenih kratic in simbolov

ACL	Seznam za nadzor dostopa (angl. Access Control List)
ACS	Storitev za upravljanje dostopa (angl. Access Control Service)
AD FS	Federacijske storitve AD (angl. Active Directory Federation Services)
API	Programski vmesnik (angl. Application Programming Interface)
ASP	Ponudnik programskih storitev (angl. Application Service Provider)
BPEL	Izvajalni jezik za poslovne procese (angl. Business Process Execution Language)
BPMN	Model in notacija poslovnih procesov (angl. Business Process Model and Notation)
CRM	Upravljanje odnosov s strankami (angl. Customer Relationship Management)
DBMS	Sistem za upravljanje podatkovnih baz (angl. Database Management System)
ERP	Integrirani poslovni informacijski sistem (angl. Enterprise Resource Planning)
FP	Oskrbovalec federacije (angl. Federation Provider)
HTTP	Internetni komunikacijski protokol (angl. Hypertext Transfer Protocol)
IaaS	Infrastruktura kot storitev (angl. Infrastructure as a Service)
IdP	Oskrbovalec identitete (angl. Identity Provider)
IT	Informacijska tehnologija (angl. Information Technology)
JSON	Zapis objektov v JavaScript (angl. JavaScript Object Notation)
LOB	Poslovna dejavnost (angl. Line Of Business)
PaaS	Platforma kot storitev (angl. Platform as a Service)
REST	Predstavitveni prenos stanja (angl. REpresentational State Transfer)
SaaS	Programska oprema kot storitev (angl. Software as a Service)
SAML	Varnostni označevalni jezik (angl. Security Assertion Markup Language)
SAS 70	Izjava o revizijskih standardih številka 70 (angl. Statement on Auditing Standards number 70)
SCM	Upravljanje oskrbovalnih verig (angl. Supply Chain Management)
SLA	Dogovor o nivoju storitev (angl. Service Level Agreement)
SMB	Majhna in srednje velika podjetja (angl. Small and Medium sized Businesses)
SOA	Storitveno-usmerjena arhitektura (angl. Service-Oriented Architecture)
SOAP	Protokol za dostop do objektov (angl. Simple Object Access Protocol)
SQL	Strukturiran povpraševalni jezik (angl. Structured Query Language)
SSO	Enotna prijava (angl. Single Sign-On)
STS	Storitev varnostnih žetonov (angl. Security Token Service)
SWT	Enostavni spletni žeton (angl. Simple Web Token)

TCO	Skupni strošek lastništva (angl. Total Cost of Ownership)
URI	Enolični identifikator virov (angl. Uniform Resource Identifier)
WS	Spletna storitev (angl. Web Service)
WWF	Microsoftova knjižnica delovnih tokov (angl. Windows Workflow Foundation)
XML	Razširljiv označevalni jezik (angl. Extensible Markup Language)

Povzetek

Diplomska naloga obravnava enega od treh storitvenih modelov računalništva v oblaku – programska oprema kot storitev (Software as a Service ali SaaS). Programska oprema kot storitev je način dostave aplikacij in vključuje arhitekturne vidike za implementacijo aplikacij ter poslovni model, ki se razlikuje od modela tradicionalnih aplikacij. Po kratkemu uvodu smo umestili tematiko diplomske naloge v široko področje računalništva. Definirali smo koncept programske opreme kot storitve, glavne lastnosti in osnovne pojme, ki so uporabljeni v diplomski nalogi. Navedli smo zrelostni model in dve enostavni delitvi aplikacij SaaS. Predstavili smo programski model, ki vključuje večnajemniški model, upravljanje dostopa, integracijo, arhitekturo kompozitnih aplikacij, metapodatkovne storitve ter nadzorovanje in merjenje. Opisali smo poslovni model, če sledimo konceptu SaaS, njegove glavne lastnosti in do kakšnih sprememb lahko pride v podjetju pri sprejemu tega modela. Poleg tega smo opisali dogovor o nivoju storitev in njegove značilnosti. SaaS ima ogromen vpliv tako na ponudnika kot odjemalca storitev. Navedli smo nekaj splošnih napotkov za ponudnika storitev. Predstavili smo tveganje, s katerim se soočajo potencialni odjemalci storitev. Opisali smo vpliv na IT oddelek in analizo TCO (Total Cost of Ownership), s katero določimo skupni strošek lastništva aplikacije. Na podlagi te analize se odjemalec lahko odloči, če je nakup dane aplikacije smotrn. Na koncu smo implementirali vzorčno večnajemniško aplikacijo.

Ključne besede:

Računalništvo v oblaku, programska oprema kot storitev, SaaS, programski model, poslovni model.

Abstract

The thesis deals with one of three cloud computing service models – software as a service. Software as a service is an application delivery model and includes architectural aspects for application implementation and business model, which is quite different than traditional application business model. After short introduction we placed subject into wide area of computer science. We defined the concept of software as a service, its main characteristics and basic terms that are used throughout the thesis. We mentioned maturity model and two simple SaaS application classifications. We presented SaaS software model, which includes multitenant model, access control, integration, composite architecture, metadata services, monitoring and measuring. We described SaaS business model, its main characteristics and changes inside enterprises that come with embracing SaaS model. Furthermore we described service level agreement and its characteristics. SaaS has big influence both for service providers and service customers. We described effects on IT and presented TCO analysis, which is used to determine total cost of application ownership. Service customer can decide whether to purchase an application on outcome of TCO analysis. At the end we developed sample multitenant application.

Key words:

Cloud computing, software as a service, SaaS, software model, business model.

1. Uvod

1.1. Motivacija

Glede na raziskave podjetja Gartner Inc. so letni stroški za lastništvo in upravljanje programske opreme štiri krat večji od stroškov začetnega nakupa. Rezultat tega je, da podjetja zapravijo več kot 75% proračuna, ki je namenjen za IT, samo za poganjanje in vzdrževanje obstoječih sistemov in infrastrukture [4]. Z vpeljavo računalniško podprtih informacijskih sistemov so podjetja vzela to kot nujen strošek poslovanja. Število aplikacij, ki bi jih podjetje potrebovalo, je lahko zelo veliko, proračun pa je pogosto zelo omejen.

Koncept SaaS omogoča podjetjem, da se naročijo na storitve ter da nakup in upravljanje infrastrukture prepustijo zunanjemu izvajalcu – ponudniku storitev. V večini primerov ponudnik storitev izvaja opravila, ki so povezana z upravljanjem infrastrukture, veliko bolj učinkovito in z manjšimi stroški, kot so to sposobni posamezni odjemalci storitev. To neposredno vpliva na zmanjšanje stroškov odjemalcev. Kot rezultat lahko proračun, ki je namenjen za IT, porabijo za več drugih aplikacij, ki pripomorejo k podpori in rasti poslovanja.

Ključni povzročitelji stroškov za katerokoli aplikacijo so stroški aplikacije same, stroški strojne opreme, ki je potrebna za poganjanje aplikacij, in človeške storitve, ki so potrebne za vzdrževanje in podporo. Cena tradicionalne programske opreme sovpada s stroški aplikacije. V večini primerov je to enkratni vnaprejšnji znesek za uporabniško licenco. Odjemalec mora sam ugotoviti, kakšni so stroški strojne opreme in človeških storitev. Za uporabo SaaS aplikacij se zaračunava naročnina, ki vključuje stroške programske in strojne opreme ter človeških storitev.

Cilji te diplomske naloge so predstaviti programski in poslovni model koncepta SaaS ter kakšen vpliv ima na odjemalca in ponudnika storitev. Namen je spoznati lastnosti koncepta SaaS, da lahko podjetja izkoristijo prednosti, ki jih ponuja.

1.2. Zgradba diplomske naloge

Drugo poglavje predstavlja kratek uvod, kam se umešča koncept SaaS. V tretjem poglavju je bolj natančno definiran koncept SaaS. Predstavljene so lastnosti modela in definicije pojmov, ki so uporabljene v tej diplomski nalogi. Četrto poglavje predstavlja programski model koncepta SaaS. V tem poglavju so predstavljeni nekateri izzivi, s katerimi se soočajo arhitekti

aplikacije, arhitekti podatkovnih baz in programerji pri načrtovanju in implementaciji aplikacij, za katere je bilo načrtovano, da so narejene po konceptu SaaS, in nekateri vzorci, ki jih lahko uporabijo pri implementaciji. Peto poglavje je namenjeno poslovnemu modelu. Predstavljene so glavne lastnosti poslovnega modela, če sledimo konceptu SaaS, in do kakšnih sprememb lahko pride v podjetju pri sprejemu tega modela. Poleg tega so predstavljeni nekateri splošni napotki, ki je dobro, da jih ponudnik storitev upošteva, in tveganje, s katerim se mora soočiti potencialni odjemalec storitev. Opisan je vpliv na IT oddelek in analiza TCO (angl. Total Cost of Ownership), s katero določimo skupni strošek lastništva aplikacije. Na podlagi te analize se odjemalec lahko odloči, če je nakup dane aplikacije smotrn. V šestem poglavju je predstavljena vzorčna večnajemniška aplikacija, ki smo jo implementirali. Navedena je platforma za razvoj in poganjanje aplikacije. Opisana je tematika, podatkovni model in avtentikacija ter prikazan primer, ki ponazarja večnajemniški model. Sledijo zaključek, seznam slik, seznam izvorne kode in seznam uporabljene literature.

2. Računalništvo v oblaku

2.1. Uvod

Računalništvo v oblaku je uporaba strojnih in programskih virov, ki so na voljo kot storitev v omrežju. Ime je dobilo po poenostavljenem simbolu, ki predstavlja kompleksno infrastrukturo, ki jo vsebuje računalništvo v oblaku. Zapletena sestava svetovnega spleta in druge podrobnosti so skrite v tej poenostavljeni shemi. Ta koncept je zelo uporaben, saj odjemalec storitev želi v največji meri izkoristiti prednosti svetovnega spleta, pri čemer ga podrobnosti pogosto sploh ne zanimajo.

Pojem storitev se uporablja za opravilo, ki je bilo sestavljeno z namenom, da se ga avtomatizira in da ga je možno večkrat uporabiti na isti način. Računalništvo v oblaku sestavljajo različni tipi storitev. Za lažje razumevanje jih razdelimo v tri modele in so prikazani na sliki 1. Vsi trije modeli potrebujejo upravljanje in administracijo. Ti modeli se imenujejo infrastruktura kot storitev, platforma kot storitev in programska oprema kot storitev [6].

Stranke dostopajo do storitev samo preko definiranih vmesnikov. Za vmesnikom se lahko nahaja kompleksna logika, ki je stranka ne rabi poznati niti razumeti, da lahko storitve vseeno uporablja.

Slika 1: Storitveni modeli [18]

2.2. Storitveni modeli

2.2.1. Infrastruktura kot storitev

Infrastruktura kot storitev (angl. Infrastructure as a Service) je najbolj osnoven model računalništva v oblaku, kjer ponudnik nudi računalniške vire (strežnike, mrežno opremo, shranjevanje podatkov in prostor za podatkovni center) kot storitev. Pogosto so ti viri na voljo stranki preko virtualizacije, ki te vire upravlja. Na ta način stranka najame računalniške vire, namesto da si kupi lastno infrastrukturo. Storitev lahko vključuje dinamično razširljivost. Dinamična razširljivost pomeni, da se viri dinamično dodajajo in odvzemajo glede na potrebe aplikacije. Tako lahko stranka nemudoma dobi več virov, kot jih je bilo sprva predvidenih. Poleg tega stranka in ponudnik skleneta dogovor o nivoju storitev, ki določa, kakšne nivoje dosegljivosti in odzivnosti mora ponudnik izpolnjevati.

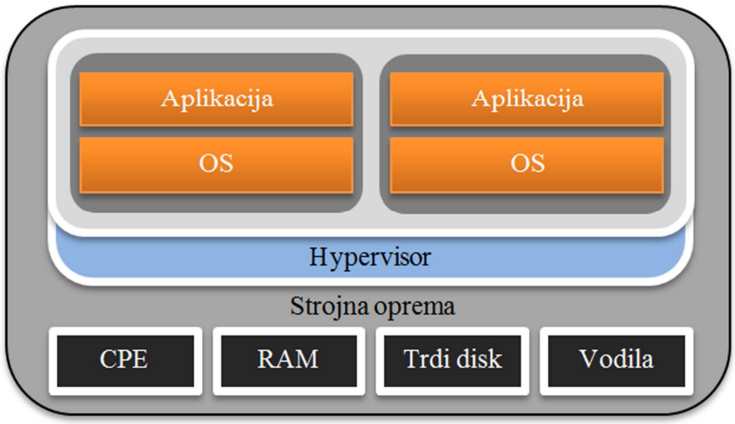
Tipični odjemalci so podjetja z veliko raziskovalnimi projekti. Te storitve omogočajo znanstvenikom in medicinskim raziskovalcem dostop do dodatne infrastrukture.

En od najbolj znanih ponudnikov infrastrukture kot storitve je Amazon s svojo storitvijo Amazon Elastic Compute Cloud (Amazon EC2). To je spletna storitev, ki nudi razširljive računske vire v oblaku. EC2 vsebuje spletni vmesnik, ki omogoča strankam dostop do virtualnih naprav. Okolje je dinamično razširljivo, kar pomeni, da se infrastrukturni viri avtomatično povečajo ali zmanjšajo glede na potrebe. Na ta način stranka dobi vire, ki jih potrebuje in za katere plača uporabo na uro. Amazon podpira operacijske sisteme Linux, Solaris in Windows.

Virtualizacija

Virtualizacija pomeni uporabo računalniških virov za oponašanje drugih računalniških virov ali celih računalnikov. Nudi platformo za optimizacijo računalniških virov, ki je enostavno razširljiva. Takšna platforma je optimalna za poganjanje storitev. Brez virtualizacije je oblak zelo težko upravljati. Virtualizacija je pomembna, ker predstavi kompleksne računalniške vire na enostaven način.

Virtualizacija ločuje vire in storitve od fizične infrastrukture. Za lažjo predstavo je na sliki 2 predstavljen logični diagram. Z virtualizacijo je možno na enem fizičnem sistemu poganjati več aplikacij in operacijskih sistemov. To se imenuje particioniranje. Tako dobimo več virtualnih naprav, ki so med seboj izolirane. To pomeni, da to ne vpliva na ostale, če pride do napake na eni od virtualnih naprav, saj so med seboj neodvisne. Poleg tega se podatki ne delijo med virtualnimi napravami. Virtualno napravo je možno predstaviti in shraniti kot eno samo datoteko. Tako lahko enostavno ugotovimo, katere storitve tečejo na njej.



Slika 2: Logični diagram virtualizacije [21]

Za upravljanje virtualnih naprav se uporablja t.i. »hypervisor« (včasih imenovan tudi angl. virtual machine manager). To je operacijski sistem, ki teče neposredno na strojni opremi. Gostujoči operacijski sistem (npr. Windows, Linux, FreeBSD, …) teče na enem nivoju višje. Prednost hypervisorja je, da kontrolira in razporeja strojne vire med virtualne naprave. Tako se gostujoči operacijski sistem ne zaveda, da teče na virtualni napravi.

2.2.2. Platforma kot storitev

Platforma kot storitev (angl. Platform as a Service) je model, kjer ponudnik nudi računalniško platformo ter paket sistemskih in programskih rešitev za poganjanje aplikacij. To vključuje operacijski sistem, izvršilni programski jezik, podatkovno bazo, aplikacijski strežnik in spletni strežnik. Na PaaS lahko gledamo kot na evolucijo gostovanja spletnih strani. Ponudniki so nudili celotno izvajalno okolje za razvoj spletnih strani. PaaS pa nudi upravljanje nameščene programske opreme skozi celoten življenjski cikel.

Slaba stran modela PaaS je, da si primoran uporabljati določene sistemske in programske komponente. Za odjemalca to ni najboljše, če želi zamenjati ponudnika storitev, če s trenutnim ni zadovoljen. Tako se lahko zgodi, da mora del aplikacije ponovno sprogramirati, kar pa mu lahko povzroči veliko stroškov.

Kot primer platforme kot storitve lahko navedemo Google App Engine in Force.com, ki je uradno razvijalno okolje za Salesforce.com. Od bolj poznanih so tu še Windows Azure, AppJet, Etelos in Qrimp.

2.2.3. Programska oprema kot storitev

Programska oprema kot storitev (angl. Software as a Service) predstavlja poslovne aplikacije, ki so nameščene pri ponudniku in dostavljene kot storitev. Glavna prednost tega modela je, da

stranka plača glede na uporabo in da ni začetnih stroškov za licence. V resnici je potrebno pred uporabo teh storitev izvesti integracijo med storitvami v oblaku in obstoječimi sistemi. Podjetja imajo takojšnjo korist, ker na ta način znižajo glavne odhodke. Dodatna prednost je tudi fleksibilnost, saj imajo običajno možnost testiranja nove programske opreme, preden podpišejo pogodbo, in lahko nadaljujejo z uporabo, če jim aplikacija ustreza.

Obstajajo SaaS aplikacije, ki jih je možno poganjati v oblaku z zelo nizkimi stroški. To so aplikacije z ogromno uporabniki, kjer uporabniki delajo popolnoma enake stvari. Primer takšne aplikacije je Yahoo Mail. To je možno, ker lahko ponudnik optimizira strojno in programsko opremo ter komunikacijo samo za določen tip bremena.

Facebook, Skype, eBay, GoogleApps so narejeni za masovno uporabo. Vse te aplikacije se uporabljajo tudi za poslovne namene.

3. Koncept SaaS

3.1. Uvod

SaaS je programska oprema, ki je nameščena pri ponudniku programske opreme in do katere se dostopa preko interneta.

Danes SaaS pomeni koncept, po katerem je razvito čedalje več aplikacij. Po tem modelu imamo eno različico programske kode za vse stranke, ki uporabljajo naš program, vendar je ta različica implementirana po večnajemniški arhitekturi (angl. multi-tenant architecture). Tipično SaaS aplikacijo prodaja ponudnik ali posrednik, ki poveže SaaS aplikacije različnih prodajalcev in jih nato ponuja svojim strankam kot celovito rešitev.

Ponudniki SaaS aplikacij uporabljajo drugačen model licenciranja programske opreme kot prodajalci tradicionalne programske opreme. SaaS aplikacije se pogosto prodajajo z naročnino, ki jo stranka plača z namenom uporabe aplikacij.

SaaS ponudnik hrani aplikacije in podatke distribuirane po svojih strežnikih, ki se običajno nahajajo na različnih lokacijah. Na strežnikih tečejo identične instance iste aplikacije in tako je namestitev popravkov in nadgradenj transparentna. Ponudnik mora imeti implementiran mehanizem, ki namesti posodobitve na vse lokacije, da vsi uporabniki uporabljajo iste verzije aplikacij in da vsi uporabniki ob istem času dobijo posodobitve. Uporabniki dostopajo do aplikacij preko spletnega brskalnika ali preko »smart-client« aplikacije. Marsikateri prodajalec pripravi programske vmesnike (angl. application programming interface), s katerimi izpostavijo aplikacijske podatke in funkcionalnosti, da lahko razvijalci te funkcionalnosti uporabijo pri svojih aplikacijah. Obstaja množica varnostnih mehanizmov, s katerimi zavarujemo občutljive podatke pri prenosu in pri shranjevanju, kot sta na primer kodiranje podatkov in seznami za kontrolo dostopa. Poleg tega nekateri ponudniki zagotovijo uporabnikom orodja, s katerimi lahko uporabniki spreminjajo podatkovne sheme, tokove in druge vidike delovanja aplikacij glede na uporabnikove potrebe.

3.2. Opredelitev temeljnih pojmov

Storitev (angl. service)

Storitev je entiteta, ki nudi zaokroženo funkcionalnost, ki jo je možno ponovno uporabiti za različne namene. Dostop je omogočen z uporabo predpisanih vmesnikov. Eden od načinov implementacije so spletne storitev, ki omogočajo interakcijo med napravami v omrežju in vsebujejo vmesnik, ki je v formatu, ki ga naprave lahko procesirajo [43]. V ta namen spletna storitev vsebuje vmesnik, ki je definiran v formatu WSDL (angl. Web Service Description Language). WSDL je jezik, ki temelji na standardu XML, ter vsebuje informacije, kako se storitev kliče, kakšne parametre pričakuje in kakšne podatkovne strukture vrača [44].

Ponudnik storitev (angl. application service provider)[1]

Ponudnik je podjetje, ki trži storitve preko omrežja. Programska oprema, ki je na voljo po tem modelu, se imenuje »Software as a Service« ali včasih »on-demand software«. Zagotoviti mora določen nivo izvajanja storitev. To vključuje med drugim tudi strežniško in omrežno infrastrukturo. Če si infrastrukture ne lasti sam, mora od lastnika infrastrukture (običajno ponudnika oblaka) dobiti potrebne garancije.

Stranka/odjemalec storitev (angl. tenant)

Odjemalec storitev je pojem, ki se v tem dokumentu nanaša na podjetje, ki pri ponudniku storitev naroči oziroma zakupi SaaS storitve.

Končni uporabnik (angl. end user)

Končni uporabnik (ali samo uporabnik) je zaposleni delavec odjemalca storitev, ki bo redno uporabljal naročene storitve.

Tradicionalni model[2]

Tradicionalne aplikacije (včasih imenovane tudi »on-premise software«) so zasnovane na modelu z velikimi začetnimi stroški uporabniških licenc in letnimi stroški podpore. Pomembne funkcije so vključene v začetnem paketu, ki ga pokrije uporabniška licenca. Poleg tega sklenjeni dogovor običajno vključuje letne zneske za posodobitve in podporo. Večje število uporabnikov lahko povzroči tudi večje stroške osnovnih sredstev zaradi potrebe po dodatnih strežnikih in dodatni IT podpori. Stroški licenc običajno temeljijo na metrikah, ki niso povezane z uporabo aplikacije (npr. tip strežnika, število procesnih enot…).

Tipična poslovna aplikacija potrebuje namestitev strojne opreme, strežnikov, sistema za varnostne kopije, mrežno opremo, zato da zadostimo vsem uporabnikom, ki dostopajo do programske opreme tako od znotraj kot od zunaj lokalne mreže. Prav tako mora biti

[1] Povzeto po [45].
[2] Povzeto po [4].

vzpostavljen varnostni sistem, ki preprečuje nedovoljene dostope do računalniških virov. Tradicionalne aplikacije so običajno močno prilagojene potrebam uporabnikov. Prilagajanje prinese določeno količino stroškov in zahteva določeno količino vloženega dela.

Celotno sprotno vzdrževanje in upravljanje mora zagotoviti stranka. Prav tako je stranka odgovorna za zagotavljanje logične in fizične varnosti, usposabljanja končnih uporabnikov in podporo. Tradicionalni programski model po arhitekturi ni večnajemniški.

3.3. Glavne značilnosti koncepta SaaS

3.3.1. Večnajemniška arhitektura

Večnajemništvo je arhitekturni model, ki se uporablja z namenom povečati razširljivost aplikacije. Isto instanco aplikacije lahko hkrati uporablja več odjemalcev in večnajemništvo zagotavlja, da so odjemalci ločeni med seboj in da vsak dostopa samo do lastnih podatkov. Model zagotavlja ponudniku storitev maksimalno razširljivost in nižje stroške podpore z vplivanjem na uporabniški vmesnik, poslovno logiko in/ali podatkovno bazo za vse odjemalce in uporabnike. Večnajemniški model, ki je implementiran na vseh nivojih aplikacije, se smatra kot optimalni model. Kljub temu ni nujno potreben na vseh nivojih, da izkoristimo vse prednosti. Prihranki, ki so posledica tega modela, so ključni, da lahko odjemalcem ponudimo storitve po nižji ceni [17].

3.3.2. Konfigurabilnost

SaaS aplikacije podpirajo prilagoditve, kar se doseže z množico nastavitev [7]. Z drugimi besedami, odjemalec lahko spremeni nastavitve oziroma nekatere parametre aplikacije, ki vplivajo na funkcionalnost in uporabniške vmesnike. Tako ima vsak posamezen odjemalec svoje nastavitve oziroma vrednosti parametrov. Pogosto ponudnik storitev nudi že določene privzete nastavitve.

3.3.3. Pospešena namestitev

SaaS aplikacijam je lažje namestiti popravke in posodobitve, zato lahko spremembe nameščamo pogosteje. Na to pozitivno vpliva več faktorjev. Aplikacija je nameščena pri ponudniku, kar pomeni, da ni fizičnega naseljevanja programa k stranki. Aplikacija ima običajno eno konfiguracijo in vsi podatki so nameščeni pri ponudniku, kar posledično pomeni, da sta razvoj in testiranje hitrejša. Poleg tega imamo na voljo orodja za spremljanje

uporabnikovega vedenja (angl. web analytics), da lažje ugotovimo, kje se splača aplikacijo izboljšati. Pospešena namestitev je dodatno omogočena z uporabo agilnih metodologij razvoja (npr. Scrum, Extreme programming) [7].

3.3.4. Odprti protokoli

Ker SaaS aplikacije nimajo dostopa do sistemov znotraj podjetja odjemalca, običajno nudijo integracijske protokole in programske vmesnike, ki delujejo v omrežju. Običajno so ti protokoli zasnovani na podlagi HTTP (angl. hypertext transfer protocol), REST (angl. Representational State Transfer), SOAP (angl. Simple Object Access Protocol) in JSON (angl. Javascript Object Notation) [7].

Danes je še vedno problem, da imajo različni ponudniki storitev svoje programske vmesnike. To pomeni, da smo včasih primorani, da vgradimo podporo za več vmesnikov. Upravljanje več vmesnikov pomeni večjo količino programiranja in napak, če pride do sprememb v aplikacijah. Zato je to danes en od najbolj pomembnih področij za standardizacijo.

Razširjenost SaaS aplikacij in drugih internetnih storitev ter standardizacija programskih vmesnikov je povzročila razvoj t.i. »mashup« aplikacij [7]. To so manjše aplikacije, ki združujejo podatke, prikaze in funkcionalnosti iz različnih virov v eni sami storitvi.

3.3.5. Funkcionalnosti za sodelovanje

Danes obstaja že kar nekaj SaaS aplikacij, ki imajo vgrajene funkcionalnosti za sodelovanje, ki omogoča izmenjavo informacij med uporabniki. Čeprav je možno, da so nekatere od teh funkcionalnosti integrirane tudi v tradicionalne aplikacije, je sodelovanje med uporabniki iz različnih podjetij možno samo, če je programska oprema gostovana pri enem ponudniku [7].

3.3.6. Skalabilnost[3]

Skalabilnost je lastnost sistemov, omrežja ali procesa, ki omogoča prilagajanje kapacitet obremenitvam, da so sistemi čim bolj učinkovito izrabljeni. V ta namen se uporabljajo različni tehnološki koncepti, ki omogočajo učinkovito razdeljevanje virov, npr. optimizacija časa zaklepanja, deljenje sredstev, kot so niti in omrežne povezave, predpomnjenje podatkov in particioniranje obsežnih podatkovnih baz. Za aplikacije, ki so narejene po modelu SaaS, je zelo pomembno, da podpirajo veliko število uporabnikov. Število uporabnikov je še toliko večje, saj SaaS aplikacije uporabljajo uporabniki iz različnih podjetij in ne samo iz enega.

[3] Povzeto po [8, 46].

Naloga arhitekta je, da zasnuje sistem, ki se ga bo dostopalo preko interneta in kjer se lahko število uporabnikov hitro povečuje.

Skalabilnost lahko zagotovimo na dva načina: aplikacijo premestimo na večji in močnejši strežnik ali pa aplikacijo poganjamo na več strežnikih. Če se za poganjanje aplikacije uporablja star strežnik, je boljša izbira nakup novega strežnika. V splošnem pa velja, da je za SaaS aplikacije bolj primerno poganjati aplikacije na več strežnikih, da lahko zagotovimo uravnavanje obremenitve. V tem primeru mora biti aplikacija ustrezno zasnovana, da lahko teče na poljubnem številu strežnikov, kjer na vsakem strežniku teče ena ali več instanc iste aplikacije.

3.3.7. Samooskrba z viri

Stranke lahko dobijo storitve v oblaku brez dolgotrajnega postopka. Stranka enostavno zahteva določeno količino virov (procesorski viri, skladišče podatkov, programska oprema) od ponudnika storitev [1].

Ta proces je ravno nasproten od procesa pri tipičnem podatkovnem centru. Ko nek oddelek implementira novo aplikacijo, mora poslati zahtevo podatkovnemu centru za dodatne vire. Podatkovni center dobi podobne zahteve od različnih oddelkov in te zahteve mora razvrstiti in ugotoviti razpoložljivost obstoječih virov. Po potrebi morajo kupiti dodatno strojno opremo, ki pa jo je potrebno še ustrezno nastaviti za poganjanje aplikacij. Nabavni procesi znotraj podjetja pa lahko vzamejo kar nekaj časa, odvisno od politike znotraj podjetja.

3.3.8. Obračun storitev in merjenje

Za zaračunavanje odjemalcem je potrebno meriti njihovo uporabo aplikacij, virov, hitrost in količino prenesenih podatkov ipd. Zato potrebujemo vgrajene storitve, ki le-to merijo. Merijo se tudi storitve, ki so na voljo brezplačno širši populaciji (npr. GoogleMail).

3.3.9. Nadzor delovanja

SaaS aplikacije morajo imeti tudi okolje za upravljanje storitev. To je integrirano okolje za upravljanje fizičnega okolja in IT sistemov, ki mora zagotavljati ustrezen nivo storitev. Z drugimi besedami, upravljanje storitev mora vsebovati nadzor in optimizacijo množice storitev. Ključna točka je delovanje celotnega sistema. To je način, s katerim ponudnik dokaže odjemalcu, da izpolnjuje svoje obveznosti iz pogodbe. Čeprav nadzor delovanja običajno zagotavlja ponudnik storitev, si odjemalci včasih priskrbijo lastna orodja.

3.3.10. Varnost

SaaS aplikacija se namesti v oblak, kar pa predstavlja določen varnostni problem. Ponudnik storitev mora zagotoviti, da je dostop do informacij onemogočen drugim podjetjem in zunanjim vdorom ter da se varnostni vidiki upoštevajo tudi pri prenosu podatkov.

3.3.11. Zrelostni model[4]

Z vidika programskega arhitekta je zrela SaaS aplikacija razširljiva, zasnovana po večnajemniški arhitekturi in jo je za različne potrebe odjemalcev možno nastaviti.

Večnajemniška arhitektura pomeni, da isto aplikacijo uporablja večje število strank. Skupni podatki, ki niso vezani na posamezno stranko, so vidni vsem strankam, obenem pa stranka vidi samo svoje podatke in ne tudi podatkov drugih strank. Arhitektura mora biti seveda tako zasnovana, da je možno podatke razlikovati glede na to, kateri stranki pripadajo. S tako arhitekturo se maksimalno poveča deljenje sredstev med posameznimi odjemalci, medtem ko je še vedno možno razlikovati med podatki, ki pripadajo različnim odjemalcem.

Če isto instanco aplikacije poganjamo za različne odjemalce, je programska koda skupna vsem. Torej ne moremo enostavno spreminjati aplikacije, da zagotovimo zahtevam katerega od odjemalcev. Namesto programiranja aplikacije za vsakega odjemalca posebej je potrebno aplikacijo implementirati na način, da odjemalec z metapodatki nastavi aplikacijo, da bo zadostovala njegovim potrebam. Tu se programski arhitekt sooči z izzivom, kako zagotoviti nastavitve aplikacije, da si bodo odjemalci na enostaven način le-to nastavili brez dodatnega razvoja ali stroškov za ponudnika programske opreme.

Zgoraj so navedene lastnosti dovršene SaaS aplikacije. Ko pogledamo določeno aplikacijo, pa lahko vsebuje samo eno ali dve od navedenih lastnosti, ki še vedno sledi zahtevam poslovnega procesa. Programski arhitekt se lahko odloči, da namenoma ne sledi vsem lastnostim, saj je lahko v nasprotnem primeru izvedba neustrezna glede na zahteve poslovnega procesa.

Dovršenost SaaS aplikacije je možno ponazoriti z zrelostnim modelom, ki je prikazan na sliki 3. Zrelostni model ima štiri nivoje, ki se med seboj razlikujejo glede na to, koliko in katere od navedenih lastnosti dovršene SaaS aplikacije so implementirane.

[4] Povzeto po [8].

3.3.11.1. Nivoji zrelostnega modela

Prvo nivo

V prvi nivo zrelostnega modela spadajo aplikacije, ki niso zrele SaaS aplikacije. Vsak odjemalec ima svojo prilagojeno verzijo iste aplikacije.

Aplikacije, ki temeljijo na odjemalec-strežnik modelu, je možno dokaj enostavno in brez večjih posegov v arhitekturo pretvoriti v aplikacijo, ki zadostuje prvemu nivoju zrelostnega modela. Taka aplikacija vsebuje malo prednosti dovršene SaaS aplikacije, vseeno pa nudi zmanjšanje stroškov ponudniku z združevanjem strojne opreme in administracije.

Drugi nivo

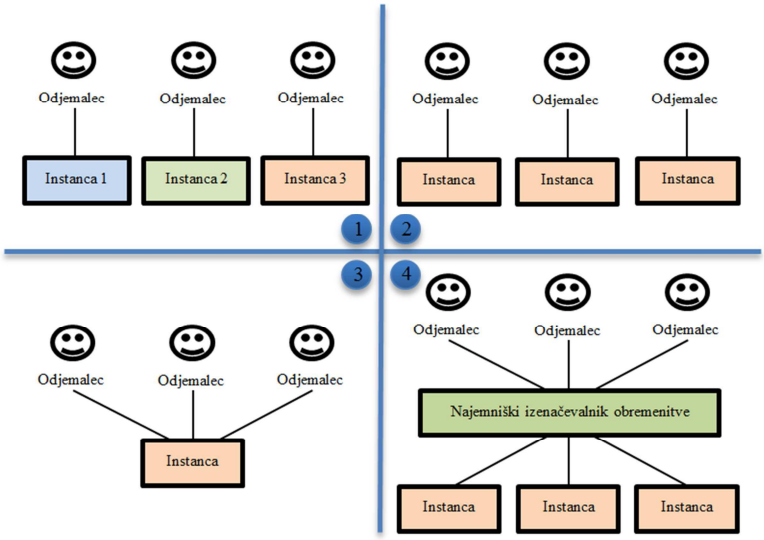
Za aplikacije na drugem nivoju je značilna večja fleksibilnost, saj jih je možno nastaviti z metapodatki. Odjemalci še vedno uporabljajo ločene instance, ki so nameščene pri ponudniku. Razlika v primerjavi s prvim nivojem je, da so instance identične in uporabljajo isto programsko kodo. Ponudnik lahko zadosti individualnim potrebam odjemalcev z množico nastavitev, ki vplivajo na izgled in obnašanje aplikacije. Posledično je tudi veliko lažje vzdrževanje aplikacije, če ponudnik spremembe striktno naseli vsem strankam. Slaba stran tega nivoja je, da je potrebno v primeru migracije implementirati veliko arhitekturnih sprememb, če dane aplikacije že v začetku ni bilo možno nastavljati.

Tretji nivo

Na tretjem nivoju so aplikacije, ki so narejene po večnajemniški arhitekturi in si jih lahko odjemalci nastavijo glede na svoje potrebe. Ponudnik ima nameščeno eno instanco aplikacije, ki jo uporabljajo vsi odjemalci. To posledično pomeni večji napor pri razvoju, saj je potrebno zagotoviti, da podatki ostanejo izolirani in da kljub eni instanci odjemalci ne opazijo, da isto instanco uporabljajo tudi drugi. Ta nivo zagotavlja nižje stroške zaradi manjših potreb po strežniških virih, saj si pomnilniške in računske kapacitete deli večje število odjemalcev. Slabost tega nivoja je omejena skalabilnost, saj aplikacij na tem nivoju ne upravlja najemniški izenačevalnik obremenitve.

Četrti nivo

Na četrtem nivoju ponudnik gosti večje število odjemalcev na skupini identičnih instanc aplikacije, ki jih upravlja najemniški izenačevalnik obremenitve. Podatki posameznih odjemalcev so logično ločeni in vsak odjemalec si nastavi aplikacijo z metapodatki. SaaS aplikacijo je možno razširiti na poljubno število odjemalcev brez sprememb v arhitekturi in posega v programsko kodo. Kapaciteto sistema je možno dinamično povečati ali zmanjšati glede na obremenjenost in s tem je zagotovljena tudi skalabilnost. Popravke in dodelave je še vedno možno enostavno namestiti, saj so vse instance še vedno enake.



Slika 3: Zrelostni model SaaS aplikacij [8]

3.3.11.2. Izbira zrelostnega modela

Naloga ponudnika je, da izbere nivo zrelostnega modela, po katerem bo zgradil svojo aplikacijo. Načeloma strmimo k višjim nivojem, včasih pa zaradi zahtev odjemalca to ni mogoče. Zrelost SaaS aplikacije je možno predstaviti kot zvezno lestvico z ločenimi podatki in programsko kodo na eni strani ter skupnimi podatki in isto programsko kodo na drugi strani. Kje na zvezni lestvici se bo nahajala aplikacija, je odvisno od poslovnih, arhitekturnih in operativnih potreb ter obzirnosti strank.

3.4. Tipi SaaS aplikacij

Danes obstaja ogromno aplikacij, ki so narejene po SaaS modelu. Da imamo lažji pregled nad njimi, sta v nadaljevanju navedeni dve delitvi.

3.4.1. Delitev glede na funkcionalnost

- Celovita programska oprema (angl. packaged software)
- Programska oprema za sodelovanje (angl. colaborative software)
- Orodja za razvoj in upravljanje (angl. enabling and management tools)

Celovita programska oprema

To je največji del trga SaaS aplikacij. Sem spadajo integrirani poslovni informacijski sistemi (angl. enterprice resource planning), sistemi za upravljanje odnosov s strankami (angl.

customer relationship management), upravljanje oskrbovalnih verig (angl. supply chain management), upravljanje s financami (angl. financial management) itn. Te integrirane rešitve so osredotočene na posamezne procese, kot je na primer vodenje plač zaposlenih. Ti produkti imajo nekaj skupnih značilnosti: implementirani so za posamezen poslovni proces, ki si ga lahko stranka prilagodi svojim potrebam, in so v večini nameščeni v oblaku, ker je upravljanje za stranko običajno prezahtevno.

Primer take programske opreme je NetSuite, ki vsebuje več različnih aplikacij, ki pokrivajo naslednja področja: CRM, ERP, upravljanje s financami, elektronsko poslovanje in strokovna orodja za avtomatizacijo.

Programska oprema za sodelovanje

Ta del tržišča je danes zelo zanimiv, saj je internet dostopen vsepovsod. V to kategorijo spadajo aplikacije, ki so osredotočene na kakršenkoli način sodelovanja. Vključujejo spletne konference, upravljanje z dokumenti, projektno vodenje, neposredno sporočanje in elektronsko pošto. Bilo je samo vprašanje časa, kdaj bodo podjetja te aplikacije preselila v oblak. Glavni razlog za to je, da morajo biti te aplikacije dostopne vsepovsod v podjetju in iz različnih lokacij.

Eden od najbolj znanih ponudnikov je Google s svojim paketom GoogleApps, ki med drugim vključuje elektronsko pošto, socialno omrežje, neposredno sporočanje, upravljanje z dokumenti itd.

Orodja za razvoj in upravljanje

V to kategorijo spadajo razna orodja, ki jih potrebujejo razvijalci pri razvoju SaaS aplikacij. Čeprav so verjetno nekatere od teh storitev nameščene v privatnem oblaku v lastnem podatkovnem centru, obstaja veliko ponudnikov, ki nudijo uporabo teh storitev. V to kategorijo spadajo storitve iz področij testiranja, nadzora in upravljanja, razvoja, varnosti ter skladnosti s predpisi in vodenja.

Tudi če je podjetje migriralo aplikacije v oblak, mora še vedno opraviti testiranje, kot bi ga sicer. Testiranje mora vključevati funkcijsko, stresno, skladnostno, performančno, integracijsko testiranje, testiranje posameznih enot in upravljanje zahtev. S tem, ko uporabijo storitev testiranja, lahko prihranijo ogromno časa in denarja. Danes je na trgu že veliko ponudnikov, ki nudi testne platforme (npr. HP, IBM, Sogeti).

Da se odjemalci prepričajo, da ponudnik zagotavlja dogovorjene nivoje storitev, lahko odjemalci uporabijo lastno nadzorovanje. Nadzorovanje je še toliko bolj pomembno, ko imamo več aplikacij in je potrebno nadzorovati kombinacijo le-teh. Neustar, Compuware in Keynote Systems so le nekatera izmed podjetij, ki nudijo te rešitve kot storitve.

Nekateri razvijalci že uporabljajo oblak kot okolje za razvoj programske opreme. Tako so razvojna orodja ponujena kot storitve v oblaku namesto enega notranjega razvojnega okolja. Za zagotavljanje varnosti lahko uporabimo produkte za upravljanje z identitetami, ki so dostopni kot storitve. Kot storitve so danes dostopni tudi antivirusni programi. Symantec, McAfee, Kaspersky Labs so le nekatera od podjetij, ki sledijo trgu in nudijo antivirusne rešitve v oblaku.

Naloge iz področja skladnosti in upravljanja so zapletena in časovno potratna opravila. Zaradi tega je tudi smiselno, da podjetja razvijejo storitve za zagotavljanje teh zmožnosti. Nekatere od teh zmožnosti so upravljanje s popravki, načrtovanje neprekinjenosti delovanja, odkritje zapisov in sporočil ter različne zahteve za upravljanje.

3.4.2. Delitev glede na uporabnike

- Poslovne spletne storitve (angl. line-of-business services)
- Potrošniške spletne storitve (angl. consumer-oriented services)

Poslovne storitve

Poslovne storitve so dostopne podjetjem in organizacijam vseh velikosti. To so obsežne poslovne rešitve, ki jih je možno nastaviti za vsako stranko posebej. Običajno so to rešitve, ki olajšajo poslovne procese, kot so finance, materialno in skladiščno poslovanje ter odnosi s strankami. Te rešitve se običajno prodajajo z naročnino. Primer takšne aplikacije je že predhodno omenjeni NetSuite.

Potrošniške storitve

Potrošniške storitve so dostopne širši množici. Največkrat so te storitve ponujene zastonj, prihodki pa so odvisni od oglaševanja. Redko pa so tudi te storitve ponujene v zameno za naročnino. V to kategorijo med drugim spadata dve aplikacije, ki spadata v sam vrh po številu uporabnikov: Facebook in Gmail.

4. Programski model

4.1. Večnajemniška arhitektura

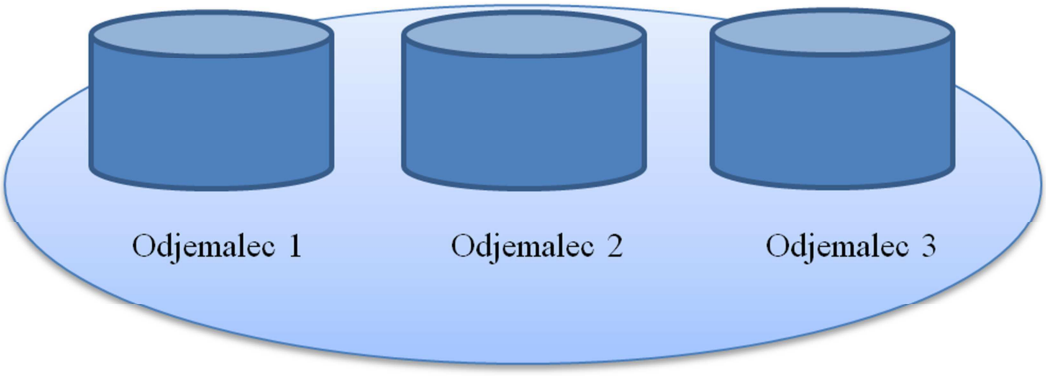
4.1.1. Pristopi za načrtovanje podatkovnega modela po večnajemniški arhitekturi[5]

Implementirana aplikacija vsebuje privzete zavihke, polja, poizvedbe in relacije, primerne danemu problemu. Pri tem imajo različne organizacije svoje zahteve, ki jim z nerazširljivim podatkovnim modelom ne bomo mogli zagotoviti. V nadaljevanju so opisani trije različni pristopi za reševanje tega problema:

- ločene podatkovne baze,
- skupna baza, ločene sheme in
- skupna baza, skupna shema.

4.1.1.1. Ločene podatkovne baze

Na sliki 4 je prikazan pristop, kjer ima vsak odjemalec svojo podatkovno bazo, ki jo lahko razširi glede na svoje potrebe. Prikazane so tri hipotetične podatkovne baze s tremi hipotetičnimi odjemalci in njihovimi identifikatorji.



Slika 4: Pristop z ločenimi podatkovnimi bazami [9]

[5] Poglavje je povzeto po [9, 38].

Podatkovne baze so lahko nameščene:

- na ločenih strežnikih: namestitev baze na ločene strežnike je primerna za odjemalce, ki imajo visoke zahteve glede izolacije podatkov, velike količine podatkov v bazi, veliko število uporabnikov ali druge specifične zahteve. Ta način povzroči velike stroške, ki so dodatno povečani zaradi redundantnih sistemov, pri večjem številu odjemalcev pa je oskrbovanje sistemov oteženo. Prednosti tega načina sta varnostni vidik, ki je tu na nivoju dostopa do strežnikov, in predvidljivo delovanje sistema, saj si odjemalci ne delijo strojnih virov (kot sta na primer pomnilnik in centralna procesna enota). Torej na delovanje ne vplivajo aktivnosti drugih odjemalcev.
- na ločenih virtualnih napravah: kreiranje ločenih virtualnih naprav za vsakega odjemalca povzroči manjše stroške kot ločeni strežniki, pri čemer je izolacija podatkov še vedno zagotovljena. Zaradi virtualizacije vsako bazo nadzoruje ločen proces, ampak deljenje virov je še vedno neučinkovito, saj odjemalec ne more koristiti neuporabljenih virov drugih odjemalcev.
- na skupnem strežniku oziroma virtualni napravi: pri tem načinu so baze nameščene na istem strežniku oziroma virtualni napravi in nadzoruje jih isti proces. Vsako bazo se dodeli svojemu odjemalcu in ta informacija se zabeleži v metapodatke. Pri tem načinu se varnost zagotovi na nivoju dostopa do baz, s čimer je dosežena tudi izolacija podatkov. V primerjavi s prejšnjima načinoma tu nastanejo nižji stroški, ker se viri bolj učinkovito delijo med odjemalce. Slaba stran je, da na delovanje vplivajo aktivnosti drugih odjemalcev.

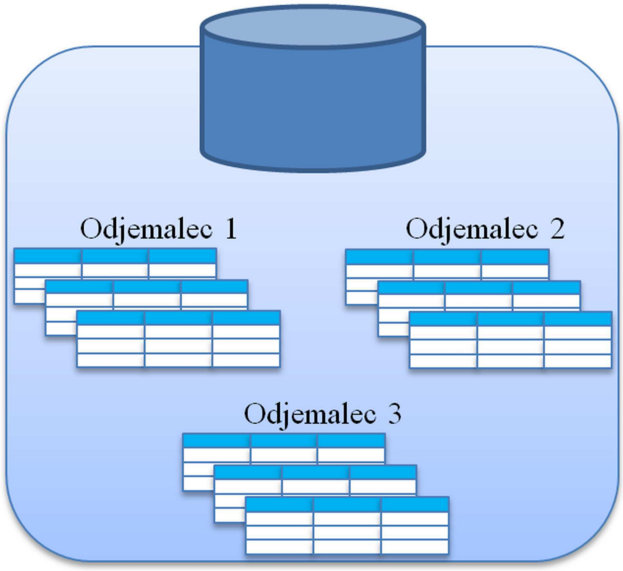
Podatkovni model je enostavno razširiti. Odjemalec lahko bazo prilagodi svojim potrebam toliko, kolikor mu dovoljuje uporabniški vmesnik in programska logika. Pri tem lahko kreira nova polja, tabele, poizvedbe in relacije. V primeru večjih napak lahko enostavno obnovimo bazo iz varnostne kopije, saj se za razliko od drugih dveh pristopov ne potrebujemo ozirati na baze drugih odjemalcev. Poleg tega razvoj zahteva minimalen dodaten čas v primerjavi s tradicionalno aplikacijo.

Pristop z ločenimi bazami je najboljši, ko odjemalci vztrajajo, da so njihove baze ločene in ne bodo pustili, da bi se delile z drugimi. To velja še posebno za bančne in medicinske podatke ter podatke, ki jih zaradi zakonskih predpisov ni dovoljeno hraniti izven države izvora. Skratka ta pristop je najboljši za odjemalce, ki so pripravljeni plačati za večjo varnost in večjo fleksibilnost.

4.1.1.2. Skupna baza, ločene sheme

Na sliki 5 je prikazan pristop, kjer so v isti bazi podatki od več odjemalcev, ki so ločeni po shemah. Vsak odjemalec ima svoj sklop tabel, kar pomeni, da je podatkovni model enostavno razširiti. Za vsakega odjemalca se kreira shema in podatek o pripadajočih shemah se shrani v metapodatke. Tabele se kreirajo s privzetimi polji, odjemalec pa lahko po potrebi doda in spreminja polja in tabele. Za zagotovitev varnosti je potrebno kreirati strukturo dovoljenj. Vsakemu odjemalcu se kreira drug uporabniški račun, ki je avtoriziran samo za določeno shemo. Na ta način je zagotovljena srednja stopnja izolacije podatkov.

Pri tem pristopu nastanejo manjši stroški, saj je možno na isti strežnik namestiti baze več odjemalcev. Podobno kot pri pristopu ločenih baz razvoj zahteva minimalen dodaten čas v primerjavi s tradicionalno aplikacijo. V primerjavi s prejšnjim pristopom je razlika predvsem v tem, kako je možno podatke obnoviti v primeru napak. V tem primeru ne moremo obnoviti podatkovne baze iz varnostne kopije, ker s tem prepišemo podatke vsem odjemalcem ne glede na to, če je tudi pri drugih odjemalcih prišlo do napak ali izgub podatkov. K obnovitvi podatkov je potrebno pristopiti drugače. Administrator podatkovne baze mora obnoviti bazo na začasni strežnik, pobrisati podatke v tabelah in uvoziti podatke iz začasne baze v tabele v produkcijski bazi. Le-to je lahko zahtevno in časovno potratno opravilo. Zato je ta pristop najboljši za aplikacije, kjer je število tabel majhno (npr. do 100 tabel).



Slika 5: Pristop s skupnimi podatkovnimi bazami in ločenimi shemami [9]

4.1.1.3. Skupna baza, skupna shema

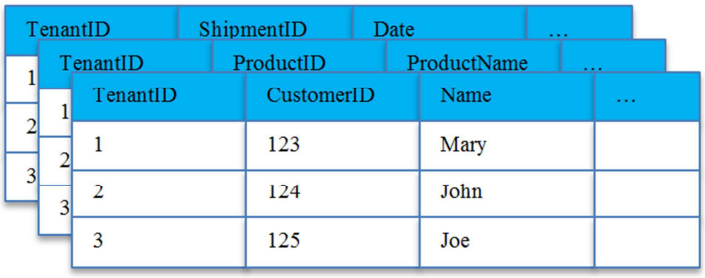
Ta pristop vključuje eno podatkovno bazo in eno shemo, ki si jo deli več odjemalcev. Ker se v istih tabelah nahajajo zapisi različnih odjemalcev, se tabelam doda polje TenantID, s katerim

enolično določimo odjemalca, kateremu pripada zapis. Slika 6 prikazuje tri različne tabele, v katere se shranjujejo podatki vseh odjemalcev.

Pri tem pristopu imamo najnižje stroške, saj je možno na istem strežniku gostiti največ odjemalcev v primerjavi z obema drugima pristopoma. Slabost je, da je potrebno pri razvoju aplikacije vložiti dodaten napor pri varovanju podatkov. Potrebno je preprečiti, da odjemalec vidi podatke od kogarkoli drugega in to v primeru tako načrtovanega napada na sistem kot v primeru še neodpravljenih napak. Posledično to pomeni, da je potrebno pri vsaki poizvedbi upoštevati polje TenantID. Pri tem pristopu je še toliko bolj pomembno, da opravimo obsežno in pazljivo testiranje, ki mora vsebovati tudi hitrost izvajanja poizvedb. Opcijsko je potrebno dodati dodatne indekse.

Obnovitev podatkov je zelo podobna kot pri ločenih shemah. Razlikuje se v tem, da je potrebno iz tabele pobrisati samo zapise, ki pripadajo določenemu odjemalcu. Poleg tega je potrebno paziti na število zapisov, ki se brišejo in ponovno vstavijo. Če je teh zapisov veliko, se lahko močno pozna pri delovanju aplikacije drugih odjemalcev.

Ta pristop je najbolj primeren, ko je pomembno, da isto aplikacijo poganja veliko število odjemalcev, da je število strežnikov majhno in da so zainteresirani odjemalci pripravljeni popustiti pri zahtevah glede izolacije podatkov. Zagotavlja najnižje stroške upravljanja podatkovnega strežnika in varnostnih kopij za nižjo ceno aplikacije.



Slika 6: Pristop s skupno bazo in skupnimi shemami [9]

4.1.2. Vzorci za realizacijo večnajemniške arhitekture

4.1.2.1. Vnaprej predvidena polja[6]

Za potrebe posameznih odjemalcev tabelam dodamo fiksno število dodatnih polj, ki jih lahko odjemalci poljubno uporabijo. Dodatna polja imajo lahko vnaprej določen tip podatkov ali pa ne. V primeru, da so tipi polj vnaprej določeni, lahko za preverjanje podatkov uporabimo vgrajene funkcije podatkovne baze in aplikacije. Na ta način je uporaba dodatnih polj

[6] Povzeto po [9].

omejena, kar lahko odjemalcu predstavlja oviro. Druga možnost je, da polja nimajo določenega tipa podatkov. V ta polja je možno shranjevati podatke kakršnegakoli tipa. V tem primeru preverjanja podatkov ni in je potrebno to funkcionalnost dodatno implementirati. Za ta namen se tip podatka shrani v metapodatke, ki se lahko prav tako shranjujejo na bazo. Tako lahko kreiramo eno tabelo, kamor se shranjujejo metapodatki vseh dodatnih polj, ali pa kreiramo za vsako dodatno polje svojo tabelo za shranjevanje metapodatkov. V primeru ločenih tabel za metapodatke dodatnih polj bi vsaka taka tabela običajno vsebovala tri kolone: TenantID, naziv dodatnega polja in tip dodatnega polja.

Slabost tega pristopa je zelo omejena razširljivost podatkovnega modela, ki je omejena s številom dodatnih polj. Potrebno je natančno premisliti, koliko polj bomo dodali in kakšnih tipov bodo, če se odločimo določiti tipe. Če je polj premalo, odjemalec ne bo mogel izkoristiti aplikacije, kot bi si želel. Če je polj preveč, imamo v bazi neuporabljena polja, ki zasedajo neizkoriščen prostor, in tudi podatkovni model takšne baze ni optimalen.

4.1.2.2. Pari ime-vrednost[7]

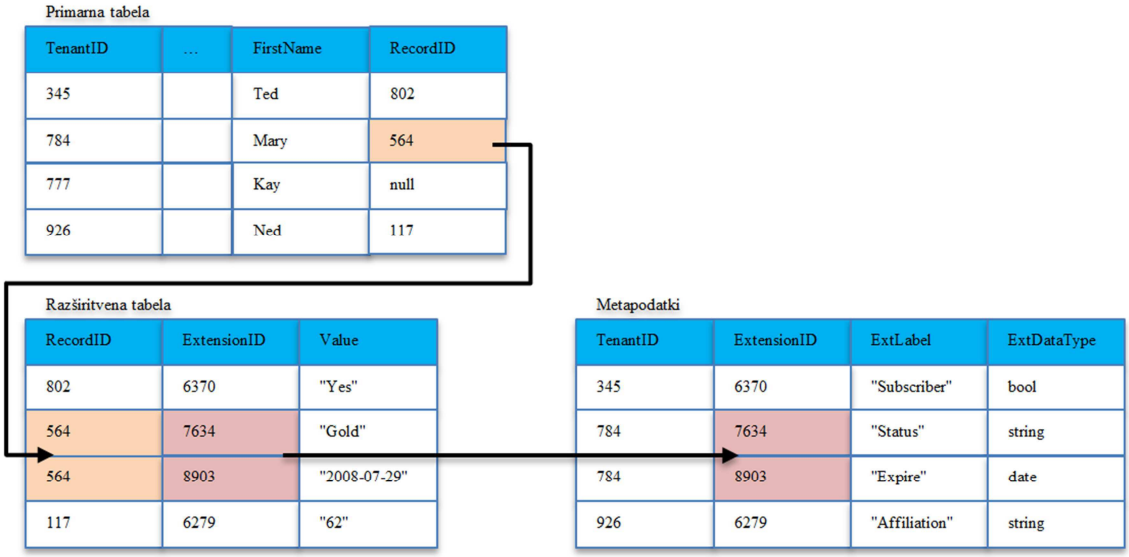
Ta pristop omogoča odjemalcem poljubno število dodatnih polj. Tu se dodatna polja shranjujejo v ločeno tabelo ter z metapodatki, ki se nahajajo v tretji tabeli, določimo naziv in podatkovni tip vsakega dodatnega polja.

Pri shranjevanju zapisa v dodatno tabelo se zgodi več stvari. Najprej se zapis vstavi ali posodobi v primarni tabeli: shranijo se samo vrednosti polj v primarni tabeli brez dodatnih polj. Nato se kreira enolični identifikator in se ga shrani v polje RecordID v primarni tabeli. Potem se pa shranijo še dodatna polja v ločeno tabelo z naslednjimi podatki: RecordID, ki se veže na istoimensko polje v primarni tabeli, ExtensionID, s katerim ločimo dodatna polja med seboj, in vrednost dodatnega polja, ki je pretvorjena v niz. Slika 7 prikazuje primarno tabelo, tabelo z dodatnimi podatki, tabelo z metapodatki in medsebojne relacije.

Ko se naredi poizvedba iz primarne tabele, se pogleda še, če obstajajo v dodatni tabeli ustrezni zapisi. Vrednosti ustreznih zapisov se vrnejo skupaj s podatki iz primarne tabele, pri čemer se še ustrezno pretvorijo glede na metapodatke. Da so pravi zapisi pretvorjeni v ustrezne tipe, zagotavlja polje ExtensionID.

Pri implementaciji tega pristopa je potrebno imeti v mislih tudi razširitev poslovne logike, da lahko aplikacija uporabi dodatne podatke, in uporabniških vmesnikov za prikaz ali vnos dodatnih podatkov. Vmesnik za nastavitve mora tako vsebovati mehanizme za spreminjanje vseh treh nivojev.

[7] Povzeto po [9].



Slika 7: Razširitev tabele s poljubnim številom dodatnih polj [9]

Ta pristop zagotavlja poljubno razširljivost podatkovnega modela. Slabosti tega pristopa sta večja kompleksnost podatkovnih funkcij, kot so na primer indeksiranje in poizvedbe. Posledično je potrebnega več napora za obnovitev podatkov odjemalca, saj je prisotna relacija med primarno tabelo in tabelo z dodatnimi polji. To je najboljši pristop, če odjemalec potrebuje čim večjo fleksibilnost pri razširitvi podatkovnega modela, če le odjemalci ne potrebujejo ločenih baz.

4.1.2.3. Razširitvena kolona tipa XML

Ta pristop omogoča razširitev podatkovnega modela s poljubnim številom dodatnih podatkov. V tabeli definiramo dodatno kolono tipa XML neodvisno od tega, koliko odjemalcev imamo in koliko dodatnih vrednosti bodo odjemalci imeli.

V metapodatke se za vsakega odjemalca shrani shema XML, ki služi za preverjanje dodatne kolone. Ko odjemalec zahteva razširitev tabele z novim poljem, se generira nova shema za preverjanje XML in shrani v metapodatke. V shemo se običajno shrani tudi naziv dodatnih polj.

Slabosti tega pristopa sta počasnejše izvajanje poizvedb in večja kompleksnost podatkovnih funkcij, kot sta indeksiranje in poizvedbe. Pristop se običajno uporablja, ko aplikacija uporablja XML za komunikacijo z drugimi spletnimi storitvami.

4.1.2.4. Zaščita tabel v podatkovni bazi[8]

Tabele v podatkovni bazi lahko zavarujemo z ukazoma GRANT in REVOKE. To sta ukaza, s katerima omogočimo uporabnikom in uporabniškim skupinam dostop in spreminjanje podatkov. Z ukazom GRANT avtoriziramo eno ali več operacij na tabeli ali drugem objektu na podatkovni bazi in z REVOKE odstranimo predhodno določene pravice.

```
GRANT SELECT, UPDATE ON [TableName] TO [UserName];
REVOKE SELECT, UPDATE ON [TableName] FROM [UserName];
```

Izvorna koda 1: Kontrola dostopa do objektov na podatkovni bazi

S tema ukazoma dodamo ali odstranimo uporabniški račun na seznam za nadzor dostopa (angl. access control list) za dano tabelo ali objekt. Ker imajo SaaS aplikacije običajno ogromno število uporabnikov, aplikacije za dostop do podatkovne baze ne uporabljajo uporabniška imena končnih uporabnikov, ampak se za vsakega odjemalca kreira dodatni uporabniški račun, preko katerega se vsem končnim uporabnikom danega odjemalca omogoči dostop do podatkovne baze. Na ta način je potrebno ta postopek izvesti samo, ko ponudnik programske opreme pridobi novega odjemalca.

Ta pristop je primeren za ločene sheme in ločene podatkovne baze. Pri ločenih bazah lahko podatke izoliramo s tem, da omejimo dostop do same baze glede na odjemalca, ki mu baza pripada. Vseeno pa lahko ta pristop uporabimo, da zagotovimo dodaten nivo varnosti. V nekaterih primerih smo celo prisiljeni uporabiti ta pristop, ko želimo omejiti dostop do tabel za različne uporabnike in želimo imeti interno varnostno shemo, ki je vezana na tabele. V teh primerih posledično ni dovolj kreiranje samo enega uporabniškega računa za dostop do podatkovne baze. Če je možno omejitev dostopa vezati na uporabniške skupine, potem lahko kreiramo uporabniške računa na podlagi uporabniških skupin, da se izognemo kreiranju računov za vsakega uporabnika posebej.

4.1.2.5. Pogled za filtriranje podatkov odjemalcev[9]

Za filtriranje podatkov, da odjemalci vidijo samo svoje podatke, se pogosto uporabljajo pogledi. V SQL-u je pogled (angl. view) virtualna tabela, ki je definirana z rezultatom poizvedbe SELECT. Poglede lahko uporabimo v poizvedbah in procedurah na isti način kot navadne tabele. Naslednji primer kreira pogled TenantEmployees, ki prikaže samo zaposlene, ki pripadajo posameznemu odjemalcu:

[8] Povzeto po [9].
[9] Povzeto po [9].

```
CREATE VIEW TenantEmployees AS
SELECT * FROM Employees WHERE TenantID = SUSER_SID()
```

Izvorna koda 2: Primer pogleda za filtriranje podatkov odjemalcev

Ta trditev pridobi identifikator uporabniškega računa, ki trenutno dostopa do baze, in ga uporabi za filtriranje vrstic, ki jih mora prikazati pogled. Pri tem primeru smo predpostavili, da se za dostop do baze uporabi uporabniški račun odjemalca in ne končnih uporabnikov, ter da se pri zapisih v tabelo uporabi isti identifikator. Vsakemu odjemalcu dodamo dostop do tega pogleda, ne pa tudi do tabele. Na ta način ima vsak odjemalec dostop samo do lastnih podatkov.

Ta pristop je nekoliko bolj kompleksen kot določanje pravic na tabelah, ampak je primeren, ko si več odjemalcev deli iste tabele.

4.1.2.6. Kodiranje podatkov[10]

Eden od načinov za zaščito podatkov je kodiranje. S kodiranjem dosežemo, da so naši podatki varni tudi v primeru, ko do njih pride tretja oseba.

Poznamo dva načina kodiranja: simetrično in asimetrično. Pri simetričnem kodiranje se ključ generira za kodiranje in dekodiranje podatkov. Podatke, ki so zakodirani s simetričnem ključem, je možno dekodirati z istim ključem. Težava tega pristopa je pri prenosu ključa samega. Ne glede na to, ali ključ pošljemo skupaj s podatki ali ločeno, je možno ključ prestreči. Pri asimetričnem kodiranju se uporabljata dva ključa: javni in privatni ključ. Podatke, ki so zakodirani z javnim ključem, je možno dekodirati samo s privatnim ključem in obratno. V splošnem so javni ključi dostopni vsem, ki želijo komunicirati z imetnikom, medtem ko se privatne ključe hrani na varnem.

Asimetrično kodiranje zahteva občutno več procesorske moči kot simetrično. Ustrezen par asimetričnih ključev porabi pri kodiranju 100 ali celo 1000-krat več časa kot kodiranje z ekvivalentnim simetričnem ključem. Za SaaS aplikacije je tako uporaba asimetričnih ključev neustrezna. Boljša rešitev je t.i. »key wrapping«, ki združuje prednosti obeh načinov.

Key wrapping

Key wrapping je način za zaščito kodirnih ključev pri prenosu v omrežju ali shranjevanju v nezaupljiven skladišču podatkov. Za vsakega odjemalca se kreirajo trije ključi: simetrični in par asimetričnih ključev. Za kodiranje podatkov se uporabi simetrični ključ, ker je kodiranje in dekodiranje hitrejše. Za dodaten nivo zaščite se uporablja še kodiranje in dekodiranje simetričnega ključa z asimetričnimi ključi.

[10] Povzeto po [9].

Ko se uporabnik prijavi v aplikacijo, le-ta uporabi impersonacijo za dostop do podatkovne baze, pri čemer uporabi varnostni kontekst odjemalca. Baza nudi aplikaciji dostop do privatnega ključa odjemalca in aplikacija lahko uporabi dani ključ za dekodiranje simetričnega ključa. S simetričnim ključem je nato možno pisati in brati podatke.

Slaba stran kodiranja podatkov je ta, da ni možno nastaviti indekse neposredno na kodirane kolone. Tako je potrebno pri izbiri kodiranih kolon napraviti kompromis med varnostjo podatkov in hitrostjo poizvedb.

Indeksiranje kodiranih kolon[11]

Na kodirane kolone ne moremo nastaviti indeksov. Kljub temu obstaja način, da uporabimo indekse, da nam pospeši izvajanje poizvedbe. Za vsak indeks dodamo dodatno kolono, katere vrednost je določena z izračunom na primer vrednosti MAC (angl. message authentication code). MAC je deterministična vrednost in posledično lahko na to dodatno kolono nastavimo indeks, saj se isti podatek vedno pretvori v isto kodo. Za izračun vrednosti MAC rabimo določiti začetni vektor, za katerega je najbolje, da je naključno določen, shranjen v tabelo in zaščiten s privatnim ključem.

Slabost tega pristopa je, da izpostavimo nekaj informacije kodiranih kolon. Tako je možno izvesti analizo frekvenc istih podatkov. Poleg tega so poizvedbe kompleksnejše in smo posledično primorani uporabljati poglede, ki nam do neke mere poenostavijo poizvedbe.

4.1.2.7. Razširljivost podatkovne baze[12]

Z rastjo velikosti podatkovne baze in števila uporabnikov se ustrezno podaljšuje čas poizvedb in iskanja. SaaS aplikacije hkrati uporablja tudi po tisoč uporabnikov, zato so te aplikacije posebej občutljive na te spremembe. Zaradi tega je potrebno aplikacije ustrezno načrtovati, da pri večjem številu uporabnikov ne pride do prevelikih upadov pri delovanju.

Razširljivost podatkovne baze lahko zagotovimo na dva načina: podatkovno bazo premestimo na boljši strežnik z močnejšim procesorjem, z več delovnega pomnilnika in s hitrejšim dostopom do diska ali pa naredimo več particij podatkovne baze in različne particije poganjamo na več strežnikih. Uporabljajo se različne strategije, odvisno od tega, ali imamo skupno ali namensko podatkovno bazo. Glavna načina za razširitev podatkovne baze se imenujeta replikacija in particioniranje.

Replikacija

Replikacija pomeni kopiranje celotne ali samo dela podatkovne baze na drugo lokacijo, pri čemer je potrebno imeti kopije sinhronizirane z originalno bazo. Poznamo dve glavni vrsti

[11] Povzeto po [20].
[12] Povzeto po [9].

replikacije: »single master« in »multi master« replikacija. Za »single master« replikacija gre, ko dovolimo pisanje samo v originalno podatkovno bazo (angl. replication master). Ta pristop je uporaben za distribucijo podatkov. Primer za to bi bil, ko imamo več poslovnih enot iste prodajne verige, ki potrebujejo imeti dnevno posodobljene cene produktov, ki morajo biti vedno dosegljive in konsistenčne v vseh enotah. »Multi master« replikacija pa je, ko dovolimo pisanje v katerokoli kopijo baze. Sistemi za upravljanje podatkovnih baz imajo vgrajene mehanizme za sinhronizacijo sprememb med različnimi kopijami podatkov. Primer uporabe take replikacije bi bila aplikacija, do katere se dostopa iz več različnih mest, kjer mora imeti vsaka točka pravice tudi za pisanje, ampak zaradi ogromne količine podatkov potrebujemo bolj lokalno dostopne podatke.

Particioniranje

Zelo enostaven način za razširitev podatkovne baze se imenuje particioniranje, s čimer podatke razdelimo na več delov, kjer razdeljene podatke shranimo v drugo podatkovno bazo ali pa v druge tabele v isti podatkovni bazi. S tem, ko podatke razdelimo po več tabelah, se poizvedbe hitreje izvršujejo. Posledično se izboljša delovanje podatkovne baze in poenostavi vzdrževanje. Particioniranje lahko izvedemo tako, da premaknemo celotne tabele ali pa tabele razdelimo vertikalno ali horizontalno. Horizontalno particioniranje pomeni, da podatkovno bazo razdelimo na dva ali več manjših delov. Vsi deli imajo enako shemo in strukturo, ampak vsebujejo manjše število vrstic v tabelah. Pred particioniranjem je potrebno podatke analizirati, da med tabelami obstaja čim manj relacij. V nasprotnem primeru bodo naše poizvedbe vsebovale nepotrebne operatorje `UNION`, kar lahko vpliva na hitrost izvrševanja poizvedb. Vertikalno particioniranje pomeni, da v različne baze shranimo različne kolone tabel. Na ta način imajo vse baze enako število vrstic.

V večini primerov se velikost baze ves čas povečuje. Zato je smiselno, da imamo implementirano dinamično particioniranje, da lahko ves čas zagotavljamo ustrezen nivo delovanja.

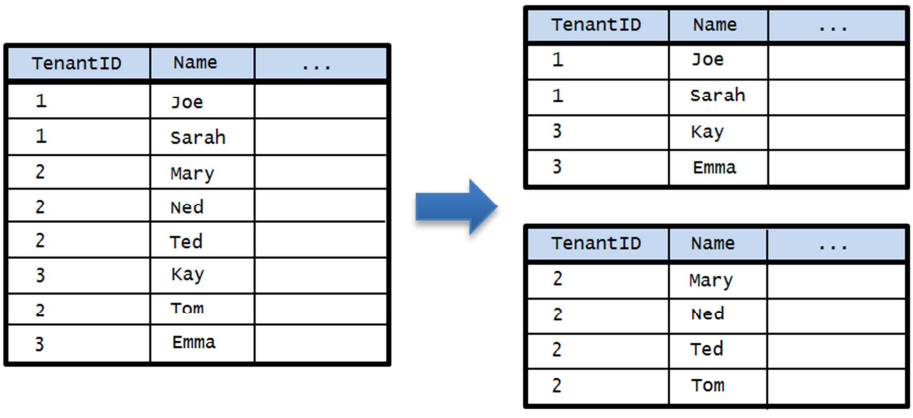
Najbolj pogost in priljubljen način particioniranja je sistem za upravljanje distribuirane podatkovne baze (angl. distributed database management system – DDBMS). Distribuirana podatkovna baza je baza, ki jo sestavljata dve ali več datotek – particije, ki se nahajajo na geografsko ločenih lokacij in so dosegljive po omrežju – vozlišča. Uporabniki dostopajo do podatkov, ki se nahajajo na različnih particijah, ne da bi se tega zavedali. Sistem DDBMS vsebuje tudi mehanizme za upravljanje, nadzor in sinhronizacijo particij. Da slučajno ne pride do izgub podatkov, so particije replicirane, da imamo redundantne kopije podatkov, saj je v primeru težav (npr. prekinjena oskrba električne energije) SaaS aplikacija nedosegljiva.

Horizontalno particioniranje na osnovi podatkov odjemalcev

Podatkovno bazo, ki si jo deli več odjemalcev, je potrebno zmanjšati, ko nič več ne dosega osnovnih meril učinkovitosti delovanja, ko hkrati do baze dostopa preveliko število uporabnikov, ko velikost baze povzroči prepočasno izvedbo poizvedb ali ko vzdrževalna opravila začnejo vplivati na dosegljivost podatkov.

Najenostavnejši način za razširitev podatkovne baze je horizontalno particioniranje po podatkih posameznih odjemalcev. Skupne podatkovne baze so primerne za horizontalno particioniranje, ker ima vsak odjemnik svoj nabor podatkov, tako da je enostavno dobiti individualne podatke odjemnika in jih premakniti. Pri tem je potrebno paziti, podatke katerih odjemalcev pustimo skupaj. Katerim odjemalcem dodelimo skupno particijo, je odvisno od več faktorjev: zahtev odjemalcev glede aplikacije, števila uporabnikov in količine podatkov individualnih odjemalcev. Glede na to imamo tudi različne metode particioniranja, ki so v veliki meri odvisne od metrik, glede na katere izvajamo particioniranje. Dane metrike je možno dobiti, če imamo v sistem vgrajen nadzor. Na ta način lahko dobimo natančne podatke o uporabi aplikacije za vsakega odjemalca posebej. Poleg tega je zelo verjetno, da bo s časom potrebno ponovno particioniranje, ker je pričakovati, da se bo odjemalec razvijal in spreminjal način dela. Poleg tega je potrebno izbrati tako strategijo particioniranja, ki ne bo pretirano vplivala na delovanje sistema. Slika 8 prikazuje enostaven primer particioniranja, kjer so particije določene glede na količino podatkov posameznih odjemalcev.

Ta pristop je primeren v primeru, ko imamo za različne odjemalce skupno bazo in skupno shemo, saj zagotavlja način, ki particionira podatkovno bazo, ne da pri tem obremenjuje delovanje aplikacije. Delovanju aplikacije zagotovo škodi, če podatke istega odjemnika porazdelimo po več strežnikih.



Slika 8: Primer horizontalnega particioniranja na osnovi podatkov odjemalcev

Razširljivost podatkovne baze z ogromno količino podatkov

Če imamo odjemalce, ki hranijo in uporabljajo velike količine podatkov, se lahko zgodi, da velikost podatkovne baze naraste do meje, ko je smiselno za podatkovno bazo takega

odjemalca nameniti ločeno bazo in ločen strežnik. Izzivi so v tem primeru podobni, kot če imamo tradicionalno aplikacijo. Za ogromno podatkovno bazo je najlažje zagotoviti razširljivost z nakupom večjega in zmogljivejšega strežnika. Sčasoma ta pristop postane nesmotrn. Tako je potrebno podatkovno bazo razdeliti na enega ali več dodatnih strežnikov. Preden pa bazo particioniramo, je potrebno analizirati podatke, da lahko določimo najboljši način particioniranja. Tu je nekaj splošnih napotkov, ki se jih je priporočljivo držati:

- Uporabi replikacijo za kreiranje kopij podatkov, ki se redko spreminjajo. Nekateri podatki se redko ali celo nikoli ne spreminjajo, ko so enkrat vneseni, in nekateri podatki se spreminjajo v samo določenem časovnem obdobju, nato pa se arhivirajo. Takšni podatki so idealni za replikacijo.
- Podatke hrani blizu podatkov, na katere se sklicujejo. Upoštevati je potrebno relacije med podatki in se odločiti, kako podatke razdeliti in nad katerimi podatki izvesti replikacijo za kreiranje kopij podatkov, ki so namenjeni samo branju. Pri tem želimo imeti čim manj komunikacije med particijami podatkovne baze.
- Identificirati je potrebno podatke, ki jih ne smemo particionirati. Neustrezni podatki za particioniranje so na primer zaloge materiala v skladiščih.
- Uporabi »single master« replikacijo, kjer je to mogoče.

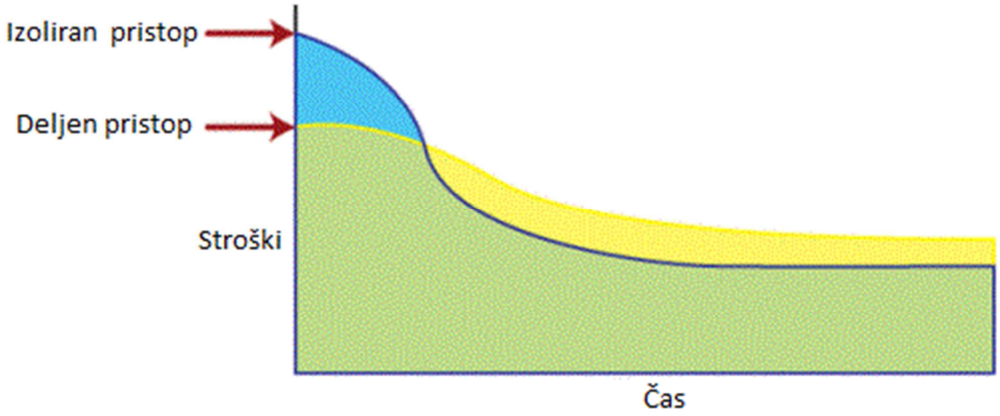
4.1.3. Izbira pristopa implementacije večnajemniške arhitekture[13]

Vsak od opisanih pristopov ima svoje prednosti in slabosti. Potrebno je pretehtati poslovne in tehnične vidike, ki lahko vplivajo na našo izbiro modela. V nadaljevanju so opisani nekateri od teh vidikov.

4.1.3.1. Ekonomski vidiki

Slika 9 prikazuje stroške dveh hipotetičnih aplikacij, kjer modra krivulja predstavlja aplikacijo z deljenimi viri in rumena predstavlja aplikacijo, kjer viri niso deljeni med odjemalci. Aplikacije, ki so optimizirane za deljenje virov med odjemalci, običajno zahtevajo večji napor pri razvoju. Do tega pride zaradi večje kompleksnosti pri razvoju arhitekture, ki podpira deljenje virov. To pripelje do večjih začetnih stroškov. Hkrati to pomeni manjše stroške tekočega poslovanja, ker takšna arhitektura omogoča, da na istem strežniku poganjanja isto aplikacijo več odjemalcev kot sicer. Na izbiro pristopa vpliva tudi količina začetnega kapitala in čas, v katerem moramo dokončati aplikacijo, da jo ponudimo na trg. To lahko pomeni, da si ne moremo privoščiti dolgotrajnega in obsežnega razvoja.

[13] Povzeto po [9].



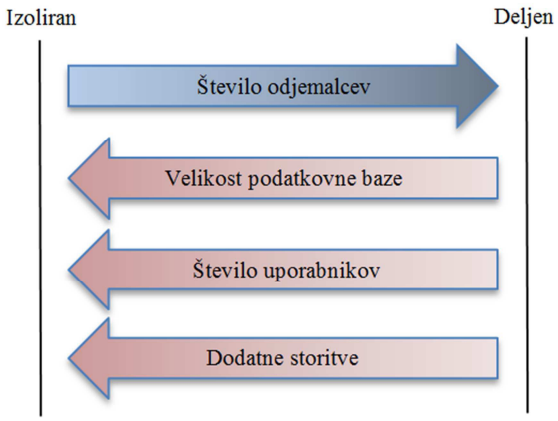
Slika 9: Stroški v odvisnosti od časa za dve hipotetične SaaS aplikacije [9]

4.1.3.2. Varnostni vidiki

Bodoči kupci imajo stroge zahteve glede varnosti, zato je potrebno s pogodbo zagotoviti močna varnostna jamstva. Podatki so močno zavarovani tudi, če niso fizično ločeni. Da zagotovimo varnost podatkov, ki se fizično nahajajo na isti lokaciji, je potrebno uporabiti bolj prefinjene načine, kot sta na primer uporaba kodiranja podatkov in pogledov za filtriranje podatkov odjemalcev v kombinaciji s kontrolo dostopa do objektov. Poleg tega potrebujemo zagotoviti tudi varen fizičen dostop do strežnikov.

4.1.3.3. Vidiki glede odjemalcev

Število, vrsta in zahteve odjemalcev prav tako vplivajo na izbiro pristopa (slika 10). Oceniti moramo, koliko odjemalcev pričakujemo. Več kot jih imamo, bolj je smiselno izbrati deljen pristop. Poleg tega moramo predvideti tudi okvirno velikost podatkovne baze. Če pričakujemo, da bodo odjemalci shranjevali ogromne količine podatkov, potem je bolj smiseln princip z ločenimi bazami.



Slika 10: Faktorji, ki vplivajo na izbiro pristopa [9]

Število uporabnikov in dodatne storitve, kot so na primer varnostne kopije in restavriranje teh kopij, prav tako vplivajo na izbor pristopa. Več kot imamo uporabnikov in dodatnih storitev, lažje je implementirati aplikacijo, če izberemo izolirani pristop.

4.1.3.4. Pravni vidiki

Pri razvoju programske opreme je potrebno upoštevati tudi pravne omejitve, kot so na primer predpisi o varnosti in arhiviranju podatkov. Ti predpisi so lahko v splošnem v vsaki državi drugačni.[14]

4.1.3.5. Usposobljenost razvojne ekipe

Ali imajo programski arhitekti, programerji in analitiki primerna znanja za razvoj arhitekture po deljenem pristopu? Če nimamo, moramo najeti nove kadre ali izobraziti obstoječi kader.

4.2. Identiteta in upravljanje dostopa

Spletna identiteta je množica atributov, ki opisujejo posameznika v digitalnem prostoru. Upravljanje z identitetami (angl. identity management) so procesi in tehnologije, ki upravljajo življenjski cikel identitete in odnos do poslovnih aplikacij in storitev. Tipične funkcionalnosti upravljanja z identitetami so: avtentikacija (angl. authentication), avtorizacija (angl. authorization), enotna prijava (angl. single sign-on), upravljanje uporabnikov (angl. user management), določanje pravic (angl. provisioning/deprovisioning) in revizija (angl. audit) [32].

Naš cilj je integracija tradicionalnih in SaaS aplikacij. Podjetja imajo običajno implementiran učinkovit model za upravljanje z identitetami znotraj podjetja. Ko je potrebno obstoječe aplikacije integrirati s tradicionalnimi aplikacijami, se ta model običajno poruši. Rešitev je federacijska identiteta (angl. federated identity), ki zagotavlja enotno prijavo tudi za integrirane aplikacije iz različnih okolij.

Za uspešno implementacijo federacijskega upravljanja identitete je potrebno izbrati pravo tehnologijo, ki ne bo vzela preveč časa za razvoj. Tipične možnosti so ali lastniške rešitve SSO (npr. spletni agenti) ali rešitve SSO, ki temeljijo na odprtih standardih ali standardih z javno objavljenimi specifikacijami (npr. federacijska identiteta). Lastniške rešitve SSO so že dalj časa v uporabi, vendar se je za njih izkazalo, da se za implementacijo porabi veliko časa, stroškov, rešitve so kompleksne in pojavi se veliko varnostnih vprašanj [2].

[14] V Sloveniji to področje ureja Zakon o varstvu osebnih podatkov (ZVOP-1, Ur. l. RS, št. 94/07-UPB1).

Za SSO se danes v veliki meri uporablja standard SAML (angl. Security Assertion Markup Language). Ključna prednost tega standarda je, da je možno isto rešitev uporabiti za katerokoli podjetje, ki isto uporablja SAML. Dejstvo, da vodilne SaaS aplikacije uporabljajo SAML, pomeni, da se skupnost zavzema za neke splošne modele glede upravljanja z identitetami in varnostjo. SAML je prevladujoč standard glede federacijskega upravljanja z identitetami. Standard definira množico XML formatov, ki predstavljajo identiteto, atribute ter protokole za zahteve in odzive na dostop informacij o dostopu.

Glavni princip standarda SAML je trditev (angl. assertion ali claim). To je izjava, ki jo zaupna stranka naredi o nekom drugem. Na primer strežnik, ki mu zaupamo, poda trditve o uporabnikih in njihovih pravicah. Posamezna aplikacija ne potrebuje dostopa do zbirke podatkov o uporabnikih ali da zaupa uporabniku – potrebuje samo zaupnost izvoru trditev. Trditve so omogočene za komunikacijo med osebo in napravo ter med napravami. Tako se lahko isti standard uporablja za dostop preko spletnega portala in za transakcije, ki potekajo v ozadju.

4.2.1. Rešitve na osnovi trditev[15]

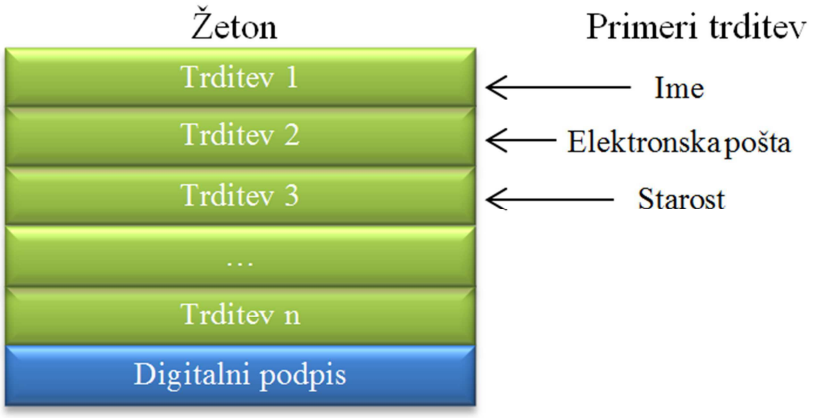
Rešitev na osnovi trditev je način, kako aplikacija pridobi informacije o identiteti uporabnikov znotraj podjetja, del drugega podjetja ali internetnih uporabnikov. Zagotavlja konsistenten pristop za avtentikacijo uporabnikov za aplikacije, ki se nahajajo tako v oblaku kot znotraj podjetja.

Kot je že omenjeno, je spletna identiteta množica podatkov o nekom ali nečem. Identiteto imajo lahko različne entitete: računalniki, aplikacije, najpogosteje pa smo to ljudje. Ne glede na to, ali je entiteta človek ali ne, smo za vsako entiteto z identiteto uporabljali naziv uporabnik.

Ko se digitalna identiteta prenaša po omrežju, je to samo skupek bitov. Danes je običajno, da se množici bitov, ki vsebujejo podatke o identiteti, reče varnostni žeton ali samo žeton (angl. token). Žeton vsebuje eno ali več trditev in digitalni podpis. Vsaka trditev nosi neko informacijo o uporabniku. Na sliki 11 je prikazan primer žetona.

Trditev lahko predstavlja karkoli o uporabniku. To so podatki o uporabniškem imenu, identifikatorju uporabniške skupine, ipd. Katere trditve vsebuje žeton, je odvisno od tega, katere trditve potrebuje aplikacija. Trditve prav tako določajo pravice uporabnika, kot je na primer dostop do datotek ali omejitev odobritve zneska nakupa. Danes je običajno, da so žetoni definirani s standardom SAML.

[15] Povzeto po [3].



Slika 11: Primer žetona [3]

Izdajatelj žetona digitalno podpiše vsak žeton, da je možno preveriti izvor žetona in se zavarovati pred neavtoriziranimi spremembami. Digitalni podpis izdajatelja se pošlje kot del žetona.

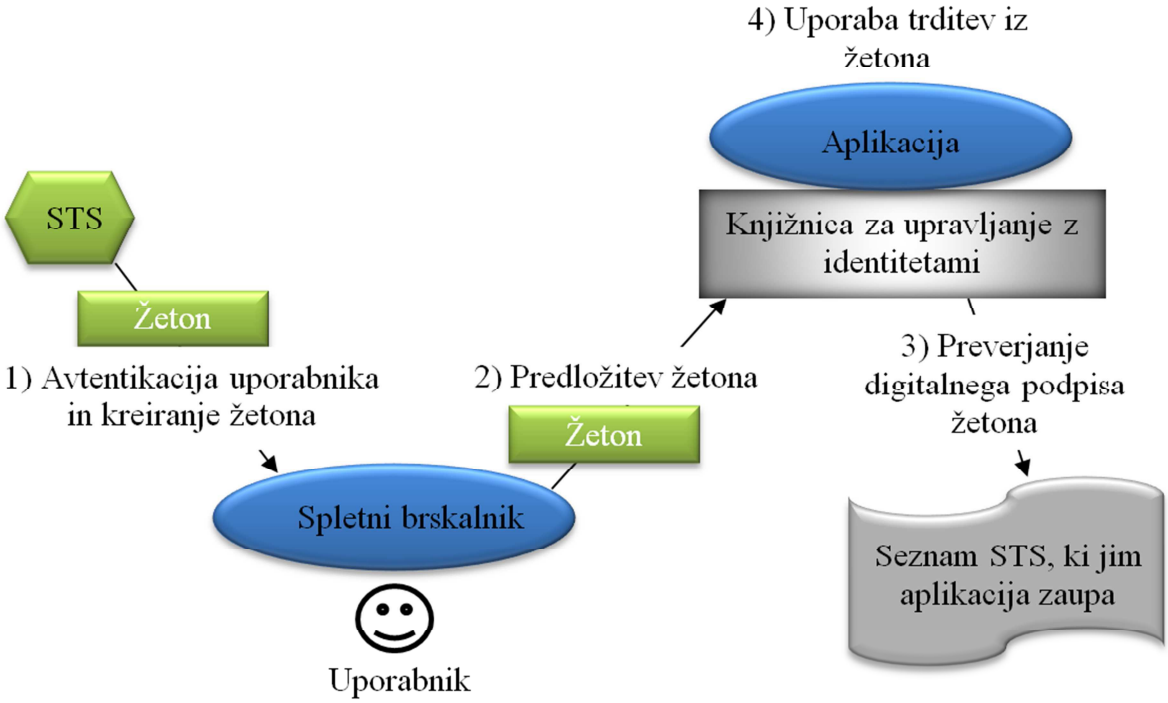
Žetone kreirajo storitve varnostnih žetonov (angl. security token service). Običajno spletni brskalnik ali kak drug klient pošlje zahtevo storitvi STS za žeton za trenutnega uporabnika. Za te zahteve se uporabljajo različni pristopi (npr. protokol Kerberos, LDAP ali preverjanje uporabniškega imena in gesla). Na ta način storitev STS preveri, da je uporabnik tisti, kdor trdi, da je. Zahteva običajno vsebuje niz URI, ki nosi informacijo o tem, do katere aplikacije želi uporabnik dostop. Storitev STS vsebuje informacije o uporabnikih in aplikacijah ter na podlagi teh informacij tvori žetone.

Pomembna razlika je, da sta avtentikacija in avtorizacija uporabnikov ločeni in da aplikacija nič več ne potrebuje vsebovati avtentikacije, ker to namesto nje naredi storitev STS. Potrebno je nastaviti aplikacijo tako, da zaupa dani storitvi. Poleg tega je potrebno storitev STS nastaviti, da izda prave trditve za uporabnika in aplikacijo, ki jih aplikacija lahko (posredno) uporabi pri avtorizaciji. Tako mora aplikacija vsebovati preslikovanje zunanjih avtentikacijskih v notranje avtorizacijske trditve.

Prednost tega pristopa je ta, da so vse trditve zbrane na enem mestu namesto, da so porazdeljene po različnih sistemih. Tako lahko iste žetone uporabimo za avtentikacijo na različnih spletnih straneh ali aplikacijah. Poleg tega lahko storitev STS zaupamo zunanjemu oskrbovalcu identitet (angl. identity provider), ki mu pravimo tudi izdajatelj žetonov, ali oskrbovalcem identitet, saj jih je v splošnem primeru lahko več.

4.2.2. Oskrbovalec identitete[16]

Slika 12 prikazuje enostaven scenarij uporabe žetonov. Ko uporabnik želi dostopati do aplikacije, ta generira zahtevo za avtentikacijo. Zahteva je zakodirana in vključena v nizu URI, ki se posreduje spletnemu brskalniku in naprej storitvi STS. STS zahtevo dekodira in izvrši avtentikacijo (npr. s preverjanjem prijavnih poverilnic ali sejnih piškotkov). V primeru uspešne avtentikacije STS vrne ustrezne trditve združene v žeton skupaj z digitalnim podpisom (na sliki je to prikazano kot korak 1). Ko klient ali spletni brskalnik prejme žeton, ga posreduje aplikaciji (korak 2). Aplikacija mora biti nastavljena, da zaupa dani storitvi STS. Za procesiranje žetona aplikacija uporabi knjižnico za upravljanje z identitetami (angl. identity library), ki omogoča delo z žetoni in protokoli, ki žetone prenašajo. Aplikacija z uporabo knjižnice preveri digitalni podpis žetona in pogleda na seznam storitev STS, ki jim zaupa (korak 3). Če se storitev STS nahaja na seznamu, potem aplikacija sprejme žeton, ustrezno uporabi trditve, ki so v žetonu (korak 4), uporabnik je avtenticiran in brskalnik naloži želeno aplikacijo.



Slika 12: Uporaba žetonov s strani spletnega brskalnika [3]

4.2.3. Prijava z identitetami različnih ponudnikov identitet

V nekaterih situacijah bo aplikacija sprejemala žetone od več različnih oskrbovalcev identitet, kar pomeni, da zaupa več različnim storitvam STS [3]. To velja še posebej za javno dostopne

[16] Povzeto po [3, 31].

aplikacije. Na primer lahko imamo aplikacijo, ki je dostopna na internetu in se je možno vanjo prijaviti z identiteto, ki jo je izdal Facebook, Google ali Microsoft. Glede na to, da ima ogromno uporabnikov odprte račune pri teh storitvah, je smiselno, da jih uporabimo, da uporabnikom ni potrebno odpirati ločenega računa za našo aplikacijo.

Za avtentikacijo mora spletni brskalnik izbrati enega izmed več oskrbovalcev identitet. Da lahko to opravi, mora uporabnik priskrbeti dodatne informacije, za kar obstaja več načinov:

- za avtentikacijo uporabnik navede elektronski naslov (npr. *@hotmail.com – Microsoft account),
- aplikacija vsebuje ločeno prijavno okno za vsakega oskrbovalca identitete (ta način je danes popularen pri javno dostopnih aplikacijah, kjer se lahko uporabniki prijavi s Facebook računom) ali
- uporabnik pri prijavi izbere tudi oskrbovalca identitete.

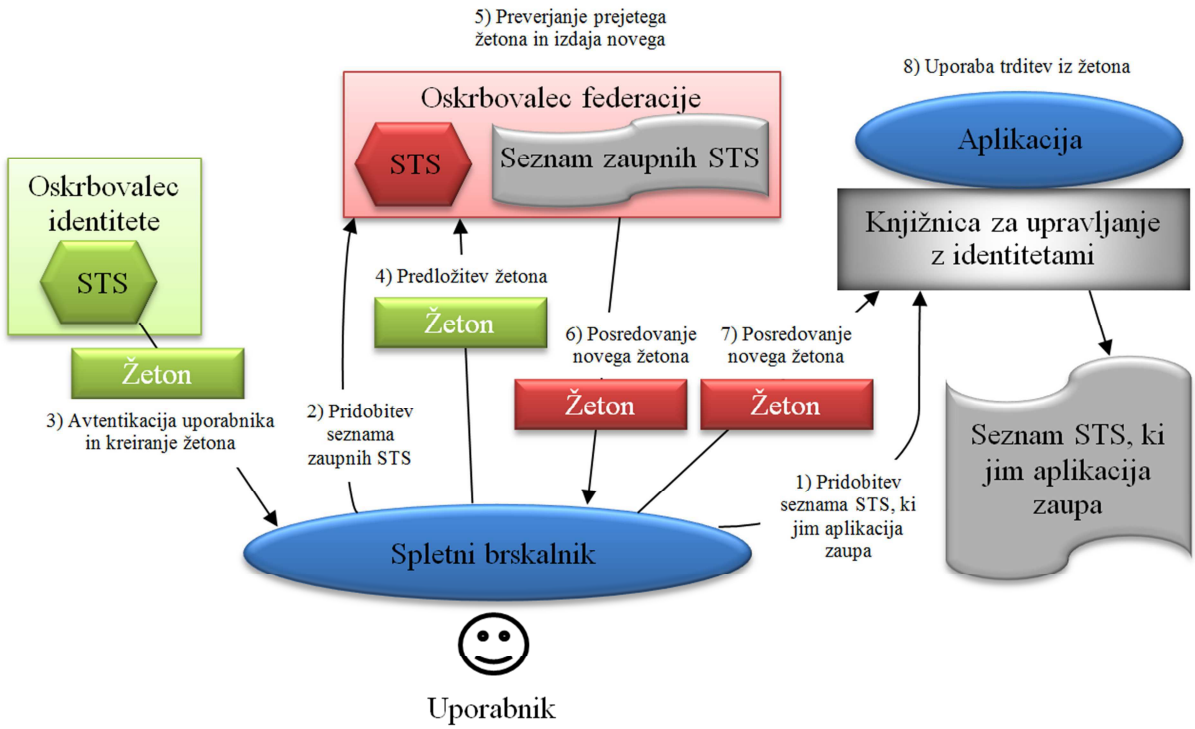
Po drugi strani obstaja veliko primerov, ko ni smiselno sprejemati žetone od različnih oskrbovalcev identitet. Dovoliti uporabnikom, da se prijavijo v poslovne aplikacije, ki so vitalnega pomena za poslovanje, s Facebook računom, ni dobra ideja [3]. Je pa ta princip uporaben za poslovne aplikacije, ki morajo biti dostopne zaposlenim, partnerjem in uporabnikom, pri čemer je aplikacija dostopna preko interneta ter želimo imeti čim manj težav s prijavami uporabnikov.

4.2.4. Oskrbovalec federacije

Včasih želimo omogočiti dostop do aplikacije tudi poslovnim partnerjem – uporabnikom, ki ne pripadajo naši organizaciji oziroma podjetju. Ti novi uporabniki že imajo svoje digitalne identitete, ki jih je izdal v splošnem drugi oskrbovalec identitet kot ta, ki mu zaupa aplikacija. Načeloma bi morali novim uporabnikom kreirati nove identitete, ampak temu bi se radi izognili. Rešitev je sistem, ki omogoča, da neka storitev STS zaupa drugi storitvi STS. Ko hoče novi uporabnik dostopati do aplikacije, se zahteva za avtentikacijo preusmeri iz storitve STS, ki je izdala naše identitete, na storitev, ki je izdala identitete novih uporabnikov. Ta pristop se imenuje federacijska identiteta (angl. identity federation), kjer aplikacija neposredno zaupa samo storitvi STS od oskrbovalca federacije (angl. federation provider).

Slika 13 prikazuje splošen primer in je v praksi bolj uporaben, ker pokrije več situacij. Spletni brskalnik najprej od aplikacije pridobi seznam storitev STS, ki jim aplikacija zaupa (korak 1). V tem primeru aplikacija zaupa samo eni storitvi STS, ki pripada nekemu oskrbovalcu federacije. Brskalnik se poveže z njegovo storitvijo in od njega dobi seznam storitev STS, ki jim oskrbovalec federacije zaupa (korak 2). Če je na seznamu storitev STS, ki je izdala identiteto uporabnika, jo kontaktira. Ta storitev izvrši avtentikacijo uporabnika in vrne žeton

brskalniku (korak 3). Tega žetona brskalnik ne more posredovati aplikaciji, ker aplikacija ne zaupa tej storitvi, ki je izdala ta žeton. Namesto tega brskalnik posreduje žeton oskrbovalcu federacije (korak 4). Oskrbovalec preveri žeton, če ga je izdala storitev, ki ji zaupa. Nato izda nov žeton (korak 5), ki ga vrne brskalniku (korak 6). Brskalnik lahko nato novi žeton posreduje aplikaciji, saj ga je izdala storitev, ki ji aplikacija zaupa (korak 7). Pri tem postopku uporabnik ne vidi zahtev in prenosa žetonov. Na ta način dostopa do aplikacije brez ločene prijave, s čimer je zagotovljena tudi enotna prijava.[17]



Slika 13: Federacija [3]

4.2.5. Pretvorba trditev[18]

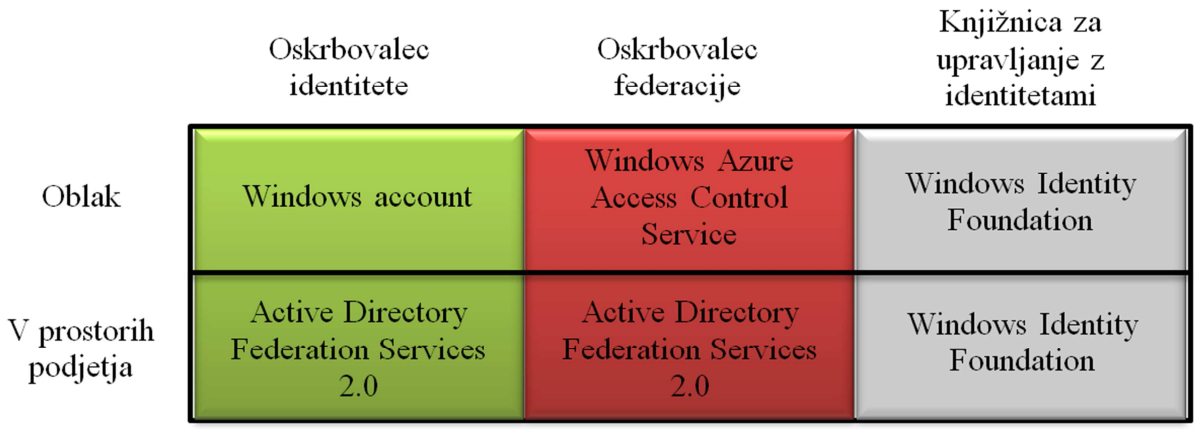
Oskrbovalec federacije iz prejetega žetona generira nov žeton. Tu se pojavi vprašanje, katere trditve bo nov žeton vseboval. Storitev STS bo vstavila trditve glede na pravila pretvorbe trditev. Ta pravila določajo, katere trditve se odstranijo, katere trditve se enostavno prepišejo in za katere trditve se opravi pretvorba (angl. claims transformation), preden se jih doda v žeton. Na ta način storitev STS odda žeton, ki vsebuje drugačne trditve, kot jih je prejela.

Pretvorba omogoča več funkcionalnosti. Tako je možno med drugim pretvoriti nize v numerično kodo (npr. ime skupine uporabnikov »Administrator« pretvoriti v numerično kodo, ki jo potrebuje aplikacija). Možno je izpustiti trditve, ki niso pomembne za aplikacijo. Možno je dodati nove trditve, kot je na primer, katera storitev STS je primarno izdala žeton.

[17] Povzeto po [3].
[18] Povzeto po [3].

4.2.6. Implementacija upravljanja identitet[19]

Za implementacijo upravljanja identitet imamo na izbiro več produktov. Tako lahko kombiniramo produkte različnih ponudnikov pod pogojem, da ponudniki upoštevajo standarde. V nadaljevanju so prikazane (slika 14) in opisane tehnologije, ki jih Microsoft ponuja za tradicionalne in SaaS aplikacije.



Slika 14: Primeri Microsoftovih tehnologij za kontrolo dostopa [3]

Microsoft account[20]

Microsoft account je Mirosoftova javna storitev STS. Najbolj znana aplikacija, ki uporablja žetone od Microsoft account, je Outlook.com. Uporaba storitve ni vezana na Microsoftove izdelke, ampak je možno žetone uporabiti tudi v lastnih aplikacijah. Žetoni, ki se kreirajo, so zelo enostavni. Vsebujejo samo globalni enolični identifikator uporabnika, ki ga lahko aplikacija uporabi za prepoznavo uporabnika. Druge lastnosti uporabnika (npr. ime in naslov) mora aplikacija voditi v lastni podatkovni bazi.

Windows Azure Access Control Service[21]

Access Control Service (ACS) je del platforme Windows Azure, teče v oblaku in ga lahko uporabimo za oskrbovalca identitete. Poleg podpore za upravljanje z identitetami nudi še podporo za pretvorbo trditev, saj vsebuje orodje za definiranje pravil za te pretvorbe.

Za zahteve žetonov podpira različne standarde: WS-Federation, WS-Trust, OpenID 2.0 in OAuth 2.0. ACS sprejema in izdaja žetone v različnih formatih kot so SAML 1.1, SAML 2.0 in Simple Web Token.

Active Directory Federation Services 2.0

Active Directory Federation Services 2.0 je komponenta, ki zagotavlja enotno prijavo in je nameščena na operacijskem sistemu Windows Server. ADFS 2.0 lahko nastopa v vlogi

[19] Povzeto po [3].
[20] Microsoft account je novo ime za Microsoft Live ID [33].
[21] Do pomladi 2012 se je ACS imenoval Windows Azure AppFabric Access Control.

oskrbovalca identitete ali oskrbovalca federacije ali celo v obeh vlogah hkrati. Uporablja se lahko za aplikacije, ki se izvajajo striktno znotraj podjetja, za aplikacije, do katerih se dostopa preko interneta, ali za oboje. ADFS 2.0 podpira različne standarde: SAML 2.0, WS-Federation[22] in WS-Trust[23].

Windows Identity Foundation

Windows Identity Foundation je del .NET Framework ter vsebuje množico razredov in funkcionalnosti za implementacijo upravljanja z identitetami, kot so sprejemanje žetonov, preverjanje digitalnega podpisa in dostop do trditev. Podpira žetone, ki so bili kreirani po formatu SAML 1.1 ali SAML 2.0. Poleg tega podpira tudi protokola WS-Federation in WS-Trust. Z razširitvami te knjižnice je možno uporabljati tudi druge protokole. Microsoft ima med drugim implementirano razširitev, ki omogoča podporo za standard OAuth 2.0.

WIF vsebuje tudi funkcionalnost za kreiranje lastnih storitev STS, če ne želimo uporabiti AD FS 2.0 ali ACS.

4.3. Integracija podatkov in aplikacij

Danes se čedalje več informacij in poslovnih procesov nahaja izven požarnega zidu podjetja na SaaS platformah, zato je potrebno zagotoviti integracijo sistemov, da odpravimo podvojeno vnašanje in vzdrževanje podatkov. Podvojenost podatkov prinaša dodatne stroške, ki se jim želimo izogniti. V ta namen potrebujemo arhitekturo, s katero lahko pripeljemo podatke v logično infrastrukturo, da lahko sistemi sodelujejo med seboj ne glede na to, ali so nameščeni zunaj ali znotraj podjetja [10].

V večini primerov implementacija SaaS aplikacije vsebuje prenos podatkov iz ene ali več aplikacij ali podatkovne shrambe v nov sistem. Prenos podatkov lahko vključuje [10]:

- prenos in transformacijo podatkov iz obstoječe tradicionalne aplikacije,
- nastavitev SaaS aplikacije, da je odvisna od podatkov iz tradicionalne aplikacije, in
- nastavitev tradicionalne aplikacije, da je en del funkcionalnosti te aplikacije odvisen od podatkov, ki nastanejo v SaaS aplikaciji.

V mnogih primerih integracija s SaaS aplikacijo pomeni kreiranje podatkovnih odvisnosti, ki zahteva, da so podatki preneseni in sinhronizirani med različnimi aplikacijami, da se olajša procesiranje. Za ta namen se uporablja posrednik za integracijo.

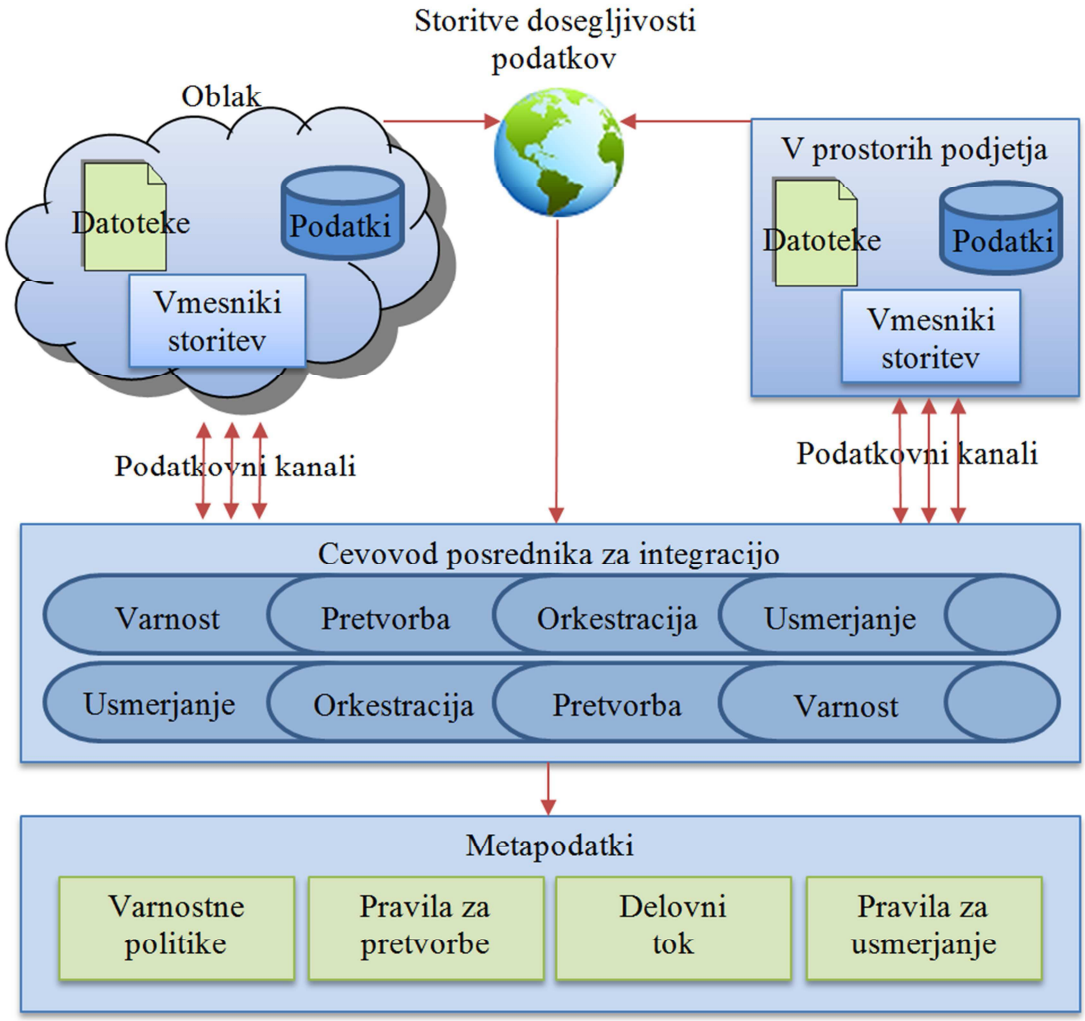
---

[22] WS-Federation je arhitekturni model za upravljanje s federacijskimi identitetami. Omogoča uporabo modela storitev varnostnih žetonov in protokola za reševanje zahtev glede identitet tako za spletne aplikacije kot za spletne storitve v različnih razmerjih zaupanja.

[23] WS-Trust je standard, namenjen za kreiranje, obnovitev in preverjanje žetonov. Omogoča vzpostavitev razmerja zaupanja med udeleženci za varen prenos sporočil.

4.3.1. Posrednik za integracijo[24]

Posrednik za integracijo je sistem, ki omogoča pretok poslovnih informacij med različnimi sistemi po različnih strojnih in programskih platformah. Z metapodatki so definirana pravila, ki usmerjajo in pretvarjajo informacije. Podatki lahko izvirajo iz različnih izvorov, uporabljajo lahko različne protokole in neskladne formate. Naloga posrednika za integracijo je vzeti podatke iz različnih izvorov, ugotoviti kje in kako se podatki procesirajo ter posredovati vsak del podatkov do ponora v takem formatu, da ga lahko ponorni sistem uporabi. Na sliki 15 je prikazana shema posrednika za integracijo.



Slika 15: Posrednik za integracijo [10]

Moduli, ki sestavljajo cevovod, določajo, kako so podatki procesirani, posredovani in integrirani s podatki na ponoru. Storitve z metapodatki nudijo pravila, da lahko posamezen modul opravi svojo nalogo. Posamezna operacija integracije običajno vsebuje naslednje korake:

[24] Povzeto po [10].

- varnost: modul za varnost nad vhodnimi podatki izvrši določene operacije, kot so preverjanje podatkovnega izvora ali digitalnega podpisa, dekodiranje podatkov in preverjanje varnostnih tveganj (npr. virusi). Varnostne operacije lahko sodelujejo z obstoječimi varnostnimi politikami za upravljanje dostopa.
- validacija: modul za validacijo primerja podatke z ustreznimi shemami in zavrže neskladne podatke ali pa jih posreduje modulu za pretvorbo v ustrezen format.
- sinhronizacija: modul za sinhronizacijo uporabi delovni tok in pravila, s katerimi so določene spremembe nad podatki ter kako so podatki posredovani do cilja in v kakšnem vrstnem redu. V primeru, da posamezno zaporedje v delovnem toku ni bilo uspešno zaključeno, mora ta modul zagotoviti konsistentnost podatkov. Pri tem se uporabijo transakcije ali nadomestna logika za razrešitev takih problemov.
- usmerjanje: modul za usmerjanje definira cilj za vsak kos podatkov. Vsebuje lahko enostavno posredovanje podatkov iz izvora do cilja ali obsežnejšo logiko, kot je npr. določanje cilja glede na vsebino.

Storitve dosegljivosti podatkov (angl. data availability service) zagotavljajo način, s katerim posrednik za integracijo zazna, kdaj so na voljo novi podatki.

4.3.1.1. Vzorci dosegljivosti podatkov

Sinhronizacija podatkov pomeni prenos novih in spremenjenih podatkov od izvora do cilja (angl. data sink), da zagotovimo skladnost podatkov v različnih sistemih. Prenosi se vršijo glede na vnaprej predpisane intervale ali ob sprožitvi dogodkov [10]. Obstajajo trije osnovni vzorci, ki se uporabljajo za proženje sinhronizacije podatkov med izvorom in ciljem:

- periodično poizvedovanje (angl. polling): periodično poizvedovanje pomeni, da cilj aktivno preverjanja status podatkov na izvornem sistemu. Poizvedovanje se tipično vrši v enakih časovnih intervalih. Slabost tega vzorca je, da lahko v primeru velikega števila sistemov, ki jih mora preveriti, prekoračimo čas, ki ga imamo na voljo za poizvedovanje [39].
- potiskanje (angl. push): potiskanje je ravno nasprotje periodičnega poizvedovanja. V tem primeru zahtevo za prenos podatkov poda podatkovni izvor, ko pride do sprememb podatkov ali v enakih časovnih intervalih [40].
- objavi in naroči (angl. publish and subscribe): ta pristop je kombinacija obeh prejšnjih vzorcev. Podatki se ne pošljejo nemudoma, ko pride do sprememb. Namesto tega izvorni sistem pošlje sporočilo ciljnemu, da so podatki na voljo. Pred tem mora biti ciljni sistem naročen na obvestila o spremembah. V mnogih »objavi in naroči«

sistemih se uporablja posrednik, ki upravlja objavljena sporočila in beleži naročnike [41].

Za različne podatke so primerni različni pristopi. Lahko se odločimo, da v aplikaciji uporabimo kombinacijo pristopov. Izbira pristopa oziroma pristopov je odvisna od števila ciljev in obveznosti odražanja sprememb v realnem času. V določenih primerih se izbere pristop, s katerim izberemo srednjo pot zaradi nasprotujočih interesov. Na primer potiskanje je najprimernejše, ko je potrebno imeti posodobljene podatke. Po drugi strani pa to ni primeren pristop, če obstaja večje število ciljev, ker je v tem primeru pošiljanje podatkov zavzame veliko količino omrežnih in računskih virov ter posledično tudi negativno vpliva na delovanje aplikacije. Ne glede na to, kateri pristop izberemo, je potrebno določiti pravila za frekvenco glasovanja, format, itd [10].

4.3.1.2. Vzorci prenosa podatkov[25]

Podatki se lahko prenašajo na sinhroni ali asinhroni način. Sinhroni prenos se vrši ob določenih časovnih intervalih, pri čemer se udeleženec poveže na izvor in od njega zahteva podatke. Udeleženec pri tem pričakuje, da bo podatke prejel nemudoma.

Pri asinhronem prenosu se pošiljanje in prejem podatkov zgodita ob različnih časih. Asinhroni prenosi običajno temeljijo na sporočilih. Ko izvor pošlje podatke, ne pričakuje takojšnjega odziva. Ko prejemnik obdela zahtevo, pošlje izvoru povratno sporočilo, da je bila zahteva obdelana. Poleg tega je možno asinhrono komunikacijo realizirati tudi z »objavi in naroči« pristopom.

4.3.1.3. Vzorci pretvorbe podatkov[26]

Pretvorba podatkov pomeni pretvoriti neko množico podatkov iz formata izvornega podatkovnega sistema v format ponornega podatkovnega sistema. Pretvorba sestoji iz več korakov. Najprej je potrebno vhodne podatke preveriti z ustreznimi formati in shemami. Nato je potrebno izvršiti podatkovno preslikavo, ki preslika posamezne podatkovne elemente iz izvornega sistema v ponorni sistem in pri tem zajame kakršnekoli spremembe. Nato je potrebno izvršiti še samo pretvorbo, ki pretvori podatke v ciljni format. Primer pretvorbe je pretvorba iz formata EDIFACT v XML.

[25] Povzeto po [10].
[26] Povzeto po [10].

4.4. Arhitektura kompozitnih aplikacij

Kompozitna aplikacija je aplikacija, kjer so poslovne funkcije in informacije učinkovito integrirane za končnega uporabnika. Poslovne informacije so realizirane v obliki šibko sklopljenih kompozitov, ki jih združujemo v celote, ki predstavljajo realizacijo neke poslovne funkcije. Dobro zasnovana kompozitna aplikacija ima veliko prednosti, kot so na primer zmanjšano podvojeno vnašanje podatkov, boljše sodelovanje med uporabniki, boljši pregled nad pomembnimi opravili in njihovimi statusi ter boljši pregled medsebojno povezanih poslovnih informacij [10]. Podatki lahko prihajajo iz različnih izvorov. Če na te tokove podatkov gledamo kot na celoto, potem lažje določimo razmerja med podatki iz različnih izvorov. Tako lahko potrebne informacije zberemo skupaj, jih združimo in prikažemo uporabniku kot zaokroženo celoto.

Spletne storitve, aplikacije in druge vire se združi na način, da tvorijo kompozitno aplikacijo, ki združuje poslovno in procesno inteligenco v en paket. Kreiranje take aplikacije ni enostavno. Potrebno je integrirati različne aplikacije, protokole in tehnologije v smiselno celoto, pri čemer ni nujno, da so aplikacije, protokoli in tehnologije združljive med seboj [10]. Slika 17 prikazuje shemo takšnega sistema.

Kompozitna aplikacija je pojem, ki je pogosto povezan s storitveno usmerjeno arhitekturo. SOA (angl. service oriented architecture) je fleksibilen in modularen arhitekturni model, ki služi za implementacijo strukturirane poslovne logike. Arhitektura SOA, ki je na kratko predstavljena v nadaljevanju, nudi veliko prednosti, zato je smiselno, da jo uporabimo pri implementaciji kompozitnih aplikacij. V splošnem pa lahko kompozitne aplikacije obstajajo tudi brez SOA.

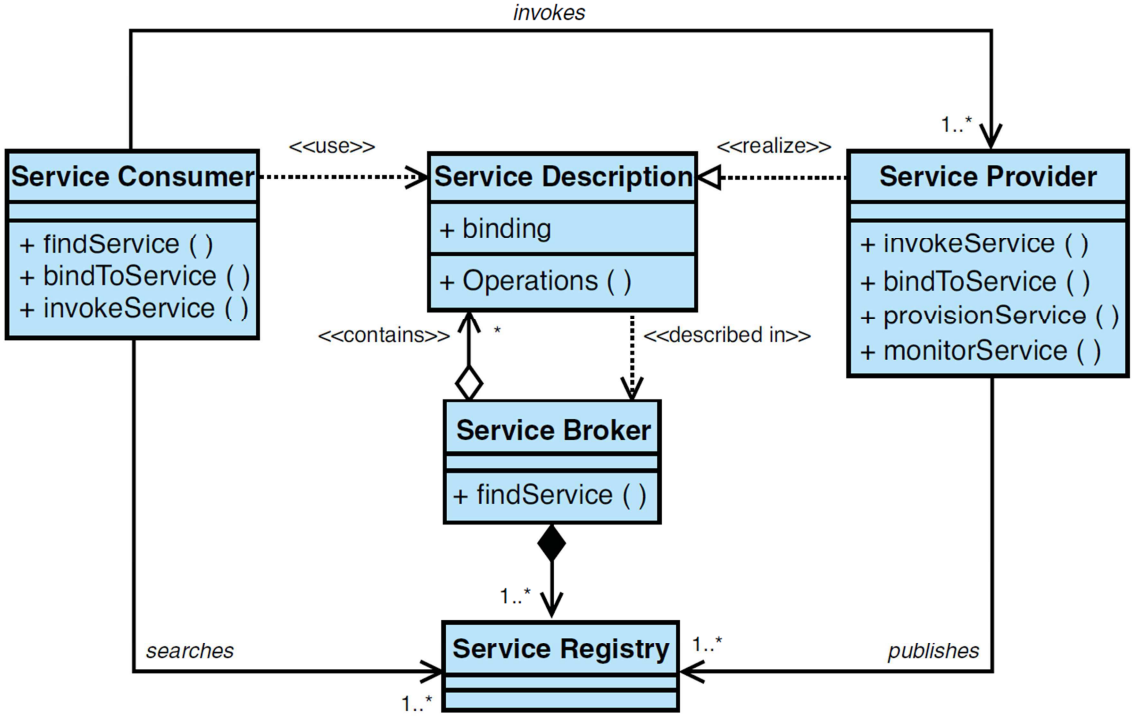
4.4.1. Storitveno usmerjena arhitektura[27]

SOA je arhitekturni model, kjer je rešitev predstavljena v obliki med seboj povezanih storitev. Storitev predstavlja logično in neodvisno poslovno funkcijo, ki je implementirana kot samostojna komponenta, ki jo je možno večkrat uporabiti. Storitve so brez stanja in so neodvisne od stanja drugih storitev. Odvisne so samo od njihovih funkcionalnosti.

SOA zahteva, da so storitve šibko sklopljene (angl. loose coupling). SOA nudi pristop, ki je zasnovan na odprtih standardih in splošnih sporočilih, ki niso specifični za katerokoli platformo ali programski jezik. Na ta način se doseže visoko stopnjo interoperabilnosti, kar implicira lažjo integracijo.

[27] Povzeto po [23].

Na sliki 16 so predstavljeni osnovni elementi arhitekture SOA: odjemalec storitev (angl. service consumer), ponudnik storitev (angl. service provider), opis storitve (angl. service description), posrednik storitev (angl. service broker) in register storitev (angl. service registry). Ponudnik objavi storitev skupaj z vmesnikom, ki vsebuje informacije, ki jih odjemalec potrebuje, da storitev najde in se nanjo poveže. Vmesnik ne vsebuje tehničnih podrobnosti implementacije, saj jih odjemalec ne rabi poznati. Vmesnik se objavi v registru storitev, po katerih išče odjemalec z različnimi iskalnimi operacijami.



Slika 16: SOA kot arhitekturni stil [23]

4.4.2. Nivoji arhitekture kompozitnih aplikacij[28]

Najnižji arhitekturni nivo so izvori podatkov, ki so shranjeni na nekem pomnilniškem mediju ali pa so rezultat procesiranja. Izvori lahko vključujejo aplikacije, notranje podatkovne baze, SaaS aplikacije, spletne storitve, datoteke in ostale izvore podatkov. Veliko SaaS aplikacij je opremljenih s programskimi vmesniki, ki izpostavijo različne lastnosti in metode za direktno uporabo.

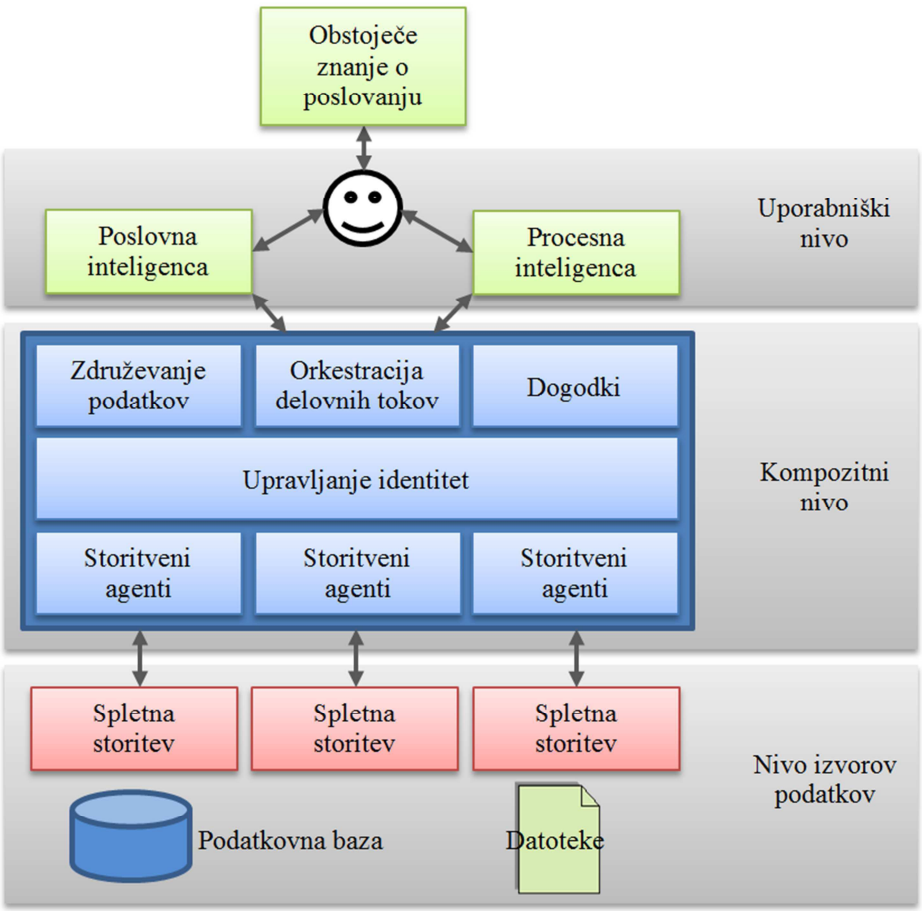
Kompozitni nivo je nivo, kjer se podatki združijo in so predstavljeni uporabniku v novi združeni obliki. Funkcija tega nivoja je pretvarjanje podatkov v poslovne in procesne informacije ter obratno.

Tehnične zmožnosti na tem nivoju vsebujejo komponente, ki upravljajo dostop, podatke, delovne tokove in pravila. Komponenta za upravljanje identitet zagotavlja, da so uporabniki

[28] Povzeto po [10].

ustrezno avtenticirani in avtorizirani, ter upravlja s poverilnicami za komunikacijo s spletnimi storitvami. Upravljanje z identitetami je bolj podrobno opisano v poglavju 4.2. Naloga komponente za združevanje podatkov je integracija in sinteza podatkov iz različnih izvorov glede na entitetni model. Ker je danes za poslovanje zelo pomembno hitro prilagajanje na spremembe, je komponenta delovnih tokov osrednja komponenta, saj zagotavlja usmerjanje zapletenih zaporedij opravil v poslovnih procesih. Komponenta za upravljanje dogodkov omogoča pošiljanje in prejemanje obvestil, tako da lahko uporabniki nadzorujejo poslovne aktivnosti in informacije. Kompozitni nivo se poveže z izvori podatkov preko storitvenih agentov (angl. service agents), katerih naloga je upravljanje povezav in izmenjava sporočil z izvori.

Uporabniški nivo predstavi kompozitne podatke uporabniku s centraliziranim, integriranim in opravilno orientiranim uporabniškim vmesnikom. Uporabniški vmesnik mora biti konsistenten, učinkovit in pregleden ter se držati standardov skozi celotno aplikacijo. Nuditi mora tako informacije za odločanje kot funkcionalnosti za izvajanje opravil. To je verjetno največji potencial storitveno orientiranega IT: združiti najboljše vidike različnih aplikacij in podatkovnih izvorov v eno aplikacijo, ki je poravnana s poslovanjem. Ta aplikacija je osredotočena na potrebe uporabnika namesto na zmožnosti in omejitve posameznega sistema.



Slika 17: Arhitektura kompozitne aplikacije [10]

4.4.3. Implementacija delovnih tokov

Komponenta delovnih tokov je osrednja komponenta, ki zagotavlja implementacijo poslovnih procesov. Delovni tok in proces sta sorodna koncepta. Proces je možno predstaviti z delovnim tokom. Delovni tok (angl. workflow) je zaporedje povezanih korakov, ki se izvajajo v določenem zaporedju. To pomeni, da se mora prejšnji korak do konca izvršiti, preden se lahko naslednji začne.

Za implementacijo aplikacij z delovnimi tokovi Microsoft nudi svojo rešitev Windows Workflow Foundation, ki vsebuje knjižnice in grafični vmesnik za modeliranje poslovnih procesov. WWF omogoča grafično kreiranje delovnih tokov ter spreminjanje in dodajanje komponent, ki so implementirane v programskih jezikih, kot je na primer C# ali VB.NET [24].

Business Process Model and Notation (BPMN)[29]

BPMN je grafična predstavitev poslovnih procesov. Je standard za modeliranje poslovnih procesov (angl. business process modeling), ki nudi grafično notacijo za navajanje poslovnih procesov v diagramih. BPMN verzije 2.0 nudi tudi semantiko za izvrševanje. Namen BPMN je podpora upravljanju poslovnih procesov in sprememb, do katerih pride tekom poslovanja (angl. business process management). Notacija je dovolj intuitivna, da jo razumejo poslovni analitiki, ki definirajo procese, in hkrati sofisticirana, da lahko razvijalci implementirajo rešitve, ki bo izvrševala kompleksne procese. BPMN vsebuje neuraden mehanizem za pretvorbo grafične notacije v izvajalni jezik, kot je na primer BPEL. Pretvorba je samo delna, saj ni vedno možno generirati kode iz BPMN modelov.

Business Process Execution Language (BPEL)[30]

BPEL (angl. business process execution language) je jezik za orkestracijo spletnih storitev. Procesi prenašajo informacije izključno z uporabo vmesnikov spletnih storitev. Uporablja se za abstrakcijo sodelovanja in zaporedja iz različnih spletnih storitev ter tvori formalno definicijo procesov, kjer so definicije zasnovane na standardu XML.

Windows Workflow Foundation (WWF)[31]

WWF je komponenta okolja .NET Framework za razvoj aplikacij, ki temeljijo na delovnih tokovih. Posamezni koraki v delovnih tokovih so tu poimenovani aktivnosti. Vsebuje orodje za grafično modeliranje ter stroj za razvrščevanje in izvrševanje delovnih tokov. Delovne tokove je možno izvršiti na tri načine:

---

[29] Povzeto po [25, 26].
[30] Business process execution language (BPEL) je krajše ime za Web services business process execution language (WS-BPEL).
[31] Povzeto po [27, 28].

- z uporabo WorkflowInvoker, ki izvrši delovni tok v isti niti, v kateri teče aplikacija, ki je zahtevala izvršitev,
- z uporabo WorkflowApplication, ki izvrši delovni tok v ločeni niti, ali
- z uporabo WorkflowServiceHost, ki izvrši delovni tok v ločeni WCF (angl. Windows Communication Foundation) storitvi.

Delovni tok se izvršuje, dokler je prisotna še kakšna aktivnost ali dokler vse aktivnosti, ki se izvajajo, čakajo na vnos. V tem primeru je delovni tok prost, jedro shrani podatke na primer v podatkovno bazo in sprosti pomnilnik, ki ga je zasedal delovni tok. Po določenem času ali po prejemu sporočila ga jedro ponovno naloži v pomnilnik.

Poleg tega jedro upravlja tudi tok izvajanja aktivnosti, upravlja podatke za izvrševanje aktivnosti (npr. argumenti in spremenljivke), ima vgrajeno sledenje izvajanih aktivnosti, ki se beležijo, ter nudi razširitve za poljubne funkcionalnosti in vizualno razhroščevanje.

V sklopu WWF so vključena tudi orodja za uvoz in izvoz v jezik BPEL ter aktivnosti, ki predstavljajo BPEL [29].

4.5. Metapodatkovne storitve[32]

Metapodatki (angl. metadata ali včasih angl. metacontent) opisujejo strukturo in pomen podatkov in aplikacij. To so podatki, ki vsebujejo informacije o načinu, namenu, času kreiranja, avtorju, lokaciji podatkov ali uporabljenih standardih. Shranjeni so v skladišču metapodatkov (angl. metadata repository), ki se lahko nahaja skupaj z vsebino ali pa ločeno. Če so metapodatki shranjeni ločeno, so običajno shranjeni v podatkovni bazi ali v datoteki. Nekatere funkcije se bolje izvajajo s podatkovno bazo, saj ta omogoča hitrejše delovanje zaradi strukture in indeksov. Priporočljivo je, da se vsi metapodatki hranijo v enotnem skladišču podatkov. Tako jih je možno enostavno prenesti iz testnega v produkcijsko okolje. Poleg tega je pomembna tudi ustrezna struktura metapodatkov, da je možno po njih iskati in jih ponovno uporabiti.

Zrela SaaS aplikacija vsebuje metapodatkovne storitve, ki uporabnikom zagotavljajo način, s katerim si aplikacijo nastavijo glede na svoje potrebe (angl. customization). V tipičnem scenariju se nastavljajo štiri področja:

- uporabniški vmesniki in označevanje z logotipi: odjemalci pogosto hočejo uporabniške vmesnike prilagoditi svojim potrebam in jih označiti kot svoje, zato SaaS aplikacije pogosto vsebujejo funkcije za spreminjanje grafike, barv, pisav itd. S

[32] Povzeto po [8, 30].

pojavom CSS (angl. cascading style sheets) je postalo spreminjanje uporabniškega vmesnika enostavno. Nastavitve morajo vsebovati tudi prilagajanje izpisov za tiskanje.

- delovni tokovi in poslovna pravila: da zagotovimo čim širši množici odjemalcev, mora naša SaaS aplikacija imeti funkcionalnosti za spreminjanje delovnih tokov in poslovnih pravil, ki vplivajo na izvajanje poslovnih procesov. Implementacija delovnih tokov je na kratko opisana v poglavju 4.4.3.
- razširitve podatkovnega modela: razširitve podatkovnega modela so opisane v poglavju 4.1.
- kontrola dostopa: tipično je vsak odjemalec odgovoren za kreiranje posameznih računov za končne uporabnike in za določanje, do katerih funkcionalnosti in virov ima dostop. Pravice dostopa in omejitve so za vsakega uporabnika spremljane z uporabo varnostnih politik, ki jih mora nastaviti vsak odjemalec. Število neuspelih poizkusov prijave ter najmanjša in največja dovoljena dolžina gesel so tipični primeri nastavitev.

Da zagotovimo fleksibilnost za nastavitve aplikacije, so te opcije hierarhično razporejene v sklope. Vsaka stranka si nastavi sklope nastavitev od danih področij glede na svoje potrebe. Poleg sklopov na najvišjem nivoju ima na voljo tudi poljubno število podsklopov, ki skupaj tvorijo hierarhijo. Uporabnik mora določiti tudi, ali podsklopi podedujejo ali prepišejo nastavitve sklopa na višjem nivoju.

SaaS aplikacije običajno nastavljajo stranke same za razliko od tradicionalnih aplikacij. Zato je zelo pomembno, da ponudnik programske opreme naredi pregleden vmesnik za te nastavitve. V idealnem primeru lahko stranke same nastavijo aplikacijo skozi čarovnik ali kakšen drugačen enostaven in intuitiven način, ki stranko ne preobremeni z informacijami. Ne glede od načina implementacije vmesnika mora biti jasno razvidno, kaj lahko stranka spremeni na posameznih sklopih in kaj ne.

4.6. Nadzorovanje in merjenje[33]

Odjemalec podpiše dogovor o nivoju storitev (angl. service level agreement) za uporabo aplikacij s ponudnikom programske opreme. SLA določa operacijske standarde, ki jih mora ponudnik zagotoviti. SLA je pravno zavezujoča pogodba ter kršitev take pogodbe lahko za ponudnika SaaS pomeni veliko izgubo in negativno vpliva na ugled podjetja. Zaradi tega je zelo pomembno nadzorovati aplikacijo za kakršenkoli znake o problemih. Te probleme je potrebno odpraviti preden povzročijo izpad ali slabše delovanje aplikacije.

[33] Povzeto po [2].

V sklopu nadzorovanja aplikacije se nahaja merjenje različnih parametrov. Dane meritve so lahko osnova za zaračunavanje uporabe storitev, če sta se ponudnik in odjemalec tako dogovorila. Poleg tega so meritve potrebne tudi za sam nadzor aplikacije. Meritve morajo obsegati odzivnost podatkovne baze, število procesov v čakalni vrsti, zahteve za dostop do trdega diska, dosegljivost omrežja, itn.

Naloga ponudnika je, da meritve analizira in ustrezno ukrepa. Ponudnik mora biti sposoben iz meritev ugotoviti, če kje prihaja od ozkih grl, in ustrezno ukrepati, da jih odpravi. Dani podatki so osnova za načrtovanje kapacitet. Cilj načrtovanja kapacitet je zagotoviti zadostne in hkrati ne prekomerne vire, da zagotovimo zahtevam odjemalcev v pravem času. Načrtovanje kapacitet poteka v treh korakih. Prvi korak vključuje samo merjenje. V drugem koraku potrebujemo predvideti delovno breme, zmodelirati različne kombinacije in izmed teh kombinacij izbrati najcenejšo, ki bo opravila nalogo v okviru odzivnih časov in storitvenih nivojev, ki jih potrebujemo. Tretji korak je izdelava modela, ki ga je potrebno preveriti. Zato mora biti načrtovanje kapacitet interaktivno in zahteva konstantno preverjanje modelov. Pri tem je pomembno, da natančno predvidimo delovno breme.

Poleg tega so meritve pomemben vir podatkov za poslovanje. Iz teh podatkov lahko ugotovimo, kakšno je delovanje našega IT oddelka v primerjavi s tekmeci, kakšno je delovanje v primerjavi s pričakovanji odjemalcev, poslovnih partnerjev in delničarjev.

Za začetek pa mora podjetje določiti, kaj se meri in spremlja, kako ti podatki pomagajo pri poslovanju, ali oddelek za upravljanje dobi prave informacije v pravem času, ali je možno iz danih podatkov predvideti možne napake in jih odpraviti, preden se zgodijo.

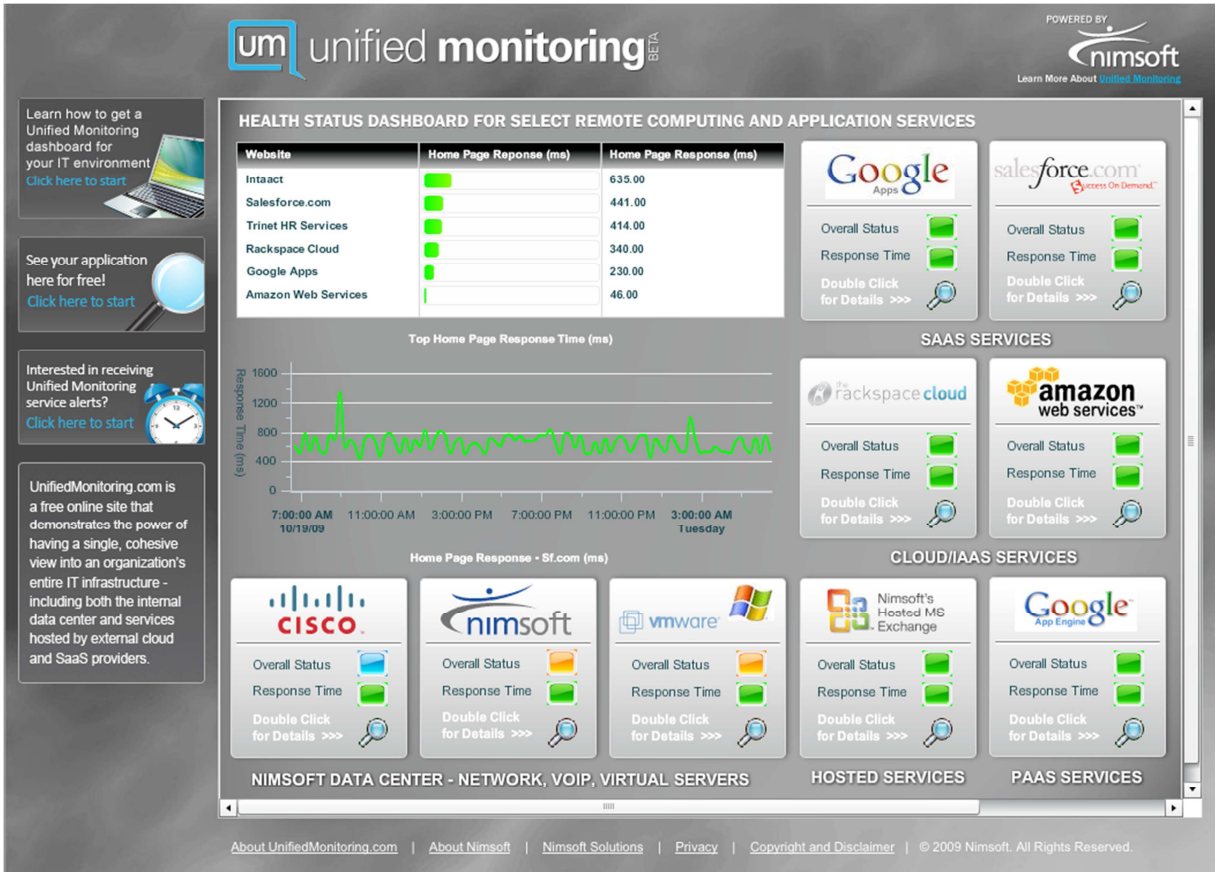
4.6.1. Nadzorovanje

Nadzorovanje (angl. monitoring) pomeni, da smo investirali v tehnologije, ki nam dajejo vpogled v operacije, ki se izvajajo. Potrebno je omogočiti nadzor in upravljanje poslovnih aktivnosti, dosegljivosti in delovanja aplikacij. Veliko podjetij uporablja pregledne plošče (angl. dashboard). Pregledna plošča je vmesnik, ki vsebuje različne storitve in kaže, če delovanje dosega zastavljene cilje. Pregledna plošča mora vsebovati informacije tudi iz oblaka. Obstaja že kar nekaj ponudnikov, ki nudijo orodja za nadzor ponudnikov storitev v oblaku. Slika 18 prikazuje primer pregledne plošče, ki pripada podjetju Nimsoft.

4.6.1.1. Nadzor dosegljivosti aplikacije[34]

Zagotoviti visoko dosegljivost aplikacije je za ponudnika SaaS aplikacij ena od najpomembnejših prioritet. Izpad posameznega strežnika ali podatkovnega centra lahko pripelje do velike izgube podatkov in produktivnosti večine uporabnikov. Za ponudnike programske opreme, ki so iz tradicionalnega modela prešli na SaaS model, je zagotavljanje visoke dosegljivosti aplikacije nov in nepoznan izziv. Priporočljivo je, da ponudnik v aplikacijo implementira podporo za osnovne tehnike, kot je na primer mehanizme za alarme. Ob sistemski napaki se mora sprožiti alarm in zahteve se morajo prenesti na redundantne strežnike, da uporabnik ne občuti izpada. Zato je zelo pomembna replikacija podatkov in aplikacij ter vzpostavitev robustnega ogrodja za redundanco. Posebno pozornost je potrebno nameniti nadzoru šibkih členov, kot je na primer povezava na bazo.

Če imamo implementiran mehanizem za alarme, to še ni dovolj. Če se alarm sproži in se nanj noben ne odzove, potem ne moremo reči, da je alarm del procesa. Potrebno je zagotoviti, da so definirani procesi, ki predpisujejo akcije v primeru sistemskih napak.



Slika 18: Pregledna plošča podjetja Nimsoft [11]

[34] Povzeto po [8].

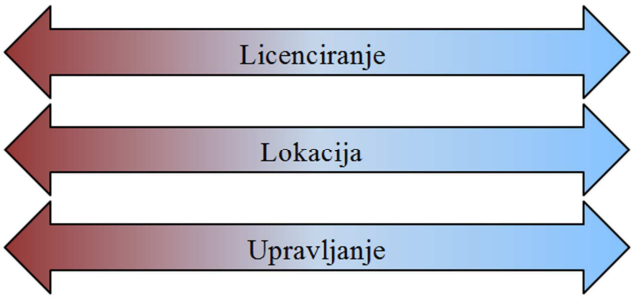
4.6.1.2. Nadzor delovanja aplikacije[35]

Odjemalci pričakujejo, da ponudnik zagotovi sprejemljiv nivo delovanja aplikacije. Do določene točke je nivo delovanja aplikacije določen z SLA. Če se uporabnikom zdi, da je aplikacija počasna in neodzivna, je malo verjetno, da bodo podaljšali naročnino na aplikacijo. Poleg tega lahko svoje izkušnje objavijo na spletu in tako lahko aplikacija dobi negativen ugled. To dejstvo velja tudi v obratnem primeru. Če je aplikacija hitra in odzivna, se aplikaciji poveča ugled, uporabniki so zadovoljni in obstaja večja verjetnost, da bodo podaljšali naročnino in začeli uporabljati še druge storitve, ki jih ponuja ponudnik programske opreme.

[35] Povzeto po [8].

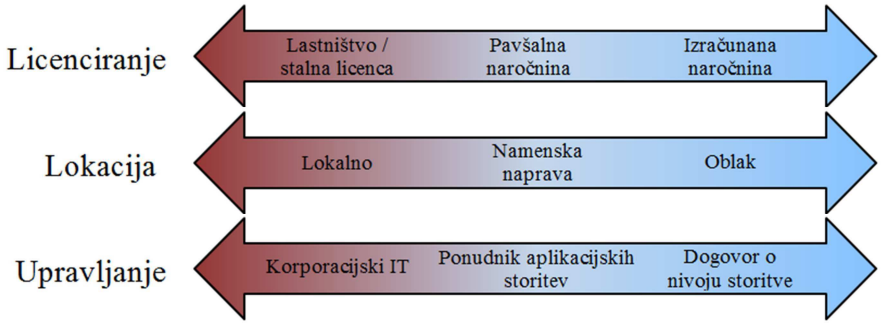
5. Poslovni model

V pravi obliki modela SaaS ima ponudnik aplikacijo nameščeno na lastnih strežnikih in ponuja dostop do aplikacije preko interneta v zameno za naročnino. V praksi razlike med SaaS in tradicionalnimi aplikacijami niso binarne, ampak se razlikujejo po treh dimenzijah: kako je programska oprema licencirana, kje se nahaja in kako se upravlja. Vsakega od teh vidikov se da predstaviti kot neprekinjeno zvezo s tradicionalno programsko opremo na eni strani in SaaS na drugi strani (slika 19) [10].



Slika 19: Tri dimenzije, po katerih se SaaS aplikacije razlikujejo med seboj [10]

Vsako neprekinjeno zvezo se da v grobem razdeliti v tri večje dele, ki predstavljajo tradicionalen, SaaS in med njima hibriden pristop (slika 20). Ponudnik lahko glede na okoliščine določi strategijo, ki iz vsake od teh treh dimenzij neodvisno od ostalih dveh izbere model, po katerem strankam ponudi svoje rešitve [10].



Slika 20: Glavni segmenti vseh treh dimenzij [10]

Licenciranje

Pri tradicionalnih aplikacijah prodajalec vsaki stranki zaračuna enkraten znesek za uporabo leteh. Pri pravih SaaS aplikacijah stranka plača glede na uporabo aplikacije, kjer je znesek običajno odvisen od števila izvedenih transakcij [10]. To je tako imenovan »consumption-based pricing model«. Obstaja še vmesna možnost, kjer stranka plača naročnino za vsakega

uporabnika za določeno časovno obdobje (npr. mesec ali četrtletje). V tem obdobju stranka lahko neomejeno uporablja storitve, za katere plačuje naročnino [10]. Temu modelu pa se reče »subscription-based pricing model«.

Omeniti velja še »freemium model«, kjer je omejena funkcionalnost storitve na voljo brezplačno, celotna funkcionalnost pa se zaračuna glede na uporabo ali pa z naročnino za določeno časovno obdobje.

Strežniška infrastruktura

SaaS aplikacije so nameščene pri prodajalcu samem, medtem ko so tradicionalne aplikacije nameščene pri stranki na infrastrukturi, ki si jo sama priskrbi. Obstaja še model, kjer prodajalec priskrbi strojno in programsko opremo, ki se fizično nahaja pri stranki, ampak stranka nima vpogleda v komponente[36]. Sistem je stranki na voljo kot črna škatla [10].

Upravljanje

Ena od nalog IT oddelka je upravljanje računalniških virov v skladu s potrebami. Ti viri vključujejo konkretne investicije v omrežno opremo, strežnike in platformo za aplikacije. Prav tako mora IT zagotavljati podporo uporabnikom in reševanje težav ter odpravljaje problemov glede varnosti, zanesljivosti, dosegljivosti in zmogljivosti. Ker je to obsežno delo, se podjetja včasih odločijo, da upravljanje prepustijo zunanjemu izvajalcu, ki je osredotočen na IT upravljanje. Na drugi strani imamo SaaS aplikacije, ki jih v celoti upravlja ponudnik storitev. Stranka pri tem modelu podpiše dogovor o nivoju storitev, s katerim se ponudnik zaveže, da bo zagotavljal dogovorjene nivoje kvalitete, dosegljivosti in podpore uporabnikom [10].

5.1. Vplivi na poslovni model

Da ponudnik programske opreme določi vsako od teh treh dimenzij, mora pri tem upoštevati veliko faktorjev, ki vplivajo na razmerje med ceno in nadzorom nad aplikacijo. Tu so navedeni nekateri faktorji [10]:

- politični faktorji: včasih pride do tega, da pride do nasprotovanj znotraj družbe, ko ljudje na vodilnih položajih vztrajajo, da se določene funkcionalnosti obdržijo znotraj družbe in pod nadzorom lastnega IT oddelka. V teh primerih vsi ostali vidiki niso pomembni.
- tehnični faktorji: SaaS aplikacije so običajno fleksibilne, da jih lahko stranka prilagodi svojim potrebam, vendar ima ta pristop določene omejitve. Za določene aplikacije potrebujemo specializirano tehnično znanje za delovanje in podporo teh aplikacij ali

[36] Primer takšnega sistema je Mirapoint Message Server, ki nudi aplikacijo za elektronsko pošto, operacijski sistem, strojno infrastrukturo in podatkovno skladišče v enem paketu.

pa stranka zahteva spremembe, ki jih ni možno posplošiti in implementirati brez programiranja po meri. V takih primerih ni možno narediti aplikacije po SaaS modelu.

Poleg tega moramo upoštevati količino podatkov, ki se bodo redno prenašali. Pasovna širina internetne povezave je tipično manjša od pasovne širine v lokalni mreži znotraj podjetja. Zaradi tega lahko prenosi trajajo nekaj ur, medtem ko bi isti prenos znotraj lokalnega omrežja trajal nekaj minut. Iz tega sledi, da je potrebno pri načrtovanju programske opreme upoštevati tudi zakasnitev v omrežju. Po potrebi je potrebno implementirati predpomnjenje (angl. cache) ali skupinsko pošiljanje podatkov (angl. batch).

- finančni faktorji: nakup programske opreme predstavlja določeno tveganje za podjetje in stranka mora ugotoviti, če se ji SaaS v primerjavi s tradicionalno aplikacijo sploh splača. Začetni stroški SaaS aplikacij so vsekakor nižji, na dolgi rok pa to morda ne drži več. Celoten strošek SaaS aplikacije je med drugim odvisen od števila uporabnikov, od količine nastavitev, ki so potrebne za integracijo z obstoječimi sistemi, in od obstoječe infrastrukture, ki jo ima stranka že na voljo. Poleg tega je v interesu stranke počakati s prehodom na SaaS aplikacije, če je pred kratkim pridobila aplikacije, ki se jim vrednost naložbe še ni povrnila.
- pravni faktorji: stranka se mora pozanimati, če ponudnik programske opreme zadovoljuje zakonskim predpisom. Predvsem tu lahko pride do razhajanja pri raznih poročilih, ki jih je stranka dolžna posredovati državi, ter pri zakonskih omejitvah glede varstva podatkov.

5.2. Spreminjanje poslovnega modela

Glavni argumenti, zakaj so SaaS aplikacije primernejše od tradicionalnih, so naslednji štirje [4]:

- z istim proračunom dobimo več,
- ni podcenjevanja človeških storitev,
- SaaS omogoča boljše upravljanje rasti in
- odgovornost SaaS ponudnika.

5.2.1. Z istim proračunom dobimo več

SaaS model omogoča direktno in izmerljivo gospodarsko prednost v primerjavi s tradicionalnim pristopom.

Obstaja veliko argumentov, ki dajejo prednost SaaS aplikacijam, zato se mnogo podjetij odloča za spremembe poslovnega modela. Spreminjanje poslovnega modela lahko vsebuje [8]:

- spreminjanje lastništva programske opreme,
- prerazporejanje odgovornosti za infrastrukturo in upravljanje (strojne opreme in profesionalnih storitev),
- nižanje cen za zagotavljanje storitev skozi specializacijo in ekonomijo obsega ter
- preusmeritev na tržišče manjših in srednje velikih podjetij.

5.2.1.1. Razporeditev stroškov[37]

V tipičnem podjetju se IT proračun porabi za tri stvari:

- programska oprema,
- strojna oprema in
- profesionalne storitve.

Od teh treh dejavnikov je programska oprema najbolj neposredno vključena v upravljanje informacij, kar je končni cilj kateregakoli podjetja. Strojna oprema in profesionalne storitve so nujno potrebne komponente, ampak so samo sredstva za dosego ciljev, ki zagotavljajo programski opremi, da zagotavlja učinkovite rezultate upravljanja informacij.

Za tradicionalno programsko opremo se proračun v veliki meri porabi za strojno opremo in profesionalne storitve. Tako za programsko opremo ostane razmero majhen del proračuna (slika 21).

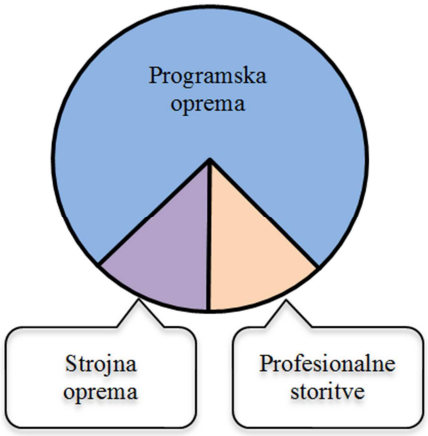
Slika 21: Tipična razporeditev proračuna za okolje s tradicionalnimi aplikacijami [8]

Pri tem modelu se proračun za programsko opremo primarno porabi za licence kopij poslovnih aplikacij. Proračun za strojno opremo gre za namizne in prenosne računalnike za končne uporabnike, strežnike, na katerih se nahajajo podatki in aplikacije, ter omrežne

[37] Povzeto po [4, 8].

komponente. Proračun za profesionalne storitve je namenjen za podporo, ki skrbi za namestitev in podporo strojne in programske opreme, ter za svetovalce in za razvoj sistemov, ki so narejeni po naročilu.

Za podjetje, ki je osredotočeno na SaaS, pa zgleda delitev proračuna drugače (slika 22).



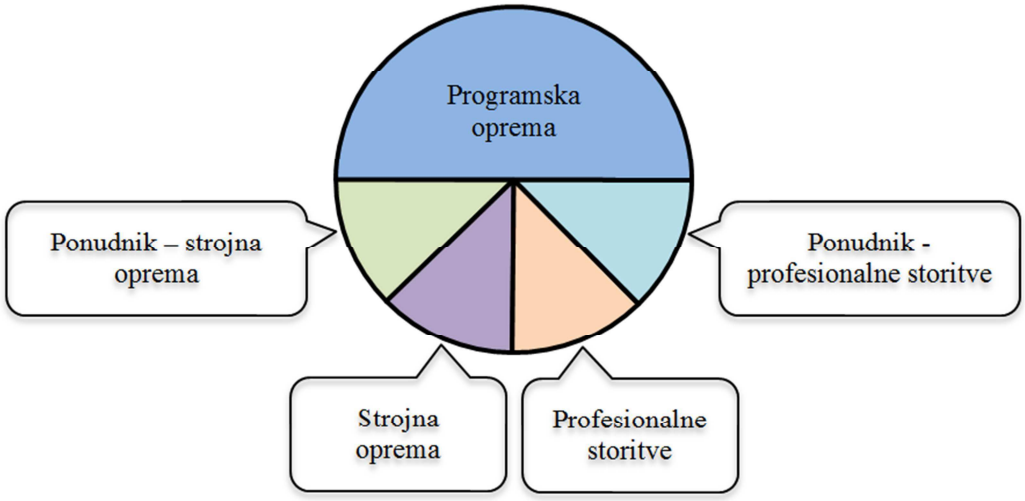
Slika 22: Tipična razporeditev proračuna za okolje s SaaS aplikacijami [8]

Po tem modelu stranka dostopa do aplikacij, ki so nameščene na infrastrukturi ponudnika storitev. Tako stranki ni potrebno kupiti in vzdrževati strojne opreme. Ta opravila so prepuščena ponudniku, ki ima za to namenjen kader. Poleg tega aplikacije, do katerih se dostopa preko interneta, imajo precej nižje strojne zahteve, kot pa lokalno nameščene verzije. To pomeni, da se stranki podaljša čas, preden mora zamenjati računalnike. Končni rezultat je, da stranki na voljo ostane veliko večji del proračuna, ki ga lahko porabi za nakup programske opreme, ki se ga običajno porabi za naročnine za nove SaaS aplikacije.

5.2.1.2. Ekonomija obsega[38]

Slika 22 je malo zavajajoča, saj ponudnik del naročnine, ki jo stranka plača, porabi za strojno opremo in profesionalne storitve, kar je prikazano tudi na sliki 23. Razlika je v ekonomiji obsega in večnajemniški arhitekturi. Ekonomija obsega nudi prihranke stroškov, saj je zaradi večnajemniške arhitekture možna visoka stopnja deljenja računalniških virov. Ponudnik lahko na enem strežniku gosti na primer pet odjemalcev, medtem ko bi po tradicionalnem modelu vsak odjemalec potreboval kupiti svoj strežnik ali celo več strežnikov, če so problem izenačevanje obremenitve in visoka dosegljivost. Ponudnik lahko dodatno zniža stroške, če ima vzpostavljeno dinamično skalabilno okolje in avtomatizira čim več upravljavskih opravil. To pomeni, da ne glede na to, da se stroškom strojne opreme in profesionalnih storitev ne da izogniti, lahko ponudnik ponudi svoje rešitve po nižji ceni in stranke lahko dobijo več funkcionalnosti programske opreme pri isti količini proračuna.

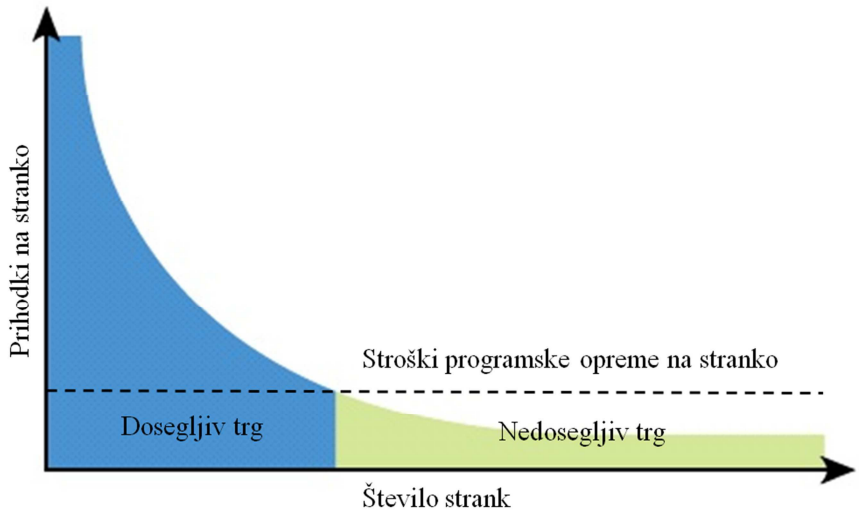
[38] Povzeto po [4, 8].



Slika 23: Tipična razporeditev proračuna za okolje s SaaS programsko opremo (prikazuje tudi stroške, ki jih ponudnik storitev porabi za strojno opremo in profesionalne storitve) [8]

5.2.1.3. Trg manjših in srednje velikih podjetij

Povpraševanje po določenih kategorijah trgovskega blaga (npr. knjige) ima obliko eksponentne porazdelitve. V takem scenariju je vsako leto izdanih tisoč novih knjig, ampak le nekaj od njih doseže nivo najbolj prodajanih. Ostali proizvodi ostanejo v tako imenovanem dolgem repu (angl. long tail). Tako imamo veliko število proizvodov manjših naklad. Preprodajalci se osredotočijo na prodajo najbolj prodajanih izdelkov, ker ne morejo skladiščiti po 1000 izvodov vsake knjige. Spletni preprodajalci nimajo težav s skladiščenjem, saj lahko pošiljko pošljejo direktno iz kakšnega večjega skladišča. Spletni preprodajalci lahko oglašujejo in prodajajo veliko več priljubljenih naslovov in to na enako lahek način kot najbolj priljubljene. Prodaja velikega števila naslovov nizkih naklad lahko pripelje do velikih prihodkov [8].

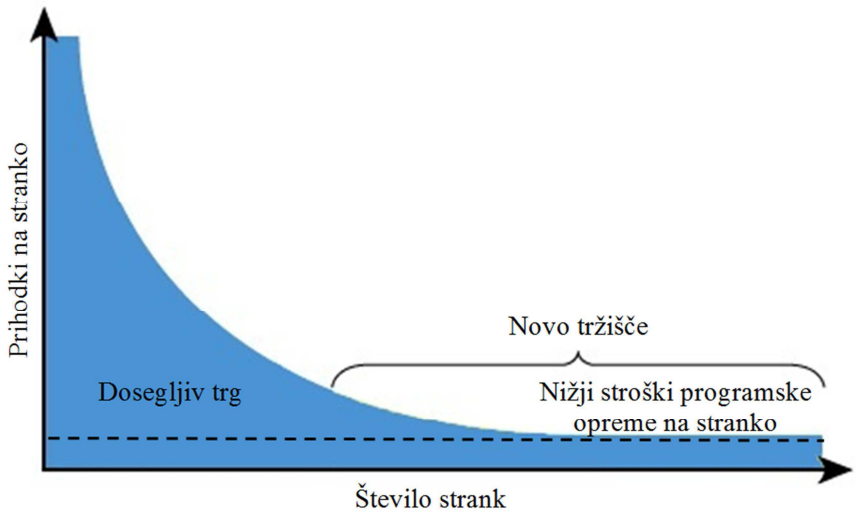


Slika 24: Krivulja števila možnih strank za trg poslovnih aplikacij [8]

Prodajalci poslovnih aplikacij se soočajo z isto krivuljo dohodkov v odvisnosti od števila možnih strank (slika 24).

Dolgi rep predstavlja veliko število možnih strank – majhnih in srednje velikih podjetij, ki imajo zaradi manjšega obsega poslovanja na voljo tudi manjši proračun, ki ga namenijo za IT [34]. Posledično si ne morejo privoščiti nakupa večjih poslovnih aplikacij, ki so pogosto prilagojene potrebam posameznih strank. Ti programski paketi pogosto zahtevajo namenski strežnik in kader za podporo ter včasih vključujejo tudi namestitev pri stranki in obiske ekip za zagotavljanje storitev od ponudnika. Stroški takšnega poslovnega modela doprinesejo k najnižji ceni, po kateri si lahko ponudnik še privošči prodajati programsko opremo [8].

Ponudniki programske opreme lahko nudijo svoje rešitve po občutno nižjih cenah, če jim uspe odpraviti prekomerne stroške vzdrževanja. To lahko naredijo tako, da uporabijo ekonomijo obsega, da združijo in centralizirajo strojno opremo in storitve. S tem stranka potrebuje veliko manjšo kompleksnost lastne IT infrastrukture. Iz tega sledi, da je SaaS ponudnikom na voljo veliko več potencialnih strank kot ponudnikom tradicionalne programske opreme (slika 25) [8].



Slika 25: Nov del trga, ki je dostopen SaaS ponudnikom, zaradi nižjih cen aplikacij [8]

5.2.1.4. Primeri cen najbolj znanih SaaS aplikacij

V tabeli 1 so prikazane cene ene najbolj znanih SaaS aplikacij: Salesforce.com CRM. Podjetje trži svojo rešitev glede na število uporabnikov in ne glede na dejansko uporabo. Salesforce je aplikacija za upravljanje odnosov s strankami (angl. customer relationship management), ki je na voljo v oblaku. Salesforce.com ne nudi lokalne verzije svojega programa. Aplikacija je zgrajena na moderni arhitekturi, kar omogoča skalabilnost, zanesljivost in dosegljivost ter integracijo z drugimi aplikacijami preko spletnih storitev. Vzpostavljeno imajo ogromno bazo uporabnikov in ekosistem partnerjev, ki dopolnjujejo njihove rešitve.

Salesforce.com CRM Professional edition	Salesforce.com CRM Enterprise edition	Salesforce.com CRM Unlimited edition
65$/mesec za vsakega uporabnika	125$/mesec za vsakega uporabnika	250$/mesec za vsakega uporabnika

Tabela 1: Cene aplikacije Salesforce.com CRM [36]

5.2.2. Ni podcenjevanja človeških storitev

Drugi faktor, zakaj so SaaS aplikacije primernejše od tradicionalnih, je podcenjevanje stroškov zaposlenih, ki so povezani s poganjanjem tradicionalnih aplikacij. Izračun dejanskih stroškov storitev, ki jih opravljajo zaposleni, ni enostaven in rezultat tega je, da so ti stroški včasih izpuščeni iz analize skupnih stroškov lastništva programske opreme (angl. total cost of ownership). Posledično to pripelje do neustrezne primerjave.

Podjetja si ne morejo več privoščiti razkošja gledati samo na stroške nakupa strojne in programske opreme, ampak morajo preveriti, kakšen vpliv imajo nakupi in zaposleni, ki skrbijo za aplikacije, skozi celoten življenjski cikel. Podjetja običajno razumejo in podrobno pregledajo stroške programske in strojne opreme, stroški zaposlenih pa običajno niso dovolj podrobno upoštevani. Je pa zelo pomembno, da določimo vse stroške in kako vplivajo na TCO, saj to omogoča učinkovito vodenje poslovanja.

Glede na Gartner Inc. se 75% IT proračuna porabi za vzdrževanje in poganjanje obstoječih sistemov in infrastrukture, ki je namenjena za aplikacije. Poleg tega verjamejo še, da stranke porabijo 4-krat več za upravljanje in lastninjenje aplikacij, kot pa stane licenca [4].

5.2.3. SaaS omogoča boljše upravljanje rasti[39]

Tretji argument je, da SaaS aplikacije rastejo z rastjo poslovanja. Podjetjem se ni potrebno odločati o tipu aplikacije, ki jo potrebujejo glede na velikost podjetja, ampak se lahko odločijo striktno glede na potrebe poslovanja.

Če vzamemo primer podjetja, ki ima 100 zaposlenih, ki potrebujejo dostop do aplikacije, potem mora podjetje v primeru tradicionalne aplikacije kupiti vnaprej vseh 100 licenc. V primeru SaaS aplikacije pa podjetje ne potrebuje infrastrukture za poganjanje aplikacije, niti ne potrebuje izobraziti IT kadra. Tako lahko kupi na primer samo 10 licenc in kasneje po potrebi dokupi dodatne. To je še posebej pomembno, ko ima podjetje omejen proračun. Iz

[39] Povzeto po [4].

tega sledi, da so SaaS aplikacije veliko bolj prijazne za model rasti podjetja kot pa tradicionalne aplikacije.

Drugi primer je fleksibilnost SaaS aplikacij. V primeru združitev ali pridružitev podjetij je ureditev oziroma postavitev IT infrastrukture problematična. Lahko traja celo nekaj mesecev preden je vzpostavljena infrastruktura, da lahko uporabniki dostopajo do storitev na drugih lokacijah. V kolikor SaaS aplikacija ni odvisna od podatkov iz tradicionalnih aplikacij, je ta problem manjši, saj za uporabo aplikacij uporabnik potrebuje samo internetno povezavo, spletni brskalnik ter uporabniško ime in geslo.

5.2.4. Odgovornost SaaS ponudnika

Četrti argument je, da SaaS ponudnik nosi večjo odgovornost, ker aplikacijo prodaja za naročnino namesto za klasično licenco. Stranke imajo običajno večji vpliv na SaaS ponudnika kot na ponudnika tradicionalnih aplikacij. Stranke plačujejo naročnino za dobo veljavnosti pogodbe. Ponudniki se zavežejo, da bodo izpolnjevali pogoje pogodbe SLA, in posledično so finančno motivirani, da zagotovijo zadostno podporo in nivo delovanja. Če ponudnik ne izpolnjuje pogojev pogodbe SLA, lahko stranke zadržijo plačilo naročnine in uveljavljajo pogoje iz pogodbe SLA. Ponudniki tradicionalnih aplikacij dobijo veliko vnaprejšnje plačilo za njihovo aplikacijo in imajo malo obveznosti, ko je enkrat aplikacija nameščena [4].

5.3. Dogovor o nivoju storitev

Dogovor o nivoju storitev (angl. service level agreement) je sklenjena pogodba med dvema udeležencema, pri čemer je prvi udeleženec stranka drugi pa ponudnik storitev. To je lahko pravno zavezujoča pogodba ali pa neformalna pogodba (za razmerja med notranjimi oddelki). SLA so poimenovane tudi pogodbe med ponudniki storitev in strankami, čeprav dejansko ni nobenega dogovora o nivoju storitev, ampak so le-ti določeni s strani ponudnika storitev [12].

V SLA so zapisani splošni dogovori o storitvah, prednostnih nalogah, odgovornostih, garancijah in pooblastilih. Za vsako posamezno področje obsega storitve morajo biti definirani nivoji. Tako lahko SLA definira razpoložljivost, uporabnost, delovanje, operacije in druge atribute storitve, kot je na primer obračun storitev. Nivo storitve je lahko določen tudi kot minimalen in povprečen, s tem stranke vedo, kaj lahko pričakujejo. V nekaterih pogodbah so določene tudi kazni v primeru neskladnosti SLA. Pomembno je tudi dejstvo, da se SLA nanaša na to, kakšen nivo storitve stranka dobi, ne pa kakšen nivo storitve izroči ponudnik [12].

5.3.1. Vsebina SLA

SLA običajno vključuje naslednje segmente za obdelavo [13]:

- definicija storitev: to je najbolj pomemben del SLA, saj opisuje storitve in način, po katerem so te storitve na voljo.
- merjenje: ključni del SLA obravnava nadzor in merjenje nivojev delovanja storitev. Glavno je, da je možno meriti izvajanje vsake storitve ter da je možno rezultate analizirati in predstaviti. V nadaljevanju tega poglavja so predstavljeni primeri metrik, ki se običajno uporabljajo.
- odpravljanje problemov: namen upravljanja problemov je zmanjšati negativne učinke nepredvidenih dogodkov. Ta del običajno določa, da mora biti na voljo primeren postopek za obravnavo in reševanje nenačrtovanih incidentov ter da morajo biti predvidene tudi preventivne dejavnosti za zmanjševanje pojavov nenačrtovanih dogodkov.
- obveznosti stranke: pomembno je tudi, da se stranka zaveda, da ima obveznosti do ponudnika storitev, da jim pomagajo pri procesu dostave storitev. V SLA je običajno definirana relacija, ki je dvosmerna entiteta. Tipično je stranka dolžna zaposlenim delavcem ponudnika storitev zagotoviti dostop, opremo in sredstva, ki jih potrebujejo za morebitno delo v prostorih stranke.
- garancije: ta del SLA pokriva kvaliteto storitev, nadomestila, terjatve tretjih oseb, pravna sredstva za kršitve, izjeme in kako ravnati v primeru višje sile.
- varnost: varnost je še posebej pomembna značilnost vseh SLA. Stranka mora zagotoviti nadzorovan fizičen in logičen dostop do svojih prostorov in informacij. Prav tako mora tudi ponudnik storitev spoštovati in upoštevati varnostne politike in postopke, ki jih stranka zahteva.
- odpravljanje večjih problemov: v tem poglavju so opisani ukrepi, ki so navedeni, če se slučajno pripetijo večje težave, z namenom, da je poslovanje čim manj moteno.
- prenehanje pogodbe: v tem poglavju so navedene naslednje teme: prekinitev po koncu začetnega obdobja, prenehanje zaradi koristi, izredne odpovedi in plačila ob prenehanju SLA.

5.3.2. Upravljanje pogodbe SLA[40]

Da zagotovimo, da se izpolnjujejo pogoji pogodbe SLA, je potrebno določiti nivoje, za katere se morajo vsi udeleženci strinjati. Iz SLA morajo biti razvidni udeleženci in njihove odgovornosti. Zelo pomembno je tudi, da udeleženci redno komunicirajo med seboj, čeprav ni problemov s sistemom. S tem, ko se aplikacija dlje časa uporablja, dobijo IT delavci, končni uporabniki in poslovni partnerji boljši pregled v smislu, kaj je res pomembno za delovanje aplikacije. Lahko se spremeni okolje, obremenitev sistema, strankina pričakovanja in potrebe ter lahko so na voljo boljše metrike, meritvena orodja in procesi. Zato je smiselno, da se dopustijo letni popravki SLA glede na novo pridobljene informacije.

Ko se oblikuje pogodba ter so nivoji identificirani in dogovorjeni, je priporočljivo, da dodamo še določbe o pogodbenih nagradah in kaznih. V večini primerov so te nagrade in kazni denarne. V vsakem primeru pa morajo udeleženci pregledati delovanje sistema vsaj enkrat letno.

Slej ko prej želimo vedeti, kako aplikacija vpliva na poslovne rezultate. Na primer, ko opazimo, da se število spletnih kupcev manjša zaradi predolgih odzivnih časov aplikacije, takrat je to vprašanje SLA, saj neposredno vpliva na prihodke. Tako je smiselno, da ponudnik storitev v času največje obremenitve poveča količino virov za izvajanje aplikacije. V vsakem primeru je upravljanje SLA nujno potrebno, da ponudnik zagotovi ustrezno kvaliteto storitev vse od nastopa pogodbe preko celotnega delovanja do končne analize učinkovitosti.

5.3.3. Metrike pogodbe SLA

Večina metrik se nanaša na kakovost dela, ki ga opravi ponudnik storitev. Kakovostna opredelitev lahko vsebuje več posameznih metrik, ki skupaj tvorijo sprejemljive kriterije ali pa služijo kot samostojne meritve posameznih vidikov storitve [15].

Primeri metrik vključujejo [15]:

- stopnjo okvar: to so procenti ali števila, ki štejejo napake večjih rezultatov, vključno s številom proizvodnih napak na mesec, število zamujenih rokov, število zavrnjenih rezultatov, itn.
- tehnično kakovost: v primeru, da razvoj aplikacije zaupamo zunanjemu izvajalcu, to vključuje meritve tehnične kakovosti programske kode. Meritve so običajno opravljene s komercialnimi orodji, s katerimi lahko preverimo velikost programa, strukturiranost programa, stopnjo zahtevnosti in napak kodiranja.

[40] Povzeto po [14].

- razpoložljivost storitev: to kaže koliko časa so bile storitve, ki jih upravlja zunanji izvajalec, na voljo. Vključevati mora metrike, ki prikazujejo dostopnost spletnih aplikacij, in tvorjenje poročil do določene ure v dnevu. Meritve lahko prikazujejo pozitivne ali negativne rezultate in običajno vsebujejo določeno stopnjo odstopanj.
- učinkovitost delovanja: učinkovitost delovanja je težje meriti kot dosegljivost in vključuje več dimenzij. Te metrike imajo običajno hitrostno dimenzijo, ki jo predstavimo z meritvami odzivnih časov, hitrostjo prenosa podatkov, itd. Količina oziroma obseg je naslednja dimenzija, ki se odraža v količini prenesenih podatkov, številu izvedenih transakcij, itd. [35]
- zadovoljstvo s storitvami: to se nanaša na zadovoljstvo stranke glede na to, kako je stranka zaznala delovanje storitve, kar zajamemo za pomembnejše funkcije preko notranjih in zunanjih anket. V idealnem primeru te ankete izvede nevtralen zunanji izvajalec. Čeprav so te meritve subjektivne, lahko služijo kot dobro preverjanje drugih metrik.

Metrike naj bodo tako izbrane, da se jih da enostavno izmeriti. Idealno je, da se meritve izvajajo avtomatično v ozadju z minimalnim vplivom na delovanje, kar pa ni možno za vse metrike. Pri gostovanih spletnih storitvah večina podjetij potrebuje vsaj 99% dosegljivost. Veliko ponudnikov nudi 99,9% ali celo še višjo stopnjo dosegljivosti. 99,9% dosegljivost pomeni, da je aplikacija nedosegljiva do 43,2 minute na mesec, kar pa za marsikatero podjetje ni sprejemljivo. Danes je že običajna praksa, da ponudnik storitev nudi skoraj 100% dosegljivost za aplikacije, ki so kritične za poslovanje stranke, pri čemer mora biti stranka pripravljena plačati za takšno garancijo [15].

5.4. Vpliv na razvijalca programske opreme

Izberi si veliko tržišče[41]

Eden od najpomembnejših dejavnikov, ki vpliva na poslovanje ponudnikov SaaS programske opreme, je velikost ciljnega tržišča. Na dolgi rok je možno doseči donosnost poslovanja samo z nizkim skupnim stroškom lastništva in zadostno količino odjemalcev. Na ta način lahko pokrijemo visoke stroške infrastrukture z večjim številom poceni naročnin, kjer posamezna naročnina prinaša majhne marže. Dobiček, ki ga lahko pričakujemo od SaaS produkta, lahko na enostaven način predstavim z naslednjo enačbo:

dobiček = količina x (naročnina – spremenljivi stroški[pridobitev stranke, podpora,...]) – nespremenljivi stroški [5]

[41] Povzeto po [5].

Na kratki rok je naš cilj, da dosežemo dobro krivuljo rasti poslovanja, pri čemer imamo pod kontrolo stroške pridobitve. Najboljši način za to je, da se osredotočimo na tiste stranke, ki imajo največje potrebe in jih je najlažje pridobiti. Torej naš cilj naj bodo majhna in srednje velika podjetja (angl. small to medium sized businesses) ter manjši, posamezni oddelki znotraj velikih podjetij. Se je pa potrebno zavedati, da je SMB raznoliko tržišče, kjer se potrebe in zahteve med različnimi segmenti lahko močno razlikujejo.

Ustvari središče dejavnosti na spletu[42]

SaaS ponudniki so po definiciji spletni založniki. Cilj tradicionalnih spletnih založnikov je povečati promet strani in ta promet pretvoriti v registrirane uporabnike. Na spletu je samo en izvor prometa spletnih strani in to so spletne povezave (angl. links): povezave od rezultatov iskanja, spletnih oglasov, drugih spletnih strani, blogov, seznamov virov, zaznamkov, itd. Če hoče ponudnik dobiti čim več prometa na svojo stran, potem mora imeti čim več kvalitetnih povezav iz različnih virov, ki so pomembni za poslovanje ponudnika. To je najcenejši in najbolj učinkovit način ustvarjanja spletnega povpraševanja.

Imamo dve skupini spletnih povezav: prve so plačljive in začasne ter druge brezplačne in stalne povezave. Več ko ima ponudnik brezplačnih povezav, manj plačljivih potrebuje. Večina SaaS ponudnikov se poslužuje plačljivega iskanja (angl. search engine marketing) kot primarnega načina za povečanje prometa spletnih strani. Poleg tega se poslužujejo tudi tehnik optimizacije iskanja (angl. search engine optimization), da izboljšajo ključne besede na njihovih straneh. Predpogoj za to je visok PageRank[43], za katerega pa potrebujemo povezave.

Pospeševanje naravne rasti[44]

Pospeševanje naravne rasti pomeni spodbujanje povpraševanja in lajšanje nakupa z razumevanjem in izkoriščanjem nakupovalnih navad potencialnih kupcev. Skrivnost je odprava ovir za nakup in odziv na zahtevane informacije, ki jih stranka želi v procesu nakupa. Tako potrebujemo opraviti določene vnaprejšnje naložbe v t.i. svobodne tržne taktike, kot so SEO, odnosi z javnostjo, blogi, socialno trženje, interaktivne predstavitve, poizkusne verzije, itd. Tako zagotovimo, da je vsebina pripravljena, ko jo stranke potrebujejo. S povečano naravno rastjo pa se zmanjšajo povprečni stroški pridobitve novih strank in poveča celotna donosnost. Skratka ideja je, da ne uporabimo klasičnega oglaševanja, marketinške propagande, prodajnih klicev, tehničnih poizvedb in ročne obdelave naročil.

Integracija kot kompetitivna prednost pred konkurenco

Bistvo SaaS revolucije je zmanjševanje skupnega stroška lastništva aplikacije. Če poenostavimo, lahko ponudnik vzame obstoječo aplikacijo, jo predela po večnajemniškem

[42] Povzeto po [5].
[43] Več na http://en.wikipedia.org/wiki/Pagerank.
[44] Povzeto po [5].

modelu, jo postavi na internet, nakopiči stranke in aplikacijo ponudi za naročnino. To je sicer odlična poslovna priložnost za stranke, ampak postavi ponudnike SaaS programske opreme na tržišče z močno konkurenco in s cenovno občutljivimi produkti. Torej, če želimo, da se naša ponudba razlikuje od ostalih produktov, moramo našo SaaS aplikacijo povezati z ostalimi produkti na omrežju, ki skupaj nudijo večjo produktivnost v temeljnih poslovnih procesih. Te priložnosti se pojavijo predvsem pri integraciji notranje-zunanjih in čisto zunanjih sistemov [5]. Transparentna integracija je ključnega pomena, da stranka dobi čim večjo dodano vrednost iz SaaS aplikacij, ampak pogosto zahteva veliko napora in časa za razvoj. To posledično pomeni, da pri tem nastane določena količina stroškov. Da se izognemo prekomernim stroškom, je potrebno raziskati potrebe za integracijo in narediti načrt. Dober načrt je predpogoj, da dodelimo ustrezno količino proračuna, da je integracija izvršena v pravem času in da so implementirane želene funkcionalnosti.

Če za našo aplikacijo vidimo priložnosti, ki sežejo po celem omrežju do naših strank, partnerjev, strank od naših partnerjev, potem smo na dobri poti, da bo naša aplikacija izstopala iz povprečja SaaS aplikacij [5].

Avtomatizacija in masovno prilagajanje aplikacij[45]

Stranke bodo vedno imele določene specifične zahteve. Odjemalec mora prilagoditi svoj arhitekturni model, da bo čim bolj ustregel zahtevam strank. Hkrati mora avtomatizirati čim več stvari in s tem čim bolj zmanjšati stroške konfiguracij, prilagajanj, sprotnega vzdrževanja in implementacij, ki so specifične za določeno stranko. To pomeni, da mora biti možno nastaviti čim več stvari, nastavitve morajo biti enostavne ter uporabiti mora odprte in standardne programske vmesnike. Del avtomatizacije bi moral biti tudi sistem za avtomatsko namestitev aplikacij, sprememb in popravkov, ki porabi minimalno virov.

Vrednost produkta za stranko[46]

Stranka mora čim prej dobiti vrednost iz produkta, po možnosti še preden produkt dejansko kupi. Stranka načeloma hoče imeti možnost preizkusiti produkt. To pomeni, da mora ponudnik imeti na voljo testno verzijo. Če stranka ne vidi neke dodane vrednosti, produkta enostavno ne bo kupila. Če že kupi produkt in ne dobi pričakovane dodane vrednosti, bo hitro zamenjala ponudnika, saj to lahko naredi, ker so stroški zamenjave programske opreme veliko manjši kot pri tradicionalni programski opremi.

Pomemben del razvojne ekipe so razvijalci, ki bodo oblikovali uporabniške vmesnike, da bodo nudili uporabnikom enostavno in učinkovito interakcijo z aplikacijo. Pri implementaciji produkta je potrebno razumeti, kdo bo ta produkt dejansko uporabljal, kakšne probleme imajo uporabniki, kako jih popraviti, kateri uporabniki so bolj pomembni. Pri vsem tem je potrebno

[45] Povzeto po [5].
[46] Povzeto po [42].

upoštevati tudi uporabnost, uporabniške izkušnje in oblikovanje. Na koncu mora biti produkt najboljše orodje za prodajo ne pa dober marketing. Vse uporabnikove izkušnje pri interakciji s produktom in celotnim podjetjem morajo biti osredotočene na to, da stranka čim prej dobi dodano vrednost.

Agilne metodologije razvoja[47]

Skrivnost uspeha s SaaS modelom je, da izdelamo majhno, enostavno, ampak kvalitetno aplikacijo, s katero pridemo hitro na tržišče. Nato uporabimo povratne informacije uporabnikov, da izboljšamo in dodelamo naš produkt. Pri takem modelu razvoja je čas med verzijami kratek. Bistvo tega modela je, da hitro pridemo na tržišče s svojim produktom in da nato produkt postopoma razvijamo. Ta proces mora vsekakor vsebovati vse vidike kakovosti, uporabnosti in uporabniške izkušnje, s katerimi zagotovimo, da naš produkt in poslovanje raste v pravi smeri. Glede na [37] je najbolj uporabljana agilna metodologija razvoja Scrum[48] (45%). Sledijo ji FDD[49] (18%), Lean software development (15%), XP[50] (9%), RUP[51] (5%) in druge (8%).

5.5. Vpliv na stranko

Za nakup programske opreme, ki je narejena po SaaS modelu, moramo imeti za to smotrn poslovni razlog. SaaS nudi ogromne možnosti za podjetja vseh velikosti, da se jim zmanjša tveganje za nakup programske opreme.

5.5.1. Upravljanje tveganja pridobitve programske opreme[52]

Tradicionalno je uvajanje velikih in za poslovanje kritičnih sistemov programske opreme velik podvig. Uvajanje sistemov, kot so ERP in CRM, lahko podjetje stane ogromno

[47] Povzeto po [42].
[48] Scrum je iterativna in inkrementalna agilna metoda za razvoj programske opreme. Scrum je osredotočen na vodenje projektov, kjer je težko vnaprej napraviti načrt. Osnovna enota razvoja je »sprint« oziroma iteracija, ki traja od enega tedna do enega meseca. Pred vsako iteracijo imamo načrtovalni sestanek in po vsakem iteraciji imamo pregled, kjer je pregledan napredek in izdelan načrt za naslednjo iteracijo.
[49] FDD (angl. feature driven development) je iterativna in inkrementalna agilna metoda za razvoj programske opreme, ki združuje splošno sprejete najboljše prakse v zaokroženo celoto. Te prakse so izpeljane iz odjemalčevega vidika funkcionalnosti. Glavni namen je pravočasno zagotoviti konkretno in delujočo programsko opremo.
[50] XP (angl. extreme programming) je agilna metoda za razvoj programske opreme, katere namen je izboljšati kvaliteto programske opreme in odzivnost na spremenljive uporabniške zahteve. Pri razvoju z metodo XP dostavimo nove verzije aplikacije v kratkih razvojnih korakih. Namen tega je večja produktivnost in postavitev kontrolnih točk, kjer lahko sprejmemo nove uporabniške zahteve.
[51] RUP (angl. rational unified process) je prilagodljivo procesno ogrodje za iterativni razvoj programske opreme. Glavni namen je, da si uporabniki lahko izberejo elemente procesa, ki zadostujejo njihovim potrebam pri razvoju programske opreme.
[52] Povzeto po [10].

finančnih sredstev, ki jih mora podjetje nameniti vnaprej za pokritje stroškov licenc. Poleg tega podjetje potrebuje IT strokovnjake in svetovalce za prilagoditev novih sistemov in integracijo z obstoječimi. Čas, osebje in proračunska sredstva predstavljajo določeno tveganje za tako velike projekte. Pogosto se zgodi, da si manjša podjetja ne morejo privoščiti nakupa takšnih sistemov.

SaaS dostavni model spremeni nekatere vidike. Če se odločimo za nakup SaaS programske opreme, ne potrebujemo kupovati strojne opreme za poganjanje aplikacij, kar bistveno zmanjša količino vnaprej potrebnih finančnih sredstev, vseeno pa je potrebno poskrbeti za učinkovito integracijo SaaS aplikacij z obstoječimi sistemi. Tako podjetju ni potrebno amortizirati začetne investicije. V primeru, da podjetje ni zadovoljno z rezultati SaaS aplikacije, lahko dano aplikacijo opustijo in poiščejo alternativno rešitev, saj so stroški zamenjave programske opreme veliko manjši kot pri tradicionalni programski opremi. Se je pa potrebno zavedati, da se ponudniki poslužujejo različnih načinov, ki otežujejo migracijo podatkov med aplikacijami. Prav tako odjemalcem ni potrebno skrbeti za infrastrukturo, ki bi jo morali pridobiti na začetku projekta.

V redkih primerih, ko integracija po meri ni potrebna, načrtovanje in uvajanje SaaS aplikacije zahteva zelo malo napora. Posledica tega je, da je čas zelo kratek, ko investicija začne vračati prihodke. To prav tako pripelje do tega, da so ponudniki SaaS programske opreme začeli ponujati testne verzije svojih aplikacij. Ponudniki nudijo testiranje svojih aplikacij za določeno časovno obdobje (npr. 30 dni). Na tak način lahko stranka močno zmanjša tveganje za nakup programske opreme.

5.5.1.1. Upravljanje IT[53]

Če smo se odločili za SaaS aplikacijo, nam kot odjemalcu ni potrebno več skrbeti za testiranje, namestitev popravkov, upravljanje posodobitev, nadzorovanje delovanja in zagotavljanje visoke dosegljivosti. Za to mora skrbeti ponudnik programske opreme. S tem, ko smo odgovornost za te aktivnosti prenesli na tretjo osebo, se lahko naš IT oddelek osredotoči na aktivnosti, ki neposredno podpirajo poslovni proces. Vodja informatike in IT osebje lahko več časa namenijo sodelovanju s poslovnimi enotami, posledično bolje spoznajo njihove potrebe in jim lažje svetujejo pri uporabi orodij za dosego ciljev.

[53] Povzeto po [10].

5.5.1.2. Vpliv SaaS modela na IT

Ko kupimo SaaS aplikacijo, se moramo pripraviti na ta prehod. Oceniti moramo, kako bo ta prehod vplival na obstoječi IT kader, in zagotoviti tekoč prehod. Prehod bo imel tudi določene posledice za IT oddelek, na katere se moramo pripraviti [10]:

- standardi varovanja podatkov: s tem, ko podatke prenesemo v oblak, tvegamo izgubo podatkov in nenamerno razkritje občutljivih informacij. Odjemalec mora določiti stopnjo varnosti, ki jo potrebuje, in zagotoviti, da ponudnik dosega standarde varnosti.
- garancije pogodbe SLA: nujno je potrebno skleniti obsežen dogovor o nivoju storitev, ki vključuje tudi najslabše možne primere.
- strategije prehoda oziroma migracije: na neki točki želimo zamenjati SaaS aplikacijo z drugo rešitvijo, zato je pomembno, da imamo možnost prenesti podatke iz obstoječe aplikacije. Vedeti moramo, kakšne strategije in procedure prenosa podatkov uporablja ponudnik.
- zahteve za integracijo notranjih aplikacij: odjemalec mora zagotoviti tudi, da pri migraciji na SaaS aplikacije uredi integracijo s sistemi, ki ostanejo znotraj podjetja.
- storitve za poročanje (angl. reporting services): s prehodom na SaaS se odjemalec odreče direktni kontroli podatkov, zato so natančna poročila zelo pomembna. Ugotoviti mora, kakšne storitve za poročanje nudi ponudnik in če so skladna z zahtevami poslovne informatike.

Vpliv na vlogo in odgovornosti IT oddelka

S tem, ko podjetje kupi SaaS aplikacijo, se spremeni vloga IT oddelka. Odpor do SaaS aplikacij lahko pride prav iz IT oddelka, saj so v preteklosti imeli vodje informatike veliko besedo pri tem, katere aplikacije bodo imeli. Uspešen vodja informatike sodeluje s poslovnimi enotami, jih pouči o vplivu nakupov programske opreme ter sodeluje z njimi, da lahko ugotovi, ali so njihove potrebe bolje zadoščene s SaaS ali s tradicionalno aplikacijo [10].

Vpliv zakonskih predpisov

SaaS model je čedalje bolj popularen ter ima pomemben vpliv na varnost podatkov in predpise o skladnosti. Zato je logično, da je večino podjetij zaskrbljenih glede pravnih in varnostnih zadev, ko se podatki nahajajo zunaj podjetja. Primeren začetek je, da od SaaS ponudnika pridobimo poročilo SAS 70 (angl. statement on accounting standards number 70) [16]. SAS 70 je revizijsko poročilo, v katerem so zabeležene notranje kontrole ponudnika storitev. Z natančnim pregledom poročila lahko odjemalec ugotovi, ali so notranji standardi ponudnika storitev skladni z zahtevami odjemalca.

5.5.2. Stroški aplikacije[54]

Preden se odločimo kupiti SaaS programsko opremo, se je potrebno prepričati, da je nakup smotrn. V ta namen je dobro izvesti analizo TCO (angl. total cost of ownership).

5.5.2.1. Glavni stroški

Tradicionalne aplikacije

Glavne stroške povzročajo strojna oprema, licence za aplikacije, omrežna infrastruktura, orodja za nadzor in testiranje, varnostni produkti ter prostori, ki jih potrebujemo za nemoteno poganjanje tradicionalnih aplikacij. Te izdatke je vedno potrebno plačati vnaprej.

SaaS aplikacije

Glavni stroški SaaS aplikacij so naročnine, ki jih odjemalci plačujejo, dokler uporabljajo storitve. Naročnine običajno vključujejo vzdrževanje, podporo, usposabljanje in nadgradnje ter vse stroške, ki so povezani z infrastrukturo za poganjanje SaaS aplikacije: strojna in omrežna oprema, podatkovne shrambe, podatkovne baze, administracija, itd. K glavnim stroškom doprinesejo svoj delež omrežna infrastruktura, ki je potrebna za dostop do aplikacij, ter osebni računalniki in druge naprave, ki jih uporabniki uporabljajo za dostop do aplikacij.

5.5.2.2. Stroški implementacije in namestitve

Tradicionalne aplikacije

Implementacija tradicionalne opreme tipično vključuje analizo tematike, načrtovanje, implementacijo, integracijo, testiranje, prilagoditve in namestitev rešitve. Ta opravila vzamejo določen čas, tekom katerega nastanejo stroški dela, ki predstavljajo pomemben del v celoti. Da zagotovimo ustrezen nivo delovanja, je potrebno tekom procesa implementacije preveriti kapacitete strežnikov ter skladnost operacijskih sistemov, aplikacij in strežnikov z računalniki, ki jih uporabljajo končni uporabniki. Po končani implementaciji aplikacije je potrebno še usposabljanje končnih uporabnikov in IT kadra.

SaaS aplikacije

Večino SaaS aplikacij je možno namestiti ter dati v produkcijo hitreje in ceneje kot primerljivo tradicionalno aplikacijo. Slaba stran SaaS aplikacije je, da je stranka bolj omejena pri nastavitvah aplikacije glede na morebitne specifične potrebe, saj so SaaS aplikacije narejene po večnajemniškem modelu. Težavo predstavlja tudi integracija SaaS aplikacij z

[54] Poglavje je povzeto po [4].

lokalnimi sistemi, ki so še vedno nujni za poslovanje. Problem integracije je na vseh nivojih – podatkovnem, aplikacijskem, nivoju poslovnih procesov in uporabniških vmesnikov.

5.5.2.3. Sprotni stroški infrastrukture

Tradicionalne aplikacije

V primeru nepredvidenih težav mora podjetje diagnosticirati problem in se v čim krajšem možnem času odzvati, za kar so potrebna orodja za nadzor in upravljanje. Ta orodja služijo tudi za razne meritve, katerih namen je ugotoviti, kdaj je potrebno povečati kapacitete. Pri izračunu moramo upoštevati tudi stroške letnega vzdrževanja in podpore, redundantnih sistemov, popravil in zamenjav strojne opreme, porabe električne energije, itd. Ti stroški so razporejeni čez celoten življenjski cikel sistema, vseeno pa jih moramo upoštevati v naši analizi.

SaaS aplikacije

Za rast SaaS aplikacije rabimo od infrastrukture zagotoviti samo zadostno internetno pasovno širino. Do SaaS aplikacije se običajno dostopa s spletnim brskalnikom, včasih pa je potrebno končnim uporabnikom namestiti namizno aplikacijo, katere naloga je komunikacija s SaaS aplikacijo. Poleg tega je potreben tudi razvoj programskega vmesnika za integracijo z obstoječimi aplikacijami.

5.5.2.4. Stroški sprotnih operacij, usposabljanja in podpore

Tradicionalne aplikacije

V primeru tradicionalnega modela mora podjetje samo priskrbeti ustrezen kader, ki bo izvrševal nadzor, podporo, vzdrževanje ter namestitev popravkov in nadgradenj. Poleg tega mora podjetje poskrbeti za usposabljanje novih zaposlenih.

Za uspešno osvojitev ter uporabo aplikacije sta zelo pomembna podpora in uvajanje uporabnikov. Ponudnik včasih nudi uvajanje kot del začetnih stroškov, bolj pogosto pa je to naloga notranjih oddelkov. Podpora mora v čim krajšem možnem času odpraviti probleme, ki jih imajo končni uporabniki z aplikacijo, saj v nasprotnem primeru to lahko pripelje do manjše produktivnosti ali celo do odpora do uporabe aplikacije. Število uporabniških zahtev in želj se veča z uporabo aplikacije, kar pomeni, da se morajo sproti večati tudi viri za podporo. Če je aplikacija namenjena tudi zunanji uporabi, stroški podpore še toliko hitreje rastejo.

SaaS aplikacije

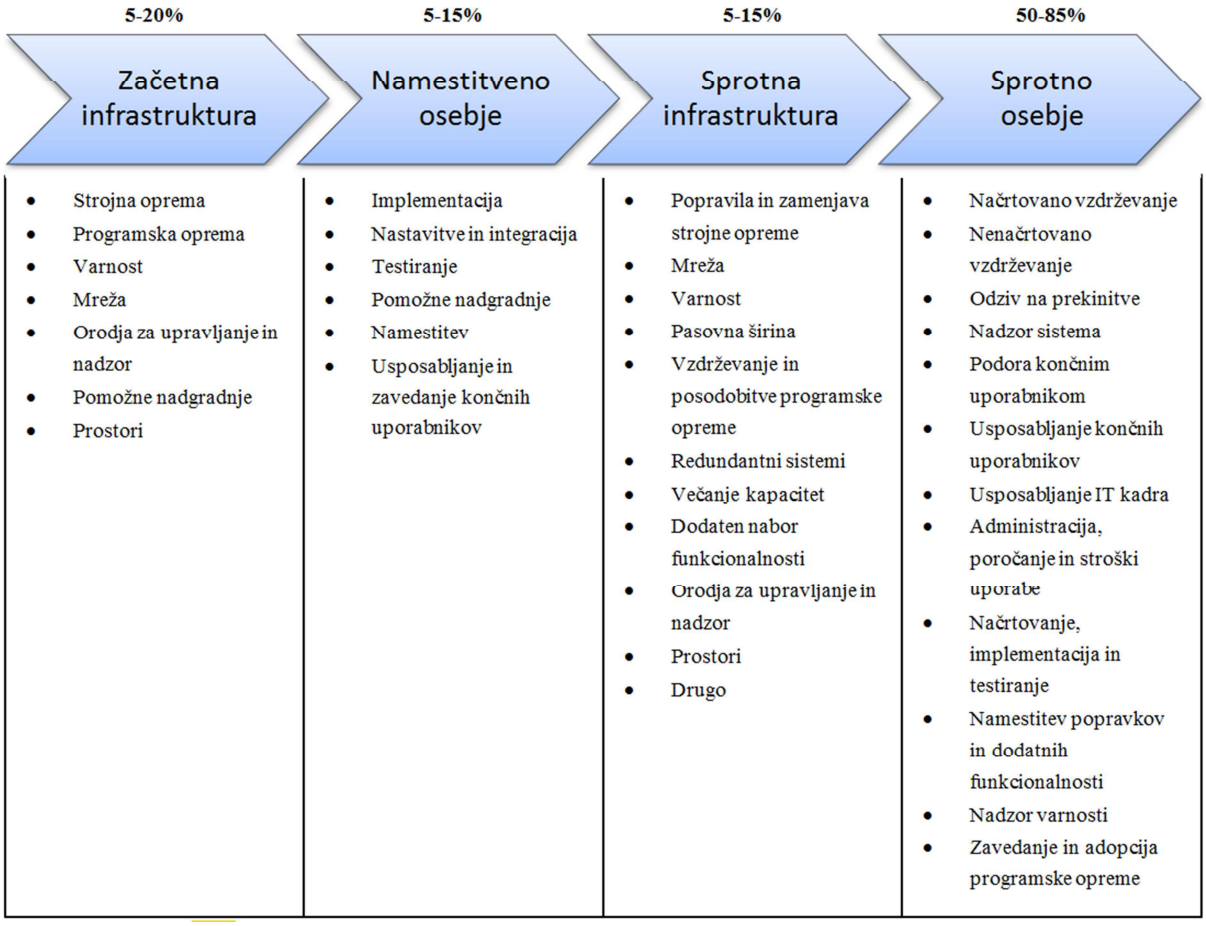
V primeru SaaS aplikacij so ponudniki tisti, ki v celoti skrbijo za aplikacijo. Stranke morajo samo zagotoviti, da imajo odprta vrata na požarnem zidu in da imajo zadostno pasovno širino, ki končnim uporabnikom omogoča nemoteno uporabo aplikacije.

Za SaaS ponudnika se prodaja aplikacije ne konča s podpisom pogodbe, saj se lahko zgodi, da stranka ne podaljša pogodbe, ko le-ta poteče. To pomeni, da je v interesu ponudnika, da uporabniki sprejmejo in uporabljajo aplikacijo. Zaradi tega razloga ponudniki aplikacijo implementirajo na način, da je enostavna za uporabo, ter nudijo začetno in sprotno usposabljanje, ki je običajno že vključeno v naročnini.

5.5.2.5. Nematerialni stroški

Nematerialne stroške je težje izmeriti in jih je zato težje vključiti v analizo TCO. Nekateri od faktorjev, ki vplivajo na nematerialne stroške, so naslednji:

- zanesljivost in dosegljivost: nedosegljivost in počasno delovanje aplikacije pomenita izgubljen čas in odpor zaposlenih do uporabe aplikacije. Kakšne nivoje delovanja aplikacije se je ponudnik zavezal izpolnjevati, ko je bila sklenjena SLA pogodba?
- povezljivost: kako enostavna je integracija z drugimi aplikacijami?
- nastavljivost: koliko napora je potrebno za nastavitve aplikacije, da se nastavi po potrebah stranke?
- varnost: stroški varnostnega vdora so lahko ogromni, če je napadalcu uspelo pridobiti zaupne informacije ali če so te informacije dostopne konkurenci. Kakšne so varnostne politike ponudnika programske opreme? Ali so skladne z notranjimi varnostnimi politikami odjemalca?
- razširljivost: ko potrebe uporabnikov rastejo, jim mora sistem slediti. Koliko je ponudnik prilagodil aplikacijo za rast uporabe in kakšni stroški pri tem nastanejo?
- kapacitete: uporabo aplikacij v podjetju je težko predvideti, kar pomeni, da je tudi načrtovanje kapacitet težavno. Posledice so lahko počasno delovanje na eni strani in neizrabljena infrastruktura na drugi. Pri SaaS aplikaciji je to lažje upravljati kot pa pri notranji tradicionalni aplikaciji.
- drugi stroški: človeški viri in glavni izdatki, ki jih porabimo za implementacijo notranje aplikacije, nam zmanjkajo pri drugih projektih ali pa odložijo nakup drugih produktov in storitev, ki so prav tako pomembni za delo zaposlenih.



Slika 26: Razporeditev stroškov tradicionalne programske opreme [19]

5.5.2.6. Rezultati analize TCO

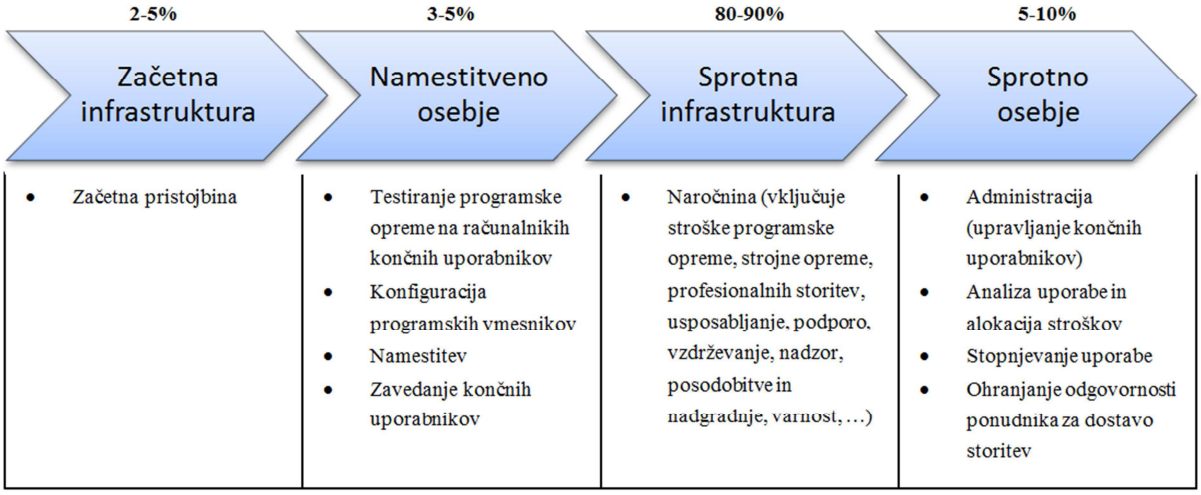
Tradicionalne aplikacije

Glede na [4] največji del stroškov tradicionalnih aplikacij predstavljajo stroški osebja, ki ga potrebujemo za nadzor, vzdrževanje in nadgradnje aplikacije ter za usposabljanje in podporo končnim uporabnikom. Napačna ocena teh stroškov lahko posledično močno vpliva na poslovanje odjemalca. Na sliki 26 so prikazani sklopi stroškov za tradicionalno okolje, ki smo jih predhodno opisali. Pri vsakem sklopu so navedena pripadajoča opravila in razmerja stroškov med posameznimi sklopi. Razmerja stroškov so navedena v razponu od-do, ker stroški niso enako porazdeljeni v vseh podjetjih.

SaaS aplikacije

Glede na [4] največji del stroškov SaaS aplikacij predstavlja naročnina, ki vsebuje stroške nadzora, vzdrževanja in nadgradenj aplikacije ter usposabljanja in podpore končnim uporabnikom. Poleg tega je iz razmerij stroškov na sliki 27 razvidno, da so začetni stroški in stroški namestitve opazno manjši kot pri tradicionalnih aplikacijah. Poleg tega je navedeno

veliko manj opravil, ker jih je mnogo vključenih v naročnini. Glede na to, da glavnino stroškov predstavlja naročnina, je možno določiti celoten strošek aplikacije razmeroma natančno.



Slika 27: Razporeditev stroškov SaaS programske opreme [19]

6. Implementacija vzorčne večnajemniške aplikacije

Za praktičen del diplomske naloge smo se odločili implementirati vzorčno večnajemniško aplikacijo. Pri načrtovanju podatkovne baze smo uporabili pristop skupnih shem, ker zagotavlja gostovanje največjega števila odjemalcev. Posledično to pomeni, da je strojna oprema veliko bolj učinkoviti izrabljena. Pri tem pristopu je pomembno zagotoviti izolacijo podatkov, saj se v istih tabelah nahajajo podatki različnih odjemalcev. V ta namen smo uporabili poglede za filtriranje podatkov odjemalcev ter zaščiten dostop do tabel in pogledov. Aplikacija ima spletni vmesnik in se izvaja na lokalnem računalniku.

6.1. Priprava delovnega okolja

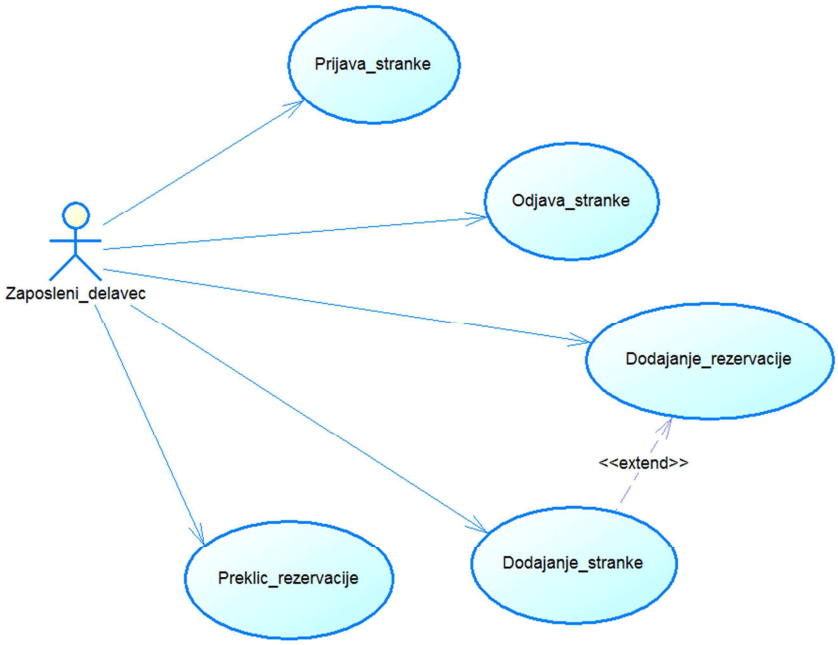
V diplomski nalogi je kot primer navedenih nekaj produktov podjetja Microsoft, ki smo jih uporabili pri implementaciji vzorčne večnajemniške aplikacije. Odločili smo se, da uporabimo okolje .NET 4.0 in programski jezik C#. Posledično smo morali namestiti ustrezno razvojno okolje in knjižnice. Pri razvoju smo uporabili sistem za upravljanje podatkovnih baz Microsoft SQL Server 2008 R2, razvojno okolje Visual Studio 2008 ter knjižnico za upravljanje z identitetami, ki vsebuje Windows Identity Foundation runtime package (WIF) in Windows Identity Foundation software development kit (WIF SDK). Pri testiranju smo uporabljali ASP.NET Development Server.

6.2. Problemska domena

Praktičen del diplomske naloge obsega enostavno aplikacijo za vodenje rezervacij in prijav strank v hotelu. Na sliki 28 je prikazan diagram primerov uporabe. Aplikacija vsebuje seznam obstoječih in nekdanjih strank ter omogoča vnos novih strank. Za vsako stranko, ki želi rezervirati prostore, vodimo naslednje informacije: naziv, naslov, pošto, številko kreditne kartice in vrsto kreditne kartice. Vsaka stranka ima lahko poljubno število rezervacij, ki jih je možno opraviti največ za leto vnaprej. Poleg tega dovolimo rezervirati termin do največ 30 dni in ne dovolimo vnašanja rezervacij za nazaj. Predpostavili smo, da z rezervacijo ne rezerviramo točno določene sobe ampak tip sobe. Tip sobe določa število postelj oziroma število oseb, ki jih je lahko nastanjenih v sobi. Z eno rezervacijo lahko rezerviramo večje število sob, ki lahko vsebujejo različno število postelj. Tako za vsak tip sobe vodimo še število rezerviranih sob. V splošnem bi lahko šifrant tipov sob razširili z dodatnimi lastnostmi

sobe. V primeru sobe z dvema ločenima posteljama in sobe s pogradom bi lahko vodili kot ločena tipa sob.

Prijava stranke v hotel je možna samo v primeru predhodne rezervacije. Na podlagi rezervacije mora zaposleni delavec stranki dodeliti ustrezne sobe. Pri tem ga aplikacija omejuje, da ne more dodeliti več sob ali sob z več posteljami, kot je bilo določeno z rezervacijo. Odjavo stranke je možno opraviti samo v primeru predhodne prijave.

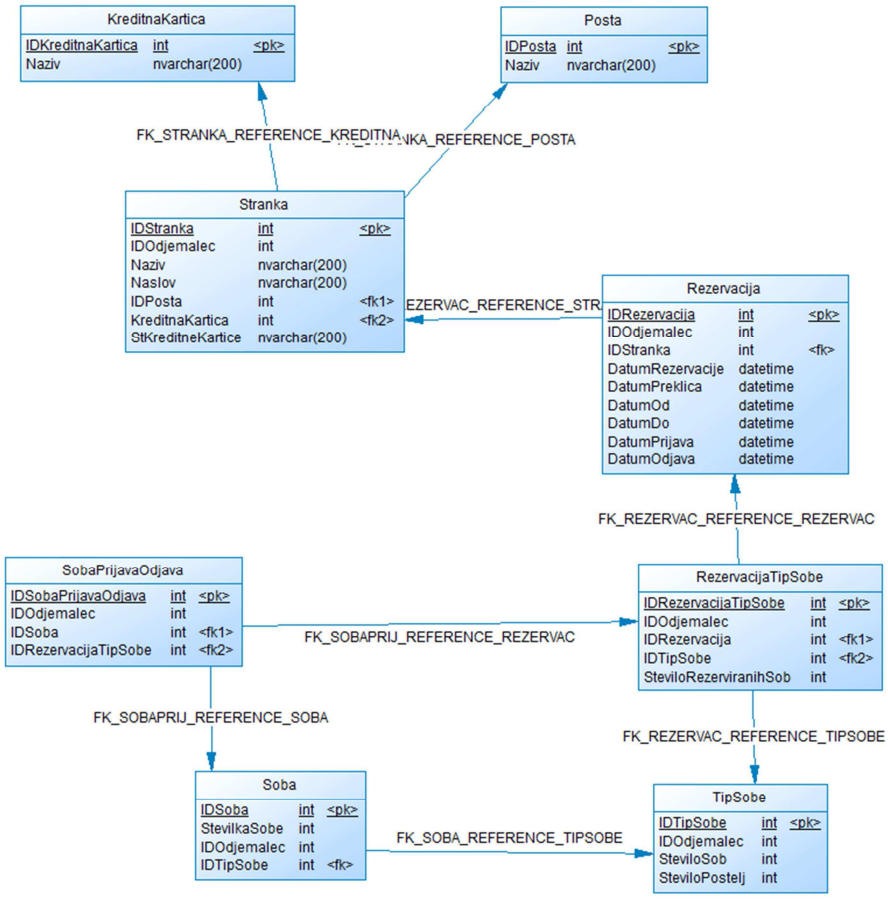


Slika 28: Primeri uporabe vzorčne večnajemniške aplikacije

6.3. Razvoj aplikacije

6.3.1. Podatkovni model

Implementacijo aplikacije smo začeli z načrtovanjem podatkovnega modela. Na sliki 29 je prikazan fizični podatkovni model, ki smo ga naredili z orodjem Power Designer. Podatkovni model smo tekom načrtovanja večkrat preverili, če ustreza vsem primerom uporabe. Pri tem smo upoštevali, da bo našo aplikacijo uporabljalo večje število odjemalcev, zato ima večina tabel dodatno kolono IDOdjemalec. Edini tabeli, ki sta skupni vsem odjemalcem sta šifranta pošt in vrst kreditnih kartic. To je manjša poenostavitev, ampak v splošnem primeru bi morali tudi tema tabelama dodati kolono IDOdjemalec, saj se lahko zgodi, da odjemalci aplikacije sprejemajo različne vrste kreditnih kartic. Poleg tega uporaba šifranta pošt v splošnem ni smiselna, saj imamo lahko stranke iz celega sveta in vzdrževanje šifranta pošt vseh držav ni primerno.



Slika 29: Podatkovni model vzorčne večnajemniške aplikacije

Pred kreiranjem objektov v podatkovni bazi smo kreirali uporabniške račune. Za vsakega odjemalca smo kreirali en uporabniški račun na operacijskem sistemu Windows in en uporabniški račun na sistemu za upravljanje podatkovne baze. Za način avtentikacije uporabniških računov podatkovne baze smo določili Windows avtentikacijo.

Na podatkovnem modelu niso prikazani pogledi, ki smo jih kreirali. Za vsako tabelo s kolono IDOdjemalec smo kreirali svoj pogled, ki vsebuje vse kolone, kot jih vsebuje tabela, ampak vrne samo podatke prijavljenega uporabnika. Naslednji primer kreira pogled StrankaView, ki prikaže samo stranke, ki pripadajo posameznemu odjemalcu:

```
create view [dbo].[StrankaView]
as
select * from Stranka
where IDOdjemalec = SUSER_ID()
go
```

Izvorna koda 3: Pogled za filtriranje seznama strank

Uporabniški računi podatkovne baze privzeto nimajo omogočenega dostopa do objektov, zato je bilo potrebno dostop omogočiti. Vsem uporabnikom smo omogočili dostop do pogledov, medtem ko smo omogočili samo bralni dostop do šifrantov pošt in vrst kreditnih kartic.

```
grant select, insert, update, delete on StrankaView to podjetjeA
```

Izvorna koda 4: Primer dodeljevanja pravic

6.3.2. Aplikacija

Uporabili smo lokalno avtentikacijo uporabnikov. S tem načinom smo omogočili dostop do aplikacije samo uporabnikom, ki imajo kreirane uporabniške račune v operacijskem sistemu. Na sistemu za upravljanje podatkovnih baz smo kreirali samo en uporabniški račun za vse uporabnike istega odjemalca, zato je bilo potrebno uporabiti impersonacijo. Impersonacija je zmožnost sistema, da se za kontrolo dostopa uporabi drugo identiteto od obstoječe, s katero se je uporabnik prijavil v aplikacijo.

```
<system.web>
  <authentication mode="Windows" />
  <identity impersonate="true" userName="podjetjeA" password="podjetjeAGeslo" />
  <!– ... -->
</system.web>
```

Izvorna koda 5: Avtentikacija uporabnikov in impersonacija[55]

Za potrebe avtentikacije smo v konfiguracijsko datoteko dodali vnose, ki so navedeni v izvlečku izvorne kode 6. Za preverjanje HTTP zahtev smo uporabili razred, ki ga doda Visual Studio – SampleRequestValidator. WSFederationAuthenticationModul omogoča razvijalcem implementacijo aplikacij na osnovi trditev. ClaimsPrincipalHttpModule zagotovi, da sta upravnik varnosti (angl. principal) in pripadajoča identiteta osnovana na trditvah. Ko STS izda žeton za uporabnika, SessionAuthenticationModul izda novi varnostni žeton in ga shrani v piškotek (angl. cookie). V naslednjih zahtevah ta modul prestreže piškotek in iz njega rekonstruira uporabnikovega upravnika varnosti.

```
<system.web>
  <httpRuntime requestValidationType="SampleRequestValidator" />
  <httpModules>
    <add name="ClaimsPrincipalHttpModule"
type="Microsoft.IdentityModel.Web.ClaimsPrincipalHttpModule, Microsoft.IdentityModel,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
    <add name="WSFederationAuthenticationModule"
type="Microsoft.IdentityModel.Web.WSFederationAuthenticationModule,
Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
/>
    <add name="SessionAuthenticationModule"
type="Microsoft.IdentityModel.Web.SessionAuthenticationModule, Microsoft.IdentityModel,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
  </httpModules>
  <!– ... -->
</system.web>
```

[55] Zapis nekodiranega uporabniškega imena in gesla v konfiguracijsko datoteko ni primeren pristop za produkcijsko okolje. Za ta pristop smo se odločili zaradi potreb testiranja. Boljši način je zapis kodiranega uporabniškega imena in gesla v register. Več na: http://msdn.microsoft.com/en-us/library/72wdk8cc(v=vs.71).aspx.

```
<system.webServer>
  <modules runAllManagedModulesForAllRequests="true">
    <add name="ClaimsPrincipalHttpModule"
type="Microsoft.IdentityModel.Web.ClaimsPrincipalHttpModule, Microsoft.IdentityModel,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
preCondition="managedHandler" />
    <add name="WSFederationAuthenticationModule"
type="Microsoft.IdentityModel.Web.WSFederationAuthenticationModule,
Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
preCondition="managedHandler" />
    <add name="SessionAuthenticationModule"
type="Microsoft.IdentityModel.Web.SessionAuthenticationModule, Microsoft.IdentityModel,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
preCondition="managedHandler" />
  </modules>
</system.webServer>
```

Izvorna koda 6: Izvleček iz konfiguracijske datoteke spletne aplikacije

Zaradi impersonacije je bilo potrebno ustrezno sestaviti niz za povezavo na podatkovno bazo. Za sestavo niza smo uporabili razred SqlConnectionStringBuilder in niz shranili v konfiguracijsko datoteko, da lahko do niza dostopajo tudi drugi objekti.

```
protected void Page_Load(object sender, EventArgs e)
{
    /* ... */
    SqlConnectionStringBuilder conBuilder = new SqlConnectionStringBuilder();
    conBuilder.DataSource = "?";
    conBuilder.InitialCatalog = "?";
    conBuilder.IntegratedSecurity = true;
    updateConfigFile(conBuilder.ConnectionString);
}
private void updateConfigFile(string connectionString)
{
    XmlDocument XmlDoc = new XmlDocument();
    XmlDoc.Load(AppDomain.CurrentDomain.SetupInformation.ConfigurationFile);
    foreach (XmlElement xElement in XmlDoc.DocumentElement as XmlElement)
    {
        if (xElement.Name == "connectionStrings")
        {
            xElement.FirstChild.Attributes[2].Value = connectionString;
            break;
        }
    }
    XmlDoc.Save(AppDomain.CurrentDomain.SetupInformation.ConfigurationFile);
}
```

Izvorna koda 7: Sestavljanje niza za povezavo na podatkovno bazo

Za potrebe avtorizacije lahko uporabimo knjižnico WIF, ki nudi dva vmesnika IClaimsPrincipal in IClaimsIdentity. Pri avtentikaciji uporabnikov operacijski sistem tvori žeton, ki vsebuje trditve z uporabniškim imenom ter pripadajočimi uporabniškimi skupinami, ki jih lahko uporabimo pri avtorizaciji. Za preverjanje avtentikacije smo uporabili naslednjo programsko kodo:

```
IClaimsPrincipal claimsPrincipal = Page.User as IClaimsPrincipal;
IClaimsIdentity claimsIdentity = (IClaimsIdentity)claimsPrincipal.Identity;
string userName = claimsIdentity.Name;
if (claimsIdentity.IsAuthenticated)
{
    /* ... */
}
```

Izvorna koda 8: Uporabniško ime in preverjanje avtentikacije prijavljenega uporabnika

6.3.3. Primer izpisa – seznam strank

Za prikaz konkretnega primera uporabe aplikacije smo izbrali seznam strank. Slika 30 prikazuje primer seznama strank za enega od odjemalcev. Nato smo se na bazo prijavili kot sistemski administrator in izbrali prvih 200 vrstic. Kot lahko vidimo na sliki 31, se v bazi nahajajo tudi zapisi drugih odjemalcev, ki jih aplikacija ne prikaže.

Slika 30: Pregled strank v aplikaciji

IDStranka	IDOdjemalec	Naziv	Naslov	IDPosta	KreditnaKartica	StKreditneKartice
1	263	Janez Novak	Slovenska cesta 111	1000	1	12345123451234512345
2	263	Mojca Novak	Slovenska cesta 111	1000	1	67890678906789067890
3	261	Dejan Župančič	Tržaška 123	1000	3	12121212121212121212
4	265	Miha Kovač	Slovenska cesta 333	2000	2	12312312312312312312312
5	265	Marija Kovač	Slovenska cesta 333	2000	2	45645645645645645645645

Slika 31: Šifrant strank v podatkovni bazi

Za branje podatkov iz podatkovne baze smo implementirali ločene razrede, v katerih so poizvedbe za vračanje (angl. select), vstavljanje (angl. insert) in posodobitve (angl. update) podatkov. V nadaljevanju je naveden primer programske kode, ki vrne seznam strank. Na podoben način smo realizirali tudi ostale poizvedbe v tem in ostalih razredih.

```
public class StrankaDB
{
    private string connectionString;
    private SqlDataAdapter da;
    private DataSet dsStranke;

    public StrankaDB()
    {
        ConfigurationManager.RefreshSection("connectionStrings");
        this.connectionString =
ConfigurationManager.ConnectionStrings["DBConnectionString"].ToString();
        this.dsStranke = new DataSet("StrankeDataSet");
    }

    /* ...  */

    public DataTable PridobiSeznamStrank()
    {
        String commandString = " select IDStranka, Naziv, Naslov, PostaNaziv,
StKreditneKartice, KreditnaKarticaNaziv from SeznamStrankaView ";
        if (this.connectionString == null)
            throw new Exception("NAPAKA! Niz za povezavo na podatkovno bazo ni
inicializiran!");
        SqlConnection con = new SqlConnection(connectionString);
        DataTable dt = null;

        try
        {
            con.Open();
            this.da = new SqlDataAdapter(commandString, con);
            this.da.FillSchema(this.dsStranke, SchemaType.Source, "Stranke");
            this.da.Fill(this.dsStranke, "Stranke");
            dt = this.dsStranke.Tables["Stranke"];
        }
        catch (SqlException e)
        {
            throw new InvalidOperationException("NAPAKA!. Podatkov ni bilo možno
prebrati!", e);
        }
        finally
        {
            con.Close();
        }
        return dt;
    }
}
```

Izvorna koda 9: Izvleček razreda StrankaDB – pridobitev seznama strank

7. Zaključek

V diplomski nalogi je izpostavljenih nekaj izzivov, s katerimi se soočajo razvijalci programske opreme. Predstavili smo večnajemniško arhitekturo, ki omogoča zmanjšanje stroškov z večjo izrabo strojne opreme. Ko aplikacijo prenesemo v oblak, je potrebno izvesti integracijo z obstoječimi sistemi. Integracija je lahko zelo obsežna in lahko vzame veliko časa. Predstavili smo integracijo aplikacij, podatkov in identitet. Glavni namen integracije identitet je omogočiti enotno prijavo za uporabnike, kar zagotovimo z implementacijo rešitve na osnovi trditev in žetonov. Poleg tega smo v diplomski nalogi zajeli še arhitekturo kompozitnih aplikacij, metapodatkovne storitve ter nadzorovanje in merjenje.

Predstavljen je tudi poslovni model in kakšen vpliv ima koncept SaaS tako na odjemalca kot ponudnika storitev. Kljub vsem prednostim, ki jih SaaS ponuja, pa ni realno pričakovati, da ga bodo podjetja nemudoma uvedla v poslovanje. To v veliki meri velja predvsem za ponudnika programske opreme. V primeru, da ima ponudnik implementirane tradicionalne rešitve, ki jih uporablja zadostno število uporabnikov, bi porabil ogromno denarnih sredstev in časa za prenovo programske opreme po konceptu SaaS. Poleg tega je za tako prenovo potrebno ogromno znanja in izkušenj, ki jih razvijalci tradicionalnih aplikacij običajno nimajo. Čeprav ima ponudnik dovolj znanja, pa potrebuje zadostno velik kader, saj ne more v celoti opustiti vzdrževanja programske opreme, ki jo uporabljajo stranke. Zaradi tega je uvedba koncepta SaaS veliko lažja za novo nastala podjetja.

Z nekaterimi izzivi, ki so predstavljeni v diplomski nalogi, smo se soočili tudi pri implementaciji vzorčne aplikacije. Uspešno nam je uspelo realizirati podatkovni model po večnajemniški arhitekturi in zavarovati dostop do podatkov tako na nivoju aplikacije kot na nivoju podatkovne baze. Do vseh podatkov ima dostop le sistemski administrator podatkovne baze. Čeprav načela storitvene usmerjenosti pri vzorčni aplikaciji niso prišla do izraza, danes ta arhitektura predstavlja temelj SaaS aplikacij. SOA tako lahko pomeni naslednji korak izobraževanja.

Kazalo slik

Kazalo izvorne kode

Literatura

[1] J. Hurwitz, Cloud Computing For Dummies, Hoboken: Wiley Publishing, Inc., 2010, pogl. 2., 10., 11., 12., 17., 20.

[2] J. F. Ransome, J. W. Rittinghouse, Cloud Computing Implementation, Management and Security, New York: Taylor & Francis Group, 2010, pogl. 5.

[3] (2012) Claims-based identity for Windows. Dostopno na: http://download.microsoft.com/download/7/D/0/7D0B5166-6A8A-418A-ADDD-95EE9B046994/Claims-Based%20Identity%20for%20Windows.pdf

[4] (2012) Software-as-a-Service; A Comprehensive Look at the Total Cost of Ownership of Software Applications. Dostopno na: http://www.winnou.com/saas.pdf

[5] (2012) Software-as-a-Service Success: The Top Ten Dos and Don'ts of SaaS. Dostopno na: http://chaotic-flow.com/media/software-as-a-service-success-top-ten.pdf

[6] (2012) Cloud Computing. Dostopno na: http://en.wikipedia.org/wiki/Cloud_computing

[7] (2012) Software as a service. Dostopno na: http://en.wikipedia.org/wiki/Software_as_a_service

[8] (2012) Architecture Strategies for Catching the Long Tail. Dostopno na: http://msdn.microsoft.com/en-us/library/aa479069.aspx

[9] (2012) Multi-Tenant Data Architecture. Dostopno na: http://msdn.microsoft.com/en-us/library/aa479086.aspx

[10] (2012) Software as a Service (SaaS): An Enterprise Perspective. Dostopno na: http://msdn.microsoft.com/en-us/library/aa905332.aspx

[11] (2012) Nimsoft Launches Unified Dashboard for On-Premise, Cloud Monitoring. Dostopno na: http://www.mspmentor.net/2009/10/20/nimsoft-launches-unified-dashboard-for-on-premise-cloud-monitoring/

[12] (2012) Service-level agreement. Dostopno na: http://en.wikipedia.org/wiki/Service-level_agreement

[13] (2012) The Service Level Agreement. Dostopno na: http://www.sla-zone.co.uk/index.htm

[14] (2012) Five Key Points for Every SLA. Dostopno na: http://content.dell.com/us/en/enterprise/d/large-business/key-points-for-sla.aspx

[15] (2012) An Introduction to Service-Level Agreements (SLAs). Dostopno na: http://www.chnsourcing.com/article/Article/abc/163420070809133104.html

[16] (2012) The Rise of SaaS and Your Regulatory Risks. Dostopno na: http://www.ecommercetimes.com/story/61448.html

[17] (2012) SaaS architecture. Dostopno na: http://www.progress.com/docs/whitepapers/public/SaaS/SaaS-Architecture.pdf

[18] (2012) Cloud computing, Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS). Dostopno na: http://www.wolfframeworks.com/cloudcomputing.asp

[19] (2012) Software As A Service: Why It Matters. Dostopno na: http://www.eco-bridge.com/uploads/EB-WP-SaaS-WhyItMatters.pdf

[20] (2012) Indexing encrypted data. Dostopno na: http://blogs.msdn.com/b/raulga/archive/2006/03/11/549754.aspx

[21] (2012) Developing Data Acquisition Systems on the Mac with Virtualization Technology. Dostopno na: http://www.ni.com/white-paper/6521/en

[22] (2012) Indexing encrypted data. Dostopno na: http://blogs.msdn.com/b/raulga/archive/2006/03/11/549754.aspx

[23] A. Arsanjani, K. Holley, 100 SOA Questions Asked and Answered, Boston: Pearson Education Inc., 2011, pogl. 1

[24] (2012) Workflow application. Dostopno na: http://en.wikipedia.org/wiki/Workflow_application

[25] (2012) Business Process Model and Notation. Dostopno na: http://en.wikipedia.org/wiki/BPMN

[26] (2012) BPEL and BPMN Tutorial. Dostopno na: http://www.pnmsoft.com/resources/bpm-tutorial/bpel-and-bpmn-tutorial/

[27] (2012) Windows Workflow Foundation. Dostopno na: http://en.wikipedia.org/wiki/Windows_Workflow_Foundation

[28] (2012) Windows Workflow Overview. Dostopno na: http://msdn.microsoft.com/en-us/library/dd489465.aspx

[29] (2012) BPEL for Windows Workflow Foundation March CTP. Dostopno na: http://www.microsoft.com/en-us/download/details.aspx?id=20222

[30] (2012) Metadata. Dostopno na: http://en.wikipedia.org/wiki/Metadata

[31] S. Kumaraswamy, S. Latif, T. Mather, Cloud Security and Privacy, Sebastopol: O'Reilly Media, Inc., 2009, pogl. 5

[32] (2012) Identity management. Dostopno na: http://en.wikipedia.org/wiki/Identity_management

[33] (2012) What is Microsoft account? Dostopno na: http://windows.microsoft.com/en-US/windows-live/sign-in-what-is-microsoft-account

[34] (2012) Long tail. Dostopno na: http://en.wikipedia.org/wiki/The_Long_Tail

[35] (2012) SLA metrics. Dostopno na: http://www.networkworld.com/newsletters/nsm/2001/01083536.html

[36] (2012) Salesforce.com. Dostopno na: http://www.salesforce.com/crm/editions-pricing.jsp

[37] (2012) Silly Agility: The Myth of the SaaS Agile Product Manager. Dostopno na: http://www.softletter.com/Resources/RicksBlogonSaaS/tabid/99/PostID/13/Default.aspx

[38] (2012) Multi-Tenant Database Architecture. Dostopno na: http://iablog.sybase.com/kleisath/index.php/2009/09/multi-tenant-database-architecture-part-1/

[39] (2012) Polling (computer science). Dostopno na: http://en.wikipedia.org/wiki/Polling_(computer_science)

[40] (2012) Push technology. Dostopno na: http://en.wikipedia.org/wiki/Push_technology

[41] (2012) Publish–subscribe pattern. Dostopno na: http://en.wikipedia.org/wiki/Publish_and_subscribe

[42] (2012) Software-as-a-Service: the danger of not knowing what you don't know. Dostopno na: http://saascamp.com/blogs/mural/archive/2008/09/17/Software_2D00_as_2D00_a_2D00_Service_3A00_-the-danger-of-not-knowing-what-you-don_1920_t-know.aspx

[43] (2012) Service (systems architecture). Dostopno na: http://en.wikipedia.org/wiki/Service_(systems_architecture)

[44] (2012) Web service. Dostopno na: http://en.wikipedia.org/wiki/Web_service

[45] (2012) Application service provider. Dostopno na: http://en.wikipedia.org/wiki/Application_service_provider

[46] (2012) Scalability. Dostopno na: http://en.wikipedia.org/wiki/Scalability