Arhitekturni projekat

TravelPlan

Članovi tima:

Petar Trifunović 16955
Maša Nešić 16774

Naziv tima:

Muffin Time

Sadržaj

1. Kontekst i cilj projekta

TravelPlan je zamišljen kao web aplikacija namenjena grupama ljudi koji planiraju zajedničko putovanje/odmor. Osnovna ideja njene realizacije jeste da jedan korisnik kreira putovanje u koje dodaje druge korisnike kao svoje saputnike. Nakon dodavanja, i kreator putovanja i njegovi saputnici imaju iste privilegije za upravljanjem putovanja. Putovanjem se može upravljati dodavanjem i menjanjem različitih parametara, poput lokacije, vremenskog okvira, smeštaja, dodatnih aktivnosti i zaduženja putnika. Svim saputnicima obezbeđuje se jedinstven pogled na šemu organizacije putovanja, i lak pristup sopstvemin zaduženjima vezanim za bilo koje od putovanja u kojima korisnik učestvuje.

Cilj ovog projekta je da olakša proces grupnog planiranja i donošenja zajedničkih odluka po pitanju putovanja (ili odmora, izleta, ...), omogućavajući laku i uniformnu komunikaciju među njegovim učesnicima. Istovremeno omogućavajući svakom pojedinačnom klijentu jednostavan pregled svih sopstvenih zaduženja i parametara putovanja u kojima učestvuje.

2. Arhitekturni zahtevi

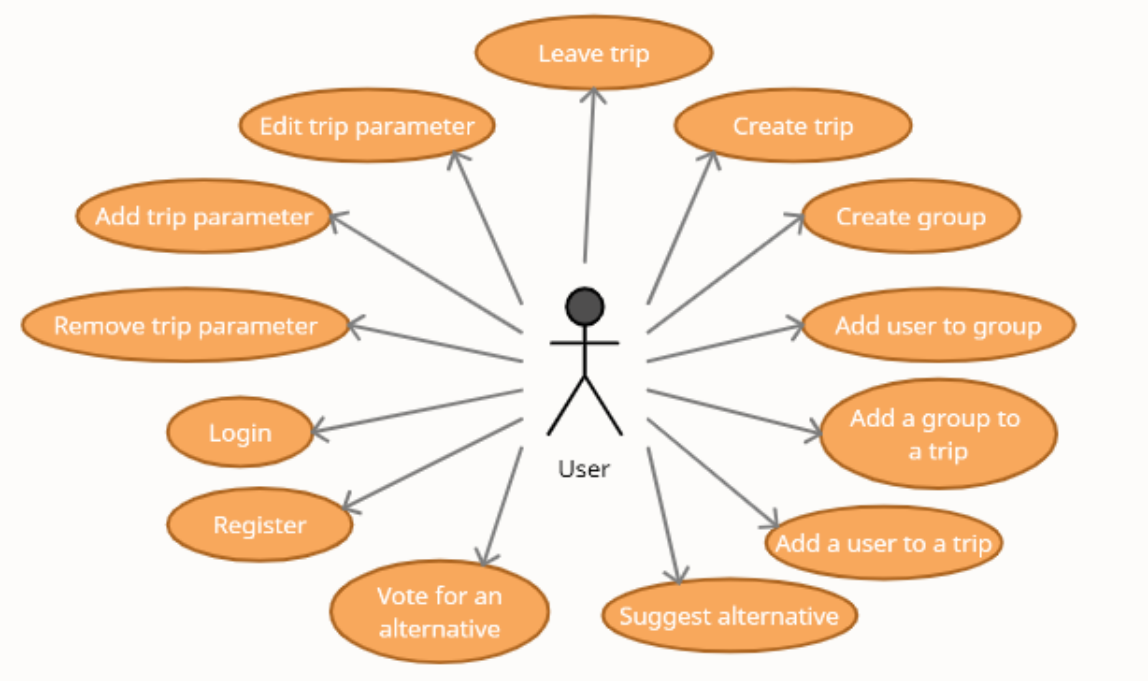
U ovom odeljku biće prikazani arhitekturni zahtevi vezani za realizaciju *TravelPlan* sistema, uključujući arhitekturno značajne slučajeve korišćenja, odnosno glavne funkcionalne zahteve, ne-funkcionalne zahteve, odnosno atribute kvaliteta i tehnička i poslovna ograničenja.

2.1. Arhitekturno značajni slučajevi korišćenja

Glavni funkcionalni zahtevi *TravelPlan* sistema su:

- Kreiranje korisničkog naloga i logovanje na isti
- Kreiranje grupa korisnika (grupisanje saputnika sa kojima korisnik često putuje)
- Kreiranje putovanja
- Dodavanje grupa ili drugih korisnika kao saputnika u putovanje
- Omogućavanje kolaboracije između saputnika na jednom putovanju
- Dodavanje i izmena parametara putovanja
- Mogućnost praćenja sopstvene liste zaduženja za sva svoja putovanja (korisnik dobija notifikaciju kada dodje do neke izmene u njegovim zaduženjima)
- Predlaganje alternative za određene parametre putovanja i mogućnost glasanja od strane saputnika za jednu alternativu. Klijenti koji su deo tog putovanja bivaju obavešteni o aktivnom glasanju
- Sistem notifikacije saputnika o promenama na posmatranim putovanjima
- Skladištenje podataka

Navedeni funkcionalni zahtevi opisani su u nastavku i Use case dijagramom sistema.



Use case dijagram sistema

2.2. Ne-funkcionalni zahtevi

Pri projektovanju i realizaciji sistema teži se ka ostvarenju sledećih ne-funkcionalnih zahteva, odnosno aributa kvaliteta:

- **Pouzdanost** – sistem mora biti pouzdan u toj meri da se omogući perzistencija svake akcije klijenta, tako da se u potencijalnom slučaju gubitka komunikacije između akcija, čitavo stanje može restaurirati na osnovu perzistiranog stanja.
- **Performanse** – sistem je interaktivne prirode, te je potrebno smanjiti vreme odziva i obezbediti što manju latenciju pri propagaciji promena načinjenih od strane jednog klijenta na kopije koje poseduju ostali aktivni klijenti. Shodno tome, potrebno je prilagoditi i propusnu moć sistema.
- **Dostupnost** – potrebno je da je aplikacija bude dostupna 24/7.
- **Modifikabilnost** – potrebno je omogućiti relativno laku promenu sistema projektovanjem komponenata koje imaju visok stepen kohezije i slabu međusobnu spregu.
- **Skalabilnost** – potrebno je da aplikacija može da podrži povećanje broja korisnika i samim tim povećanje broja zahteva za sinhronom i asinhronom komunikacijom u jedinici vremena i povećanje broja simultanih konekcija. Takođe, potrebno je obezbediti i adekvatan i efikasan rad permanentnog skladišta podataka sa povećanjem količine podataka koju skladišti.
- **Upotrebljivost** – potrebno je da aplikacija bude intuitivna i jednostavna za korišćenje.
- **Sigurnost** – potrebno je primeniti principe autentifikacije i autorizacije kako bi se omogućila kontrola pristupa sadržaju, tako da samo učesnici nekog putovanja mogu da vide njegove detalje. Pri prenosu osetljivih podataka kroz mrežu (username i password) potrebno je izvršiti njihovu enkripciju.

2.3. Tehnička i poslovna ograničenja

Zbog prirode sistema i problema koje želi da reši, nameću se određena tehnička i poslovna ograničenja:

- **Intuitivan korisnički interfejs** - *TravelPlan* sistem je po prirodi interaktivan sistem, namenjen opštem profilu korisnika, pa je potrebno obezbediti prijatan i intuitivan korisnički interfejs kako bi se omogućilo lako i jednostavno korišćenje aplikacije svakom korisniku.
- **Sinhrona i asinhrona komunikacija** - Ovaj sistem zahteva dva različita tipa komunikacije – sinhronu komunikaciju između klijentskog dela sistema i serverskog dela sistema, i asinhronu komunikaciju prilikom propagacije izmena od jednog klijenta ka ostalim, pa je neophodno da arhitektura i dizajn sistema podrže oba ova načina komunikacije.
- **Skrivenost šeme baze podataka** – Korisnicima je potrebno prikazati podatke na njima prikladan način i od njih skriti način reprezentacije datih podataka u samom skladištu. Kako bi se ovo postiglo, sama ideja se mora inkorporirati u dizajn i arhitekturu sistema i obezbediti izolacija pogleda podataka u odnosu na njihovu šemu.
- **Pristup preko web-a** – Potrebno je obezbediti mogućnost korisnicima da pristupe sistemu putem weba, te shodno tome koristiti adekvatne web tehnologije koje će omogućiti svu zahtevanu komunikaciju.

Poslovna ograničenja sistema baziraju se i na parametrima putovanja. Zavisno od tipa putovanja ograničeno je koje tipove aktivnosti ono može da sadrži. Takođe, dužina trajanja putovanja, broj različitih lokacija i broj odabranih smeštaja za jedno putovanje su međusobno uslovljeni. Ispunjenost ovih ograničenja mora se obezbediti samim dizajnom aplikacije.

3. Arhitekturni dizajn

U ovom odeljku biće prikazan arhitekturni dizajn *TravelPlan* aplikacije, u vidu arhitekturnih obrazaca koji će biti korićeni i opšteg prikaza arhitekture aplikacije. Takođe, dizajn će u ovom odeljku biti sagledan iz još dve perspektive – strukturne i bihevioralne.

3.1. Arhitekturni obrasci

Arhitektura *TravelPlan* aplikacije uspostavljena je korišćenjem sledećih arhitekturnih obrazaca:

- *Layered obrazac*

  Struktura komponenata *TravelPlan* sistema biće uređena po *Layered* arhitekturnom obrascu, podelom na tri osnovna sloja – klijentski, serverski i sloj baze podataka.

  Klijentska komponenta predstavljaće dodirnu tačku sa korisnicima, u vidu interfejsa prikazanog u Web browser-u.

  Serverska komponenta biće srednji sloj i posrednik između druga dva sloja, odvajajući klijentski deo aplikacije od dela namenjenog skladištenju podataka nastalih kao proizvod korisničkih akcija. Klijent će imati mogućnost sinhrone komunikacije sa serverom preko *RESTful API* poziva, kao i asinhrone, korišćenjem usluga *message broker*-a.
  Sama serverska komponenta biće dalje podeljena na dva specijalizovana sloja. Prvi od njih biće

namenjen poslovnoj logici aplikacije. Svrha drugog sloja biće upravljanje bazom podataka korišćenjem *Entity Framework Core ORM-a*.

Sloj baze podataka biće namenjen trajnoj perzistenciji podataka nastalih kao proizvod rada aplikacije.

- *Repository obrazac*

TravelPlan aplikacija će za perzistenciju podataka koristiti centralizovano skladište, u vidu *SQL Server baze podataka*. Kao konektor između skladišta i ostatka sistema koristiće se usluge koje pruža *Entity Framework Core ORM*. Skladište će biti pasivnog tipa, nad kojim će serverska strana sprovoditi akcije izvršavanjem transakcija.

- *Publish/subscribe obrazac*

Publish/Subscribe obrazac će u sistem biti uključen preko *message broker* komponente servera. Preko nje će se obezbediti asinhrono obaveštavanje klijentske aplikacije od strane servera o važnim događajima koji nastupaju u toku rada aplikacije. Ovakva komunikacija serverske i klijentske strane rezultovaće slanjem i prikazivanjem notifikacija aktivnim korisnicima aplikacije, ukoliko nastupi događaj za koji su zainteresovani.

Gledano iz ugla korisnika, važni događaji o kojima će biti automatski obaveštavani notifikacijama jesu promene u njihovim zaduženjima ili drugim važnim parametrima koji se tiču nekog od putovanja u kojem učestvuju.

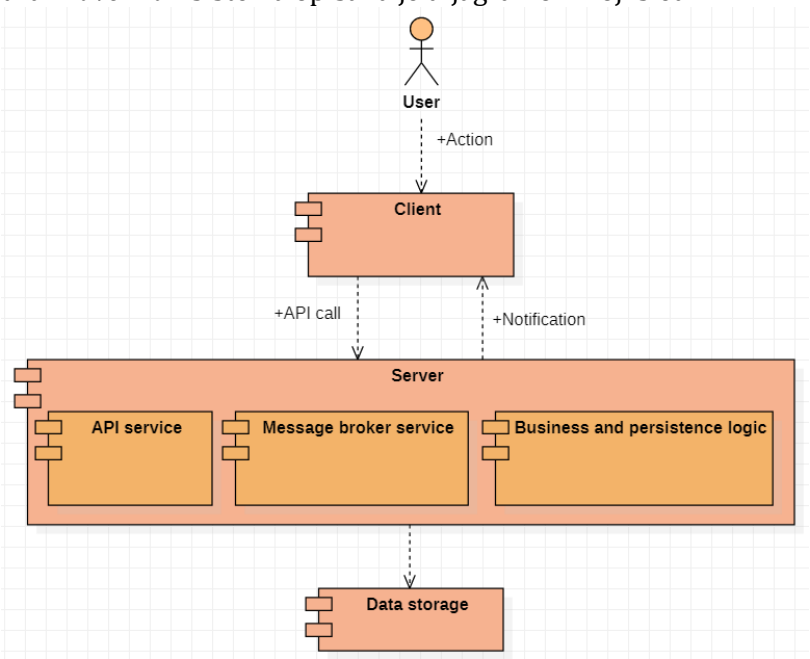
- *MVC obrazac*

Serverska strana će klijentskoj pružati usluge *RESTful API*-ja, uređenog po MVC obrascu. API će sadržati klase *controller*-a kao tačke pristupa uslugama API-ja, *model* će se kao izdvojena komponenta baviti domenskim klasama i njihovom perzistencijom, ali sama *view* komponenta ovog obrasca neće postojati u standardnom obliku, imajući u vidu da serverska strana neće direktno kreirati delove korisničkog interfejsa. Umesto toga, API će klijentskoj strani vraćati odgovor u vidu važnih podataka predstavljenih u JSON formatu.

Za interfejs će biti zadužena klijentska strana, izrađena kao „fat client“ tip u Vue.js framework-u, koji će interno imati svoje reprezentacije pogleda i modela.

3.2. Generalna arhitektura

Generalna arhitektura *TravelPlan* sistema opisana je dijagramom koji sledi.

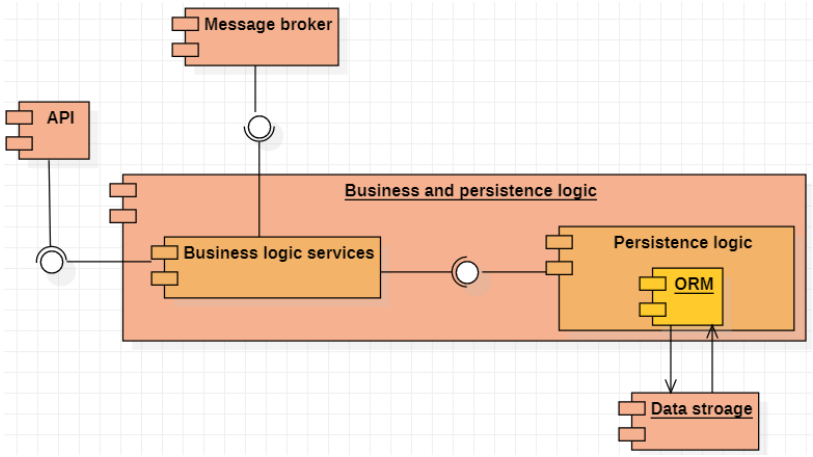


Arhitekturna skica

3.3. Strukturni pogledi

Komponentni dijagram komponente za biznis logiku i logiku perzistencije

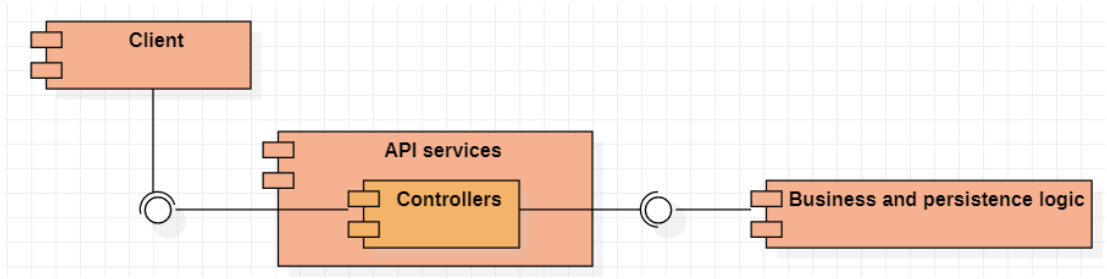
Ova komponenta imaće dva zaduženja. Prvo je sprovođenje akcija unutar same aplikacije u skladu sa biznis logikom, nametnutom domenom problema i namenom aplikacije. Drugo zaduženje je uspostavljanje dvosmerne komunikacije sa skladištem podataka putem *Entity Framework Core ORM-a*, u cilju perzistiranja generisanih podataka i pribavljanja istih kada rad aplikacije to zahteva.



Komponentni dijagram komponente za biznis i logiku perzistencije

Komponentni dijagram API komponente

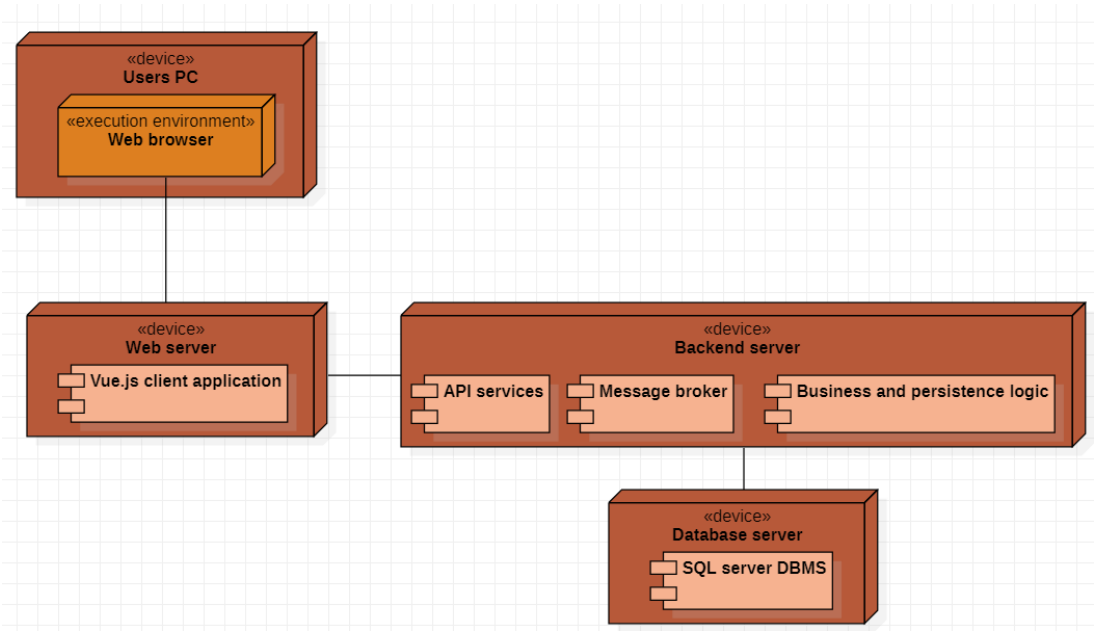
Klijentska strana ostvarivaće komunikaciju sa serverskom putem usluga Web API-ja koje server pruža. Sama API komponenta sadrži podkomponente u vidu kontrolera vezanih za pristupne tačke (endpoint-e). Klijent će pristupati ovim tačkama navođenjem odgovarajućeg URL-a u API zahtevu i čekanjem odgovora od kontrolera nakon toga.



Komponentni dijagram API komponente

Dijagram raspoređivanja

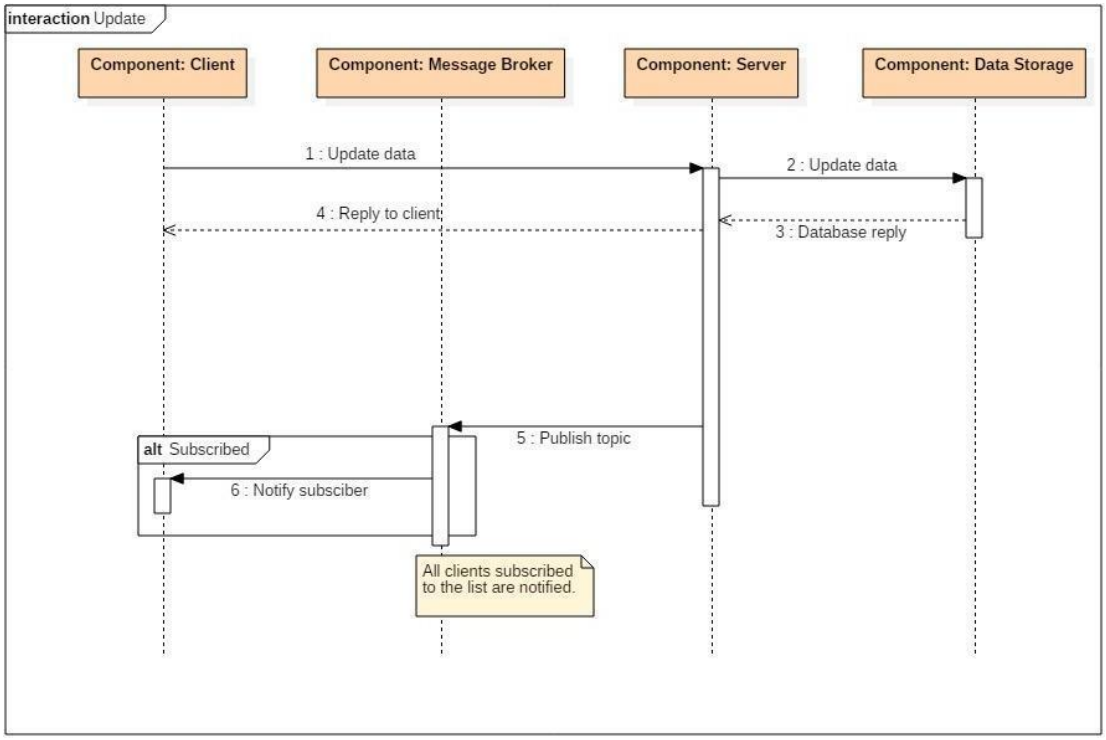
Na sledećoj slici prikazan je raspored komponenti sistema po čvorovima.



Dijagram raspoređivanja

3.4. Bihevioralni pogledi

Navedeni dijagram ilustruje bihevioralni pogled sistema na primeru operacije ažuriranja. Klijent šalje zahtev za ažuriranje serveru, koji obrađuje zahtev i komunicira sa bazom podataka po potrebi. Zatim, server ostvaruje komunikaciju sa message broker-om putem "publish" operacije. Svi klijenti koji posmatraju, odnosno koji su subscribe-ovani na dato putovanje, (nad kojim je učinjena izmena) bivaju obavešteni da je došlo do izmena.



Dijagram sekvence sistema

3.5. Implementaciona pitanja

U ovom odeljku navedene su biblioteke, komponente i okviri (frameworks) koji će biti korišćeni za implmentaciju *TravelPlan* aplikacije.

Vue.js – frejmvork za izradu klijentskih SPA web aplikacija

ASP.NET Core Web API aplikacija – Serverska aplikacija

Entity Framework – Objektno-relacioni maper (ORM framework)

SQL Server – Baza podataka

4. Analiza arhitekture

4.1. Potencijalni rizici u implementaciji i strategije prevazilaženja

Obzirom na to da je sistem projektovan tako da se oslanja na centralizovano skladište, porastom broja korisnika, količine podataka koju ti korisnici generišu i geografske razuđenosti korisnika može doći do situacije gde skladište postaje usko grlo sistema. Ukoliko se ovakav problem pojavi, može se rešiti prelaskom na distribuiranu realizaciju skladišta podataka. Može se izvršiti vertikalna i horizontalna fragmentacija podataka i podela odgovornosti upravljanja fragmentiranim podacima na više različitih čvorova. Dok se problem geografske razuđenosti može prevazići replikacijom čitavog skupa čvorova koji predstavlja celo skladište u više instanci koje će biti pozicionirane tako da obezbeđuju minimizaciju vremena odziva za sve klijente. Sa porastom količine podataka koja je smeštena u skladištu, treba razmisliti i o opciji da se zastareli podaci premeste u trajnije skladište kako bi se rasteretila glavna komponenta skladištenja kojoj se regularno pristupa.

Dodatno, obzirom na to da se koristi *third party* komponenta za potrebe *message broker*-a, može se desiti da se vremenom izgubi podrška za izabranu komponentu. U tom slučaju bilo bi neophodno zameniti tu komponentu i njoj zavisne delove implementacije klijentske i serverske komponente sistema. Kako bi se ovaj prelazak učinio što „bezbolnijim“, poželjno je u procesu projektovanja sistema minimizirati delove implementacije koji direktno zavise od odabrane *third party* komponente.