Sustav upravljanja svečanim događajima

Kadić, Ante

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

SUSTAV UPRAVLJANJA SVEČANIM DOGAĐAJIMA

Ante Kadić

Zagreb, studeni 2017.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, 11.11.2017.

Predgovor

Zahvaljujem svom mentoru Aleksandru Radovanu na ukazanom povjerenju i pruženoj pomoći tijekom izrade ovog završnog rada.

Zahvaljujem Josipi Vidić, voditeljici restorana Gastro Globus, na suradnji tijekom izrade praktičnog dijela završnog rada. Također joj zahvaljujem na korisnim savjetima i strpljenju.

Od srca zahvaljujem svojoj obitelji i prijateljima na pruženoj potpori tijekom studija.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Cilj ovog rada je poboljšati poslovnu efikasnost voditelja restorana i/ili hotela u kojemu se organiziraju svečani događaji, kongresi, vjenčanja i sl.

Naime, u restoranima gdje se organiziraju svečani događaji najbitniji je sustav praćenja rasporeda samih događaja. Taj posao može s vremenom postati poprilično kompleksan iz razloga nepredviđenih komplikacija kao što su: otkazivanje događaja, promjena datuma, dogovorenog broja ljudi, financija itd.

Praktičnim dijelom ovog završnog rada je ponuđeno poslovno rješenje u obliku web aplikacije koja bi trebala olakšati i ubrzati, ali i povećati efikasnost samih voditelja. Aplikacija mora ponuditi uvid u povijest odrađenih događaja te uvid u razne izvještaje pomoću kojih se mogu donositi bolje poslovne odluke. Jedan od modula je i uvid u efikasnost samih voditelja u obliku postotaka uspješnosti realizacije dogovorenih događaja. Isto tako korisno je imati i popis svih dosadašnjih klijenata, a najbitniji modul koji aplikacija mora ponuditi je kalendar s događajima za dodavanje, brisanje ili ažuriranje samih događaja.

Ključne riječi: hotel, svečani događaji, voditelji, web aplikacija, kalendar.

Summary

The goal of this work is to improve business efficiency of restoran managers in managing solemn events, congresses, weddings etc.

The most important thing in managing solemn events is tracking and managing events schedule. That task can become pretty complex because of unpredictable complications such as: canceling event, changing date, number of people attending, finances etc.

Thru practical part of this final work is offered business solution in form od web application which should make easier and faster, but also improve efficiency of managers.

Application needs to offer insite in event history and also in all kinds of reports which leads to better business decisions. One of modules is insite in managers efficiency in form of success percentage of event realization. It is also helpful to have list of all former clients, but the most important module is event calendar for adding, deleting and updating events.

Key words: hotel, solemn events, managers, web application, calendar.

Sadržaj

1. Uvod

Glavni motiv realizacije ovog rada dobiven je pri saznanju trenutnog načina poslovanja u organiziranju svečanih događaja. Naime, voditelji u većini imaju blok papira s nacrtanim tablicama te zapisuju „ručno“ upite klijenata te njihove rezervacije. Ako klijent s vremenom odustane od rezerviranog događaja, onda se mora brisati s papira kako bi navedeni datum ostao slobodan za nekog drugog zainteresiranog klijenta. Postoje neka bitna poslovna pitanja, na koja je na ovaj način teško ili gotovo nemoguće odgovoriti. Na primjer pitanja kao što su: koliko je bilo maturalnih večera u prošloj godini, ili koliko postoji nadolazećih vjenčanja ove, a koliko iduće godine? Koja godina je bila najprofitabilnija? Koji voditelj dogovori najviše događaja, a kod kojeg najčešće klijenti odustanu?

Zamišljena je aplikacija u kojoj svaki voditelj ima uvid u zajednički kalendar u kojem jednim klikom miša može unesti novi događaj. To bi značilo da i jednim klikom miša može izbrisati otkazani događaj. Isto tako istovremeno može više voditelja imati sastanak s različitim klijentima i unositi događaje u raspored. Unosom novog događaja svi voditelji iz istog hotela vide unesene promjene u stvarnom vremenu. To su samo neke od mogućnosti koje nudi aplikativno rješenje ovog završnog rada.

Kroz ovaj rad je detaljno opisana arhitektura sustava upravljanja svečanim događajima, odnosno praktičnog dijela ovog rada. Arhitektura je objašnjena od same baze podataka, preko servisnog sloja pa sve do same web aplikacije. Tu će se vidjeti smisao i prednosti arhitekture sastavljene od više servisa te koliko je zapravo lakše održavati aplikaciju ukoliko je ona podijeljena na više samostalnih komponenti, umjesto da je sve dio jednog velikog modula.

Zatim se čitatelja pobliže upoznaje s tehnologijom na kojoj se bazira sustav te ukratko njena povijest, kako je nastala te neke prednosti zbog kojih je ista i odabrana za realizaciju ovog sustava.

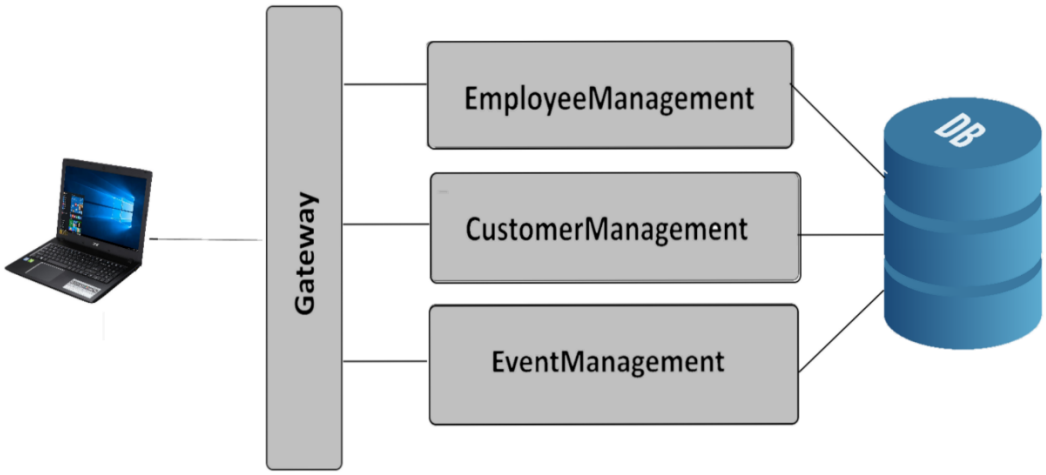
Osim same tehnologije opisan je i uzorak dizajna (engl. *Design pattern*) koji je odabran, ili kako će se kasnije vidjeti, je jedini pravi odabir za razvoj web aplikacije.

2. Arhitektura sustava

Praktični dio rada je podijeljen u tri sloja:

- Klijentska web aplikacija
- Servisni sloj kao poslovna logika
- Baza podataka

Na taj način je osigurana podjela najbitnijih dijelova sustava tako da svaki sloj izvršava svoju zadaću neovisno o drugima. Isto tako je bitno da se na taj način omogućuje lakše buduće održavanje koje može biti podijeljeno na veći broj razvojnih inženjera. Također je omogućena eventualna buduća nadogradnja sustava. Npr. jedan od potencijalnih budućih zahtjeva bi bio da se omogući mobilna aplikacija za isti sustav u smislu da se koriste isti podaci. To bi značilo da se klijent koji koristi aplikaciju na webu može ulogirati svojim podacima na mobilnu aplikaciju i imati uvid u iste podatke. To je na ovoj arhitekturi omogućeno jer se sva poslovna logika odvija na servisnom sloju, a i pristup podacima se vrši preko servisa. Arhitektura sustava je prikazana slikom (Slika 2.1.).

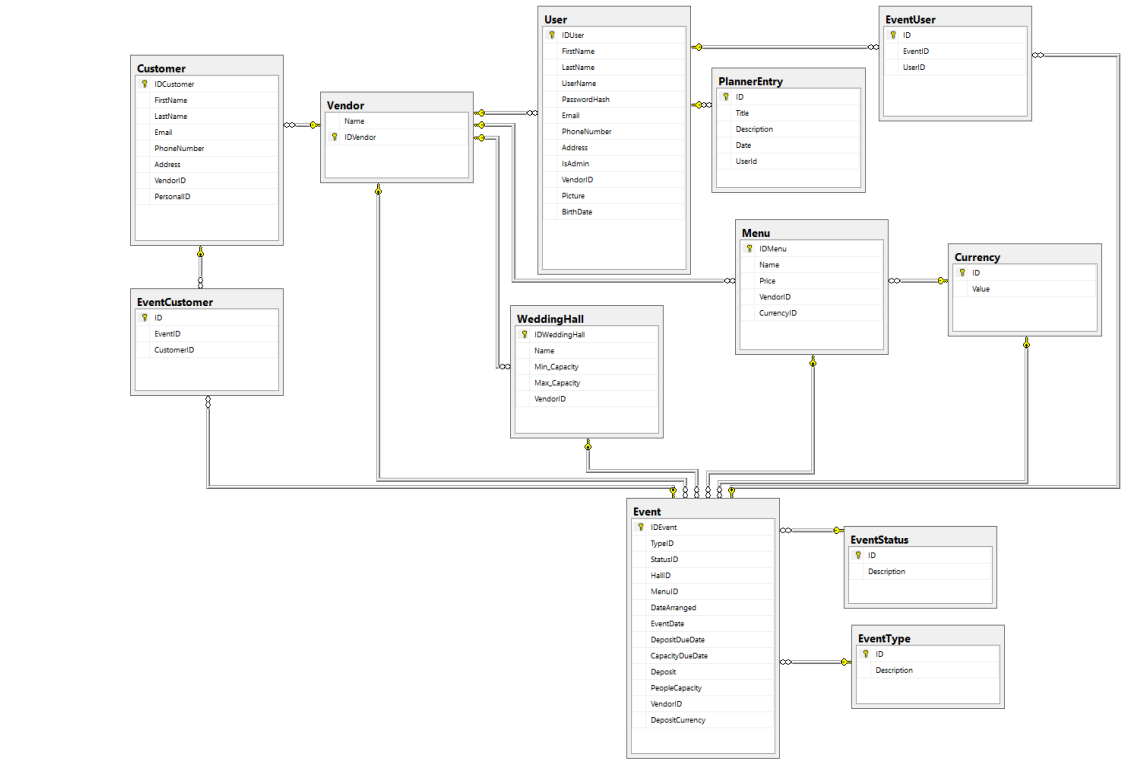


Slika 2.1. Arhitektura

Na slici (Slika 2.1) je vidljivo je da je klijentska aplikacija razdvojena od servisnog sloja i komunicira isključivo s *Gateway* servisom. Takvim razdvajanjem briga (engl. *separation of concerns*) je omogućeno da bude više različitih klijentskih aplikacija koje koriste ovaj servisni sloj. Baza podataka je skrivena od klijentske aplikacije te pristup prema njoj imaju samo tri servisa: *EmployeeManagement, CustomerManagement* i *EventManagement.*

2.1. Baza podataka

Za bazu podataka je odabran Microsoft SQL server tako da cijeli sustav zapravo leži na Microsoftovoj platformi. Na slici (Slika 2.2.) se može vidjeti dijagram baze podataka dok je na tablici (Tablica 2.1.) prikazan detaljan opis svake tablice.



Slika 2.2. Dijagram baze podataka

Tablica 2.1. Opis tablica

Tablica	Opis
Customer	Poslovni klijent. Osoba koja dogovara željeni događaj.
Vendor	Restoran koji je korisnik sustava. Jedan hotel/restoran predstavlja jednog vendora.

User	Korisnik aplikacije odnosno voditelj koji je zaposlen u jednom restoranu (vendor).
Event	Događaj koji se dogovara od strane klijenta.
WeddingHall	Dvorana odnosno sala za vjenčanje unutar restorana.
PlannerEntry	Privatni događaj jednog korisnika. Unosi se u privatni planer (sastanci, bilješke…).
Menu	Jelovnik dogovoren za događaj.
Currency	Valuta.
EventUser	Vezna tablica između događaja i korisnika. Tu se vidi koji voditelj vodi koji događaj.
EventType	Tip događaja: vjenčanje, maturalna večera, catering, kongres, proslava obljetnice, obiteljska večera.
EventStatus	Status događaja: rezervirano, kapareno, završeno, obustavljeno.
EventCustomer	Vezna tablica između događaja i klijenta.

Za pristup bazi podataka je korišten Entity Framework. To je jedan od najpoznatijih i najraširenijih ORM-ova (engl. *Object-relational Mapping*) na svijetu. Glavna zadaća svih ORM-ova je da olakšaju developerima pristup bazi podataka i to na način da rade s objektima okruženja u kojem rade. Rade na način da mapiraju objekte sustava na objekte baze podataka (tablice, poglede i procedure). Budući da se sustav upravljanja svečanim događajima bazira na .NET okruženju, logično je da je korišten i Microsoftov Entity Framework.

2.2. Servisni sloj

Na servisnom sloju se odrađuje glavna poslovna logika cijelog sustava. Radi se o REST servisima što znači da je način komuniciranja između klijentske aplikacije i servisa u REST obliku (Representational State Transfer).

REST je softverski stil arhitekture weba, odnosno set principa koji definira kako bi web standardi poput HTTP i URI-a trebali biti korišteni. Korišten je kako bi opisao željenu web arhitekturu, identificirao postojeće probleme, usporedio alternativna rješenja i osigurao da proširenje protokola neće kršiti temeljna ograničenja koja čine web uspješnim. Ideja REST-a jest da se umjesto kompleksnih mehanizama poput CORBE, RPC ili SOAP-a koristi jednostavan HTTP za uspostavljanje veze između uređaja. Sistemi koji udovoljavaju REST ograničenjima nazivaju se RESTful sistemi, a oni obično komuniciraju preko HTTP-a putem HTTP metoda.[1]

Tehnologija koja je odabrana za arhitekturu servisnog sloja je WCF (*Windows Communications Foundation*). WCF je alat za distribuirano programiranje koji je uveden s .NET-om 3.0. Korištenjem ovog alata, servis postaje dostupan vanjskim pozivima različitih tehnologija. Na primjer, jednom zatvorenom sustavu, gdje su sva računala na istom operacijskom sustavu i koja koristi različite TCP protokole radi ostvarivanja što boljih performansi, mogu pristupiti i vanjski korisnici korištenjem web servisa baziranim na XML protokolu kako bi izvršavali svoje funkcionalnosti bez obzira na programski jezik ili operacijski sustav.[2]

Servisni sloj se sastoji od četiri servisa: prolaznik (engl. *Gateway*), upravljanje korisnicima (engl. *User management*), upravljanje klijentima (engl. *Customer management*) i upravljanje događajima (engl. *Event management*). *User management, Customer management i Event management* predstavljaju tri glavna modula ovog sustava, dok je *Gateway* pristupni servis koji služi kao komunikacijska spona između klijenta i servisnog sloja. Više o pojedinom servisu je opisano u sljedećim poglavljima.

Kako je sustav podijeljen u tri glavna modula, tako postoji isto toliko repozitorija koji predstavljaju apstraktni pristup bazi podataka. No to ne znači da samo jedan servis ima

[1] Sara Kuserbajn, REST arhitektura, Veleučilište u Šibeniku, Šibenik, 2016. str. 3.
[2] Ana Jovanović, Windows Communication Foundation,Prirodno-matematički fakultet, Kragujevac, 2009.str.9

pristup jednom repozitoriju jer pripadaju istom modulu, nego svaki servis ima pristup svim repozitorijima, a samim time i bilo kojem dijelu baze podataka.

Svaki repozitorij implementira generičko sučelje (engl. *interface*) `IRepository`:

```
public interface IRepository<T>
    {
        IEnumerable<T> GetAll(int vendorId);
        T Get(int id);
        T Update(T entity);
        void Remove(int id);
        T Add(T entity);
    }
```

Kôd 2.1. Sučelje repozitorija

Implementirajući ovo sučelje svaki repozitorij mora implementirati CRUD metode (CREATE, READ, UPDATE i DELETE) za određeni tip. No sve ostale potrebne metode su dodane u sam repozitorij. Tako je npr. metoda za dohvaćanje događaja samo za određenog voditelja dodana u repozitorij događaja (engl. *Event Repository*):

```
public IEnumerable<Event> GetEventsByUser(int userId);
```

Kôd 2.2. Metoda za dohvaćanje događaja

Osim što implementiraju opisano sučelje, svi servisi nasljeđuju i baznu klasu naziva *ServisBase*. Time je postignuta sigurnost servisa od prestanka rada. Naime, ukoliko dođe do iznimke (engl. *Exception*) u nekoj od metoda, u bilo kojem servisu, taj servis će se ugasiti. Jedina zaštita od takvog scenarija je da svaka metoda bude okružena blokom koji ju štiti od toga (*try – catch blok*). U baznoj klasi se nalazi sljedeća metoda:

```
protected T ExecuteFaultHandledOperation<T>(Func<T>
codeToExecute)
{
try
{
            Tracer.Trace($"Executing {MethodName}....");
            T result = codeToExecute.Invoke();

            Tracer.Trace($"Succesfully executed
            {MethodName}");

            return result;
}
```

```
catch (FaultException ex)
{
            if (ex.Message.EndsWith("not found in database"))
                  Tracer.Trace(ex,
                  Core.Common.TraceLevel.Warning);
            else
                  Tracer.Trace(ex);

                  throw;
}
catch (Exception ex)
{
            Tracer.Trace(ex);
            throw new FaultException(ex.Message);
}
}
```

Kôd 2.3. Metoda bazne klase servisa

Ta metoda kao parametar prima metodu koja se treba izvršiti i koja kao povratnu vrijednost vraća generički tip. Ta metoda će biti proslijeđena iz samog servisa koji nasljeđuje *ServisBase.* Tako da sve metode na servisima imaju tijelo okruženo sljedećim izrazom:

```
return ExecuteFaultHandledOperation(() =>
{
});
```

C# programski jezik omogućuje zapisivanje i prosljeđivanje metoda u obliku Lambda izraza. Lambda izraz je anonimna funkcija koje se može koristiti za stvaranje delegata ili tipova stabla izraza (engl. *Expression tree types*). Koristeći lambda izraze, mogu se pisati lokalne funkcije koje mogu biti proslijeđene kao argumenti ili vraćeni kao rezultat poziva funkcije. Lambda izrazi su posebno korisni za pisanje LINQ izraza.[3]

Zadaća statičke klase `Tracer` iz kôda (`Kôd 2.3.`) je da zapisuje (engl. *log*) događaje, informacije i greške u datoteku i/ili konzolu koja predstavlja servis. Ima dvije javne metode

[3]https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions

`Trace` gdje se zove ona koja je u tom trenutku potrebna. Jedna kao prvi parametar očekuje iznimku (engl. *Exception*), a druga poruku koju je potrebno zapisati.

Zapisivanje informacija, a pogotovo iznimaka i greški je od iznimne važnosti u svakoj ozbiljnoj aplikaciji. Time je olakšano praćenje eventualne greške koja se dogodila, a da se najčešće ne zna što se dogodilo ili kada se točno dogodilo.

i na taj način će svaka iznimka biti obrađena, a servis će nastaviti dalje sa svojim radom.

2.2.1. Event Management

Uzevši u obzir samu svrhu ovog sustava i njegov cilj, može se zaključiti da je *Event management* najbitniji i najprometniji servis. Osim samih događaja, sve obrade podataka koje se tiču jelovnika ili dvorana također idu kroz ovaj servis. Sučelje *Event managementa* je prikazano sljedećim kodom:

```
public interface IEventService
    {
        Event GetEvent(int id);
        IEnumerable<Event> GetAllEvents(int vendorId);
        IEnumerable<Event> GetEventsByUser(int userId);
        Event UpdateEvent(Event ev);
        Event AddEvent(Event ev);
        void DeleteEvent(int id);
        IEnumerable<WeddingHall> GetWeddingHalls(int
vendorId);
        IEnumerable<WeddingHall> GetWeddingHallsEvents(int
vendorId);
        WeddingHall GetWeddingHall(int id);
        void UpdateEventDeposit(int eventId, decimal deposit,
int currency);
        void UpdateEventState(int eventId, int newState);
        void UpdatePeopleNumber(int eventId, int peopleNo);
        void UpdateEventMenu(int eventId, int menuId);
        IEnumerable<Menu> GetMenus(int vendorId);
        Menu AddMenu(Menu menu);
        Menu UpdateMenu(Menu menu);
        WeddingHall AddWeddingHall(WeddingHall weddingHall);
        WeddingHall UpdateWeddingHall(WeddingHall
weddingHall);
```

```
    }
```

Kôd 2.4. Sučelje *Event management* servisa

2.2.2. User Management

User management je jako bitan modul koji omogućuje upravljanje podacima i informacijama o korisnicima samog sustava. Na taj način je omogućen pristup podacima o svakom voditelju i njegovim poslovnim učincima preko kojih se mogu raditi daljnji izvještaji. To je jako korisna funkcionalnost u klijentskoj aplikaciji. Sučelje *User managementa* je prikazano sljedećim kodom:

```
public interface IUserService : IDataService
    {
        User GetUser(int id);
        IEnumerable<User> GetAllUsers(int vendorId);
        User UpdateUser(User ev);
        void UpdateUserImage(int id, string picture);
        void UpdateUserPassword(int id, string password);
        User AddUser(User ev);
        void DeleteUser(int id);
        User ValidateUser(string userName, string password);
        void AssignUserToEvent(int eventId, int userId);
        IEnumerable<PlannerEntry> GetPlannerEntries(int
userId);
        PlannerEntry UpdatePlannerEntry(PlannerEntry entry);
        PlannerEntry AddPlannerEntry(PlannerEntry entry);
    }
```

Kôd 2.5. Sučelje *User management* servisa

2.2.3. Customer Management

Kao i svaki ozbiljniji sustav, i ovaj ima modul koji je zadužen za upravljanje klijentima. Uvijek je korisno imati bazu podataka svih svojih dosadašnjih klijenata. Iako ima samo četiri osnovne funkcionalnosti, čime je i najmanji servis sustava, otvoren je za potencijalna proširivanja funkcionalnostima. Sučelje *Customer managementa:*

```
public interface ICustomerService : IDataService
    {
        Customer GetCustomer(int id);
        IEnumerable<Customer> GetAllCustomers(int vendorId);
```

```
        Customer UpdateCustomer(Customer ev);
        Customer AddCustomer(Customer ev);
    }
```

Kôd 2.6. Sučelje *Customer management* servisa

2.2.4. Gateway

Gateway predstavlja prolaz klijentske aplikacije prema servisima. Zamišljen je kao kanal koji nudi metode svih servisa prema vanjskim sustavima, a u ovom slučaju klijentskoj web aplikaciji. Svaka metoda nudi REST pristup za čiji potpis URI-a mora znati aplikacija koja ga koristi. .NET nudi mogućnost opisa metoda atributima, a jedan od mogućih je *WebInvoke* atribut koji nudi mogućnost na WCF-u da se metodi pristupi putem REST-a. Tako metoda za dohvat događaja s navedenim atributom izgleda ovako:

```
[WebInvoke(Method = "GET",
            ResponseFormat = WebMessageFormat.Json,
            UriTemplate = "event/{id}")]
public EventDTO GetEvent(string id);
```

Kôd 2.7. RESTful metoda

Atributom je opisano sve što aplikacija koja zove metodu mora znati. Web metoda kojom se mora pozvati metoda je GET, format odgovora koji metoda vraća je JSON, a URI kojim se pristupa metodi je *event/{id}* gdje se na mjesto {*id}* stavlja ID stvarnog događaja kojeg se dohvaća.

Same klase koje predstavljaju tablice iz baze podataka nisu izložene prema vanjskim sustavima nego se koriste takozvani objekti za prijenos podataka (engl. *Data Transfer Object, DTO*). Tako je svaki objekt koji se mora prenijeti iz servisnog sloja u web aplikaciju mapiran na pripadajući *DTO.* Iz prethodnog kôda (Kôd 2.7) se vidi da metoda za dohvaćanje događaja (engl. *Event*) zapravo vraća mapirani *EventDTO.*

2.3. Web aplikacija

Klijentska aplikacija preko servisa pristupa bazi podataka i prezentira podatke korisniku. Za ovaj završni rad klijenta predstavlja web aplikacija pisana u ASP.NET Core tehnologiji, programskom jeziku C#. Dvije jako bitne komponente aplikacije su predmemorija (*engl. Cache*) i podatkovna usluga (*engl. Data service*). Uzorak dizajna (engl.

Design pattern) kojim je aplikacija strukturirana je MVC (*Model-View-Controller*). To je najkorišteniji i najpoznatiji *design pattern* u programiranju web aplikacija.

2.3.1. MVC design pattern

Jedan jako bitan izazov koji imaju svi programeri je odvajanje korisničkog sučelja (engl. *User interface*) od poslovne logike. Za tu svrhu postoji jako puno *design patterna,* no za web aplikaciju najkorišteniji je MVC. Jednostavno se tokom godina pokazao kao najprikladniji za tu svrhu.

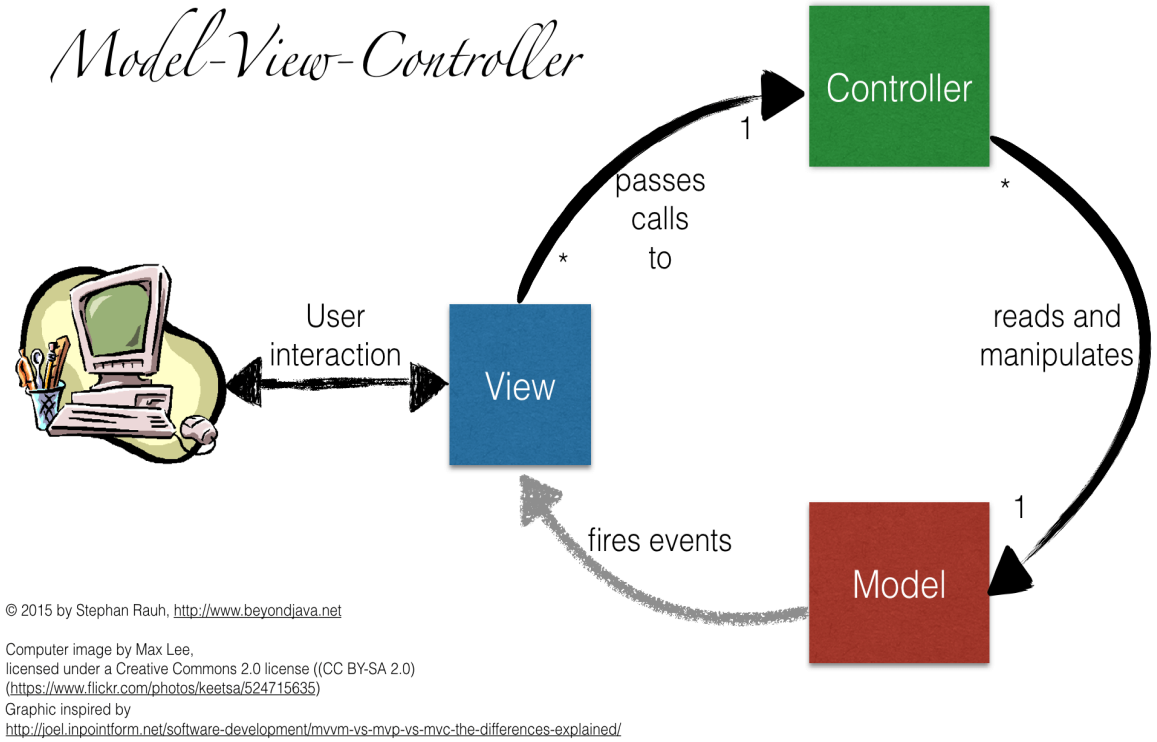
MVC *pattern* je dobro uspostavljeni način grupiranja aplikacijskih funkcija u objekte. Njegove varijacije postoje barem od ranih dana *Smalltalka*, jednog od prvih objektno-orijentiranih jezika. MVC je visoko-razinski uzorak – adresira arhitekturu aplikacije i klasificira objekte prema glavnim ulogama koje imaju u aplikaciji, umjesto da se „spuštaju“ prema specifičnostima.[4]

Model-View-Controller (MVC) pattern dijeli arhitekturu na tri bitne grupe od kojih se i sastoji njegovo ime:

- Model (engl. *Model*)
- Pogled (engl. *View*)
- Upravitelj (engl. *Controller*)

Model predstavlja podatak, odnosno klasu koja prenosi podatak. To može biti podatak dobiven ravno iz baze podataka. *View* je zapravo kako i sam naziv kaže, pogled koji korisnik vidi. To je jedini dio sustava u koji korisnik ima uvid. Sami dizajn neke stranice je prikazan u *viewu.* Na *Controlleru* se zapravo odrađuje logika sustava. On odrađuje korisničke zahtjeve te dohvaća modele i prosljeđuje ih u *view* na prikaz korisniku. MVC *design pattern* prikazan je slikom (Slika 2.3.).

[4] http://www.dummies.com/web-design-development/mobile-apps/the-model-view-controller-mvc-design-pattern/



Slika 2.3. Model-View-Controller[5]

2.3.2. Predmemorija aplikacije

Cache je dio aplikacije koji služi za čuvanje podataka spremnima za dohvat. Razlog tome je da se ne mora svakim korisničkim zahtjevom ići skroz preko servisnog sloja do baze podataka. Konkretno u ovoj aplikaciji je poprilično dugo putovanje jednog podatka: klijentska aplikacija, dva servisa, repozitorij, baza podataka te cijeli put natrag. Tako da sustav uz *cache* dobiva na brzini i performansama.

Kod prijave korisnika u sustav, *cache* se inicijalno napuni podacima koje čuva spremnima za korisničke upite. Svako spremanje podataka u bazu zahtjeva i njegovo ažuriranje kako bi u svakom trenutku bio u ispravnim stanju. Nije nužno sve podatke držati u *cache*-u, ali one čiji je dohvat jako čest ili kojih ima jako puno, svakako jest korisno. Ono što se u ovoj aplikaciji smatra nužnim čuvati u *cache*-u su: zaposlenici, klijenti, događaji, jelovnici i dvorane.

Osim što poboljšava performanse sustava, ujedno i omogućuje pristup podacima kada jedna od komponenti sustava (neki servis ili baza podataka) nije dostupna.

[5] https://www.beyondjava.net/blog/model-view-whatever/

2.3.3. Podatkovna usluga

Data service u aplikaciji predstavlja ono što i repozitorij u servisima. Predstavlja apstrakciju baze podataka odnosno pristup podacima. Poznajući arhitekturu sustava prema slici (Slika 2.1. Arhitektura), moguće je zaključiti da *data service* web zahtjevima (engl. *web request*) šalje upite na *Gateway* servis. On zatim, ovisno o zahtjevu, preko određenog modulskog servisa ide u bazu podataka po tražene podatke. Iz perspektive web aplikacije, ona zapravo ne zna i ne mora znati od kuda dolaze podaci. Jednostavno šalje upite preko *data service*-a, čija je zadaća da komunicira s *Gateway* servisom.

Zadaća ove komponente je da određuje kojim web metodama (GET, POST, PUT ili DELETE) će slati zahtjeve. Isto tako jedino ova komponenta web aplikacije zna da se komunikacija sa servisnim slojem odrađuje u JSON formatu.

3. ASP.NET Core

Tehnologija u kojoj je pisana web aplikacija je ASP.NET Core, koja je na dan pisanja ovog rada najnovija generacija ASP.NET okruženja. Razvijena je od Microsoft zajednice i nudi jako velike i obećavajuće mogućnosti. Jedan od glavnih razloga je to što je prva Microsoftova tehnologija čije aplikacije je moguće instalirati na razne platforme, a ne kao do sada samo na Windows operacijski sustav.

Na radost mnogih .NET programera, Microsoft se napokon odlučio na otvoren kod (engl. *Open source*). Na taj je način približio .NET programske jezike programerima i organizacijama, koji su ih možda jedino zbog dosadašnjeg zatvorenog kôda izbjegavali. Sada cijela programerska zajednica može pridonijeti razvoju *frameworka*.

Osim otvorenosti kôda i mogućnosti instaliranja na različitim platformama, bilo je još razloga za promjene, barem što se tiče ASP.NET-a. Od prvih dana ASP.NET Web Formsa preko popularnog ASP.NET MVC-a, Microsoft je stigao do svojeg *open source* radnog okvira (engl. *Framework*). Sigurno je da su morali postojati neki nedostaci u prijašnjim radnim okvirima, ako se Microsoft odlučio na tako veliki zahvat razvijanja nečega „iz nule“.

3.1. ASP.NET Web Forms

Poznato je da je HTTP (*Hypertext Transfer Protocol*) protokol koji ne čuva stanje (engl. *stateless*). Jedna od zadaća web aplikacije je da komunikacijom između klijenta i poslužitelja održava stanje tako da korisnik ima dojam kao da koristi aplikaciju na svom računalu. Zapravo da ima osjećaj da koristi desktop aplikaciju u kojoj je stanje očuvano u svakom trenutku (engl. *statefull*).

S Web Forms platformom, Microsoft je pokušao sakriti *Hypertext Transfer Protocol,* s njegovim unutarnjim neočuvanjem stanja, i *Hypertext Markup Language* (HTML), koji je u ono vrijeme bio nepoznanica za mnoge programere, modeliranjem korisničkog sučelja kao hijerarhiju kontrola s poslužiteljske strane (engl. *server-side control objects).* Svaka kontrola je vodila evidenciju o svojem stanju preko zahtjeva (engl. *requests*), renderirajući samu sebe u HTML kada je bilo potrebno i automatski povezujući događaje s klijentske strane (engl.

client-side events) (npr. klik gumba) s pripadajućim kodom obrade događaja (engl. *event handler code*) na poslužiteljskoj strani.[6]

Ono što se programerima uglavnom svidjelo kod *Web Formsa* je bilo dodavanje kontrola na formu metodom povuci-spusti (engl. *Drag and drop*) što je omogućavalo brzu izradu web stranice, za razliku od pisanja HTML koda. Isto tako su imali jednostavnu implementaciju validacija samih kontrola.

Ipak, s vremenom se pokazalo da je aplikaciju na ovoj platformi teško održavati. Pogotovo ako je aplikacija već velika i dugo u produkciji. Bila je to dobra ideja, ali s vremenom se ipak pokazalo da to nije idealan način programiranja weba. Neki od bitnijih nedostataka *Web Formsa*:

- Automatsko generiranje HTML-a dodajući kontrole se pokazalo da ne daje potpunu kontrolu programeru, što nikako nije dobro.
- Prenošenje stanja između klijenta i poslužitelja preko zahtjeva i odgovora (engl. *request and response*) otežava promet odgovora poslužitelja i time bitno otežava rad aplikacije.
- Životni ciklus jedne stranice je možda najbitniji dio za shvatiti kod ove tehnologije, a malo programera je u potpunosti razumije zbog njene kompleksnosti.

Iz razloga što je već dugo prisutna, trenutno ima jako puno aplikacija diljem svijeta pisanih u *Web Forms* platformi koje treba održavati. Iz tog razloga je i potražnja za programerima koji je razumiju i dalje velika. No u Microsoftu su shvatili da je nekako prirodni način programiranja web aplikacija zapravo MVC *pattern* pa su se i oni počeli okretati u tom smjeru.

3.2. ASP.NET MVC

U prijašnjem poglavlju 2.3.1. je pobliže opisan MVC *pattern*, a na slici (Slika 2.3.) je prikazan odnos pojedinih komponenti. Može se vidjeti kako je za razliku od *Web Forms* metode ovdje prezentacijski dio (*View*) u potpunosti odvojen od programske logike. Štoviše u *View* dijelu su samo HTML i JavaScript programski jezik koji pripadaju klijentskoj strani,

[6] Adam Freeman, *Pro ASP.NET Core MVC*, 6. izdanje, Apress 2016. str. 3.

dok je u ostalom dijelu aplikacije (*Controller* i *Model*) programski jezik od same serverske logike (u ovom slučaju C#). Na taj način serverski dio mogu održavati programeri koji znaju C# jezik (engl. *Backend*), a klijentsku stranu mogu održavati programeri koji znaju JavaScript, HTML i CSS (engl. *Frontend*).

Microsoft je 2007. godine predstavio svoju platformu MVC-a pod nazivom ASP.NET MVC i time obradovao veliki dio .NET programera, a s vremenom vjerojatno i sve ostale koji su do sada radili na ASP.NET platformi. Prvi dio naziva je ostao ASP.NET iz razloga što je napravljen nad postojećom ASP.NET platformom. To se brzo pokazalo kao pravi korak s njihove strane te su od 2007. godine do 2015. godine došli do svoje šeste verzije. No, očito su i tu našli neke nedostatke zbog kojih su se odlučili napraviti cijelu novu platformu, time odustavši od postojeće.

Od trenutka kada je napravljena, imalo je smisla za Microsoft da napravi MVC *Framework* preko postojeće ASP.NET platforme, koja je imala mnogo solidnih niskorazinskih funkcionalnosti koje su omogućile glavni početak u procesu razvijanja i koje su već bile poznate i razumljive od strane ASP.NET programera.

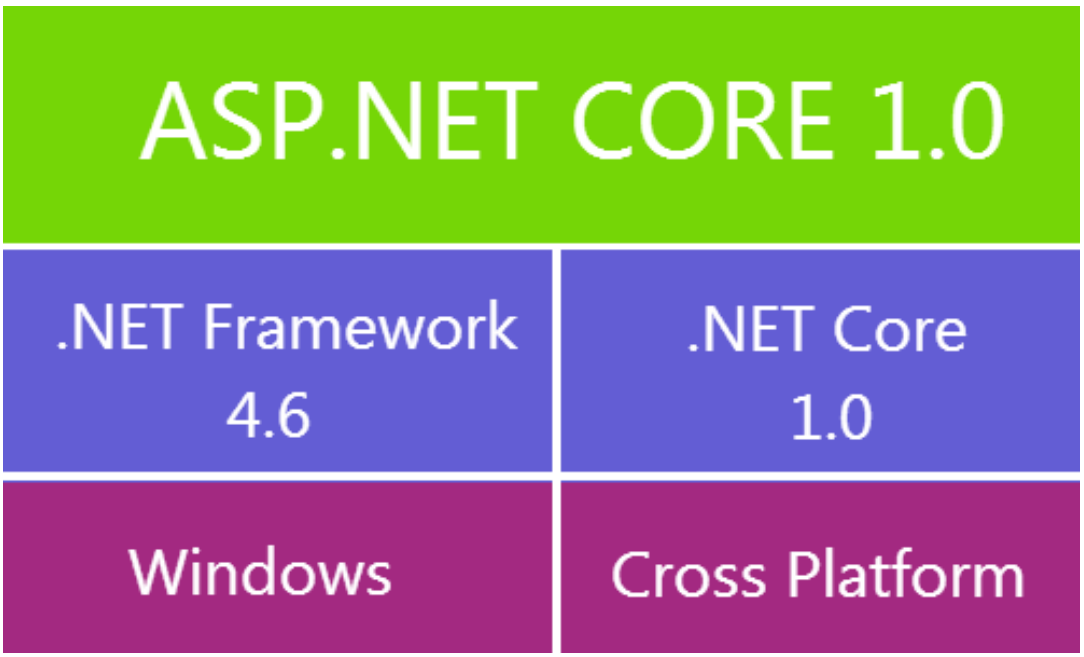
Kako je MVC *Framework*-u rasla popularnost, Microsoft je počeo uzimati neke od bitnih funkcionalnosti i dodavati ih u *Web Forms.* Rezultat je bio nadasve čudan, gdje su funkcionalnosti dizajnirane za podršku MVC *Framework*-a bile proširene za podršku *Web Formsa,* s dodatnim dizajnerskim dosjetkama kako bi sve skupa zajedno odgovaralo. U isto vrijeme Microsoft je počeo proširivati ASP.NET s novim radnim okvirima za stvaranje web servisa (*Web API*) i komunikaciju u stvarnom vremenu. Novi radni okviri su nadodali svoje vlastite konfiguracijske i razvojne konvencije, od kojih je svaka imala svoje prednosti i nedostatke, i krajnji rezultat je bio fragmentirani nered.[7]

3.3. Razumijevanje ASP.NET Core-a

ASP.NET Core je samo dio nešto većeg *frameworka* .NET Core koji je *open source* i omogućen za više različitih platformi (engl. *Cross platform*). Definira tipove koji postaju nova generacija prenosivih klasnih biblioteka (engl. *class libraries*). Kada su ga počeli razvijati u Microsoftu, trudili su se da ima što općenitiju namjenu kako bi se mogao koristiti

[7] Adam Freeman, *Pro ASP.NET Core MVC*, 6. izdanje, Apress 2016. str. 5.

na više različitih mjesta. Iz tog razloga .NET Core danas ima fleksibilnu arhitekturu. Na slici (Slika 3.1) se vide Microsoft radni okviri i njihov međusobni odnos.



Slika 3.1. Microsoft radni okviri[8]

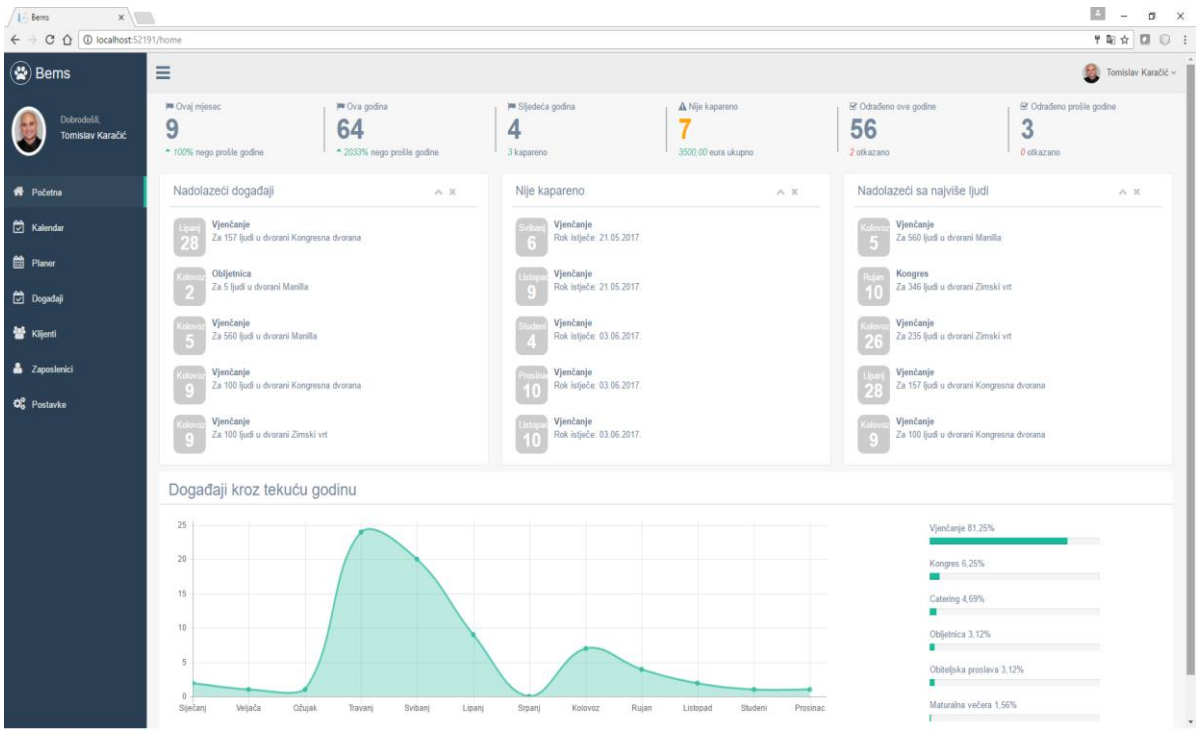
Prva verzija .NET Core 1.0 objavljena je u lipnju 2016. godine, zatim je verzija 1.1 predstavljena u studenom 2016. godine, dok je dosadašnja zadnja verzija 2.0 predstavljena u kolovozu 2017. godine. Tek su u verziji 2.0 omogućeni i ostali .NET programski jezici osim C#, kao što su F# i Visual Basic.

ASP.NET Core MVC pruža funkcionalnosti originalnog ASP.NET MVC *Frameworka* razvijenog na novoj ASP.NET Core platformi. Uključuje funkcionalnosti koje su ranije pružene sa strane Web API-a, uključuje prirodniji način generiranja kompleksnog sadržaja, i čini ključne programerske zadatke, kao što su testiranje jedinica (engl. *Unit testing*), jednostavnije i predvidljivije.[9]

[8] http://www.fusionhit.com/net-core-better/
[9] Adam Freeman, *Pro ASP.NET Core MVC*, 6. izdanje, Apress 2016. str. 5.

4. Korištenje aplikacije

Na početnoj stranici same aplikacije se mogu vidjeti razni statistički podaci vlastitog restorana. Tu se nalaze sažete najbitnije stavke kao što su prvi nadolazeći događaji, događaji koji još uvijek nisu kapareni, broj događaja kroz tekuću godinu grupirani po tipu događaja, i slične bitne informacije. Na slici (Slika 4.1.) se može vidjeti primjer početne stranice.



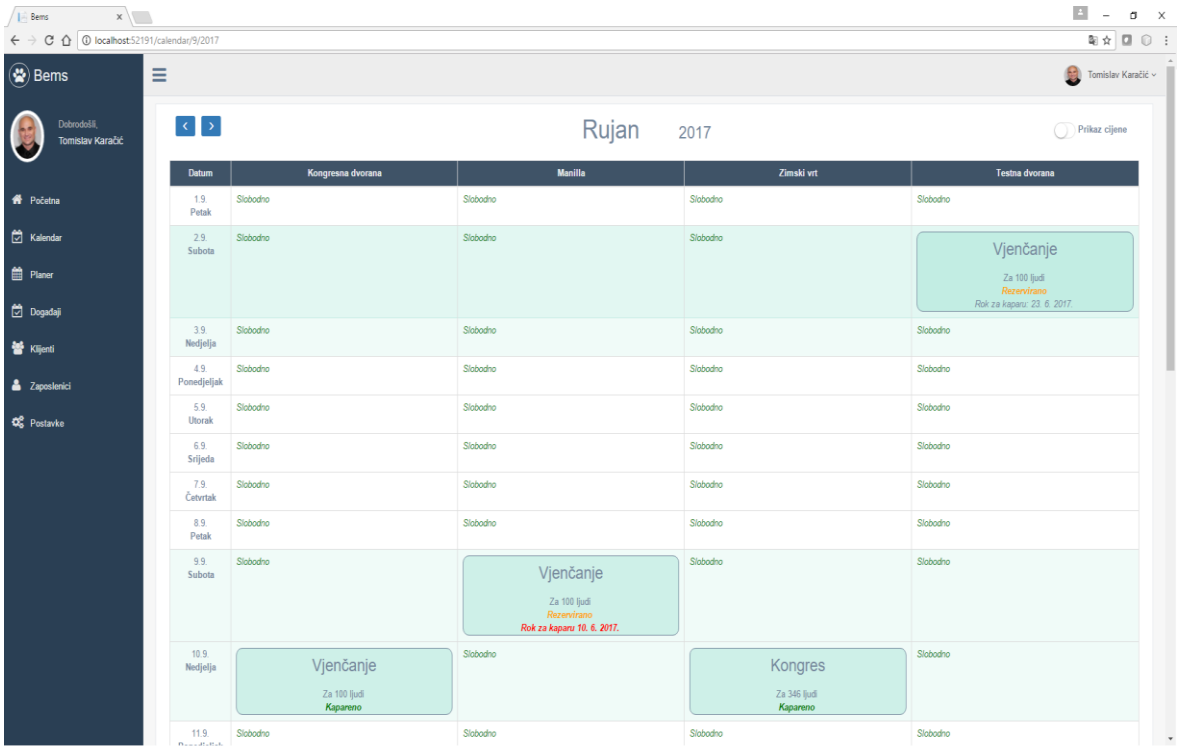
Slika 4.1. Početna stranica

4.1. Kalendar događaja

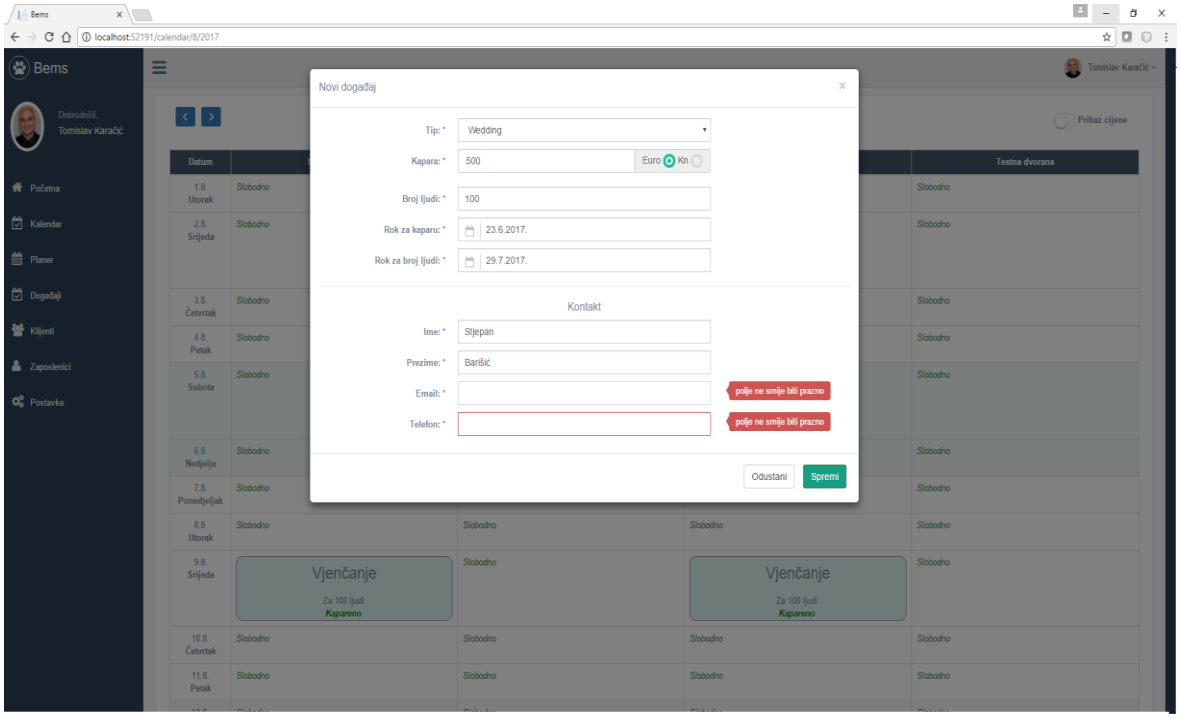
Najbitniji modul cijelog sustava, odnosno onaj od kojeg sami korisnici imaju najviše koristi, je sam kalendar događaja koji je prikazan slikom (Slika 4.2.). Stupci u kalendaru predstavljaju dvorane dok je svaki redak datum u mjesecu. Jednostavnim pritiskanjem gumba se može prolaziti po mjesecima ili čak godinama. Na taj način je jednostavan i intuitivan prikaz događaja gdje se za svaki može vidjeti u kojoj se dvorani održava i kojeg datuma u godini.

Unos novog događaja se odrađuje jednim klikom miša na željeni pravokutnik koji je presjek datuma i dvorane. Ukoliko je potrebno izmijeniti postojeći događaj, potrebno je kliknuti na pravokutnik željenog događaja. Za unos novog događaja se otvara dijaloški okvir

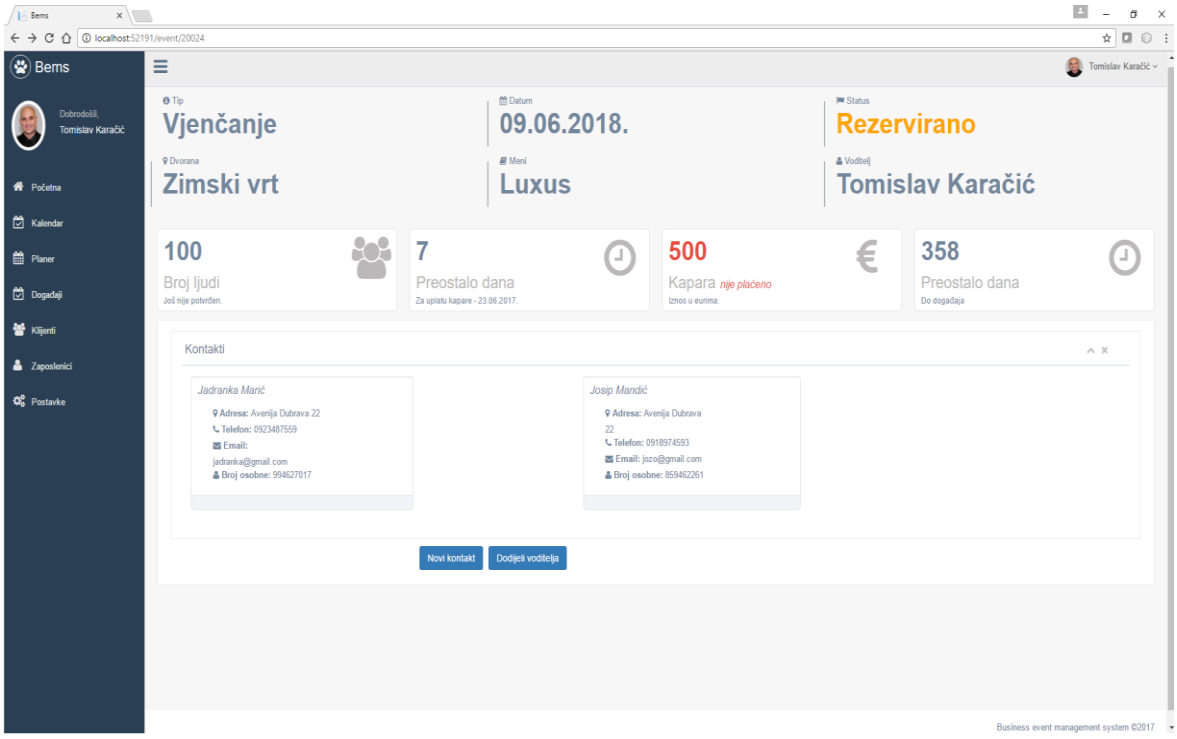
prikazana slikom (Slika 4.3.), dok je prikaz postojećeg događaja i mogućnost njegovog uređivanja prikazano slikom (Slika 4.4.).



Slika 4.2. Kalendar događaja



Slika 4.3. Novi događaj

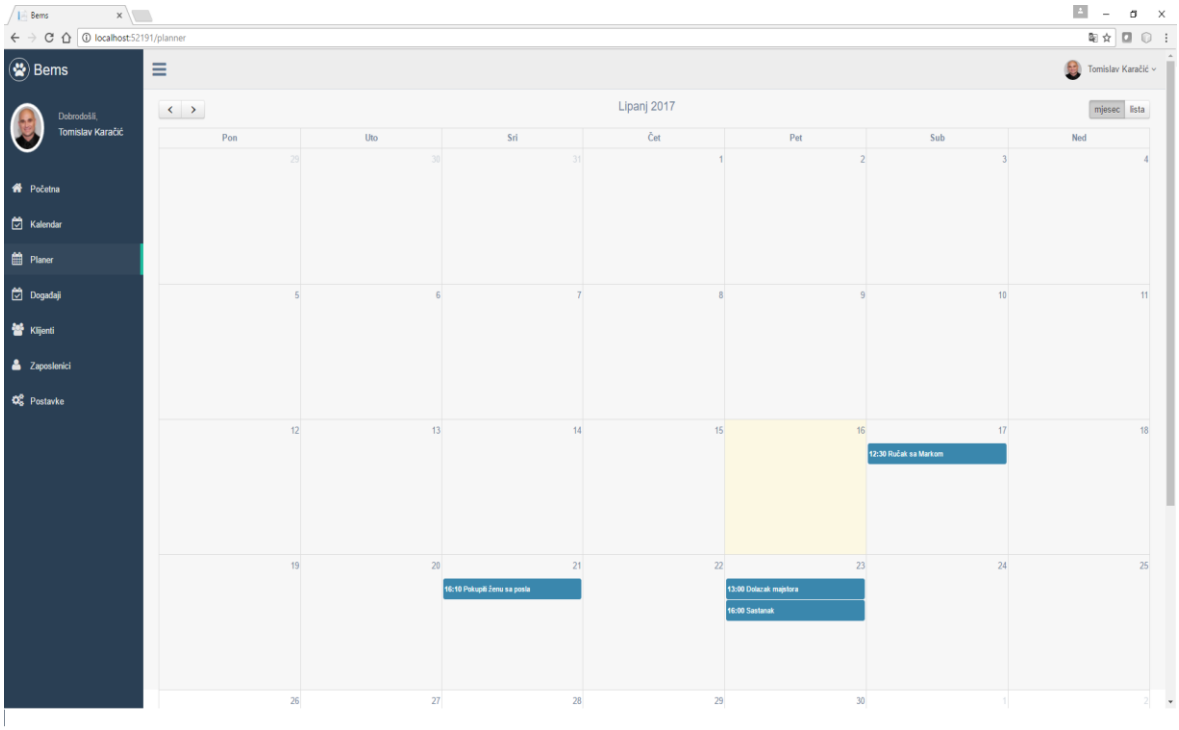


Slika 4.4. Postojeći događaj

Kod uređivanja postojećeg događaja je moguće promijeniti dvoranu, status, iznos kapare, broj ljudi, jelovnik te dodijeliti novog voditelja. Isto tako je moguće dodavati klijente koji predstavljaju kontakt osobe za odabrani događaj. Datum i dvorana se ne mogu mijenjati. Najviše iz razloga što zaposlenici u ovoj branši smatraju da mijenjanjem jedne od te dvije stavke to automatski postaje novi događaj, dok se postojeći otkazuje. Mijenjanjem dvorane ili datuma treba uzeti u obzir dostupnost dvorane za novi datum ili datuma za novu dvoranu. Sve ostale stavke su zato lako promjenjive.

4.2. Osobni planer

Kako se svaki zaposlenik prijavljuje vlastitim korisničkim imenom i svojom zaporkom, tako svaki od njih ima i svoj osobni planer. Planer je koristan za zapisivanje nekih osobnih stvari kao što su sastanci s klijentima, dogovori za ručak, neki bitni osobni zapisi i slično. Slično kao i kod kalendara događaja, i kod planera je jednostavno listati stranice po mjesecima jednostavnim pritiskanjem određenih gumba. Planer je prikazanom slikom (Slika 4.5.). Unos novog događaj se isto unosi klikom miša na slobodno polje, a uređivanje postojećeg klikom na željeni postojeći događaj.

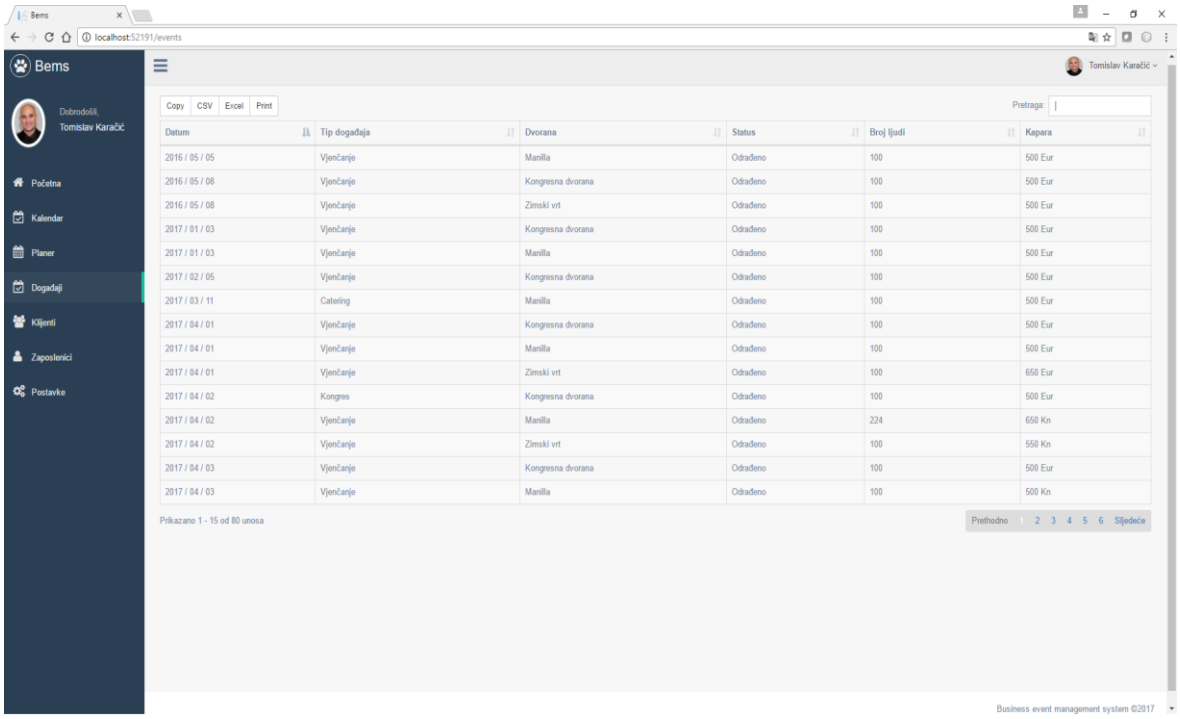


Slika 4.5. Osobni planer

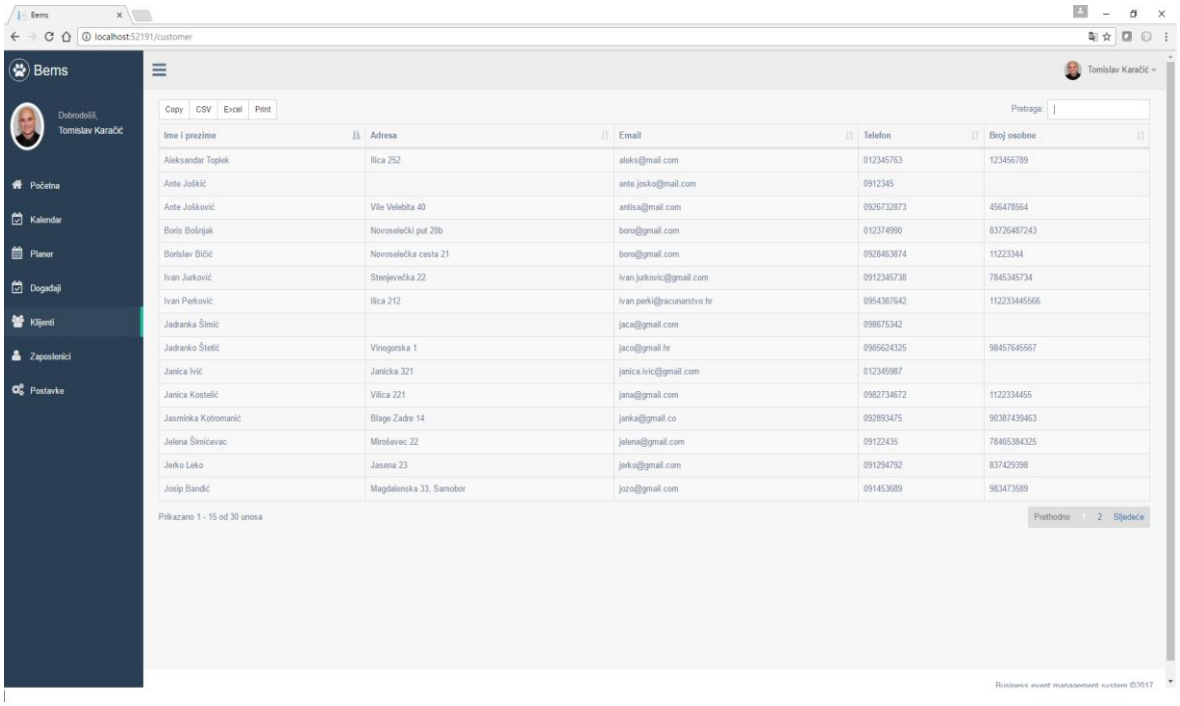
4.3. Popisi događaja i klijenata

Kao i svaka „pametna“ aplikacija, i ova čuva u bazi podataka sve dosadašnje događaje i klijente. Čuvaju se svi događaji, bili oni odrađeni, otkazani ili su tek rezervirani. Baza klijenata isto tako može korisniku bitna od velike koristi, jer voditelji ili čak vlasnici, vole vidjeti koji su im klijenti vjerni i dolaze im po nekoliko puta, ili koji možda često otkažu svoje rezervacije.

Aplikacija nudi ispis svih događaja i klijenata u obliku tablice. Mogu se sortirati po bilo kojem stupcu i lako je za pretraživati jer se nudi polje za unos teksta po kojem se pretražuje. Na slici (Slika 4.6.) je prikazan primjer popisa događaja, a na slici (Slika 4.7.) primjer popisa klijenata. Obje tablice je moguće ispisati u nekom od ponuđenih oblika (CSV ili Excel) ili isprintati.



Slika 4.6. Popis događaja



Slika 4.7. Popis klijenata

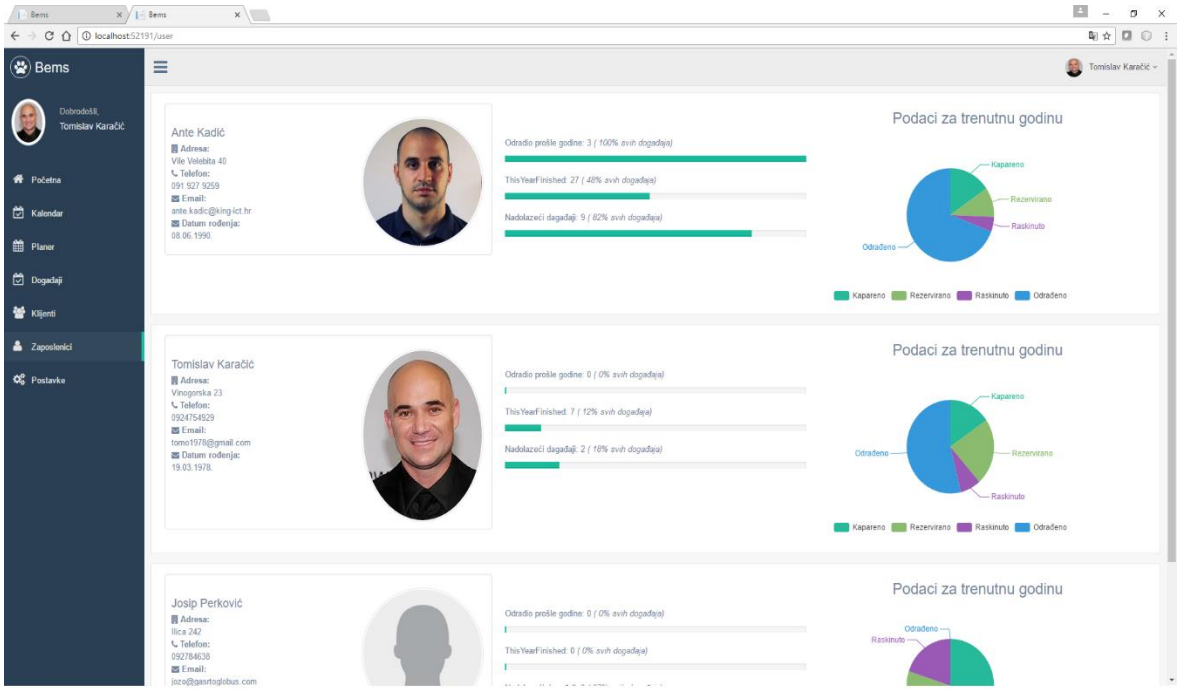
4.4. Administratorska prava

Postoje dva modula u aplikaciji za koja su potrebna administratorska prava. Zamišljeno je da voditelji koji odrađuju svakodnevne poslove i unose u aplikaciju nemaju ta prava, nego samo direktori.

Jedan nudi prikaz voditelja s njihovim informacijama o učinkovitosti, a drugi općenite postavke vezane za jelovnike i dvorane.

4.4.1. Prikaz voditelja

Kako svaki direktor voli lijepi prikaz grafikona i ostalih dijagramskih prikaza, ni ovdje neće biti zakinuti. Ovdje aplikacija nudi prikaz voditelja s njihovim osobnim informacijama, ali i informacije o poslovnim zalaganjima. Na slici (Slika 4.8.) se može vidjeti koje su to konkretno informacije.

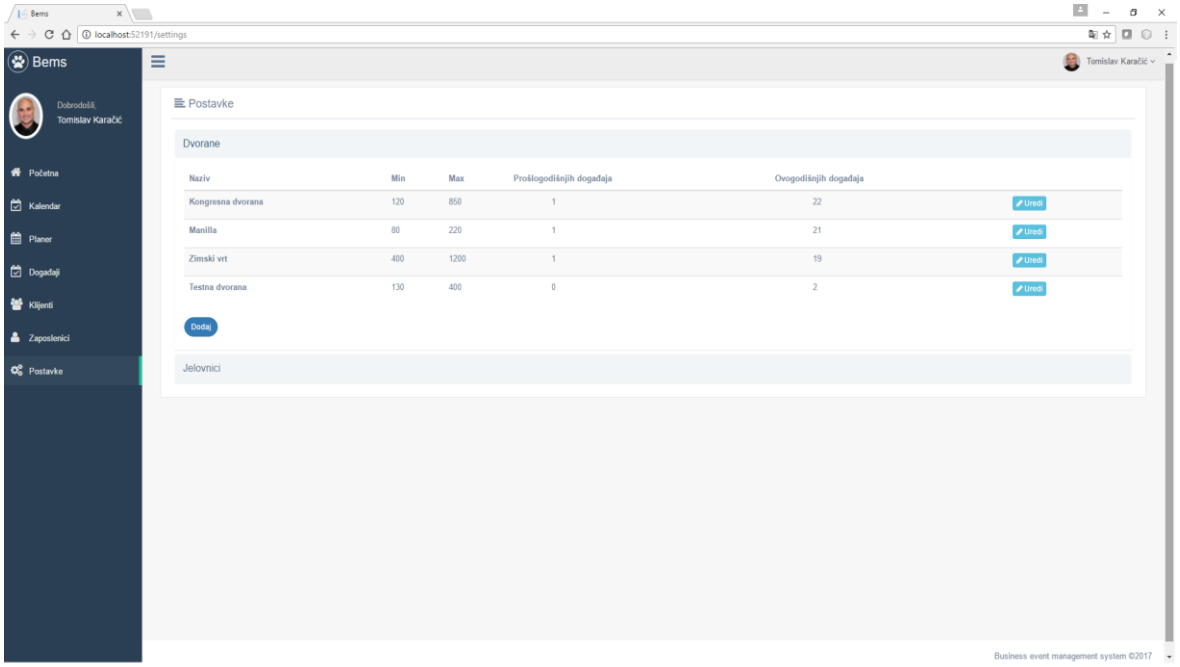


Slika 4.8. Prikaz voditelja

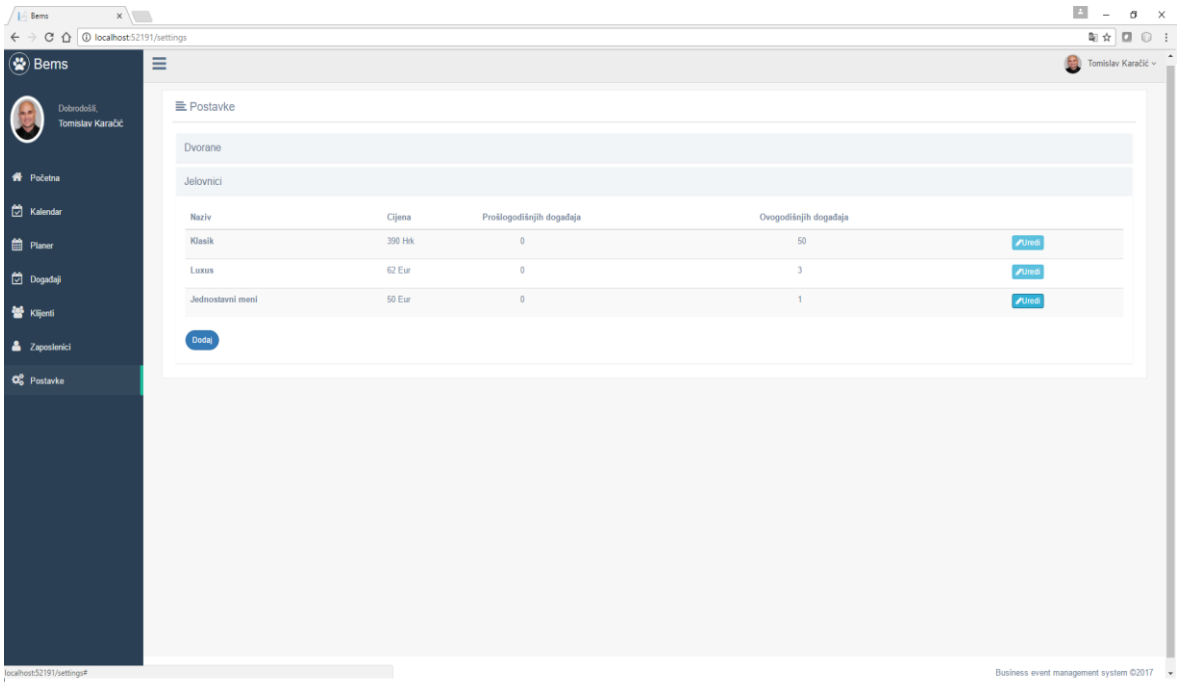
4.4.2. Postavke

Kod samih postavka je bitno da se mogu unositi dvorane i jelovnici, ali i uređivanje istih. Sučelje je, kao i sve prethodne, dovoljno intuitivno gdje su dostupni gumbi na kojima piše koji je za uređivanje, a koji za dodavanje nove stavke. Slika (Slika 4.9.) i (Slika 4.10.)

prikazuju izgled sučelja za postavke gdje se može primijetiti da se jednostavnim klikom na naslov *Dvorane* ili *Jelovnici* otvara popis odabrane stavke, a s njim i dostupni gumbi za određene akcije.



Slika 4.9. Postavke dvorane



Slika 4.10. Postavke jelovnika

Zaključak

Krajnji produkt ovog rada je poslovni informacijski sustav koji se sastoji od web aplikacije, servisnog sloja (četiri servisa) i baze podataka, namijenjen voditeljima hotela i restorana u organizaciji svečanih događaja.

Kako zapravo nije poznato da li takav sustav postoji na tržištu, zaključeno je da bi isti mogao naći svoje mjesto na istom. Nudi mnogo automatizacije već poznatog poslovanja, kojem je dosada nedostajalo konkretno čuvanje i upravljanje podacima, te dobivanje informacija na pristupačan način.

Osim što u testnoj okolini odrađuje sve unaprijed definirane operacije, korisničko sučelje je pristupačno i na lijep način prezentirano. Samo korisničko sučelje je dizajnirano na način da bude što intuitivnije tako da korisnicima prilagodba i učenje bude minimalno kratko.

Popis kratica

API	*Application Programming Interface*	aplikativno programsko sučelje
CORBA	*Common Object Request Broker Architecture*	zahtjev brokerske arhitekture zajedničkih objekata
CSV	*Comma-Separated Values*	vrijednosti odvojene zarezom
DTO	*Data Transfer Object*	objekt za prijenos podataka
HTML	*HyperText Markup Language*	označni jezik hiperteksta
HTTP	*HyperText Transfer Protokol*	prijenosni protokol hiperteksta
JSON	*JavaScript Object Notation*	notacija JavaScript objekata
LINQ	*Language Integrated Query*	jezični integrirani upit
MVC	*Model-View-Controller*	model-pogled-upravitelj
ORM	*Object Relational Mapping*	objektno relacijsko mapiranje
REST	*Representational State Transfer*	reprezentativni prijenos stanja
RPC	*Remote Procedure Call*	poziv udaljenih metoda
SOAP	*Simple Object Access Protocol*	jednostavan protokol pristupa objektu
TCP	*Transmission Control Protocol*	protokol kontrole prijenosa
URI	*Uniform Resource Identifier*	jedinstveni identifikator resursa
WCF	*Windows Communications Foundation*	temelj Windows komunikacije
XML	*EXtensible Markup Language*	proširivi označni jezik

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] KUSERBAJN S. REST arhitektura, Veleučilište u Šibeniku, Šibenik, 2016.

[2] JOVANOVIĆ A. *Windows Communication Foundation*, Prirodno-matematički fakultet, Kragujevac, 2009.

[3] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions, listopad 2017.

[4] http://www.dummies.com/web-design-development/mobile-apps/the-model-view-controller-mvc-design-pattern/, listopad 2017.

[5] FREEMAN, A. *Pro ASP.NET Core MVC*, 6. izdanje, Apress 2016.

[6] https://www.beyondjava.net/blog/model-view-whatever/, listopad 2017.

[7] http://www.fusionhit.com/net-core-better/, listopad 2017.