

**Univerzitet u Nišu**  
**Prirodno-matematički fakultet**  
**Departman za računarske nauke**



---

## **Arhitektura MVC aplikacije sa primerima u PHP-u**

---

**Master rad**

*Student:*  
Markuš Mitrinović Elizabeta  
Broj indeksa: 51/2011

*Mentor:*  
prof. dr Marko Milošević

Niš, Oktobar 2016. godine

## Sadržaj

Uvod.....	3
Softverska arhitektura .....	4
Arhitektura web aplikacija.....	11
Model-View-Controller aplikacije.....	14
MVC šablon .....	15
Model .....	16
View.....	17
Controller .....	17
Interakcija komponenti .....	18
Prednosti i mane.....	19
Popularna MVC okruženja .....	19
Zend framework.....	21
Aplikacija „Nastavnički portal“.....	25
Poređenje Zend Frameworka sa drugim razvojnim okruženjima .....	34
Zaključak.....	35
Literatura.....	36

## Uvod

Tokom studija, imala sam prilike da radim u timskom i pojedinačnom razvoju raznih aplikacija, i upoznala sam se sa mnogim poteškoćama i načinima rešavanja problema tokom rada na projektima. Način razmišljanja i znanje koje sam stekla radeći na aplikacijama u saradnji sa izvanrednim, posvećenim i iskusnim mentorima, je neprocenljivo i biće mi i te kako korisno za dalji put kroz nove aplikacije na kojima ću raditi.

Na početku sam imala mišljenje da je na projektu najbolje raditi sam jer si tada sam svoj šef, kako ti odlučiš tako će i biti. Međutim, sada sa ovim iskustvom koje posedujem mogu reći da nema ničeg boljeg od rada u timu. Mnoge su prednosti rada u timu, pre svega zadatku koji dobiješ na rešavanje si mnogo posvećeniji nego kada imaš hiljadu problema i ne znaš kojim redom krenuti. Probleme ne rešava jedna glava, imaš prilike da čuješ tuđa mišljenja i shvatanja, lakše je doći do nekih kontraprimera koji su bitni kada treba odlučiti kojim putem nastaviti. Tim poboljšava kvalitet i brzinu izrade aplikacije. Pošto je i sam problem podeljen u podprobleme koji su podeljeni među članovima timovima, i kod koji je proizvod jednog timskog rada je bolje organizovaniji i lakše razumljiviji. Međutim, od tima i timskog rada nema ničega ako tim nema dobrog vođu. Vođa tima ima jednu od najvažnijih uloga, verovatno i najviše zasluga u uspehu jednog projekta. Vođa tima se susreće sa raznoraznim problemima kako pojedinaca koji čine tim, tako i problema koji se stvaraju baš iz razloga što tim postoji. Ono što je najvažnije da bi projekat uspeo jeste jako dobra organizacija i precizno definisana pravila po kojima će svako od „igrača“ tima igrati.

Projekat „Nastavnički portal“ je nastao kao proizvod timskog rada kolege Stefana Stanimirovića, moje malenkosti, u radu sa mentorima prof. dr Markom Miloševićem i mr Ivanom Stankovićem, kojima se ovom prilikom zahvaljujem. Ovim radom želim da predstavim i podelim delić znanja koje sam stekla tokom studiranja i rada na aplikacijama.

Takođe se zahvaljujem i svim ostalim profesorima i asistentima koji su doprineli u shvatanju i rešavanju raznoraznih problema u raznim oblastima.

## Softverska arhitektura

Softverska arhitektura je skup definicija strukture sistema/projekta, odnosa među elementima sistema/projekta, i osobinama dva elementa u vezi, zajedno sa razlozima za uvođenje i konfiguracije svakog elementa, koji, ako se ne definišu precizno i konkretno, mogu dovesti do neuspeha projekta.

Dokumentovanje softverske arhitekture olakšava komunikaciju između članova tima koji rade na realizaciji projekta, čuva bitne odluke o dizajnu, i omogućava ponovnu upotrebu dizajna komponenti u novim projektima. Budući da je arhitektura za korak bliža realizaciji projekta u odnosu na zahteve, to je i njeno razmatranje detaljnije od razmatranja samih zahteva. Arhitektura obezbeđuje podelu problema na nivo detaljnosti koji odgovara veštini svakog od programera, i omogućava da veći broj programera iz više razvojnih timova radi nezavisno. Dobra arhitektura obezbeđuje laku konstrukciju, dok je loša čini skoro nemogućom. Jako je skupo da se promene u arhitekturi vrše tokom konstrukcije ili kasnije. Vreme potrebno za popravku greške nastale u arhitekturi je istog reda kao vreme potrebno za ispravku greške koja nastane prilikom realizacije tj. pisanja samog koda.

Dobra softverska arhitektura prvo zahteva pregled sa najširim opisom sistema. Ukoliko nema takvog pregleda, onda dolazi do poteškoća građenja koherentne slike iz hiljadu detalja koji potiču od nekoliko desetina pa i stotina pojedinačnih klasa i nerazumevanje načina na koje klase koje se razvijaju doprinose u radu sistema. U dokumentaciji koja opisuje arhitekturu moraju se naći i dokazi da su razmotrene sve alternative izabrane organizacije sistema, kao i razlozi zbog kojih je baš ta organizacija dobila prednost u odnosu na razmatrane alternative i opravdanje napravljenog izbora. Frustrirajuće je raditi na klasi kada izgleda da nije dobro razmotrena uloga te klase u sistemu. Ove informacije obezbeđuju jasnu sliku o toku razmišljanja koja je jako korisna tokom konstrukcije i održavanja sistema.

Dokument o arhitekturi treba da definiše glavne elemente projekta, gradivne blokove, koji mogu predstavljati, u zavisnosti od veličine projekta, jednu klasu ili podsistem koji sadrži veći broj klasa tj. rutinu koje zajedno rade i obezbeđuju funkcionalnost sistema. Svaka karakteristika koja je pobrojana u zahtevima treba biti pokrivena bar jednim gradivnim blokom. Ako je funkcionalnost zahtevana od dva ili više gradivnih blokova, tada zahtevi treba da budu kooperativni, i nikako ne smeju biti u konfliktu.

Potrebno je jasno definisati za šta je odgovoran svaki od gradivnih blokova. Svaki gradivni blok ima svoju oblast odgovornosti i treba da zna što je manje moguće o oblastima odgovornosti drugih gradivnih blokova. Ovakvom vrstom minimizacije informacije postiže se lokalizacija informacija o dizajnu unutrašnjosti jednog gradivnog bloka. Neophodno je adekvatno definisati pravila za komunikaciju za svaki od gradivnih blokova. Arhitektura treba da opiše za svaki gradivni blok koje od ostalih blokova može direktno da koristi, koje indirektno, a koje uopšte ne sme da koristi.

Dokument treba da odredi najvažnije klase koje će se koristiti, da identificuje njihove odgovornosti i način na koji te klase imaju interakciju sa drugim klasama. Dokumentacija o arhitekturi treba da sadrži opis hijerarhije klasa, prelaska između stanja i persistenciju objekata, organizaciju klasa u podsisteme ukoliko je projekat dovoljno velik.

Dokument o arhitekturi treba da opiše dizajne najvažnijih tabela i datoteka. Obično se podacima direktno pristupa samo iz jednog pod sistema ili klase, osim kod klasa i rutina koje dopuštaju pristup na kontrolisani način. Arhitektura treba da specificira organizaciju i sadržaj svih baza podataka. Treba da objasni zašto je jedna baza bolja od više njih ili obratno, da identificuje moguće interakcije sa drugim programima koji pristupaju podacima, da objasni koji će pogledi biti kreirani nad podacima itd.

Ukoliko arhitektura zavisi od konkretnih poslovnih pravila, ona treba da ih identificuje i da opiše uticaj koji ta pravila imaju na sam dizajn celokupnog sistema. Jedan primer takvog zahteva je da se od sistema zahteva da podaci kojima pristupa korisnik moraju biti ažurirani na svakih 30 sekundi.

Ponekad se korisnički interfejs specificira u vreme kreiranja zahteva, ukoliko to nije slučaj onda to treba učiniti u toku kreiranja softverske arhitekture. Dokument o arhitekturi treba da specificira najvažnije elemente koje se odnose na format web strane, grafički korisnički interfejs, interfejs komande linije itd. Pažljivo definisan korisnički interfejs pravi razliku između programa koji se sviđa korisnicima i programa koji niko ne koristi.

Arhitektura treba da bude modularna, tako da novi korisnički interfejs može da bude zamjenjen, a da ta zamena ne utiče na poslovnu politiku niti na delove programa koji ispisuju izlazne podatke. Na primer, arhitektura treba da omogući relativno lako isključenje grupe klase sa interaktivnim interfejsom i uključenje grupe klasa koje „rade“ preko komande linije. Takva mogućnost je često korisna, zato što su interfejsi komandnih linija pogodni za testiranje softvera na nivou jedinica (engl. *unit test*) i na nivou podistema (engl. *subsystem test*).

Ulaz/izlaz predstavlja još jedno područje kome treba posvetiti pažnju prilikom kreiranja arhitekture. Arhitektura treba da specificira da li se, ukoliko se čitaju podaci iz datoteke, koristi šema čitanja „gledaj napred“ (engl. *look-ahead*), „gledaj pozadi“ (engl. *look-behind*) ili „tačno na vreme“ (engl. *just-in-time*). Dokument treba da opiše na kom nivou se detektuju greške ulaza/izlaza: na nivou polja, na nivou sloga, na nivou toka podataka ili na nivou cele datoteke.

Softverska arhitektura treba da opiše plan za upravljanje oskudnim resursima. U oblastima aplikacija gde postoji memorijsko ograničenje, prilikom definisanja arhitekture neophodno je razmotriti i upravljanje memorijom. Dokument treba da proceni resurse koji se koriste u normalnim slučajevima, kao i resurse koji se koriste u ekstremnim slučajevima. U jednostavnom slučaju, procene treba da pokažu da su potrebni resursi sasvim unutar kapaciteta implementiranog okruženja koje nameravamo da koristimo. Dok u složenim slučajevima, može se zahtevati da aplikacija sama aktivnije upravlja sa sopstvenim resursima,

i tada menadžer resursa treba da bude definisan tokom kreiranja arhitekture, pri čemu se definisanju menadžera resursa treba posvetiti ista pažnja kao drugim delovima sistema.

Dokument o softverskoj arhitekturi treba da opiše pristup prema sigurnosti na nivou dizajna i na nivou koda. Ukoliko model pretnji do tada nije bio izgrađen, treba ga napraviti prilikom osmišljavanja same arhitekture. Pravila za zapis koda (engl. *coding guidelines*) treba da budu razvijena tako da se imaju u vidu sigurnosne implikacije, što obuhvata pristupe za rukovanje baferima, pristupe za rad sa podacima za koje nema poverenja (kao što su ulazni podaci korisnika, kolačići, konfiguracioni podaci, drugi spoljašnji interfejsi itd.), enkripciju, nivo detalja koje sadrže podatke o grešci, zaštitu tajnih podataka itd.

Ako postoji zabrinutost oko performansi, tada ciljane performanse moraju biti specificirane tokom kreiranja dokumenta sa zahtevima. Ciljane performanse se mogu odnositi na brzinu i na korišćenje memorije. Softverska arhitektura treba da obezbedi procene i objasni zašto arhitekti veruju da su ciljne performanse dostižne. Ukoliko u nekim oblastima postoji opasnost da se postavljeni cilj ne ispuni, arhitektura treba to jasno da iskaže. Neke oblasti mogu da zahtevaju korišćenje specificiranih algoritama ili tipova podataka kako bi se dostigle željene performanse, te je potrebno da dokument sadrži informaciju o tome. Dokument može da uključi određivanje prostornog i vremenskog budžeta za svaku od klasa ili objekata.

Dizajneri mogu brinuti o mogućnostima sistema koji se gradi da postigne ciljne performanse, da radi sa ograničenjima u resursima koja su mu nametnuta, ili da bude adekvatno podržan od strane okruženja za implementaciju. Dokument treba da demonstrira da je sistem tehnološki izvodljiv. Ukoliko neizvodljivost u ma kojoj od oblasti može dovesti do nemogućnosti u radu, dokument mora da ukaže kako su ovi elementi istraženi - kroz prototipove koji služe za proveru koncepta, kroz istraživanje ili na drugi način. Ovi rizici treba da se razreše pre nego što se krene u razvoj softvera.

Skalabilnost je sposobnost sistema da raste, kako bi mogao da izađe u susret budućim zahtevima. Softverska arhitektura mora da opiše kako će se sistem odnositi prema porastu broja korisnika, porastu broja servera, mrežnih čvorova, veličine baze podataka, veličine obima transakcija i svih dugih parametara koji mogu da utiču na rad sistema. Ukoliko se ne očekuje skalabilnost sistema ili se ona ne razmatra, u dokumentu je potrebno eksplicitno naglasiti taj stav.

Interoperabilnost je kada se od sistema očekuje da deli podatke ili resurse sa drugim softverom ili hardverom, u tom slučaju softverska arhitektura treba da opiše na koji način će to biti postignuto i da li to i na koji način utiče na rad sistema.

Internacionalizacija (engl. *Internationalization*), je tehnička aktivnost pripreme programa da podržava veći broj jezika. Internacionalizacija se prepiće sa lokalizacijom (engl. *Localization*) koja predstavlja aktivnost programa tako da podržava konkretni lokalni jezik. Pitanje internacionalizacije privlače pažnju u definisanju arhitekture interaktivnog sistema. Najveći broj interaktivnih sistema sadrži na desetine i stotine pitanja, prikaza statusa, poruka koje sadrže pomoć, poruka o greškama itd. Potrebno je proceniti resurse sa

znakovnim nizovima. Dokument o softverskoj arhitekturi treba da pokaže kako su razmatrana tipična pitanja koja se odnose na tekst, znakove i kodne strane, na koji će se način održavati i menjati tekst kako će se prevoditi tekst na duge jezike a da uticaj na kod i na korisnički interfejs bude najmanji moguć. Arhitektura može da opiše i izabere opciju da li se koristi tekst unutar koda, da se čuva u klasama i referiše na njega kroz interfejs klase ili da se smesti u resursne datoteke.

Obrada grešaka je jedan od težih problema u modernom računarstvu, te se njima bavimo nesistemski. Neki su procenili da se čak i do 90% programskog koda piše radi obrade izuzetaka, odakle sledi da se samo 10% koda odnosi na „normalne“ situacije. Čim se toliko mnogo koda bavi rukovanjem sa greškama jasno je da arhitektura treba da specificira strategiju za njihovu obradu na konzistentan način. Rukovanje greškama se često posmatra na nivou kodiranja (ne arhitekture), a ima i slučajeva u kojima se uopšte ne razmatra. Ipak, s obzirom da njegove implementacije utiču na ceo sistem, najbolje je posmatrati ga na nivou arhitekture.

Neka od pitanja koja treba razmotriti pri definisanju ovog aspekta arhitekture su:

1. Da li obrada grešaka korektivna ili se ograničava na otkrivanje? Ukoliko je korektivna, tada program može da pokuša da se oporavi od nekih grešaka, dok kod grešaka koje su ograničene na otkrivanje, program može da nastavi sa radom kao da se ništa nije dogodilo ili može da se zaustavi. U svakom slučaju korisnik mora da bude obavešten da je greška detektovana.
2. Da li je detekcija grešaka aktivna ili pasivna? Sistem može da aktivno predviđa greške (npr. provera validnosti unetih podataka od strane korisnika) ili može pasivno da odgovara na greške onda kada je nemoguće izbeći ih (npr. kada kombinacija unosa od strane korisnika doveđe do numeričkog prekoračenja). Drugim rečima, sistem može da raščišćava put ili da čisti za sobom. U oba slučaja izbor alternative ima implikacije na korisnički interfejs.
3. Kako program prosleđuje informaciju o grešci? Kada se otkrije greška, može da se odluči da se podaci koji su doveli do nje odmah ponište; odluči da se greška testira kao greška ulaskom u stanje obrade greške, ili da se čeka završetak svih obrada i da se tada korisnik informiše o tome da je tokom rada otkrivena greška.
4. Koja je strategija za rukovanje greškama? Ako arhitektura ne specificira jedinstvenu konzistentnu strategiju, tada se korisnički interfejs pojavljuje kao konfuzan kolaž različitih interfejsa i različitih delova programa, te je neophodno utvrditi strategiju za rukovanje greškama kako bi se izbeglo ovakvo ponašanje sistema.
5. Na kom nivou se rukuje greškama unutar programa? Greškama se može rukovati na mestu otkrivanja, mogu se prosleđivati klasi za rukovanje sa greškama ili se mogu prosleđivati naviše kroz lanac poziva.
6. Koja je nivo odgovornosti u validaciji ulaznih podataka za svaku od klasa? Da li je svaka klasa odgovorna za validaciju svojih podataka ili postoji grupa klasa odgovorna za validaciju podataka u sistemu? Da li klase na nekom od nivou mogu pretpostaviti da su podaci koje oni prihvataju čisti?

7. Da li se koristi mehanizam za rukovanje izuzecima koji je ugrađen u okruženje, ili se pravi sopstveni? Čak iako okruženje podržava konkretan pristup za rukovanje greškama, to ne mora značiti da je u našem slučaju taj prisut najbolji.

Dokument o arhitekturi takođe, treba da ukaže na očekivani nivo tolerancije na otkaze (engl. *fault tolerance*). Tolerancija na otkaze je skup tehnika koji uvećava pouzdanost sistema tako što otkriva greške, oporavlja sistem od grešaka (ukoliko je to moguće), i zaustavlja loše efekte grešaka ako je oporavak moguć.

Na neki od sledećih načina sistem može obezbediti neka operacija bude otporna na otkaze:

1. Sistem može čuvati međurezultate i pokušati ponovo kada detektuje otkaz. Ako je prvi odgovor pogrešan, on se može vratiti do tačke na kojoj je sve bilo u redu, pa nastaviti od te tačke.
2. Sistem može da ima pomoćni kod koji se koristi u slučaju kada se detektuje otkaz. Ako je prvi odgovor pogrešan, sistem se može prebaciti tako da koristi neku alternativnu rutinu.
3. Sistem može da koristi algoritam glasanja. Može da ima tri klase sa metodima koje obrađuju podataka, pri čemu svaki od metoda radi na drugačiji način. Po završetku svakog od metoda upoređuju se dobijeni rezultati, u zavisnosti od vrste tolerancije koja je ugrađena u sistem, on kao krajnji rezultat može dati rezultat dobijen od ta tri rezultata, npr. aritmetičku sredinu, medijanu ili nešto treće.
4. Sistem može zameniti pogrešnu vrednost sa nekom lažnom vrednošću za koju se zna da neće imati loš efekat na ostatak sistema.

Ostali pristupi obezbeđenja tolerantnosti na otkaze obuhvataju promenu sistema tako da u slučaju detekcije greške prelazi u stanje delimične operativnosti ili u stanje umanjenja funkcionalnosti. Na taj način on može da se sam isključi ili da se automatski restartuje.

Robusnost je osobina sistema da nastavi sa radom čak i kada se dogodi greška. Često, dokument o arhitekturi, specificira i robusniji sistem nego što se traži u zahtevima. Jedan od razloga za to je što sistem sastavljen od delova koji su minimalno robusni može biti manje robusan u celosti nego što je zahtevano. Arhitektura treba jasno da ukaže da li pri razvoju programeri treba da teže povećanoj intenzitivnosti ili da se orijentisu na pravljenje najjednostavnije stvari koja zadovoljava uslove i koja funkcioniše. Određivanje pristupa u ovom aspektu je veoma važno, zato što mnogi programeri u cilju da se dokažu automatski previše intenzivno razviju svoje klase. Eksplicitnim postavljanjem očekivanja, može se izbeći pojava da jedna grupa klasa bude izuzetno robusna, a druge da budu skoro neadekvatno robusne.

Najradikalnije rešenje u izgradnji softvera je da se softver uopšte ne gradi, već da se umesto toga kupi. Moguće je kupiti kontrole grafičkog interfejsa, komponente za grafike i

dijagrame, komponente za sigurnost i enkripciju itd. Jedna od najvećih prednosti programiranja u modernom okruženju je količina funkcionalnosti koja je automatski na raspolaganju. Ako arhitektura ne koristi kupljene komponente tj. komponente na raspolaganju, onda treba da opiše način na koji samostalno izgrađene komponente treba da nadmaše komponente koje su već na raspolaganju.

Ako plan razvoja zahteva korišćenje softvera koji već postoji, tada dokument o arhitekturi treba da objasni kako softver koji se ponovo koristi treba da bude u saglasnosti sa novim zahtevima.

Budući da je izgradnja softverskog proizvoda istovremeno i proces učenja i za programere i za korisnike, može se očekivati da će tokom svog razvoja proizvod da se menja. Promene proističi iz promenjivih tipova podataka i formata datoteka, iz promenjive funkcionalnosti, iz novih karakteristika itd. Promene mogu biti nove sposobnosti koje mogu nastati iz planiranih proširenja, ili mogu biti mogućnosti koje nisu uključene u prvu verziju sistema. Prema tome, jedan od najvećih izazova sa kojima se suočava arhitektura softvera je kreiranje arhitekture koja je dovoljno fleksibilna da se može prilagoditi očekivanim promenama. Arhitektura treba da sadrži i opiše jasnu strategiju za rukovanje promenama, da prikaže da su razmotrena moguća proširenja i da su u pitanju proširenja koja su najverovatnije istovremeno i najlakša za implementaciju. Ako se očekuju promene u ulaznim ili izlaznim formama, u smislu interakcije sa korisnikom, ili u zahtevima procesiranja, arhitektura treba da pokaže da su sve promene bile predviđene i da efekti ma koje od njih bivaju ograničeni na mali broj klasa.

Plan promena može biti jednostavan, kao što je rezervisanje polja za buduće korišćenje, ili dizajn baze podataka tako da se lako mogu dodati nove tabele. Ukoliko se koristi neki od generatora koda, arhitektura treba da pokaže da su predviđene promene u okviru sposobnosti generatora koda. Takođe, treba da ukaže na strategije koje se koriste radi prolongiranja obavezivanja.

Dobra specifikacija arhitekture se karakteriše razmatranjima o klasama u sistemu, o informacijama koje su skrivene od drugih klasa i o razlozima za uključivanje i isključivanje svih mogućih alternativa u dizajnu. Arhitektura treba da predstavlja konceptualnu celinu sa nekoliko dodatnih „ad-hok“ definicija. Esencijalni problem sa velikim sistemima je u održavanju njihovog konceptualnog integriteta. Dobra arhitektura treba da se uklopi u problem. Kada posmatrate arhitekturu morate imati osećaj da je rešenje lako i prirodno, arhitektura ne sme da odaje sliku da su rešenja privremeno „pomoću štapa i kanapa“ povezana u problem. Svaka od promena tokom razvoja arhitekture treba da se čisto i precizno uklopi u osnovni koncept. Dizajn sistema čiji je primarni cilj mogućnost luke modifikacije se razlikuje od dizajna sistema u kome je cilj bezkompromisno postizanje performansi – čak i u slučaju da oba sistema imaju istu funkciju.

Arhitektura treba da opiše motivaciju za svaku od najvažnijih odluka.

Dobra arhitektura softvera je u velikoj meri nezavisna od računara i nezavisna od jezika, ukoliko se ne radi o programu koji se izvršava samo na jednoj vrsti računara ili na

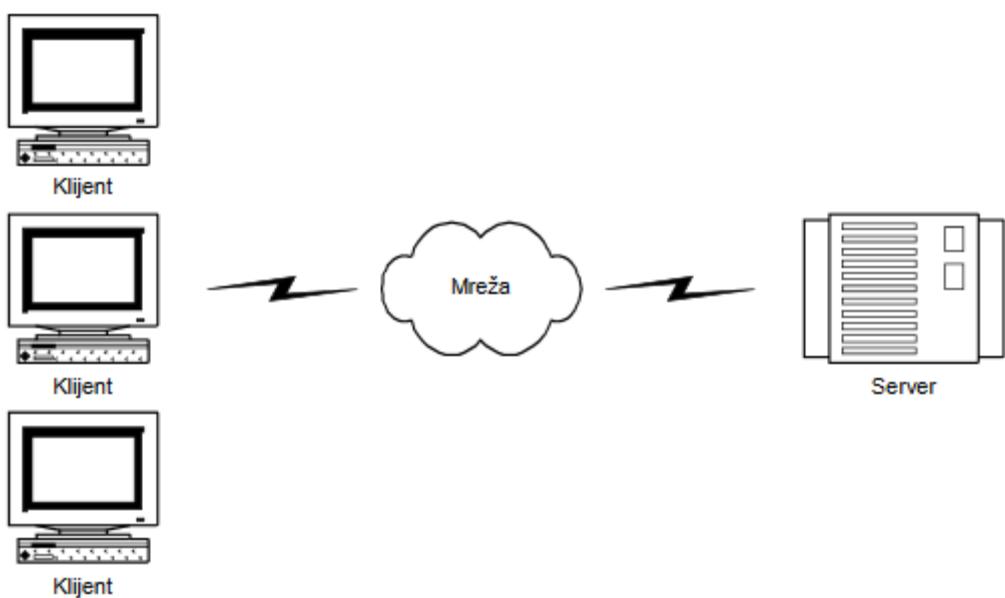
jednom jeziku. Naravno, ne može se potpuno ignorisati okruženje konstrukcije. Time što smo u najvećoj meri nezavisni od okruženja, lakše ćemo se odupreti iskušenju da previše intenzivno napravimo arhitekturu sistema ili da radimo posao koji se može bolje uraditi u fazi konstrukcije. Potrebno je postaviti jasnu liniju između nedovoljno specificiranog sistema i previše intenzivnog specificiranog sistema. Prilikom generisanja arhitekture moramo obratiti pažnji da nijedan problem koji se razmatra ne dobije više pažnje nego što zасlužuje, inače će rešenje takvog problema biti previše intenzivno dizajnirano, dok će rešenja drugih problema biti usvojena bez dovoljnog razmatranja, da bi se uštedelo na vremenu koje se izgubilo. Arhitektura treba eksplisitno da identifikuje rizična polja, da objasni zašto su ta područja rizična i koji su koraci preduzeti u cilju minimizacije rizika.

Potrebno je obratiti pažnji da dokument o arhitekturi ne sadrži nejasne, teško razumljive ili dvosmislene definicije nekog dela sistema, jer može dovesti do loše konstrukcije sistema.

Količina vremena koja je potrebna za definiciju problema, za zahteve i za softversku arhitekturu varira u zavisnosti od potreba samog projekta. Ako su zahtevi nestabilni a u pitanju je veliki formalni projekat, verovatno će biti potrebno odvojiti vremena za rad sa analitičarem zahteva kako bi dodatno pregledali zahteve i razrešili probleme zahteva koji su identifikovani u ranoj fazi konstrukcije, i na taj način dobili verziju zahteva pogodne za dalji rad. Ukoliko su zahtevi nestabilni, a u pitanju je manji neformalni projekat, potrebno je odvojiti vremena da se dobije spisak dovoljno dobro definisanih zahteva, i to tako definisanih da njihovo eventualno menjanje u budućnosti ima minimalni uticaj na konstrukciju. Na pojedinim projektima, ukoliko su zahtevi nestabilni, onda se rad sa zahtevima tretira kao poseban projekat. Ukoliko zahtevi nisu dobro urađeni, može se dogoditi da se izostave važni detalji u softverskoj arhitekturi. Izmene u zahtevima koštaju 20 do 100 puta više ako se realizuju u kasnijim fazama, te je pre početka programiranja i realizacije sistema potrebno biti siguran da su zahtevi pravi i jasno postavljeni. Tek kada se dobije spisak jasno definisanih zahteva moguće je raditi na proceni i rasporedu vremena.

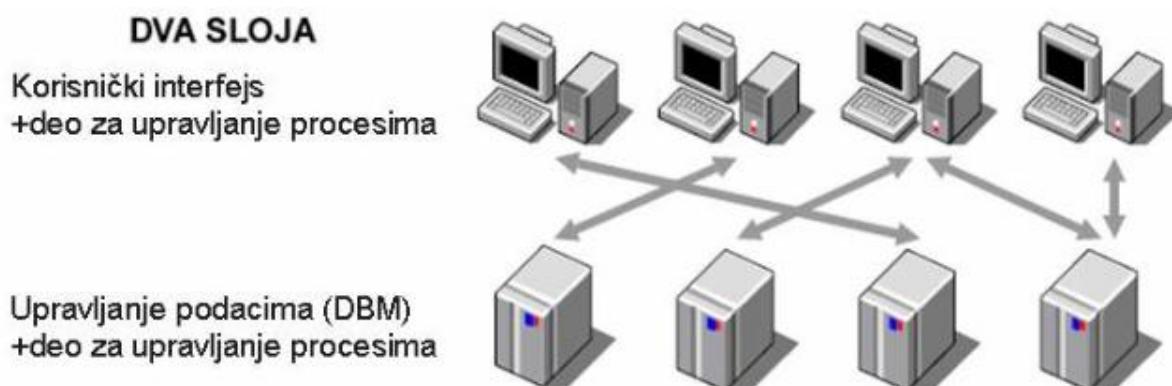
## Arhitektura web aplikacija

Nakon pažljive analize korisničkih zahteva, neophodno je doneti odluku o arhitekturi sistema. Odluka mora biti doneta na osnovu trenutnih potreba i budućeg razvoja. Odabir arhitekture zavisi od broja korisnika i računara na mreži, vrste razvojnih okruženja i programskih alata, modela i obima baze podatka, složenosti programskih procedura. Web aplikacije su dominantno bazirane na klijent/server modelu arhitekture. Klijent/server arhitektura je razvijena kao višenamenska, modularna infrastruktura zasnovana na slanju i primanju poruka sa ciljem unapređenja upotrebljivosti, fleksibilnosti, interoperabilnosti i skalabilnosti. U klasičnom sistemu za obradu podataka po klijent/server modelu mogu se uočiti tri klase komponenti: server, klijent i mreža. Namena servera je optimalno upravljanje zajedničkim resursima (najčešće su to podaci), upravljanje bazom podataka kojoj pristupa više korisnika, kontrola pristupa i bezbednost podataka, i centralizovano obezbeđenje integriteta podataka za sve aplikacije. Klijent-aplikacije vrše upravljanje korisničkim interfejsom i izvršavaju deo logike aplikacije. Računarska mreža i komunikacioni softver omogućavaju prenos podataka između klijenta i servera.

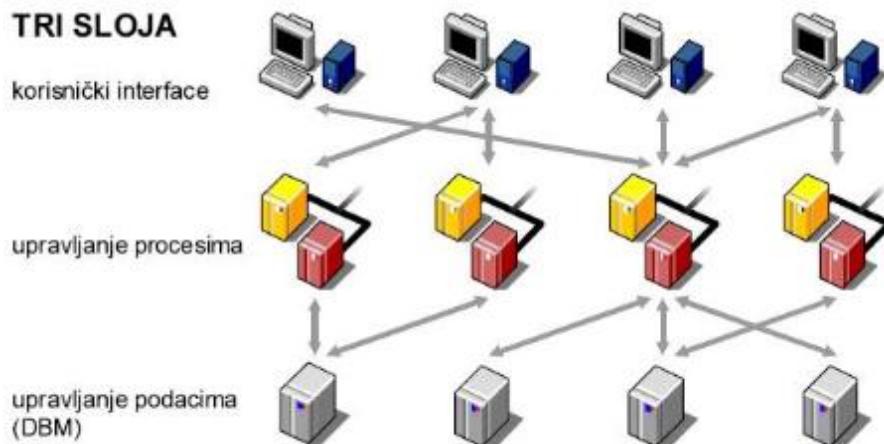


Dvoslojna arhitektura sastoji se od 3 komponente distribuirane u dva sloja – klijentskom i serverskom. Te tri komponente su: *korisnički interfejs* (prikaz podataka na ekranu, sesije, unos teksta, dijalozi...), *upravljanje procesima* (generisanje, izvođenje i nadgledanje procesa i neophodnih resursa), i *upravljanje podacima* (servisi vezani za deljenje podataka i datoteka). Jedna od osnovnih karakteristika klijent/server sistema je distribuirana obrada podataka – logika aplikacije je podeljena između klijenta i servera tako da obezbedi optimalno korišćenje resursa. Na primer, prezentacija podataka i provera ulaznih podataka su sastavni deo klijent-aplikacije, dok se rukovanje podacima, u smislu njihovog fizičkog smeštanja i kontrole pristupa, vrši na serveru. Prednosti ovakvog modela obrade podataka su centralizovano upravljanje resursima sistema i jednostavnije obezbeđenje sigurnosti podataka, dok je nedostatak skalabilnost. Pod skalabilnošću se podrazumeva osobina sistema

da omogući efikasan rad velikom broju korisnika, i da dalje povećanje broja korisnika ne izaziva drastičan pad performansi sistema.

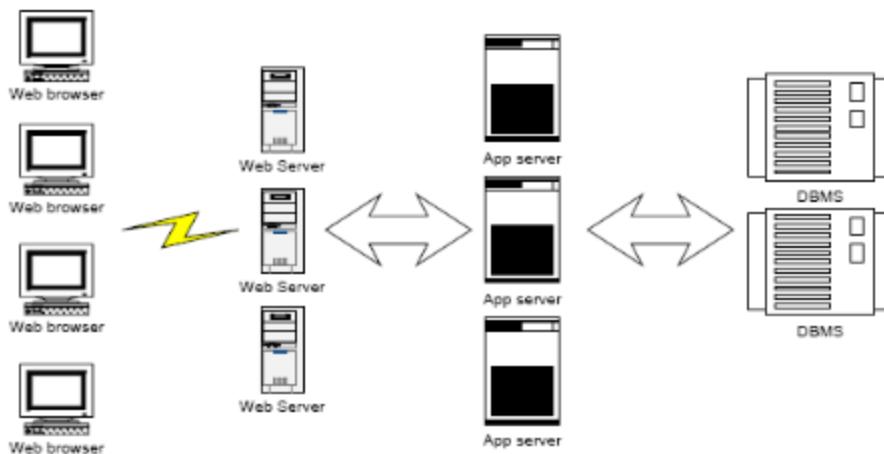


Klijent/server sistemi sa troslojnom arhitekturom predstavljaju sisteme sa tri, u velikoj meri nezavisna, podistema. *Podistem za interakciju sa korisnikom* (implementira funkcije korisničkog interfejsa), *podistem za implementaciju osnovnih funkcija sistema* – poslovnu logiku, i *podistem za rukovanje podacima*, pri čemu se pre svega misli na fizički smeštaj podataka (sistem za upravljanje bazama podataka). Za razliku od dvoslojnog modela obrade podataka gde je logika aplikacije bila podeljena između klijenta i servera, u troslojnom modelu ona se nalazi koncentrisana u takozvanom aplikacionom serveru čija je namena da izvrši programski kod koji implementira logiku aplikacije. Klijent aplikacija je namenjena samo za implementaciju korisničkog interfejsa, a funkcija sistema za upravljanje bazom podataka je isključivo fizičko rukovanje podacima. Troslojni koncept je doveo do podele programskog koda na segmente koji implementiraju tačno određene funkcije sistema. Tako organizovani sistem je jednostavniji za održavanje, jer je moguće nezavisno razvijati korisnički interfejs i logiku aplikacije. Za potrebe fizičkog rukovanja podacima najčešće se koristi neki od komercijalno dostupnih servera za tu namenu. Troslojne arhitekture sistema podrazumevaju oslanjanje na standarde u odgovarajućim oblastima, zasnovane na internet tehnologijama. Oslanjanje na standarde omogućava integraciju sistema heterogenih u pogledu korišćene hardverske i softverske opreme. Važna karakteristika troslojnih sistema je skalabilnost u pogledu povećanja broja klijenta. Povećanje propusne moći i brzine odziva servera srednjeg sloja je moguće kroz dodavanje novih serverskih mašina uz korišćenje postojećih. Sistem sa više servera karakteriše i povećava pouzdanost i fleksibilnost. Logika aplikacije se može menjati i u toku rada sistema. Moguće je efikasno vršiti balansiranje opterećenja serverskog podistema.



Daljim proširivanjem koncepta troslojnog sistema dolazi se do pojma višeslojnih sistema (engl. *multitier architecture*), gde se vrši dalja podela komponente u okviru srednjeg sloja sa ciljem još većeg povećanja skalabilnosti, odnosno performansi.

Srednji sloj je podeljen na dva sloja: jedan je namenjen za opsluživanje Web klijenata, a drugi sadrži komponente koje implementiraju poslovnu logiku sistema.



## Model-View-Controller aplikacije

Programski jezik PHP (*Hypertext Preprocessor*) se već više od 15 godina koristi za izradu dinamičkih web stranica. Inicijalno su se web stranice generisale pomoću HTML jezika, dok su se dinamičke karakteristike dodavale pomoću PHP jezika koji se uklapao u postojeću HTML stranicu. Ovakvo ispreplitanje dva programska jezika predstavlja pogodnost budući da se na jednostavan način dodavala funkcionalnost web stranicama. Velika popularnost PHP jezika dovodi do razvijanja velikih web aplikacija pisanih na ovom jeziku. Ubrzo postaje jasno da ispreplitanje dva programska jezika ne predstavlja dugoročno rešenje za velike web aplikacije. Problemi se ogledaju u skalabilnosti i teškom održavanju velikih projekata. Na ovakav način, mnogo je teže dodavati nove funkcionalnosti i ispravljati programske greške ukoliko se funkcionalni deo koda nalazi zajedno sa HTML prikazom. Jedna od osobina web aplikacija je često menjanje grafičkog interfejsa. Ukoliko se funkcionalni deo koda usko veže za HTML nije moguće zameniti izgled stranice bez menjanja postojeće funkcionalnosti koda. U cilju rešavanja ovog problema nastala su razvojna okruženja, a jedno od najpopularnijih razvojnih okruženja za programski jezik PHP je Zend Framework. Razvojna okruženja služe za potporu izgradnje i testiranje dinamičkih web stranica i aplikacija. Takođe, pomažu u razrešavanju učestalih problema koji se pojavljuju prilikom projektovanja web aplikacija. Često okruženja nude gotov skup biblioteka i alata za pristup bazi podataka, izradu i upravljanje šablonima, upravljanje korisničkom interfejsom te olakšavaju ponovnu upotrebu programskog koda.

Mnoga razvojna okruženja prate MVC strukturalni šablon (engl. *Model-View-Controller*). Osnovna prednost MVC arhitekture je razdvajanje projekta u smislene i odvojene celine, što predstavlja veliku prednost pri izradi velikih projekta na kojima radi više osoba. Iz ove osobine proizilazi i druga prednost koja se odnosi na laku izmenu, nadogradnju i budući razvoj. MVC pristup omogućuje laku promenu jednog od elementa bez velike intervencije u drugim elementima, kao i ponovno korišćenje već napravljenih elemenata. U ovakvoj arhitekturi *model* predstavlja telo, *view* oči, a *controller* je mozak projekta.

Zend okruženje je stvoreno sa ciljem jednostavnijeg stvaranja i održavanja velikih web aplikacija zasnovanih na PHP jeziku, sadrži bogat skup gotovih komponenti koje pokrivaju veliki deo učestalih problema prilikom izrade web aplikacija, dok je MVC šablon ključni deo Zend okruženja.

## MVC šablon

U programskom inženjerstvu šabloni predstavljaju skup ispravnih rešenja za česte probleme koji se pojavljuju tokom razvoja aplikacija. Neki od čestih problema odnose se na smanjivanje sprege između pojedinih komponenata, odvajanje interfejsa od poslovne logike te nezavisnost od baze podataka. Šabloni ne predstavljaju gotov deo programskog koda nego opisuju na koji način rešiti problem. MVC šablon je programska arhitektura za odvajanje programskog koda koji prezentuje problem korisniku od ostalog dela aplikacije, tj. razdvajanje modela podataka sa poslovnom politikom od korisničkog interfejsa. MVC se koristi za odvajanje pojedinih delova aplikacije i njenih komponenata u zavisnosti od njihove namene. Na ovaj način se omogućava jednostavno menjanje i stvaranje novih grafičkih maski bez promene koda vezan za poslovnu logiku aplikacije, olakšava nezavisan razvoj, testiranje i održavanje aplikacije, podelu posla među programerima u timu. MVC arhitektura se upotrebljava kako bi aplikacija bila stabilnija, sigurnija i veoma jednostavna za nadogradnju i kasnije proširenje.

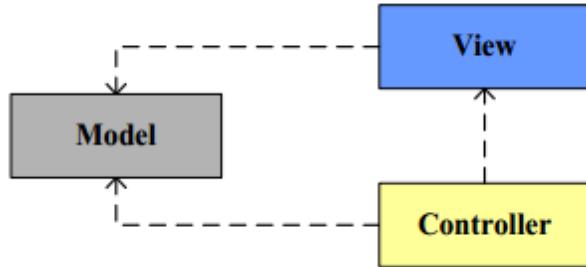
MVC arhitekturu je prvo bitno formulisao krajem sedamdesetih godina prošlog veka norveški informatičar Trigvi Riensku (norv. *Trygve Mikkjel Heyerdahl Reenskaug*), kao delo Smalltalk-76 programskog jezika. Kompanija Xerox je prva kompanija koja je primenila MVC model i to u biblioteci klase jezika Smalltalk-80. MVC je prvi put predstavljen 1979. godine. U to vreme koncept web aplikacija nije postojao, te je najpre zamišljen kao koncept za desktop aplikacije. MVC koji danas koristimo u web programiranju adaptacija prvo bitne arhitekture i danas predstavlja jedan od najčešće korišćenih arhitektura. Do popularizacije ovog šablonu u web programiranju dovela su dva najkorišćenija razvojna okruženja *Struts* i *Ruby on Rails*. Ova dva razvojna okruženja su otvorila put hiljadama drugih razvojnih okruženja koja se danas koriste u razvoju web aplikacija.

Prvi zvaničan rad u kojem je opisana MVC arhitektura izšao je 1988. godine i nosio je naziv „*A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*“, a autori su bili Glenn E. Krasner i Stephen T. Pope. Danas se MVC koristi u velikom broju modernih web i GUI razvojnih okruženja kao što su: *CodeIgniter*, *CakePHP*, *Zend Framework*, *Ruby on Rails*, *Apple Cocoa*, *Apache Struts*...

U osnovi ove arhitekture nalaze se dve centralne ideje: ponovno korišćenje već postojećeg koda i jasna raspodela zaduženja među različitim delovima sistema. Prva ideja omogućava da se jednom napisan kod uz minimalne ili čak nikakve izmene može koristiti u više različitih projekata. Druga ideja je podela sistema na više međusobno nezavisnih sistema od kojih svaka ima svoj zadatak i koje se pojedinačno mogu modifikovati, bez bojaznosti da će se te promene odraziti na ostatak sistema.

Suština MVC obrasca je razdvajanje prezentacijskog sloja od same logike aplikacije. Prezentacijski sloj ili pogled (engl. *View*) je, jednostavno rečeno, interfejs koji korisnik vidi i koji obezbeđuje interakciju sa korisnikom. Model (engl. *Model*) predstavlja logiku same aplikacije, odnosno podatke i strukturu pomoću kojih operiše. Logika MVC obrasca je takva da model „ne zna“ za postojanje pogleda i obratno. Kontroler (engl. *Controller*) ima ulogu

koordinatora. Poštovanjem logike MVC aplikacije, svaki web zahtev dolazi direktno do odgovarajućeg kontrolera, koji pomoću modela generiše potrebne podatke, prosleđuje ih pogledu i na kraju odgovarajući pogled vraća klijentu, tj. web pretraživaču. Ideja MVC arhitekture je jednostavna i zasniva se na podeli odgovornosti između ove tri glavne komponente, od kojih svaka obavlja različite zadatke.



Važno je zapaziti da pogled i kontroler zavise od model komponente, dok sa druge strane, model ne zavisi niti o pogled komponenti, niti o kontroler komponenti. Ova nezavisnost model komponente je vrlo važna jer omogućava da ta komponenta bude razvijana i testirana nezavisno od pogleda tj. prezentacijske logike.

Odvojenost pogled i kontroler komponente je sekundarne prirode čak kod nekih varijanti Model-View-Controller šablonu te dve komponente se udružuju u jednu. Ta vrsta implementacije je često zastupljena kod klijentskih desktop aplikacija, dok je kod web aplikacija granica između ovih komponenti izražena, budući da se prezentacijski sloj nalazi na strani korisnika i vidljiv je kroz web pretraživač, dok se zahtevi korisnika obrađuju na serverskoj strani tj. u kontroleru.

## Model

Model (engl. *Model*) sadrži glavne programske podatke kao što su informacije o objektima iz baze podataka, sastoji se od skupa klasa koje modeliraju i podržavaju rešavanje problema kojim se aplikacija bavi, te je taj deo obično stabilan i trajan koliko i sam problem. Sva poslovna logika aplikacije se nalazi u modelu. Postoji nekoliko načina na koje je moguće stvoriti kostur modela. Programer prvo može da pristupi kreiranju modela, definisanju klasa i atributa unutar svake klase, a kasnije na osnovu klasa kreirati potrebne tabele tj. bazu podatka. Ili pristup koji je češći, jeste da se prvo napravi i projektuje baza podataka sa potrebnim tabelama, dok se modeli kasnije generišu na način da prate napravljene tabele. Svakako je potrebno uzeti u obzir veze među tabelama u bazi podatka, tj. veze među modelima u aplikaciji. Model se u MVC arhitekturi sastoji od modela područja (engl. *Domain model*), i modela aplikacije (engl. *Application model*). Model područja sadrži glavne podatke o samoj tabeli tj. njenu strukturu, dok model aplikacije sadrži sve potrebne funkcije za upravljanje podacima tj. za upis, izmenu i brisanje podataka. Zavisno od potreba, razlikuju se tri vrste modela: konceptualni, logički i fizički model. Konceptualni sadrži samo prikaz naziva entiteta tj. tabela. Logički model sadrži i nazive atributa u tabelama, primarnih i stranih ključeva. Fizički model sadrži sve kolone u tabelama, tipove podataka u pojedinim kolonama, i predstavlja način na koji se podaci „vide“ u memoriji.

Svi podaci se dobijaju od modela, ali se on ne može direktno pozvati, već je kontroler taj koji od modela zahteva određene podatke. Model zatim obrađuje zahteve i vraća kontroleru zahtevane podatke, dok ih on dalje prosleđuje pogledu, preko kontrolera, koji ih interpretira krajnjem korisniku. Model predstavlja jednu ili više klase sa svojim stanjima koja se mogu prikazati na zahtev pogleda, a kontroler ih može menjati. Postoje aktivan i pasivan model. Pasivan model je model čije se stanje menja sa učešćem kontrolera, a aktivan model čije se stanje menja bez učešća kontrolera. Međutim, u praksi je češći scenario u kojem model ima aktivnu ulogu i predstavlja deo poslovne logike. U tom slučaju, model obezbeđuje metode koje će koristiti kada se ažurira trenutno stanje objekta. Model takođe može obavestiti pogled da je došlo do promene, da bi se osvežio trenutni prikaz. U osnovnom obliku model nema nikakvo saznanje o pogledu i kontroleru dok u proširenoj verziji može da sadrži osnovne operacije za rad sa „Posmatrač“ šablonom (engl. *Observer pattern*).

## View

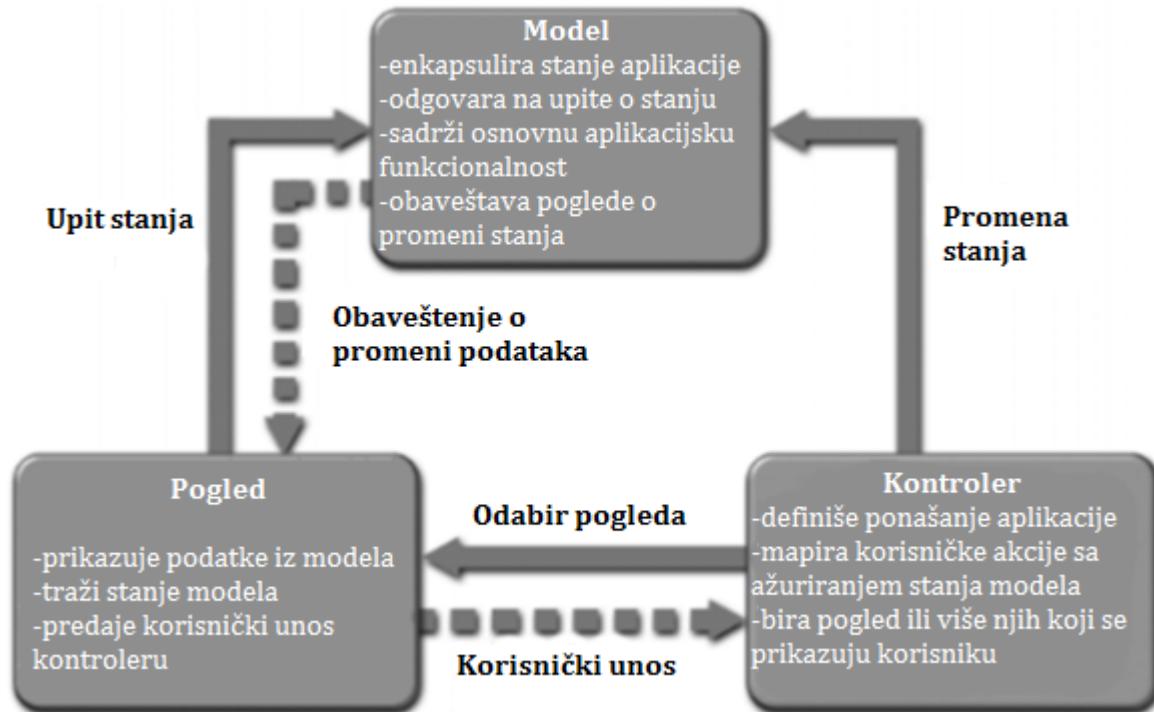
Pogled (engl. View) je poslednji sloj MVC arhitekture koji sadrži okruženje aplikacije, odnosno obezbeđuje različite načine za prezentovanje podataka koje dobija od modela, preko kontrolera. U web aplikacijama pogled sadrži: HTML, CSS, JavaScript, XML ili JSON, itd. Ne preporučuje se korišćenje HTML, CSS ili JavaScript u kontroleru. Korisnik može videti ono što pogled prikazuje, dok su model i kontroler skriveni od korisnika. Najvažnija osobina pogleda je njegova jednostavnost, tačnije pogled predstavlja vizuelni prikaz modela koji sadrži metode za prikazivanje i omogućava korisniku da menja podatke. On svakako ne treba da bude nadležan za skladištenje podataka, osim kada se koristi keširanje kao mehanizam poboljšanja performansi. Može biti odgovoran za prikaz jednog ili više objekata iz modela. Sam pogled nikad ne obrađuje podatke – njegov posao je gotov kada su podaci prikazani. Pogled ima svest o postojanju modela i preko metoda modela dobija podatke koje prikazuje.

## Controller

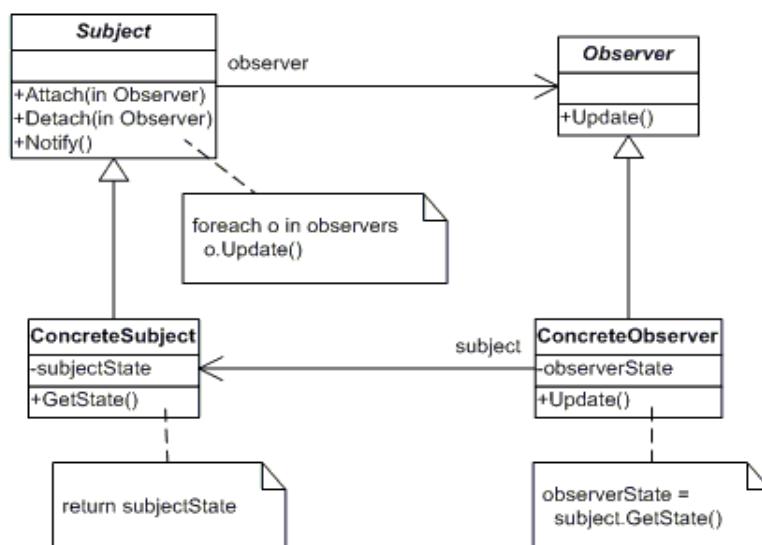
Kontroler (engl. Controller) sadrži glavne mehanizme za kontrolu programa i dogovoran je za njegov tok, tj. definiše ponašanje same aplikacije. U web aplikacijama on predstavlja prvi sloj koji se poziva kada pretraživač pozove URL adresu. Takođe, kontroler upravlja korisničkim zahtevima. Obično kontroler poziva određeni model za zadatak i zatim bira odgovarajući pogled. Dakle, kontroler predstavlja posrednika između pogleda i modela, i ima svest o postojanju i Pogleda i Modela. Kontroler interpretira ulazne podatke korisnika i prosleđuje ih do modela ili pogleda. Za razliku od pogleda, kontroler sadrži deo aplikacijske logike i ima sposobnost da utiče na stanje modela, tačnije, on odlučuje kako model treba da se promeni kao rezultat korisničkih zahteva i koji pogled treba da se koristi. Ovo se najčešće odnosi na obradu podataka koje korisnih zahteva putem pogleda. Kontroler predstavlja most koji može povezati bilo koji i bilo koji broj modela sa bilo kojim ili bilo kojim brojem pogleda i obratno.

## Interakcija komponenti

Osim podele aplikacije u tri odvojene komponente, MVC arhitektura omogućava i interakciju između njih.



Slika prikazuje komunikaciju između tri komponente MVC arhitekture. Broj pogleda u sistemu više ne igra ulogu, jer za jedan model može postojati neograničen broj pogleda, a svaki od njih može biti za drugu namenu (tabelarni prikaz podataka, prikaz podataka u vidu dijagrama i sl.). Najsloženija veza u ovom modelu prestavlja vezu između modela i pogleda. Tačnije, kako propagirati obaveštenje o promeni nekog skupa podatka unutar modela proizvoljnom broju pogleda. Ova veza pogleda se postiže upotrebom šablona „Posmatrač“ (engl. *Observer pattern*), kojim se omogućuje slanje poruka od modela prema pogledima bez stvaranja zavisnosti o konkretnim pogledima. Slika predstavlja dijagram.



Model je centralna komponenta. On direktno upravlja podacima, logikom i pravilima aplikacije. Takođe, on skladišti podatke koji se preuzimaju od strane kontrolera i prikazuju se u pogledu. Kad god postoji promena, kontroler ažurira model. Model ne mora da zna ko su kontroler i pogled.

Kontroler prihvata zahteve od klijana za izvršavanje operacije. Nakon toga poziva operaciju koja je definisana u modelu i, ukoliko model promeni stanje, obaveštava pogled o promeni stanja modela. Kontroler može da šalje komande modelu za ažuriranje njegovog stanja.

Pogled od modela traži informacije koje koristi za generisanje izlaznih podataka. On takođe obezbeđuje korisniku interfejs pomoću koga unosi podatke i poziva odgovarajuće operacije koje je potrebno izvršiti nad modelom. On u stvari, korisniku prikazuje stanje modela. Na slici je prikazana međusobna interakcija između modela, kontrolera i pogleda.

## Prednosti i mane

MVC obrazac ima nekoliko pozitivnih osobina: model se može prikazati na više načina; lakše je dodati novi pogled (na primer: novu internet stranicu koja prikazuje postojeće podatke ili deo njih); interakcija sa korisnikom se lako menja; više programera mogu raditi istovremeno i paralelno na različitim slojevima aplikacije (posebno važno u timovima specijalizovanim za pojedine slojeve, na primer, tim zadužen za implementaciju i održavanje korisničkog interfejsa, poslovne logike...).

Najvažnije prednosti MVC arhitekture su ponovno korišćenje koda i dodela sistema na međusobno nezavisne celine, odakle proizilaze i ostale prednosti. Projekti su mnogo sistematičniji, njihovi pojedinačni delovi se mogu lako menjati i poboljšavati, kod pisani na ovaj način je neuporedivo lakše testirati, paralelni razvoj aplikacije se može lako organizovati i samim tim se povećava produktivnost. Odvojeni prikaz od logike obrade podataka postiže visoku fleksibilnost i može doprineti boljem razvoju i arhitekturi aplikacije.

MVC ima, naravno, i svoje mane. Previše je kompleksan za implementaciju kod razvoja manjih aplikacija, i njegovo korišćenje u tim slučajevima dovodi do pogoršanja kako dizajna tako i performansi. Ponekad se može desiti da, usled čestih promena modela, pogled bude preplavljen zahtevima za izmenu. Ukoliko on služi za prikazivanje sadržaja, kojem je potrebno određeno vreme za stvaranje slike koji prikazuje, česti zahtevi za izmenu mogu dovesti do kašnjenja.

## Popularna MVC okruženja

*CodeIgniter* je web razvojno okruženje „*open source*“ koda pisano u PHP programskom jeziku. Cilj okruženja je ga omogući programerima razvoj složenih aplikacija korišćenjem bogatih biblioteka gotovih funkcija. Sama konfiguracija dosta je jednostavna te nije potrebna baza za svaki kontroler kao što je to slučaj u *CakePHP*-u. Prva verzija okruženja *CodeIgniter* objavljena je 28.02.2006. godine i od tada do danas prešao je veliki put i razvio se u jedan od najboljih PHP frameworka (trenutna verzija je 3.1.0). Baziran je po

MVC šablonu. Jedan od tvoraca programskog jezika PHP, Rasmus Ledfors, je na konferenciji 2008. godine pohvalio *CodeIgniter* okruženje zbog visokih performansi, velike zajednice korisnika i jednostavnosti korišćenja. Instalacija *CodeIgnitera* je jednostavna i lako se konfiguriše. Pogodan je za izradu većih aplikacija te omogućava povezivanje sa velikim brojem tipova baza (MySQL, MySQLi, MS SQL, Oracle, SQLite, ODBC) kao i povezivanje sa više baza podataka unutar jedne aplikacije. Okruženje je podržano dobrom dokumentacijom i visokim kvalitetom kodiranja. Loša strana ovog okruženja je slabija podrška za OOP strukturu.

*CakePHP* je web razvojno okruženje „*open source*“ koda izrađeno i programskom jeziku PHP koje pruža proširivu arhitekturu za razvoj, održavanje i implementaciju programa. Razvoj Cake-a počinje u aprilu 2005. godine. Inspirisan programom *Ruby on Rails*, Michal Tatarynowicz objavio ga je pod MIT licencom i omogućio da ga koriste zajednice programera. U julu 2005. godine, Larrz E. Masters, poznatiji kao *PhpNut*, preuzeo je ulogu glavnog programera ovog frameworka. Od tada je nastalo nekoliko projekata i još uvek se radi na razvoju novih. Dobro dokumentiran framework i oblikovan pomoću koncepcata *Ruby on Rails* razvojnog okruženja, može se koristiti u PHP4 i PHP5 verziji. Iako koristi mnoge koncepte *Ruby on Rails* okruženja, ne radi se o pokušaju prevođenja tog jezika u PHP programske jezike. *CakePHP* se koristi za razvoj raznolikih aplikacija, olakšava komunikaciju korisničkog interfejsa sa bazom podataka, brz je, baziran na MVC arhitekturi te je time fokusiran na objektno-orientisano programiranje. *CakePHP* je jednostavan za korišćenje pogotovo kod početnika. Jedna od karakteristika je automatsko izvršavanje određenih akcija. Ova karakteristika je veoma neobična za većinu programera, jer može doći do izvršavanja neke komande koja nije predviđena, bilo da je ona korisna ili ne. Međutim, ona doprinosi na brzini razvoja koja je pogodna za razvoj manjih aplikacija. Nedostatak ovog okruženja je manja fleksibilnost. Pomoću alata *Bake*, moguće je koristiti *CakePHP* iz komandne linije, na vrlo brz i lak način za svega nekoliko minuta može se kreirati potpuna aplikacija. Zbog unapred pretpostavljenog ponašanja komponenti teško je unositi veće promene bez menjanja samog izvornog koda komponente, iz ovog razloga *CakePHP* nije dobar odabir za aplikacije koje zahtevaju česte promene.

*Yii* framework je, takođe, web razvojno okruženje „*open source*“ koda izgrađeno u PHP jeziku. *Yii* je skraćenica od „*Yes It Is!*“. Projekat je započet početkom 2008. godine u cilju poboljšanja *PRADO* web razvojnog okruženja, te čini relativno mlado razvojno okruženje, koje je moguće slabo ispitano i broji znatno manju zajednicu korisnika u odnosu na ostala okruženja. *PRADO* – je web okruženje temeljeno na događajima i komponentama, koji uvodi nove koncepte u razvoj web aplikacija kao što su događaji i svojstva, koji zamenuju postojeće koncepte procedura, URL-a i parametara u upitima koji se javljaju u ostalim okruženjima. Nedostatak *PRADO* okruženja je bio spor odziv prilikom prevođenja složenih web stranica, te nije bio pogodan za početnike i mnoge komponente za obavljanje raznih funkcija kao što su pristup bazi podataka i obavljanje upita, validacija korisničkog unosa, automatsko generisanje kompleksnih WSDL specifikacija za servise i drugo. *Yii* predstavlja jedino razvojno okruženje za programski jezik PHP koji nalikuje okruženju *ASP.NET*.

*Symfony* framework je PHP okruženje „*open source*“ koda za web aplikacije. Agencija *Sensi Labs* stvorila je ovaj framework za potrebe vlastitih web stranica, i koristi ga i dan danas. *Symfony* se pojavio 2005. godine pod MIT Open Source licencom, i danas je jedan od vodećih frameworka za PHP razvoj web aplikacija. Odlikuje se bogatim sadržajem, podrškom velike zajednice, profesionalnom podrškom itd. Brojne stranice su izgrđene upravo korišćenjem ovog okruženja (Yahoo, Delylmotion, Opensky, phpBB...). Korisnici su brzo prihvatali *Symfony* okruženje zbog lake prepozнатljivosti i stabilnosti okruženja. Zahvaljujući svojoj aktivnoj zajednici i razvojnim inženjerima, *Symfony* se kontinuirano razvija. Dizajniran je od strane profesionalaca i za profesionalce. Karakteristike ovog okruženja su: brzina, fleksibilnost, bogatstvo resursa, ponovno upotrebljive komponente, konstantna poboljšanja koja dovode do povećane produktivnosti. *Symfony* omogućava korisniku korišćenje delova vlastitog softvera bez korišćenja celog frameworka.

## Zend framework

*Zend Framework* je razvojno okruženje otvorenog „*open source*“ koda stvoreno sa ciljem jednostavnijeg stvaranja i održavanja velikih web aplikacija i servisa zasnovan na PHP programskom jeziku. Zend je u potpunosti implementiran korišćenjem objektno-orientisane paradigmе. Komponente ovog web razvojnog okruženja su oblikovane tako da imaju što manji broj zavisnosti prema drugim komponentama. Na ovaj način je postignuta saglasnost sa jedim od osnovih koncepta objektno-orientisanog programiranja, a to je smanjivanje sprege (engl. *coupling*). Ova pogodnost omogućava korišćenje individualnih komponenata po potrebi. Zend zajednica ovu pogodnost naziva i oblikovanje po potrebi (engl. *use-at-will design*).

Iako ih je moguće koristiti odvojeno, komponente razvojnog okruženja Zend čine snažnu i proširivu biblioteku gotovih rešenja. Cilj Zenda je da pruži programerima robusnu okolinu za razvoj sopstvenih aplikacija i usluga. On nudi MVC implementaciju visoke učinkovitosti, apstraktни sloj nad bazom podataka koji olakšava upravljanje podacima i niz komponenti koje implementiraju prikaz HTML stranica te validaciju i filtriranje korisničkog unosa. Ostale komponente kao što su *Zend\_Auth* i *Zend\_Acl* omogućavaju autentifikaciju i autorizaciju korisnika prilikom pristupa resursu. Prednost Zend Framework razvojnog okruženja je velika količina gotovih komponenata koje se mogu iskoristiti za izradu sopstvenih web aplikacija ili servisa, a te komponente su temeljno testirane i isprobane.

Korišćenjem gotovih testiranih komponenata smanjuje se vreme razvoja. Na primer, validaciju i/ili filtriranje korisničkog unosa je česta operacija u svim web aplikacijama. Zavisno od nemene aplikacije, korisnički unos može biti problematičan za analizu, na primer unos e-mail adrese ili JMBG broja. Zend sadrži niz gotovih komponenata koje implementiraju većinu poslovne logike vezane za filtriranje i validaciju unosa.

Glavni sponzor projekta Zend Framework je kompanija Zend Technologies, ali i mnoge druge kompanije su doprinele razvoju ovog web razvojnog okruženja. Osnivači kompanije Zend Technologies su Andi Gutmans i Zeev Suraski, koji su poznati kroz svoj rad na PHP programskom jeziku. Smatra se da je njihova popularnost imala značajan uticaj na širenje i prihvatanje Zend okruženja u ranim fazama njegovog razvoja. Kompanije kao što su

Google, Microsoft i Strikelron udružile su se sa Zend Technologies kako bi pružili gotove komponente, odnosno interfejsa prema sopstvenim servisima i dugim tehnologijama koje žele pružiti korisnicima Zend razvojnog okruženja. Zend framework poseduje i veliku zajednicu korisnika koji međusobno razmenjuju informacije putem IRC kanala, foruma i elektronske pošte.

Razvoj Zenda je usko vezan za razvoj programskog jezika PHP. Tačnije, nastao je u cilju proširenja jezika i poboljšavanja efikasnosti u izradi aplikacija. Nakon što je PHP 3.0 postao javno dostupan 1998. godine, Andi Gutmans i Zeev Suraski počeli su raditi na njegovom poboljšanju. Cilj im je bio poboljšati performanse složenih aplikacija i podršku za modularnu izradu komponenata. Tadašnja treća verzija Zend okruženja nije bila u stanju da efikasno radi sa složenim tipovima podataka. Svoje poboljšanje prozvali su Zend Engine, sastavljeno od njihovih imena Zeev i Andi. Prvi put predstavljena je 1999. godine, a nova, četvrta verzija programskog jezika PHP je bazirana na Zend Engine-u. Nova verzija je donela brojne pogodnosti kao što su: podrška za više servera, visoke performanse, globalne serverske varijable (`$_GET`, `$_POST`, `$_COOKIE`, `$_SERVER`, `$_SESSION`), sigurniji način upravljanja korisničkim interfejsom.

PHP 4 i dalje nije bio bolji po pitanju objektno-orientisanog programiranja od verzije 3. Iako PHP nije bio stvoren za objektno-orientisano programiranje mnogi programeri su ga ipak koristili. Nova verzija Zend 2 razvojnog okruženja sadrži većinu obeležja objektno-orientisanog jezika kao što su: prosleđivanje referenci (engl. *Pass by Reference*), kopiranje objekta tj. kloniranje, vraćanje objekta. PHP verzija 5 je po uzoru na Zend 2 preuzeila sve karakteristike objektne paradigme te se od verzije 5 programski jezik PHP smatra potpuno objektno-orientisanim jezikom. Neke nove karakteristike, uz one koje uvodi Zend 2, su: destruktori, sakupljači smeća (engl. *Garbage Collection* – predstavlja oblik automatizovanog upravljanja memorije, tj. brisanje objekata iz memorije koji se više ne koriste), vidljivost varijabli objekata (*public*, *private*, *protected*). Usvajanjem objektne paradigme stvorilo se pogodno tlo za dalji razvoj biblioteke gotovih funkcionalnih modula (engl. *Framework*). Gotove komponente bile bi oblikovane po odgovarajućim obrascima što smanjuje međusobnu povezanost i time povećava modularnost. Ta inicijativa s vremenom je sazrela u Zend Framework. Tokom 2010. godine sve više web programera prelazi sa programskom jezikom Java na PHP. Razlog tome je velika jednostavnost jezika PHP u odnosu na Javu. Programski jezik Java je strogo tipizirani potpuno objektno-orientisan jezik koji se pokreće pomoću posebnog virtualnog jezičkog procesora. PHP jezik se uči znatno lakše i mnogo je fleksibilniji u odnosu na Javu, što ga čini pogodnim za početnike. Podrškom za objektno-orientisano paradigmu i razvojem okruženja poput Zend Frameworka, PHP postaje jezik pogodan za razvoj vrlo složenih web aplikacija. Neke od najpoznatijih web aplikacija današnjice napravljeni su upravo uz pomoć programskog jezika PHP (Wikipedia, Facebook, White House). Autori Zend okruženja smatraju da će kroz nekoliko narednih godina trend razvoja aplikacija korišćenjem PHP jezika naglo rasti, a primarni razlog će biti korišćenje razvojnih okruženja poput Zenda. Budućnost razvoja samog razvojnog okruženja Zend biće usmerena ka računarstvu zasnovanom na oblacima.

Za razliku od mnogih drugih okruženja, gde su biblioteke usko povezane, u Zend Framework okruženju biblioteke su skoro potpuno nezavisne, tako da svaka može da čini zasebnu komponentu. Ova karakteristika dozvoljava da se iskoristi samo jedan deo biblioteke, na primer, ukoliko je potrebno uraditi samo autentifikaciju korisnika moguće je iskoristi samo *Zend\_Auth* biblioteku, bez korišćenja ostatka okruženja. Komponente Zend okruženja mogu promeniti svoje ponašanje jednostavnom manipulacijom postavki putem definisanog interfejsa što ga čini veoma fleksibilnim, te se Zend koristi u većim projektima koji su podložni čestim promenama zahteva.

Kako PHP jezik ne propagira neki standard prilikom kodiranja, teško je održati konzistentnost velikih aplikacija, nedostatak striktnosti dozvoljava programerima da odstupe od ustavljene prakse, što dodatno može da naruši sigurnost same aplikacije. Zend sa druge strane, ide u korak sa trenutno najboljom praksom za implementiranje važnih funkcija, kao što su sigurnost aplikacije, provera unetih podataka i druge, pa se korišćenjem ugrađenih biblioteka poboljšava kvalitet koda, i smanjuje podložnost napadima. Ukoliko je projekat rađen u Zendu, mnogo je lakše snaći se u tuđem kodu i to zahteva manje vremena, nego da je to slučaj slobodnog izbora programera i pisanje koda bez ikakvih pravila.

Zend je zasnovan na paradigmama objektno-orientisanog programiranja, te se na taj način izbegava dupliranje koda, što štedi vreme programerima, i daje mogućnost da se jednom napisan kod ponovo koristi. Ovo je posebno važno za web aplikacije, jer ako se javi potreba za promenom interfejsa aplikacije, a funkcionalnost ostaje ista, potrebno je samo definisati novi pogled, ili više njih, koji na drugačiji način predstavljaju podatke dobijene iz kontrolera, što pojednostavljuje ceo proces.

Zend je „*open source*“ projekat. Iako je stvoren od strane kompanije Zend Technologies, programeri širom sveta rade na većem delu razvoja, ispravljajući greške i dodavajući nove funkcije. Pored dokumentacije, veliku podršku i pomoć prilikom rada možete dobiti od zajednice koja prati i koristi Zend. U slučaju da se pojavi neka greška ili neki problem prilikom rada, u vrlo kratkom roku, nekad za nekoliko sati, možete doći do rešenja na nekim od forumima, gde možete naći i neke predloge boljih rešenja. Zend okruženje se koristi bez naknade za licencu, i bez kupovine skupog hardvera ili softvera.

Zend poseduje i aktivan tim koji stalno radi na novim verzijama okruženja, pa se dešava da mesečno izađu jedna, dve, a nekada i tri nove verzije. Postoje i verzije koje predstavljaju kandidate i daju smernice zajednici šta mogu da očekuju u nekom od narednih verzija. Kako se Zend sastoji od nezavisnih komponenti, tako postoje testovi za svaku od njih posebno. Kada se uključi neka nova komponenta u Zend okruženje, ona mora da sadrži i testove napisane u nekom od *PHPUnit* okruženja za testiranje.

Zend prate i alati za generisanje koda. Generatori omogućuju kreiranje jedne komponente Zend aplikacije uz malo truda. Generisanje koda je implementirano pokretanjem određenog skupa naredbi preko komandne linije, i na taj način je moguće kreirati kontroler, pogled, akcije, forme, modele, pored onih koje se automatski generišu prilikom kreiranja Zend projekta.

Zend sadrži podršku za veliki broj alata i tehnologija kao što su Google Data Api, Microsoft CardSpace, Adobe Action Message Format (AMF), web servise sa internet sajtova Amazon, Yahoo, Twiter... i mnoge druge alate. Jedna od najvažnijih osobina Zend-a je podržavanje skoro svih tipova baza podataka: MySQL, Oracle, Microsoft SQL, PostgreSQL. Pristup podacima se vrši preko komponente *Zend\_Db*. Zend podržava i slanje elektronske pošte SMTP, POP3, IMAP protokolima, i kreiranje web servisa koristeći SOAP i REST protokole. Ima mogućnost kodiranja i dekodiranje JSON (JavaScript Object Notation) objekata. Kreiranje i manipulacija PDF, XLSX, DOCX, ODT i drugih dokumenata.

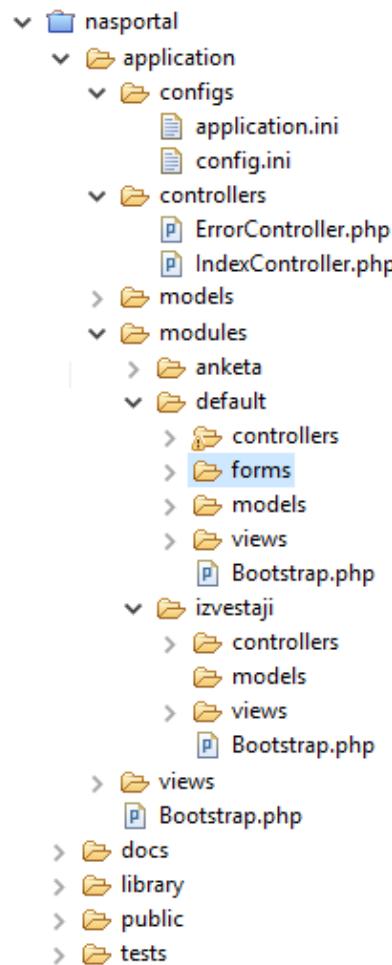
Zend okruženje dolazi sa obimnom dokumentacijom koja sadrži vodič za programere sa preko hiljadu strana, ubrzani kurs za iskusne programere, detaljne opise, video i audio materijale snimljene od strane poznatih Zend inženjera.

## Aplikacija „Nastavnički portal“

Aplikacija „Nastavnički portal“ predstavlja proširenje već postojeće aplikacije „Fakultis“ koji predstavlja informacioni sistem fakulteta, softver za sveobuhvatno praćenje aktivnosti studenata od samog konkurisanja za polaganje prijemnog ispita pa do dobijanja diplome. Nastavnički portal je projekat koji u osnovi predstavlja modul za komunikaciju nastavnika sa studentima. Ideja je da svaki od nastavnika ima svoj nalog pomoću kojeg pristupa aplikaciji u kojoj ima mogućnost pregleda rasporeda časova svih predmeta za koje je angažovan, pregleda spiska studenata koji su se prijavili da slušaju predmet, vođenje evidencije prisutnosti studenata na predavanjima/vežbama, otpremanje datoteke vezana za određeni predmet, zakazivanje predavanja, kolokvijuma i ispita rezervisanjem potrebnih resursa, slanje obaveštenja celoj grupi studenata, otkazivanje predavanja zbog nemogućnosti održavanja, evidentiranje uspeha svakog od studenata na pojedinačnim delovima ispita. Ideja ovog projekta je da se olakša administrativni deo svakog od predavača i mogućnost pristupa važnim informacijama od strane studenata.

Nastavnički portal se jednim delom oslanja na već postojeće tabele u MySQL bazi podataka, dok su za potrebe portala kreirane nove tabele. Za implementaciju je odabранa MVC arhitektura u Zend Framework okruženju, dok je za korisnički interfejs korišćen već postojeći dizajn.

Tipična Zend Framework aplikacija se sastoji od osnovnih direktorijuma koji razdvajaju različite delova aplikacije.



Svaka datoteka sadrži određene delove aplikacije. *Application* – sadrži programski kod potreban za pokretanje aplikacije. Unutar direktorijuma nalaze se dodatni direktorijumi pomoću koji se odvijaju pojedini delovi programskog koda. *Configs* – sadrži globalna podešavanja aplikacije kao što su parametri za pristup bazi, parametri za slanje elektronske pošte i drugo. *Controllers* – sadrži sve kontrolere i kod koji je namenjen za komunikaciju između pogleda i baze podataka. *Models* – sadrži klase koje opisuju svaku od potrebnih tabela iz baze podataka, poslovnu politiku i osnovnu funkcionalnost aplikacije. *Modules* – direktorijum koji postoji u aplikacijama koje se mogu podeliti u nezavisne celine, te svaka od tih celina ima svoj poddirektorijum i u svakom od njih po jedan direktorijum *Controller*, *Models*, *Views*. *Views* – sadrži skup pogleda koji predstavljaju korisnički interfejs. *Docs* – direktorijum je namenjen za čuvanje prateće dokumentacije. *Library* – direktorijum namenjen za čuvanje dodatnih programskih biblioteka (Zend, PHPExcel, PHPWord, TCPDF). *Public* – sadrži javne datoteke kao što su početna stranica *index.php*,

slike i CSS dokumenti koji čine korisnički interfejs. *Tests* – folder namenjen testovima.

Osnovni modul Zend okruženja je *Zend\_Controller* koji implementira glavni kontroler u MVC šablonu. *Zend\_Controller* presreće HTTP zahteve (engl. *HTTP request*) i zavisnosti od formata zahteva pozva odgovarajuće metode/akcije. Zend okruženje svaku stranicu aplikacije smatra jednom akcijom, a akcije su grupisane u kontrolerima. Na primer, sledeći URL

```
http://myServer.com/authentication/login
```

poziva akciju (metodu) *login* koja se nalazi u kontroleru (klasi) *authentucation*. Ukoliko se izostavi naziv akcije, podrazumevana akcija je *index*. U slučaju da aplikacije ima module, pre naziva kontrolera treba da stoji naziv modula, ukoliko je izostavljen podrazumevani modul je *default* modul. Ukoliko se ime modula, klase ili akcije sastoji iz dve reči, na primer da je naziv akcije „delovi ispita“, u URL-u bi stojalo „/delovi-isпита“ dok bi se akcija zvala „*deloviIsпитаAction*“, dakle svako veliko slovo u imenu akcije se u URL-u menja sa „-“ malo slovo. Kako kontroler predstavlja skup akcija, svaki od njih ima skup *phtml* stranica odnosno pogleda/view koje su smeštene direktorijumu koji nosi isti naziv kao kontroler. Direktorijum koji sadrži skup pogleda, nalazi se u istom modulu kao i kontroler, i čini poddirektorijum direktorijuma view/scripts.

Najčešći zadatak programera u izradi web aplikacija je da se napravi sistem prijavljivanja i odjavljivanja korisnika aplikacije. Sa stanovišta baze podataka, potrebna je tabela koja će sadržati podatke o svakom korisniku (korisničko ime, lozinka, email adresa...). Zbog bolje zaštite podataka i sigurnosti svakog korisnika preporučuje se da se u bazi čuvaju lozinke koje su kriptovane nekim od popularnih algoritama (MD5, SHA, SHA1...).

Da bi se korisniku pojavila stranica sa elementima za logovanje, pre svega je potrebno je napraviti kontroler u koji treba dodati akciju i napraviti pogled/view u istoimenom direktorijumu. Akciju ćemo nazvati *loginAction* i formirati je u kontroleru *authentucation*. U direktorijumu view/script istog modula dodaćemo folder sa nazivom *authentucation*, u koji ćemo dodati datoteku *login.phtml*.

Umesto „ručnog“ kreiranja kontrolera, akcije i pogleda moguće je koristiti zend komande upotreboom komandne linije. Komanda

```
zf create controller CONTROLLER_NAME index-action-included[=1]
MODULE_NAME
```

kreiraće kontroler sa željenim nazivom u željenom modulu i dodati *index* akciju po želji. Pored generisanja kontrolera, komanda će napraviti i direktorijum za poglедe na odgovarajućem mestu, i ukoliko je navedeno da doda *index* akciju, generisaće i *index.phtml* datoteku. Za dodavanje akcija u kontroler koristi se komanda

```
zf create action ACTION_NAME CONTROLLER_NAME view-included[=1]
MODULE_NAME
```

koja generiše navedenu akciju u željenom kontroleru sa pogledom ukoliko to želite. Zapravo, ne mora svaka akcija da ima svoj pogled, moguće je da akcija koristi pogled neke druge akcije.

Nakon generisanja kontrolera, akcije i pogleda potrebno je da se na poziv putem URL-a korisniku prikaže stranica tj. forma na kojoj će moći da unese korisničko ime i lozinku i dugme za potvrdu.

Forme se mogu generisati prostim pisanjem *html* koda u *view-u*, ili korišćenjem zend formi. Pisanje *html* koda jeste brži i jednostavniji način, ali je korišćenje zend formi bezbednije i pruža mogućnost korišćenja zend validacije, dok jednom kreirana forma može više puta da se koristi. Za kreiranje forme koristi se klasa *Zend\_Form*. Forme koje predstavljaju klase možete ručno dodati u direktorijum *forms* istog modula, ili korišćenjem zend komande

```
zf create form FORM_NAME  
MODULE_NAME .
```

U kreiranoj klasi forme u *init* funkciji pozivanjem odgovarajućih metoda klase *Zend\_Form*, formi možete dodati sve potrebne elemente, podešiti validaciju za svaki od elemenata, dok pomoću dekoratora možete napraviti željeni raspored elemenata forme.

Korisnik je preko URL-a poslao zahtev kontroleru, tj. pozvao akciju *login* koja treba korisniku da prikaže formu za logovanje, što bi značilo da se u akciji mora kreirati forma i proslediti view-u kako bi je on prikazao korisniku. Korisnik dobija formu za prikaz koju popunjava i poziva istoimenu akciju kojoj prosleđuje popunjenu formu. Ukoliko je forma validna, u ovom slučaju popunjena polja predviđena za korisničko ime i lozinku, akcija treba da prihvati unete podatke i nastavi sa daljim radom, ukoliko forma ne prolazi validaciju korisniku se prikažu poruke tj. razlog zašto nije mogao da se uloguje. Kada akcija dobije korisničko ime i lozinku potrebno je da obrati bazi podataka kako bi proverila ispravnost unetih podataka.

Svaka tabela sa kojom aplikacija treba da komunicira mora da poseduje svoj model. Modeli su smešteni u *models* direktorijumu i sastoje se od tri klase *DbTable*, *Mappers* i *Model*. Za generisanje ovih modela na osnovi tabela iz baze koriste se gotovi generatori koji ove klase generišu pozivanjem odgovarajuće komande iz komandne linije.

Akcija koja je sa view-a dobila parametre za logovanje prosleđuje parametre modelu koji se obraća bazi i proverava ispravnost unetih podataka. Nakon provere, rezultat vraća kontroleru koji u zavisnosti od rezultata prosleđuje view-u upozorenje da su uneti podaci neispravni, ili poziva drugu akciju ukoliko se korisnik uspešno ulogovao.

Zend nudi veliki broj gotovih komponenata koje nude određenu funkcionalnost što mnogo ubrzava izradu jednog projekta i čine aplikaciju sigurnijom. Često korišćene komponente koje doprinose na sigurnosti aplikacije su *Zend\_Auth* (služi za autentifikaciju korisnika i za upravljanje podacima o korisniku) i *Zend\_Acl* (brine se za odgovarajući pristup resursima, upravlja ulogama i odlučuje o tome koji korisnik sme imati pristup određenom sadržaju aplikacije). U projektu postoje sledeći nivoi pristupa: gost, nastavnik, sekretar, upravnik, dekan, administrator. Svaki od njih ima svoj skup pristupa, dok neki nivoi podrazumevaju da imaju sva prava pristupa nekog manjeg nivoa.

Da bi se smanjio rizik napada i povećala sigurnost aplikacije, Zend nudi razne komponente za validaciju korisničkog unosa. Klasa *Zend\_Validate* nudi široku mogućnost validacije raznih stvari kao što su: email adresa, provera da li uneti niz karaktera sadrži isključivo slova i/ili brojeve, kontrola dužine unetog podatka, provera IP adresa...

Prema projektu OWASP TOP 10, najčešći napadi usmereni na web aplikacije su napadi „umetanjem koda“ takozvane „injekcije“. Osim toga, zbog česte zavisnosti web aplikacija o pozadinskoj bazi podatka, najčešći oblik napada „umetanjem koda“ su SQL „injekcije“. „Obične“ PHP aplikacije direktno pozivaju metode za pristup bazi podataka te pritom koriste neke od postojećih funkcija za sprečavanje od napada. Nedostatak ovog pristupa je potreba za ručnim upravljanjem prilikom unosom prilikom svakog pristupa bazi.. Ovaj proces je izvor mnogih sigurnosnih ranjivosti jer izostanak filtriranja samo jednog unosa može u potpunosti narušiti sigurnost aplikacije. Zend nudi poseban API za pristup bazi podataka. Upotrebom tih komponenata moguće je transparentno filtrirati sve štetne unose. Zavisno o odabranoj klasi za pristup bazi podataka upiti se grade drugačije.

Primer *SELECT* naredbe koja se prosleđuje metodom *fetchAll*.

```
$sql="SELECT      id      FROM      korisnici      WHERE      korisnicko_ime=?      AND
lozinka=?";

$result=$db->fetchAll($sql, array('korisnik1','test123'));
```

Sigurniji način postiže se klasom *Zend\_Db\_Select*. Klasa služi za dinamičko stvaranje *SELECT* upita.

```
$select=$db->select()

->from ('korisnici', array ('id', 'korisnicko_ime', 'lozinka',
'email'))

->where ('korisnicko_ime=?', 'korisnik1')

->where ('lozinka=?', 'test123');
```

Nakon uspešnog logovanja korisniku se prikaže mesečni kalendar sa spiskom obaveza, tj. terminima nastave, obaveštenja drugih korisnika i ličnih podsetnika. Kalendar se popunjava takođe podacima iz baze. Pozivanjem *index* akcije u *Calendar* kontroleru pogled tj. kalendar dobija informacije koje su mu potrebne da bi prikazao raspored obaveza. *Index* akcija se obraća bazi podatka preko modela, i zahteva spisak obaveza koje ima ulogovani korisnik za tekući mesec, prosleđujući potrebne parametre. Jedan od zahteva je bio da se svaki tip obaveza bude prikazan drugom bojom. U akciji (kontroleru) se određuje boja po tipu za svaku pojedinačnu obavezu, i dodeljuju se drugi atributi (tip obaveze, da li je obaveza celodnevna ili je vazana za neki period dana i da li obaveza može da se modifikuje). Uređeni niz podataka, akcija prosleđuje pogledu, koji dobijene podatke prikazuje u kalendaru na način na koji je to definisano u kontroleru.

Пон	Уто	Сре	Чет	Пет	Суб	Нед	
	29	30	1	2	3	4	
		<b>9:00</b> Усмени део испита - Интерактивно програмирање <b>9:00</b> Писмени део испита - Управљање пројектима у ИТ <b>14:00</b> Усмени део испита - Управљање пројектима у ИТ	<b>9:00</b> Усмени део испита - Комбинаторика и теорија графова <b>9:00</b> Писмени део испита - Комбинаторика и теорија графова <b>9:00</b> Писмени део испита - Интерактивно програмирање <b>9:00</b> Писмени део испита - Комбинаторика и теорија графова <b>12:30</b> Консултације			5	
	6	7	8	9	10	11	
	<b>9:00</b> Усмени део испита - Комбинаторика и теорија графова	<b>9:00</b> Усмени део испита - Веб програмирање	<b>9:00</b> Писмени део испита - Управљање пројектима у ИТ <b>12:30</b> Консултације		<b>14:00</b> Усмени део испита - Интерактивно програмирање		12
	13	14	15	16	17	18	19
	<b>9:00</b> Усмени део испита - Управљање пројектима у ИТ	<b>15:15</b> Предавања - Интерактивно програмирање	<b>12:30</b> Консултације		<b>11:15</b> Предавања - Комбинаторика и теорија графова <b>13:15</b> Предавања - Веб програмирање		

Klikom na dan u kalendaru za dodavanje nove obaveze, ili klikom na neku od obaveza, dobija se odgovarajući dijalog u zavisnosti od akcije koja ga je pokrenula. Prilikom klika poziva se akcija *unosDogadjaja* u *Calendar* kontroleru. Ukoliko se unosi novi događaj korisniku se prikazuje forma za definisanje nove obaveze.

**Унос/Измена догађаја**

Назив:	<input type="text"/>		
Датум почетка:	<input type="text" value="03.10.2015"/> <input type="button" value="..."/>	Време почетка:	h: <input type="text" value="00"/> min: <input type="text" value="00"/>
Датум крај:	<input type="text" value="03.10.2015"/> <input type="button" value="..."/>	Време краја:	h: <input type="text" value="00"/> min: <input type="text" value="00"/>
Тип	<input type="text" value="Вежбе"/> <input type="button" value="..."/>	Видљивост	<input type="text" value="Приватна"/> <input type="button" value="..."/>
Целодневни:	<input type="checkbox"/>		
<input style="background-color: #00AEEF; color: white; border: none; padding: 2px 10px; margin-right: 5px;" type="button" value="Проследи"/> <input style="background-color: #E63333; color: white; border: none; padding: 2px 10px;" type="button" value="Одустани"/>			

Korisnik unosi podatke o obavezi i klikom na prosledi prosleđuje popunjenoj formi istoj akciji. Akcija proverava validnost forme, tj. da li su uneti svi potrebni podaci i da li je format unetih podataka odgovarajući. Ukoliko dobijena forma ne prolazi validaciju korisnik dobija poruke o tome zašto neki od podataka nije prošao validaciju. Kada forma koju akcija/kontroler dobija od pogleda prođe validaciju, generiše se objekat odgovarajuće tabele iz baze podataka. Objektu se pomoću *set* funkcije postave uneti podaci i poziva se metoda *save*. Metoda *save*, u ovom slučaju oponaša *insert*, i komunicira sa bazom podataka kojoj šalje podatke za dodavanje nove vrednosti u odgovarajuću tabelu. Nakon izvršenog upisa novog podatka u bazu podataka, akcija prosleđuje novi skup podataka pogledu koji osveži kalendar dobijenim podacima i korisniku se prikaže definisana obaveza u kalendaru.

U slučaju da se radi o izmeni definisane obaveze, klikom na nju, poziva se ista akcija samo se u ovom slučaju akciji prosleđuje parametar pomoću kog se dobija informacija na koju obavezu je kliknuo korisnik sa željom da je izmeni. Akcija se sada pre prikazivanje forme za unos i izmenu događaja, obraća bazi podataka sa zahtevom da dobije sve potrebne podatke o odabranoj obavezi. Nakon dobijenih podataka akcija popunjava kreiranu formu podacima iz baze, i prosleđuje tako popunjenu formu korisniku na prikaz. Nakon željene promene korisnik vraća podatke akciji, koja u slučaju da je forma prošla validaciju, pomoću objekta odgovarajuće tabele tj. njenog modela oponaša *update* naredbu. Nakon završene akcije od strane modela, akcija vraća podatke pogledu i korisnik vidi svoje unete promene u kalendaru.

Kada korisnik klikne na neku od obaveza koju nije sam definisao, u zavisnosti od njenog tipa dobija dijalog sa različitim mogućnostima: slanja zahteva za promenu termina ispita, evidentiranja održane ili neodržane nastave, odlaganje nastave.

Ulogovani nastavnik ima mogućnost pregleda predmeta na kojima je angažovan u tekućoj školskoj godini. Klikom na stavku glavnog menija i odabirom stavke Predmeti, aplikacija korisnika vodi na *index* akciju u kontroleru predmeti. Akcija se obraća bazi preko modela prosleđujući tekuću godinu i identifikacioni broj logovanog korisnika. Model se obraća bazi podataka preko definisanog upita i izvlači podatke o predmetima na kojima je ulogovani korisnik angažovan u tekućoj školskoj godini. Model vraća kontroleru podatke u vidu niza, koji taj niz prosleđuje svom pogledu. Podatke koje je pogled dobio od kontrolera, tj. niz predmeta, može ulogovanom korisniku prikazati na bilo koji način, da li je to tabelaran prikaz, ili je prikaz u vidu nekih celina za svaki od predmeta raspoređen u nekoliko kolona na stranici, to je sve stvar dogovora prilikom definisanja zahteva, u slučaju ove aplikacije opredelili smo se za drugu varijantu.

<b>Веб програмирање</b>	<b>Комбинаторика и теорија графова</b>	<b>Програмски језици</b>
• Информатика, мастер академске студије, III семестар <a href="#">Детаљи</a>	• Рачунарске науке 2014, мастер академске студије, Управљање информацијама, II семестар <a href="#">Детаљи</a>	• Примењена математика, мастер академске студије, Математика у финансијама, II семестар <a href="#">Детаљи</a>
<b>Дизајн и анализа алгоритма</b>	<b>Комбинаторика и теорија графова</b>	<b>Развој веб апликација</b>
• Математика, докторске академске студије, IV семестар • Информатика (Рачунарске науке), докторске академске студије, IV семестар <a href="#">Детаљи</a>	• Рачунарске науке 2014, мастер академске студије, Развој софтвера, II семестар <a href="#">Детаљи</a>	• Рачунарске науке 2014, мастер академске студије, Развој софтвера, II семестар • Рачунарске науке 2014, мастер академске студије, Управљање информацијама, II семестар <a href="#">Детаљи</a>
<b>Интерактивно програмирање</b>	<b>Комбинаторика и теорија графова</b>	<b>Управљање пројектима у ИТ</b>
• Информатика, основне академске студије, III семестар <a href="#">Детаљи</a>	• Информатика, мастер академске студије, III семестар <a href="#">Детаљи</a>	• Информатика, основне академске студије, V семестар <a href="#">Детаљи</a>
<b>Комбинаторика и теорија графова</b>	<b>Програмски језици</b>	<b>Управљање пројектима у ИТ</b>
• Математика, мастер академске студије, IV семестар <a href="#">Детаљи</a>	• Примењена математика, мастер академске студије, Математика у физици, II семестар <a href="#">Детаљи</a>	• Биологија, основне академске студије, V семестар <a href="#">Детаљи</a>

Klikom na dugme *Detalji* u odeljku bilo kog predmeta, poziva se ista akcija, kojoj se prosleđuje informacija o odabranom predmetu. U ovom delu aplikacije korisnik ima mogućnost detaljnog pregleda informacije o predmetu i tabelarni spiska studenata koji su se prijavili za odabrani predmet. Pored ove mogućnosti, korisnik može da otprema datoteke koje su vezane za ovaj predmet, jednostavnim popunjavanjem forme, i prosleđivanjem podataka kontroleru i akciji, koja prihvata podatke i vrši otpremanje datoteke na server i zapisivanje informacije o datoteci u bazi podataka, pomoću kojih korisnik kasnije može da skine (engl. *download*) datoteku. Prilikom brisanja otpremljene datoteke, akciji koja vrši brisanje se prosleđuje podatak pomoću kojeg se jednoznačno može identifikovati datoteka koju korisnik želi da obriše. Pomoću tog podatka akcija dobije sve potrebne podatke iz baze, koji su potrebni za uklanjanje datoteke sa servera i brisanje podataka o toj datoteci iz baze. Nakon uspešnog brisanja, prosleđuje se korisniku informacija da je datoteka uspešno obrisana i osveži se spisak otpremljenih datoteka.

## 7114, И231, Веб програмирање

Информатика, мастер академске студије.  
Недељни фонд 3+1+2+0, III семестар, Обавезни, 8.00 ЕСПБ

Korisnik ima mogućnost da definiše delove ispita (prvi i drugi kolokvijum, domaće zadatke i testove i drugo), da pritom definiše i broj minimalni i maksimalnih bodova koje je moguće ostvariti, i mogućnost da zakaže obavezu vezanu za odabrani predmet i deo predmeta i da prilikom zakazivanja rezerviše učionicu.

Jedan od zahteva je bio da se korisniku pruži mogućnost da na jednostavan način pošalje obaveštenje vezano za predmet svim studentima koji su se prijavili da slušaju odabrani predmet. U levom delu se prikazuje spisak poslatih obaveštenja, dok u desnom delu je prikazana forma koju je potrebno popuniti da bi se obaveštenje poslalo. Nakon popunjavanja forme, podaci se prosleđuju akciji koja prihvata poruku koju treba proslediti. Preko modela iz baze podataka dobija spisak email adresa studenata kojima je potrebno proslediti ovu poruku. Pomoću *Zend\_Mail* komponente vrši se slanje elektronske pošte svim studentima, dok se podaci o poslatom obaveštenju upisuju preko modela u bazu podataka, i na kraju završene akcije korisnik dobija novi prikaz obaveštenja.

## 7114, И231, Веб програмирање

Информатика, мастер академске студије.  
Недељни фонд 3+1+2+0, III семестар, Обавезни, 8.00 ЕСПБ

Na poslednjem jezičku ovog dela aplikacije, tabelarno je prikazan spisak održanih i neodržanih predavanja sa razlozima neodržavanja, kao i mogućnost da se takav spisak dobije u nekom elektronskom formatu koji je pogodan za slanje elektronskom poštom ili štampanje radi lične evidencije.

Još jedan od zahteva je da korisnik ima uvid u rezultate obavljenih anketa tj. uvid u podatke drugog dela aplikacije „*Fakultis*“ koja omogućava popunjavanje upitnika od strane studenata. U ovom delu, aplikacija komunicira sa bazom podataka samo radi čitanja podataka, dok je korisniku onemogućeno bilo kakvo menjanje ili brisanje podataka.

Pored navedenih mogućnosti korisniku su pružene i sledeće: administracija biografije, publikacije, konsultacija i ličnih podataka.

Podatke koje korisnik unosi koristeći aplikaciju „*Nastavnički portal*“, dostupni su drugim delovima aplikacije „*Fakultis*“ na uvid, tako studenti mogu doći do otpremljenih datoteka vezane za predmet, i informacija o korisniku/predavaču u vidu biografije, publikacija i konsultacija.

## Poređenje Zend Frameworka sa drugim razvojnim okruženjima

Kao i *Zend*, *CakePHP* je napravljen po uzoru na MVC šablon. Ipak, *CakePHP* je znatno jednostavniji za korišćenje, pogotovo kada su u pitanju početnici. Iako oba okruženja nude veliki nivo modularnosti i pristup gotovim komponentama, *CakePHP* većinu postavki i operacija definiše i obrađuje samostalno, dok *Zend* zahteva dublje poznavanje korišćene komponente. Iz ovog razloga, *CakePHP* pogodniji je za manje projekte gde se u relativno malom vremenskom intervalu mora razviti funkcionalni prototip. Izrada istog projekta korišćenjem *Zend* okruženja mogla bi trajati duže ukoliko se odabранe komponente moraju dodatno konfigurisati i oblikovati. *CakePHP* zahteva znatno manje poznavanje objektnog programiranja budući da nije potrebno ručno konfigurisati gotove komponente. Nedostatak *CakePHP* okruženja je manja fleksibilnost u odnosu na *Zend*. Zbog unapred prepostavljenog ponašanja komponenti teško je unositi veće promene bez menjanja samog izvornog koda komponente. Komponente *Zend* okruženja mogu menjati svoje ponašanje jednostavnom manipulacijom postavki putem definisanja interfejsa što ga čini vrlo fleksibilnim. Iz tog razloga se *Zend* okruženje koristi u većim projektima u kojima često dolazi do promene zahteva.

*Yii* je moderno razvojno okruženje izgrađen po MVC šablonu, nudi gotove komponente za obavljanje raznih funkcija kao što su pritup bazi podataka i obavljanje upita, validacija korisničkog unosa, automatizovano generisanje kompleksnih WSDL specifikacija za servise i drugo. Jedino okruženje koje nalikuje ASP.NET okruženju, ipak, ne predstavlja primarni izvor za početnike, ali ni za veće projekte. Razlog je manjak funkcionalnosti u odnosu na druga okruženja poput *Zend*, *CakePHP* i *Codeigniter*. *Yii* je relativno mlado razvojno okruženje što je apriori nedostatak zbog mogućih neuočenih grešaka i znatno manje zajednice korisnika u odnosu na ostala okruženja.

*Codeigniter* se takođe bazira na MVC šablonu. Tačnije, kontroleri i pogledi su nužni deo okruženja, ali model nije, i retko se koristi. Ono po čemu se ovo okruženje izdvaja od ostalih su performanse koje ga čine najbržim od spomenutih okruženja, uključujući i *Zend*. Takođe, je i jednostavan za korišćenje i prati ga velika zajednica korisnika. Neke od mana su: podrška za templejte, korisničke kontrole, interakcija sa programskim jezikom jQuery i kompatibilnosti sa PHP verzijom 4 iz razloga što se u program uključuje velika količina nepotrebnog koda što smanjuje preglednost i održavanje.

Velika prednost *Ruby on Rails* okruženja su dodaci koji se koriste prilikom razvoja aplikacije kao što su: *Scaffolding* – metoda meta-programiranja koja služi za izradu dinamičkih upita prema bazi podataka. Pozadinski jezički procesor pomoću meta podataka gradi upite i vraća rezultate aplikaciji; *WEBrick* – jednostavan ugrađeni web server; *RubyGems* – metoda upravljanja programskim paketima slična standardnim *Linux* alatima poput *yum* i *apt-get*. Osim navedenih dodataka, *Rails* podržava veliki broj tehnologija kao što su Ajax, SOAP, JavaScript, jQuery i druge. Zahvaljujući fleksibilnom načinu pristupa sloju podataka, lako je promeniti pozadinsku bazu podataka. Prilikom zamene baze podataka potrebno je izmenjati određene konfiguracijske datoteke, dok je kod *Zend* okruženja potrebno prilagođavati upite.

## Zaključak

Programski jezik PHP stekao je veliku popularnost u poslednjih 15 godina i postao je jedan od glavnih tehnologija za razvoj web aplikacija. Velika prihvaćenost i dominacija jezika PHP ogleda se i kroz brojna razvojna okruženja kao što je Zend Framework. Zend pomaže u izgradnji i testiranju dinamičkih web stranica, aplikacija i usluga. Takođe, pomaže u rešavanju učestalih problema koji se pojavljuju prilikom projektovanja web aplikacija. Kako je Zend izgrađen po osnovu MVC šablonu, komponente ovog razvojnog okruženja teže što manjem broju zavisnosti prema drugim komponentama, čime se postiže saglasnost sa jednim od osnovnih koncepta objektno-orientisanog programiranja smanjivanje sprege. Takođe, korišćenjem MVC šablonu odvaja se funkcionalni deo koda od grafičkog interfejsa.

PHP jezik je jednostavan i pogodan za početnike, ali se najčešće greške dešavaju iz razloga što je PHP nije strogo tipiziran jezik i ne zahteva precizno definisanje promenljivi, te se često dešava da se istoimena promenljiva koristi za više tipova podataka što lako može dovesti do greške prilikom izrade aplikacije, i teško uočavanje načinjene greške. Kako se pre uvođenja MVC šablonu u web aplikacije, kod potreban za jednu stranicu pisao na jednom mestu, prilikom uvođenja i učenja MVC šablonu najteži deo predstavlja kako postaviti precizne granice koji deo koda se piše u modelu, koji u pogledu i koji u kontroleru. Jedna od poteškoća korišćenja Zend okruženja je sigurno način pisanja upita pomoću *Zend\_Db\_Select* klase.

Nakon navikavanja na sve prednosti PHP jezika, i njegovih mana, savladavanja MVC šablonu i korišćenje Zend Framework okruženja, dolazite u mogućnost da na jako lak i jednostavan način napravite aplikacije, koje su luke za održavanje i pogodne za timski rad na aplikaciji, jer sama arhitektura aplikacije omogućava lakšu podelu posla među članova tima koji nezavisno jedni od drugih mogu da rade na svojim zadacima. Takođe, iz ovog razloga mogu da se dobiju mnogo kvalitetnije i lepše aplikacije, jer se modelima može baviti programer koji je ekspert za baze podataka, kontrolerima programer koji dobro barata algoritmima, dok na pogledima radi dizajner koji svojim idejama može kupiti svakog korisnika.

## Literatura

1. **W. Jason Gilmore**, *Beginning PHP and MySQL: From Novice to Professional*, 2010
2. **George Schlossnagle**, *Advanced PHP Programming*, 2004
3. **Ahsanul Bari** and **Anupom Syam**, *CakePHP Application Development*, 2008
4. **David Upton**, *CodeIgniter for Rapid PHP Application Development*, 2007
5. **Samisa Abeysinghe**, *PHP Team Development*, 2009
6. **Vikram Vaswani**, *Zend Framework: A beginner's Guide*, 2012
7. **Chris Pitt**, *Pro PHP MVC*, 2012



**ПРИРОДНО - МАТЕМАТИЧКИ ФАКУЛТЕТ  
НИШ**

**КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА**

Редни број, РБР:	
Идентификациони број, ИБР:	
Тип документације, ТД:	монографска
Тип записа, ТЗ:	текстуални / графички
Врста рада, ВР:	мастер рад
Аутор, АУ:	Елизабета Маркуш Митриновић
Ментор, МН:	Марко Милошевић
Наслов рада, НР:	Архитектура MVC апликације са примерима у PHP-у
Језик публикације, ЈП:	српски
Језик извода, ЈИ:	енглески
Земља публиковања, ЗП:	Р. Србија
Уже географско подручје, УГП:	Р. Србија
Година, ГО:	2016.
Издавач, ИЗ:	авторски репринт
Место и адреса, МА:	Ниш, Вишеградска 33.
Физички опис рада, ФО: (поглавља/страна/цитата/табела/слика/графика/прилога)	38 стр. ; граф. прикази
Научна област, НО:	рачунарске науке
Научна дисциплина, НД:	Веб програмирање
Предметна одредница/Кључне речи, ПО:	Веб архитектура, MVC шаблон, Zend framework
УДК	004.738.1/.12; 004.42; 004.455; 004.045
Чува се, ЧУ:	библиотека
Важна напомена, ВН:	
Извод, ИЗ:	У раду је представљена анализа софтверске архитектуре, документа о архитектури и проблема приликом израда веб апликација. Представљен је MVC шаблон, и приказана примена шаблона у Zend framework окружењу кроз апликацију „Наставнички портал“, са детаљнијим описом начина рада и структуре апликације.
Датум прихваташа теме, ДП:	25.10.2016
Датум одбране, ДО:	
Чланови комисије, КО:	Председник: Члан: Члан, ментор:



**ПРИРОДНО - МАТЕМАТИЧКИ ФАКУЛТЕТ  
НИШ**

**KEY WORDS DOCUMENTATION**

Accession number, ANO:	
Identification number, INO:	
Document type, DT:	<b>monograph</b>
Type of record, TR:	<b>textual / graphic</b>
Contents code, CC:	<b>university degree thesis (master thesis)</b>
Author, AU:	<b>Elizabeta Markuš Mitrinović</b>
Mentor, MN:	<b>Marko Milošević</b>
Title, TI:	Architecture of MVC application with examples in PHP
Language of text, LT:	<b>Serbian</b>
Language of abstract, LA:	<b>English</b>
Country of publication, CP:	<b>Republic of Serbia</b>
Locality of publication, LP:	<b>Serbia</b>
Publication year, PY:	<b>2016</b>
Publisher, PB:	<b>author's reprint</b>
Publication place, PP:	<b>Niš, Višegradska 33.</b>
Physical description, PD: (chapters/pages/ref./tables/pictures/graphs/applications)	<b>38 p. ; graphic representations</b>
Scientific field, SF:	<b>computer science</b>
Scientific discipline, SD:	<b>web programming</b>
Subject/Key words, S/KW:	<b>Web architecture, MVC pattern, Zend Framework</b>
UC	004.738.1/.12; 004.42; 004.455; 004.045
Holding data, HD:	<b>library</b>
Note, N:	
Abstract, AB:	In the thesis, we analyse architecture of software, document of architecture and the problems during development of web applications. We also present a MVC pattern, and shows how to use pattern in Zend framework through the application "Nastavnički portal", with detailed description of the operation and structure of the application.
Accepted by the Scientific Board on, ASB:	<b>25.10.2016</b>
Defended on, DE:	
Defended Board, DB:	President:
	Member:
	Member, Mentor: