Razvoj web aplikacija vođenih ponašanjem

Tičić, Lara

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Lara Tičić

RAZVOJ WEB APLIKACIJA VOĐENIH PONAŠANJEM

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Lara Tičić

JMBAG: 0016129266

Studij: Informacijski sustavi

RAZVOJ WEB APLIKACIJA VOĐENIH PONAŠANJEM

ZAVRŠNI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, rujan 2022.

Lara Tičić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristila drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Tema ovog završnog rada je razvoj web aplikacija vođenih ponašanjem. Na početku rada, kao uvod u glavnu temu rada, objašnjavaju se osnovni pojmovi i postupci vezani uz razvoj web aplikacija općenito; vrste web aplikacija te različiti načini razvoja i testiranja istih. Detaljnije je objašnjen razvoj web aplikacija vođen testiranjem jer je iz takvog pristupa nastala ideja o razvoju web aplikacija vođenih ponašanjem. Kako bi se definirao pristup razvoju web aplikacija vođenih ponašanjem objašnjena su načela i prakse korišteni u provođenju, te su istaknute prednosti i mane korištenja istog. Opisan je i način na koji se razvoj provodi, pri čemu je prioritet ostvarivanje poslovnih ciljeva te uspješna implementacija svih značajki aplikacije. Razvoj vođen testiranjem i razvoj vođen ponašanjem u osnovi su vrlo slični, provedena je usporedba kako bi se ukazalo na sličnosti i različitosti ova dva pristupa. Prije same izvedbe praktičnog dijela rada, uspoređena su četiri okvira koji omogućavaju provedbu razvoja web aplikacija vođenog ponašanjem: Behat, Codeception, phpspec i Kahlan. U praktičnom dijelu rada korištenjem programskog jezika PHP i alata phpMyAdmin te Behat okvira za provođenje razvoja vođenog ponašanjem razvijena je aplikacija opisana u ovome radu. Nakon teoretskog i praktičnog dijela rada doneseni su zaključci o korisnosti ovoga pristupa razvoju web aplikacija.

Ključne riječi: Web aplikacija; Razvoj web aplikacija; Razvoj vođen ponašanjem; BDD; Razvoj vođen testiranjem; TDD ; Behat

Sadržaj

1. Uvod 1

2. Razvoj web aplikacija 2

2.1. Vrste web aplikacija 2
2.2. Pristupi razvoju web aplikacija 4
2.3. Vrste testiranja web aplikacija 6
2.4. Razvoj web aplikacija vođenih testiranjem 8

3. Razvoj web aplikacija vođenih ponašanjem 10

3.1. Načela i prakse BDD-a 10
3.2. Prednosti BDD-a 12
3.3. Nedostatci BDD-a 13
3.4. BDD značajke, priče i primjeri 14
3.5. Usporedba razvoja aplikacija vođenih testiranjem i ponašanjem 16

4. Okviri za razvoj web aplikacija vođenih ponašanjem u PHP-u 18

4.1. Behat 18
4.2. Codeception 20
4.3. phpspec 22
4.4. Kahlan 24

5. Usporedba okvira 28

6. Razvoj web aplikacije u odabranom okviru 29

6.1. Opis web aplikacije 29
6.2. Realizacija funkcionalnosti aplikacije 30
6.3. Razvoj web aplikacije 32

7. Kritički osvrt na razvoj aplikacije 41

8. Zaključak 42

Literatura 43

Popis slika 44

1. Uvod

U ovome će se završnom radu u teoriji objasniti te na praktičnom primjeru pokazati razvoj web aplikacija vođenih ponašanjem.

Na početku rada će se definirati razvoj web aplikacija općenito te će se objasniti osnovni termini potrebni za razumijevanje istog. Opisati će se različite vrste web aplikacija ovisno o njihovom sadržaju i funkciji. Biti će objašnjena i važnost odabira pristupa razvoju web aplikacija ovisno o njenim specifikacijama i korisničkim zahtjevima te će se opisati najčešći pristupi razvoju web aplikacija. Razvoj web aplikacija vođen testiranjem će se detaljnije opisati jer je upravo taj razvoj bio temelj za nastajanje razvoja web aplikacija vođenih ponašanjem, objasniti će se ideja istog te će se opisati kako se provodi.

Kako bi se detaljno objasnila glavna tema rada, razvoj web aplikacija vođenih ponašanjem, biti će opisana ideja i ciljevi ovog pristupa razvoju te će se opisati na kojim se načelima i praksama isti temelji. Kako bi ga se pobliže definiralo biti će istaknute prednosti i mane korištenja ovog razvojnog pristupa. Opisati će se i koraci provođenja istog od definiranja poslovnih ciljeva do izrade funkcionalne i korisne aplikacije. Kako bi se ukazalo na njihove sličnosti i različitosti usporediti će se aplikacije vođene testiranjem i ponašanjem.

Provedba pristupa razvoju web aplikacija vođenih ponašanjem ostvaruje se korištenjem nekoga od brojnih alata i/ili okvira koji to omogućavaju, opisati će se i usporediti četiri takva okvira: Behat, Codeception, phpspec i Kahlan.

Za praktični dio ovoga završnog rada izraditi će se aplikacija korištenjem programskog jezika PHP, alata za upravljanje bazama podataka phpMyAdmin i Behat okvira za provođenje razvoja web aplikacija vođenih ponašanjem.

Na kraju rada donijeti će se kritički osvrt na izradu aplikacije te provedbu i korisnost razvoja web aplikacija vođenih ponašanjem, te će se na temelju cjelokupnog završnog rada, i teorijskog i praktičnog dijela, donijeti zaključak.

2. Razvoj web aplikacija

Web aplikacija može se definirati kao računalni program za obavljanje određene funkcije, poziva se i pokreće korištenjem web preglednika (eng. Web browser) preko interneta. U početcima razvoja interneta i svjetske mreže (eng. World Wide Web) prve su web aplikacije bile statičke, te su služile isključivo za pristup određenim informacijama. Razvoj tehnologija preglednika, infrastrukture interneta te tehnologije općenito uzrokovao je brz razvoj te gotovo potpunu transformaciju web aplikacija koje su u današnje vrijeme sigurne i dinamičke, usmjerene na krajnjeg korisnika, moguće im je pristupiti bilo kada iz bilo kojeg dijela svijeta pomoću različitih uređaja s različitim veličinama ekrana.(Jazayeri, 2007)

2.1. Vrste web aplikacija

Ovisno o njihovim karakteristikama, namjenama i sadržajima web aplikacije dijele se na različite vrste. U sljedećem će poglavlju biti opisane najčešće vrste web aplikacija u koje se može podijeliti većina modernih aplikacija.

1. Statičke web aplikacije služe za prikazivanje određenog uglavnom jednostavnijeg sadržaja, kao što su na primjer informacije o određenoj kompaniji. Ove web aplikacije mogu uz tekst sadržavati audio ili video sadržaje ili slike, ali njihov sadržaj nije fleksibilan te kako bi ga se promijenilo potrebno je mijenjati izvorni kod aplikacije. Za izradu statičkih web aplikacija dovoljni su HTML (eng. HyperText Markup Language) – programski jezik za dizajniranje rasporeda elemenata web stranica te prikaz web stranica u pregledniku i CSS (eng. Cascading Style Sheets) – stilske upute kojima se određuje dizajn svakog pojedinog elementa web stranice, kao što je vrsta i veličina fonta, boja pozadine i slično.(*Web App Development: 5 Relevant Types & Examples*, bez dat.)

2. Dinamičke web aplikacije služe za prikaz ali i upravljanje kompleksnijim podatcima, povezane su s bazom podataka te se koriste za ažuriranje iste. Tehnički su zahtjevnije za implementaciju, ali mijenjanje prikazanih sadržaja je jednostavnije nego kod statičkih, uglavnom imaju administratora koji korištenjem same aplikacije može mijenjati prikazani sadržaj i ažurirati bazu. Sadržaj koji korisnik vidi na stranici ažuriran je odnosno dohvaćen direktno iz baze prilikom svakog posjeta stranici. Mnogo je programskih jezika koji se mogu koristiti pri izradi dinamičkih web aplikacija, a jedan od najčešće korištenih je PHP (eng. PHP: Hypertext Preprocessor) koji služi za povezivanje baze podataka sa aplikacijom i strukturiranje sadržaja prikazanih na samoj aplikaciji.(*Web App Development: 5 Relevant Types & Examples*, bez dat.)

3. Responzivne web aplikacije karakterizira mogućnost prilagodbe sadržaja, izgleda i prikaza elemenata web aplikacije ovisno o uređaju, to jest veličini ekrana uređaja, na

kojem se prikazuju. Isti kod web aplikacije može se koristiti za različite prikaze aplikacije na različitim uređajima (tablet, pametni telefon(eng. smartphone)) što omogućavaju nove biblioteke (eng.libraries) i razvojni okviri (eng frameworks) zbog čega su razvoj i modifikacija web aplikacija puno brži i jednostavniji. Većina responzivnih web aplikacija uz HTML i CSS koristi i JavaScript (skriptni programski jezik za razvoj web aplikacija, koristi se za implementiranje dinamičkih značajki web aplikacije), točnije AJAX(eng. Asynchronous JavaScript and XML) tehnologiju koja komunicira s poslužiteljem (eng. browser) i omogućava asinkrono ažuriranje web aplikacija.(Shahzad, 2017)

4. Web portali (eng. Portal web application) su web aplikacije koje imaju početnu stranicu (eng. homepage) te od korisnika zahtijevaju registraciju odnosno prijavu u sustav kako bi određenim korisnicima bile prikazane samo relevantne informacije. Primjer ovakve web aplikacije su forumi ili intranet web aplikacije za zaposlenike neke tvrtke.(*Web App Development: 5 Relevant Types & Examples*, bez dat.)

5. E-trgovine (eng. E-commerce) su web aplikacije koje mogu imati značajke do sad opisanih web aplikacija, ali uz to omogućavaju realiziranje kupovine, prodaje, plaćanja i naručivanja proizvoda ili usluga. Krajnjim korisnicima omogućavaju naručivanje i plaćanje elektroničkim putem (npr. kreditnim karticama), a tvrtkama obradu narudžbi, upravljanje dostavom i slično. E-trgovine mogu u potpunosti realizirati sustav klasične (fizičke) kupovine ili prodaje.(*Web App Development: 5 Relevant Types & Examples*, bez dat.)

6. Sustavi za upravljanje sadržajem (eng. CMS - Content Management System) su web aplikacije koje služe za pohranjivanje i upravljanje sadržajem (slike, tekstualni zapisi, video zapisi i slično) kako bi se on mogao prikazati na web aplikaciji, omogućavaju administratoru da jednostavno provodi promjene i ažuriranja sadržaja web aplikacije. Najpoznatija ovakva web aplikacija je WordPress pomoću koje je vrlo jednostavno upravljati sadržajem npr. bloga ili sličnih web aplikacija.(*Web App Development: 5 Relevant Types & Examples*, bez dat.)

7. Sustavi za upravljanje učenjem (eng LMS - Learning Management System) su web aplikacije za upravljanje e-učenjem; učenicima omogućavaju pristup nastavnim materijalima te interaktivnim aktivnostima za učenje, a predavačima strukturiranje provedbe nastave, prijenos i izradu nastavnih materijala, praćenje napretka pojedinog učenika i slično. Koriste se u školama, na sveučilištima ali i za online realizaciju reznih edukacija i tečajeva.(O'Connor, 2020)

8. Web 2.0 pojam je koji opisuje moderne web aplikacije, karakterizira ga prijelaz iz statičkog u pretežno dinamički sadržaj te sve veće količine sadržaja generiranog od strane korisnika. Web 2.0 aplikacije omogućuju korisnicima koji nemaju predznanja u kreiranju i

upravljanju web sadržajima da objavljuju vlastiti sadržaj, komuniciraju s drugima te na vrlo jednostavan način aktivno sudjeluju u brojnim aktivnostima na web-u. Primjer ovakvih web aplikacija su sve češće i šire korištene društvene mreže (eng. social networks) koje su uzrokovale i omogućile velikom broju novih korisnika kreiranje i korištenje sadržaja na web-u. (Kenton, 2022)

Za izradu web aplikacije određene složenosti potrebno je prvo analizirati njezinu namjenu, složenost, zahtjeve korisnika i slično, te zatim odabrati najprikladniji pristup dizajniranju i razvoju web aplikacija. Prvi korak u razvoju web aplikacija (nakon analiza zahtjeva i definiranja specifikacija) je dizajniranje aplikacije, to jest izrada žičanog okvira (eng. wireframe) i prototipa aplikacije. Prilikom dizajniranja odabire se hoće li aplikacija biti statička, dinamička, responzivna i slično, a nakon osmišljavanja i kreiranja dizajna slijedi korak razvoja funkcionalnosti to jest programskog koda aplikacije koji se odvija korištenjem određenog pristupa razvoju web aplikacije.

2.2. Pristupi razvoju web aplikacija

Mnogo je različitih vrsta i primjena web ali i aplikacija općenito, pa postoji i mnogo različitih pristupa samom razvoju aplikacija. U sljedećem će poglavlju biti ukratko opisano nekoliko pristupa razvoju web aplikacija, tri uobičajena modela (vodopadni (eng. Waterfall Approach), spiralni (eng. Spiral Approach) i inkrementalni (eng. Incremental Approach)) te tri nešto kompleksnija pristupa (Web dizajn vođen podacima (Data-Driven Web Design), Domenom vođeno oblikovanje (eng. Domain-driven design) i Razvoj temeljen na testovima prihvatljivosti (eng. Acceptance-test driven development)).

1. Vodopadni model razvoju pristupa linearno, aktivnosti se obavljaju jedna za drugom te se izbjegava vraćanje na prethodne. Rezultat jedne faze razvoja polazište je za sljedeću. Ova metoda prikladna je za korištenje u slučaju razvoja softvera/aplikacije sa kompleksnim ali stabilnim zahtjevima.(*Software Development Approaches - AcqNotes*, 2021)

2. Spiralni model provodi se kroz više iteracija, svaka od kojih se sastoji od izrade prototipa te analize i procjene rizika istog. Na ovaj način naučeno pri izradi određene verzije pomaže pri izradi verzije u sljedećoj iteraciji. Spiralni model koristi se za razvoj softvera/aplikacija koji imaju uglavnom nepoznate domene.(*Software Development Approaches - AcqNotes*, 2021)

3. Inkrementalni model koristi niz koraka za izradu prve verzije softvera/aplikacije koja sadrži samo dio ukupno planiranih funkcionalnosti. Zatim se koraci razvoja ponavljaju te se u svakom novom ponavljanju („inkrementu“) aplikaciji dodaju nove funkcionalnosti dok se ne razvije cjelokupna, na početku planirana aplikacija. Ova je metoda prikladna za korištenje

kada je potrebno razviti aplikaciju, odnosno početnu ili osnovnu verziju aplikacije u kratkom vremenu.(*Software Development Approaches - AcqNotes*, 2021)

4. Web dizajn vođen podacima usmjeren je na korisnika te sakupljanje i analiziranje korisničkih podataka pomoću kojih se web aplikacija prilagođava zahtjevima i potrebama korisnika. Kako bi provođenje ovog pristupa razvoju web stranica imalo smisla, ključno je odrediti koje korisničke podatke treba analizirati kako bi se mogle utvrditi želje i zahtjevi korisnika. Ovaj pristup web dizajnu veliku važnost daje personalizaciji korisničkog iskustva, pojednostavljivanju i prilagođavanju korištenja aplikacije a donošenje odluka o dizajnu provodi se sukladno korisničkim očekivanjima. Ovakav pristup ima za cilj poboljšanje korisničkog iskustva, te se koristi za izrade web aplikacija o kojima organizacijama ovisi zarada.(*What Is Data-Driven Web Design? - Smartboost*, 2020).

5. Domenom vođeno oblikovanje, ili skraćeno DDD, usredotočeno je na izradu i vrlo dobro razumijevanje modela domene te pravila i procesa nad domenom, pomoću kojih se zatim definira rad i dizajn sustava. Jedna od glavnih ideja ovoga pristupa razvoju je izgradnja sveprisutnog jezika (eng. Ubiquitous Language) pomoću kojeg bi se smanjila dvosmislenost, a koji definira nazive za sve potrebne termine iz domene u samoj aplikaciji. Ovaj se pristup koristi pri izradi web aplikacija za sustave s kompleksnim domenama i logikama koje je teško organizirati. (Fowler, 2020)

6. Razvoj temeljen na testovima prihvatljivosti podrazumijeva razvoj web aplikacija u kojemu uz tim razvojnih programera sudjeluju i testeri te dionični ili krajnji kupci. Poticanje na komunikaciju i suradnju osigurava da razvojni tim radi na značajkama ključnim kupcu. Ključni aspekt ovog pristupa je automatiziranje testiranja, to jest korištenje automatiziranih testnih slučajeva koji osiguravaju da se razvoj aplikacije odvija po planu. Kroz cijeli proces razvoja koriste se testovi prihvatljivosti (eng. acceptance tests) koji se kreiraju direktno iz korisničkih zahtjeva i specifikacija prije nego što razvojni tim započne s izradom koda. Na ovaj način je omogućeno sudjelovanje i krajnjem korisniku u razvojnom procesu, jer su specifikacije definirane u svima razumljivom obliku, te je osigurano da razvojni tim radi isključivo na potrebnim značajkama aplikacije.(*What Is Acceptance Test Driven Development (ATDD)*, bez dat.)

U sljedećim će poglavljima biti detaljnije opisana još dva pristupa razvoju web aplikacija koja su po mnogim značajkama slična razvoju temeljenom na testovima prihvatljivosti.

2.3. Vrste testiranja web aplikacija

Testiranje web aplikacije ključan je i obavezan korak u njezinom razvoju. Ovisno o zahtjevima i funkcionalnostima aplikacije te pristupu razvoju iste, testiranja se provode na različite načine te u različitim fazama razvoja. Testiranje je provjeravanje svih aspekata aplikacije koje se obavezno provodi prije nego aplikacija postane dostupna korisnicima. Kako bi se testirao cjelokupan rad aplikacije potrebno je provjeriti: funkcionalnost (npr. rade li kolačići (eng. cookies) ispravno), upotrebljivost (npr. provjera sadržaja ili navigacije), sučelje (npr. provjera komunikacije između poslužitelja i aplikacije), bazu podataka (npr. da li je prilikom ažuriranja baze podataka integritet podataka očuvan), kompatibilnost (npr. kako se aplikacija prikazuje u različitim preglednicima(eng. browser)), performanse (npr. vrijeme potrebno za dohvaćanje odgovora ovisno o brzini internetske veze) i sigurnost (npr. testiranje prava pristupa stranicama).(Hamilton, 2022)

U sljedećem će poglavlju biti ukratko opisane najčešće korištene vrste testiranja.

1. Jedinično testiranje (eng. Unit testing) provode programeri, te ga se može provesti u bilo kojem trenutku implementiranja aplikacije. Jediničnim testiranjem provjerava se ispravnost rada dijela koda (neke izdvojene „jedinice“) kao što je funkcija. Jedinična se testiranja provode izdvojeno, glavni im je cilj logička provjera određenog dijela koda. Provođenje jediničnog testiranja prilikom pisanja programskog koda vrlo je bitno jer pomaže u pronalasku grešaka ili nedostataka u kodu koje bi u kasnijim fazama testiranja bilo teže pronaći.(Thanh, 2020)

2. Integracijsko testiranje (eng. Integration testing) provodi se kako bi se testirao ukupan rad više odvojenih komponenti. Često se provodi nakon jediničnog testiranja, kako bi se provjerilo rade li dijelovi programa (za koje je već obavljena provjera pojedinačne ispravnosti) ispravno kao cjelina. Koristi se za pronalaženje grešaka i/ili nedostataka u komunikaciji između komponenti, odnosno provjeru protoka podataka, a ne za provjeru ispravnosti rada određene komponente.(Thanh, 2020)

3. Ispitivanje od kraja do kraja (eng. End-to-end testing) služi za ispitivanje ispravnosti web aplikacije kao cjeline. Provodi se nakon jediničnih i integracijskih testiranja, a cilj mu je ispitivanje dobivaju li se za određene ulazne rezultate točni izlazi. Ovim se testiranjem ne može ukazati na (ne)ispravnost pojedinog dijela aplikacije, jer ne provjerava na koji se način došlo do rezultata, te je za provođenje ispitivanja od kraja do kraja potrebno prije provesti jedinične i integracijske testove. Provodi se tako da se oponašaju aktivnosti krajnjeg korisnika aplikacije, to jest simulira se korištenje aplikacije kako bi se utvrdilo radi li ona ispravno te isporučuju li se krajnjem korisniku za određene akcije ili upite ispravni odgovori.(Thanh, 2020)

4. Ispitivanje prihvatljivosti (eng. Acceptance testing) testira zadovoljava li aplikacija korisničke zahtjeve. Ispitivanje prihvatljivosti provodi se nakon do sada opisanih testiranja te ne služi primarno za pronalaženje grešaka u radu aplikacije nego ispituje jesu li uspješno implementirane specifikacije definirane od strane korisnika. Cilj ispitivanja prihvatljivosti je utvrditi da li je aplikacija funkcionalna i spremna za puštanje u rad. Može se provoditi kroz više faza (npr. α-testiranje ili β-testiranje). Provodi se tako što se pomoću povratnih informacija od testnih korisnika (bili oni dio razvojnog tima ili ne) identificiraju neispravnosti koje se zatim otklanjaju kako bi se finalna verzija aplikacije mogla izdati.(Thanh, 2020)

5. Penetracijski testovi (eng. Penetration testing) testovi su kojima se ispituje sigurnost aplikacije ali i sposobnost organizacije da reagira na potencijalne sigurnosne probleme, naziva se još i etičko hakiranje (eng. ethical hacking). Penetracijski testovi obuhvaćaju razne metode, alate i načine na koje se simuliraju hakerski napadi na aplikaciju u svrhu utvrđivanja sigurnosnih ranjivosti. Ove testove mogu provoditi ljudi, takozvani bijeli hakeri (eng. white hacker ili white hat hacker), ili gotova programska rješenja. Provođenje penetracijskih testova odvija se kroz pet koraka; planiranje (određivanje cilja i opsega testa), skeniranje mreže (kako bi se dobila povratna informacija o reakcijama aplikacije na prijetnje), ostvarivanje pristupa (korištenjem npr. SQL injekcije, određivanje mogućih načina za pristupanje podatcima), održavanje pristupa (ispitivanje mogućnosti ostvarivanja trajne prisutnosti u sustavu), analiziranje (osvrt na provedene testove, analiza otpornosti sustava).(Dizdar, 2022)

2.4. Razvoj web aplikacija vođenih testiranjem

Razvoj web aplikacija vođen testiranjem (eng. Test-driven development), ili skraćeno TDD, je vrsta razvoja web aplikacija koja podrazumijeva stalno testiranje koda dok je on još u fazama razvoja, sam kod aplikacije se piše tek nakon što je za njega već osmišljen test, te se dijelovi koda testiraju sa više testova ili testnih slučajeva. Ovaj pristup osigurava da sav kod pri razvoju aplikacije funkcionira te je napisan prema načelima pisanja „jasnog koda“ (eng. clean code). (Beck, 2002)

TDD je repetitivan pristup, testiranja pojedinačnih dijelova koda izvode se tokom cijelog razvoja aplikacije, a sam proces provođenja testova u teoriji je vrlo jednostavan. Jedan od vodećih zagovornika TDD-a, programski inženjer koji je i razvio ovaj pristup, Kent Beck opisuje testiranje kroz tri koraka koja u svojoj knjizi „*Test Driven Development: By Example*“ naziva *Red/Green/Refactor* - mantrom TDD-a:

1. Crveno (eng Red) - napišite mali test koji ne radi, možda se u početku ni ne kompajlira
2. Zeleno (eng. Green)– brzo prepravite test da proradi, što god za to bilo potrebno
3. Refaktoriranje (eng. Refactor)- eliminirajte sva dupliciranja nastala kako bi test proradio

(Beck, 2002, str ix)

TDD podrazumijeva pisanje i provođenje testova na način opisan u gore navedena tri koraka. Kako je ovaj pristup repetitivan, da bi ga se pobliže definiralo dovoljno je konkretizirati provođenje jednog testa, jer se svi kasniji obavljaju po istom principu.U nastavku ovog poglavlja detaljije su opisani koraci kojima Kent Beck opisuje jedan ciklus razvoja web aplikacije vođenog testiranjem. (Beck, 2002)

1. Brzo dodajte test - Kako bi se dodala bilo kakva nova funkcionalnost u kodu, prvo je potrebno napisati test koji je ispravan ako su zadovoljeni zahtjevi i specifikacije te funkcionalnosti. Na ovaj način razvojni programeri usredotočeni su na specifikacije, a ne na kod. U ovome je specifičnost ovog pristupa razvoju, jer je u potpunosti suprotan od uobičajenog razvijanja bilo kakvoga koda u kojemu se testira gotov kod. (Beck, 2002)

2. Pokrenite sve testove i vidite da novi nije uspio – Očekivano je da novi test neće uspjeti, to samo dokazuje da je potrebno kreirati novi kod, ali i potvrđuje da je provođenje samog testa djelotvorno. „Neuspjeh je napredak. Sada imamo konkretnu mjeru neuspjeha. To je bolje nego samo nejasno znati da ne uspijevamo.“ (Beck, 2002, str. 10)

3. Napravite malu promjenu - Potrebno je dodati kod koji će osigurati da svi testovi daju ispravan rezultat. U ovom se koraku ne treba brinuti o tome na koji način će se to zadovoljiti, potrebno je samo na bilo koji način napisati minimalan kod koji će proći sve testove.

Ovaj korak predstavlja najveći problem iskusnim programerima naviknutim na poštivanje pravila u svakom trenutku pisanja koda, te je upravo zbog toga vrlo teško prebaciti se na korištenje ovog razvojnog pristupa. Na primjer, u ovome je koraku dopušteno i korištenje „*tvrdoga koda*“ (eng. hardcode) odnosno eksplicitno definiranje varijabli i pisanje koda koji prolazi testove, ali je nefleksibilan i teže ga je kasnije mijenjati. Ovaj novo dodani kod kasnije će se refaktorirati. (Beck, 2002)

4. Pokrenite sve testove i provjerite jesu li svi uspjeli – Potrebno je provjeriti novi kod, odnosno pokrenuti sve testove i ispitati imaju li zadovoljavajuće rezultate. Novo dodani kod ne smije poremetiti funkcioniranje ranije dodanih značajki te je ključno da prolazi sve testove. Ako to nije slučaj, potrebno je dorađivati novi kod dok se ovi zahtjevi ne ispune (to jest, vratiti se na prethodni korak). (Beck, 2002)

5. Refaktorirajte za uklanjanje dupliciranja – Novo dodani kod u ovome je koraku potrebno refaktorirati odnosno „popraviti“. Bez da se naruši njegova ispravnost, treba kod učiniti lakšim za čitanje i ažuriranje. Refaktoriranje je vrlo bitan korak u pisanju bilo kakvoga koda, ono čini kod lakšim za razumjeti, olakšava pronalaženje grešaka u kodu, te olakšava i ubrzava pisanje i ažuriranje koda. (Beck, 2002)

Ovaj je proces detaljno opisan te kroz primjere objašnjen u knjizi „*Refactoring: Improving the Design of Existing Code“.* Autori definiraju refaktoriranje na sljedeći način:

> Refaktoriranje je proces promjene softverskog sustava na takav način da ne mijenja vanjsko ponašanje koda, ali poboljšava njegovu unutarnju strukturu. To je disciplinirani način za učiniti kod jednostavnijim koji minimizira šanse za događanje pogrešaka. U biti, kada refaktorirate, poboljšavate dizajn koda nakon što je napisan.
>
> (Fowler et al., 1999, str 9.)

I u ovome je koraku potrebno provoditi testiranja kako bi se osiguralo da kod, iako izmijenjen, još uvijek prolazi sve testove. Vrlo važan dio refaktoriranja je uklanjanje dupliciranja u kodu, definiranje logičnih naziva objekata i funkcija, te premještanje koda na logično mjesto. (Beck, 2002)

Razvoj web aplikacija vođen testiranjem vrlo je specifičan te nije vrlo često korišten jer su logika i implementacija kod ovog pristupa suprotni od najčešće korištenih programerskih praksi. Zbog toga je vrlo teško prihvatiti i dobro usvojiti načine korištene u ovom pristupu svakome tko ima iskustva u razvoju koda „uobičajenim“ pristupima. No korištenje ovog pristupa osigurava razvoj jasnog i funkcionalnog koda, te štedi mnogo vremena jer ne zahtijeva kasnija otklanjanja pogrešaka (eng. debugging).

3. Razvoj web aplikacija vođenih ponašanjem

Pristup razvoju web aplikacija vođenih ponašanjem (eng. Behavior-driven development), ili skraćeno BDD, osmislio je Dan North sredinom 2000-ih godina. Ovaj pristup razvoju u osnovi ima slične ideje kao i razvoj vođen testiranjem, od kojega je BDD i nastao. Glavni cilj razvoja vođenog ponašanjem je razvoj aplikacija koje ne samo da su izgrađene na dobar način te ih je lako ažurirati i mijenjati, nego i pružaju korisniku vrijednost te implementiraju sve korisničke zahtjeve. Velika se važnost pridaje upravo prepoznavanju i razumijevanju svih korisničkih zahtjeva i prije nego se započne sa samim razvojem aplikacije, a kako bi se to postiglo BDD se uvelike oslanja na komunikaciju i suradnju između poslovnih analitičara, tima razvojnih programera i testera. Ovaj pristup razvoju usmjerava cijeli proces razvoja na ostvarivanje korisničkih zahtjeva, te time osigurava da su sve značajke dobro dizajnirane i implementirane, te korisniku predstavljaju vrijednost. Velika je prednost korištenja BDD-a to što ovaj razvoj omogućava i korisnicima i cijelom razvojnom timu da sudjeluju u procesu razvoja od samog početka, jer koristi jednostavan jezik razumljiv svima kako bi se opisale sve značajke koje je potrebno implementirati a istovremeno i olakšava razvojnim programerima provođenje testiranja te izradu dokumentacije. (Smart, 2014)

U knjizi *„BDD in Action“* koju je napisao John Ferguson Smart, zagovornik agilnog pristupa razvoju web aplikacija, a za koju je predgovor napisao Dan North, autor daje definiciju razvoja vođenog ponašanjem:

> Razvoj vođen ponašanjem (BDD) skup je praksi softverskog inženjeringa osmišljenih da pomognu timovima da brže izgrade i isporuče vrjedniji i kvalitetniji softver. Oslanja se na metode agilnog i lean razvoja uključujući, posebno, razvoj vođen testiranjem (TDD) i domenom vođeno oblikovanje (DDD). Ali što je najvažnije, BDD pruža zajednički jezik temeljen na jednostavnim, strukturiranim rečenicama izraženim na engleskom (ili na materinjem jeziku dionika) koji olakšava komunikaciju između članova projektnog tima i poslovnih dionika.

(Smart, 2014, str 12)

3.1. Načela i prakse BDD-a

Kako bi se u izradi aplikacija u potpunosti iskoristile prednosti korištenja BDD-a potrebno je znati na koji način pristupiti razvoju različitih aplikacija. BDD se može ukomponirati u sve faze razvoja te se može koristiti kroz razne alate. U sljedećem će poglavlju biti objašnjene najčešće korištene prakse i načela na kojima se temelji BDD. (Smart, 2014)

1. Fokusiranje na značajke koje daju poslovnu vrijednost – Umjesto da korisnici definiraju značajke koje zatim razvojni tim implementira, korisnici skupa s razvojnim timom

moraju identificirati zahtjeve koje je potrebno implementirati. Razvojni tim mora razumjeti poslovne zahtjeve aplikacije, a nakon toga skupa s korisnicima dogovoriti koje je značajke potrebno implementirati u aplikaciji kako bi se zahtjevi ostvarili. Na taj način osigurava se da će aplikacija biti korisna za ostvarivanje poslovnih ciljeva a korisnicima je omogućeno aktivno sudjelovanje u razvoju čime je osigurano da će finalna aplikacija ostvarivati njihove zahtjeve. Ovim pristupom razvoj aplikacija je brži jer razvojni programeri rade isključivo na potrebnim i korisnim aspektima aplikacije. (Smart, 2014)

2. Cijeli tim sudjeluje u određivanju značajki – BDD je pristup razvoju koji zahtijeva stalnu komunikaciju svih uključenih u razvojni proces, no iako se na prvu čini kako ovaj pristup komplicira donošenje odluka te realizaciju zahtjeva, na ovaj se način izbjegavaju pogreške te je osigurano da se razvija upravo ono što je korisnicima potrebno. Tradicionalnim pristupom razvoju korisnik definira svoje zahtjeve poslovnom analitičaru koji ih zatim prenosi razvojnom timu, pri čemu se vrlo lako događaju krive interpretacije te može doći do gubitka bitnih informacija. U praksi BDD-a značajke određuju zajedno svi uključeni u razvoj na način da korisnici opišu potrebe aplikacije, razvojni programeri pomoću vlastitog tehničkog znanja i iskustva predlažu korisna rješenja a testeri odmah ukazuju na mogućnosti provjere određenih specifikacija. (Smart, 2014)

3. Prihvaćanje neizvjesnosti – U bilo kojem trenutku procesa razvoja postoji mogućnost da će se neke od definiranih specifikacija i/ili zahtjeva morati mijenjati. Timovi koji koriste BDD računaju na tu mogućnost, te i pretpostavljaju da će do određenih promjena doći. Kako ne bi gubili vrijeme na razvoj nepotrebnih značajki te bili sigurni da su korisnici zadovoljni, razvojni tim od korisnika traži povratne informacije već u ranim fazama razvoja, umjesto da im se rješenje predstavlja tek kada je cijeli projekt dovršen. (Smart, 2014)

4. Opisivanje značajki konkretnim primjerima – Ranije je spomenuto da se specifikacije i zahtjevi aplikacije korištenjem BDD-a opisuju na način razumljiv i korisnicima i programerima, no razgovorni je jezik previše dvosmislen te vrlo lako dovodi do nesporazuma. Kako bi se to izbjeglo, komunikacija se uvelike oslanja na korištenje primjera pri opisima potrebnih specifikacija. Pomoću primjera svaka se specifikacija može definirati na jednostavan, precizan i nedvosmislen način. (Smart, 2014)

5. Pisanje specifikacija koje se mogu izvršiti umjesto automatiziranih testova – Automatizirani testovi su korisni te često korišteni u razvoju web aplikacija, koriste se za provjeru funkcionalnosti aplikacije. Učinkovitost ovakvih testova ovisi o tome koliko su dobro napisani. Umjesto korištenja automatiziranih testova, BDD pristup preporuča korištenje specifikacija koje se mogu izvršiti (eng. executable specification) a koje obavljaju funkciju automatiziranih testova. Ovakve specifikacije provjeravaju kako aplikacija ispunjava određene

poslovne zahtjeve, pokreću se kad god se u aplikaciji izvrši promjena te određuju koje su nove značajke dovršene, te osiguravaju da nove promjene nisu narušile ispravnost postojećih značajki. Ovakve se specifikacije izrađuju pisanjem testnih kodova za svaku značajku, za što se vrlo često koriste alati kao što su Cucumber i JBehave.(Smart, 2014)

6. Pisanje specifikacija niže razine umjesto jediničnih testova – Slično kao u prethodnoj točki, BDD preporuča korištenje specifikacija niže razine umjesto jediničnih testova. Ovakve su specifikacije slične jediničnim testovima, ali su napisane tako da opisuju namjenu koda te daju konkretne primjere kako bi se kod trebao koristiti. Ovakve specifikacije mogu se pisati korištenjem nekog od alata za jedinično testiranje, ili nekog od specijaliziranih BDD alata kao što je RSpec.(Smart, 2014)

7. Generiranje „žive“ dokumentacije (eng. living documentation) – Korištenjem u prethodne dvije točke opisan specifikacija olakšava pisanje dokumentacije jer generira izvješća o radu aplikacije koja su razumljiva svima a ne samo razvojnim programerima. Time se štedi mnogo vremena jer su ovakva izvješća uvijek ažurna i opisuju rad aplikacije u svakom trenutku razvoja. Dokumentacija nastala od specifikacija viših razina korisna je poslovnim analitičarima i testerima ali i krajnjim korisnicima kako bi pratili razvoj aplikacije. Dokumentacija generirana od specifikacija nižih razina korisna je timu razvojnih programera jer na jednostavan način omogućava praćenje svake implementirane značajke te načina na koji je implementirana. (Smart, 2014)

8. Korištenje „žive“ dokumentacije za pomoć pri održavanju aplikacije – Do sada opisane specifikacije i dokumentacija vrlo su korisni i nakon završetka razvoja aplikacije, u fazi održavanja. Ove su specifikacije i dokumentacija detaljne, ažurne i relevantne te zbog toga pojednostavljuju proces održavanja i bilo kakvog mijenjanja aplikacije. Ukoliko na održavanju ne radi isti tim koji je samu aplikaciju i razvio, ova dokumentacija olakšava razumijevanje zahtjeva ali i načine na koji su implementirani s tehničke strane. Glavne prednosti korištenja „žive“ dokumentacije su to što pruža mnoštvo primjera kako ispravno testirati aplikaciju te pomaže novim programerima da razumiju poslovne ciljeve koje aplikacija implementira. (Smart, 2014)

3.2. Prednosti BDD-a

Fokusiranost na poslovne ciljeve i ostvarivanje značajki vrijednih za poslovanje i korisnika osigurava da se razvija korisna aplikacija, čime se uvelike smanjuje količina nepotrebnog koda, to jest razvojni tim je siguran da radi isključivo na bitnim aspektima aplikacije na pravi način. (Smart, 2014)

Pisanje manje nepotrebnog ili nefunkcionalnog koda vodi i do smanjenja troškova izrade web aplikacije. Briga o korisnosti aplikacije od početka razvoja osigurava minimalne greške te minimalne preinake u završnim fazama izrade, što direktno štedi vrijeme i novac. (Smart, 2014)

Ranije je spomenuto i da korištenje BDD alata olakšava održavanje aplikacije nakon što je faza razvoja završena. Ovo je vrlo bitna prednost korištenja ovoga pristupa jer je korištenjem tradicionalnih pristupa razvoju upravo održavanje najskuplje i vrlo komplicirao za provedbu jer na održavanju uglavnom ne radi isti tim koji je aplikaciju i razvio. „Živa“ dokumentacija i automatizirani testovi osiguravaju puno bržu, lakšu i sigurniju implementaciju promjena. (Smart, 2014)

Iz do sada navedenog direktno slijedi još jedna vrlo bitna prednost korištenja BDD-a, a to je brži razvojni ciklus. Automatizirani testovi olakšavaju posao testerima, razvojni programeri ne gube vrijeme na pisanje nepotrebnog koda, puno rjeđe dolazi do pogrešaka te se ne gubi vrijeme na ispravljanje istih…Ovome doprinosi i uključenost korisnika u ranim fazama razvoja, jer to osigurava da će finalna aplikacija zadovoljavati njihove zahtjeve te se ne gubi vrijeme na dorade. (Smart, 2014)

3.3. Nedostatci BDD-a

Ako se provodi na ispravan način, ovaj pristup razvoju ima značajne prednosti i za razvojni tim i za klijente, ali postoje situacije u kojima ovaj pristup nije najbolji izbor za razvojni proces.

Kao što je već objašnjeno, BDD zahtijeva visoku razinu komunikacije i suradnje, kako unutar razvojnog tima, tako i između korisnika, poslovnih dionika, poslovnih analitičara te svih uključenih u proces razvoja. BDD direktno ovisi o komunikaciji te je nemoguće u potpunosti iskoristiti ovaj pristup razvoju ukoliko svi uključeni nisu voljni na stalnu suradnju. Na primjer, ukoliko naručitelj web aplikacije nije voljan ili nije u stanju aktivno sudjelovati u definiranju zahtjeva ili kasnije u davanju povratnih informacija o proizvodu teško da će se iskoristiti sve prednosti korištenja ovoga pristupa. (Smart, 2014)

BDD se vrlo često definira kao agilan razvojni pristup, koristi kraće razvojne cikluse s ciljem bržeg izdavanja aplikacije te za definiciju korisničkih zahtjeva koristi jezik razumljiv svima umjesto tehnički zahtjevnih specifikacija. Ovaj pristup pretpostavlja da je skoro nemoguće definirati sve korisničke zahtjeve na samom početku razvoja te računa da su promjene stalno moguće i vjerojatne, te se upravo zbog toga BDD ne može u potpunosti ostvariti izvan agilnog konteksta. Shodno tome, primjerice u organizacijama sa strukturom takozvanih organizacijskih silosa (eng. organizational silos), to jest u organizacijama u kojima

skupine zaposlenika ili odjeli međusobno nemaju dobru komunikaciju te nedovoljno surađuju BDD ne može dobro funkcionirati. (Smart, 2014)

Funkcionalnost ovog pristupa razvoju, uz komunikaciju i suradnju svih uključenih, direktno ovisi i o pisanju samih specifikacija to jest provođenju testiranja. Kako bi se uspješno kreirali korisni automatizirani testovi za kompleksnije web aplikacije potrebno je znanje i iskustvo, što timovima koji tek počinju koristiti BDD predstavlja velik izazov. Loše napisani testovi mogu biti zahtjevni za održavanje te na kraju zahtijevati više vremena, a dorađivanja i mijenjanja istih predstavljati veći trošak. (Smart, 2014)

3.4. BDD značajke, priče i primjeri

U ovome će se poglavlju objasniti na koji se način od poslovnih ciljeva organizacije dolazi do konkretnih značajki koje je u aplikaciji potrebno implementirati kako bi ona bila korisna. Opisati će se najbitnija karakteristika BDD-a, a to je zapis korisničkih zahtjeva i ciljeva aplikacije koji je razumljiv svima ali i formalan, nedvosmislen te koristan u daljnjim koracima razvoja. Ovi će se elementi BDD-a objasniti na primjeru iz knjige *„BDD in Action“.*(Smart, 2014)

Razvoj web aplikacija vođen ponašanjem uvijek započinje određivanjem ukupne poslovne svrhe aplikacije, to jest poslovnog cilja za ostvarenje kojeg je aplikacija potrebna. U knjizi *„BDD in Action“* uzeta je za primjer aplikacija za vlakove, koja bi korisnicima pružala informacije o voznom redu vlakova i informacije o kašnjenjima u stvarnom vremenu, radu na pruzi i slično. Primarni poslovni cilj u tome je slučaju: „Povećanje prihoda od prodaje karata tako što će se putovanje vlakom učiniti lakšim i vremenski učinkovitijim“. (Smart, 2014, str 35)

Nakon definiranja poslovnog cilja potrebno je osmisliti kako taj cilj realizirati. Uobičajeno je analizirati implementacijom kojih bi se značajki (eng. features) mogli ispuniti korisnički zahtjevi koji vode do ostvarivanja poslovnog cilja. Na primjeru s vlakovima glavne bi značajke za implementaciju mogle biti :

- Pružanje putnicima optimalne rute puta.
- Pružanje informacija o voznom redu vlakova u stvarnom vremenu te informacija o kašnjenjima.
- Omogućavanje putnicima da spreme svoja omiljena putovanja.
- Obavještavanje putnika o kašnjenju vlaka.

(Smart, 2014, str 35)

Kako bi se ti korisnički zahtjevi pobliže objasnili moguće je da ih je potrebno pojednostaviti odnosno rastaviti na više jednostavnih zahtjeva koje je lakše implementirati. Te jednostavnije zahtjeve, koji se u BDD-u nazivaju i pričama (eng. story) potrebno je opisati na jednostavan i svima razumljiv način, za što se u BDD-u najčešće koristi sljedeći format i u ovom primjeru hrvatski jezik, ali može se koristiti bilo koji govorni jezik:

```
Kako bi < postizanje poslovnog cilja ili isporučivanje poslovne vrijednosti >
Ja kao <sudionik>
Želim <nešto>
```

(Smart, 2014, str 36)

U ovome formatu bi priča o pronalasku vlaka do određene stanice mogla izgledati ovako:

```
Priča: Saznati u koliko sati polazi sljedeći vlak za moju odredišnu stanicu
Kako bih učinkovitije planirao svoja putovanja
Kao putnik
Želim znati koji sljedeći vlak ide do mog odredišta
```

(Smart, 2014, str 37)

Kako bi se bolje razumjelo priče, te kako bi se spriječili nesporazumi u komunikaciji između razvojnog tima i naručitelja aplikacije, priče se dodatno opisuju konkretnim primjerima. Kroz konkretne primjere korisnici ili poslovni dionici koji ne moraju imati tehnička znanja o razvoju aplikacija mogu konkretno objasniti na koji način žele da aplikacija radi. Primjere je moguće opisati razgovornim jezikom, ali u tome slučaju oni mogu biti dvosmisleni te dovesti do nesporazuma. Kako bi se to izbjeglo, u BDD-u se primjeri zapisuju u strukturiranom obliku kao rečenice koje započinju ključnim riječima: s obzirom na, kada i tada, to jest u Given, When, Then formatu koji se nekad naziva i Gherkin format:

```
Given <kontekst>
When <akcija ili događaj>
Then <očekivani rezultat>
```

(Smart, 2014, str 39)

Ranije opisana priča na hrvatskom jeziku mogla bi se prema primjeru iz knjige *„BDD in Action“* opisati konkretnim primjerom u ovom Given, When, Then formatu na hrvatskom jeziku na sljedeći način (pri čemu su Parramatta i Gradska vijećnica nazivi stanica):

```
S obzirom da vlakovi zapadne linije polaze iz Parramatte u 7:58, 8:02, 8:08,
8:11
Kad želim putovati iz Parramatte u Gradsku vijećnicu u 8:00
Onda bi mi trebalo reći da uzmem vlak u 8:02
```

(Smart, 2014, str 39)

Na ovaj način opisani primjeri u ovome formatu zatim se korištenjem BDD alata i okvira prevode u testove prihvatljivosti ili specifikacije, o čemu će se više govoriti u sljedećim poglavljima. (Smart, 2014)

3.5. Usporedba razvoja aplikacija vođenih testiranjem i ponašanjem

Pristup razvoju web aplikacija vođen testiranjem i pristup razvoju web aplikacija vođen ponašanjem temeljeni su na vrlo sličnim idejama. BDD je razvijen iz TDD-a, te se često opisuje kao izvedenica TDD-a. Oba pristupa imaju prednosti i mane te se ne može reći koji je bolji, ali ovisno o aplikaciji i timu koji ju razvija moguće je odrediti koji je pristup pogodniji za određenu situaciju. U ovom će se poglavlju usporediti ova dva pristupa te će se istaknuti ključne različitosti i načini na koje one pridonose uspješnom razvoju aplikacije.

Glavna prednost BDD-a u usporedbi s razvojem vođenim testiranjem je u tome što pridaje veliku važnost razumijevanju svrhe samih korisničkih zahtjeva. BDD na jednostavan način pojašnjava zahtjeve i daje smisao testovima te time osigurava razvoj potrebnih značajki. Iako je i ideja TDD-a vrlo slična, ovaj se pristup temelji na pisanju neuspjelih testova nakon kojih se razvija kod kojim se testovi uspješno ostvaruju, što može puno lakše dovesti do kreiranja kako testova tako i koda koji nije koristan, odnosno ne vodi direktno ka cilju ostvarivanja korisnih značajki. BDD je pogodniji za definiranje općenite svrhe aplikacije i korisničkih zahtjeva viših razina, ali TDD je koristan kako bi se testirale određene značajke, klase ili dijelovi koda. (Salehi, 2016)

Autor knjige *„Mastering Symfony“* Sohail Salehi, razvojni programer i podatkovni znanstvenik (eng. Data Scientist) ističe da BDD ne može u potpunosti zamijeniti TDD, te daje komentar na odnos ova dva pristupa razvoju web apliikacija:

> Još uvijek postoje rasprave o odnosu između BDD-a i TDD-a. Imajte na umu da BDD nije zamjena za TDD. Oni zajedno rade na poboljšanju ideje o testovima prihvatljivosti. I dalje morate testirati svoje metode i klase (jedinično testiranje), ali prije toga morate biti sigurni da vaša aplikacija funkcionira onako kako je definirano i dogovoreno (testiranje funkcionalnosti).
>
> (Salehi, 2016, str 76)

Još jedna bitna prednost BDD-a je u tome što se testovi pišu u obliku koji je razumljiv svima, te time omogućava svim dionicima aktivno sudjelovanje u razvoju, a uz to je i nedvosmislen te koristan razvojnim programerima. TDD testovi pišu se u programskim jezicima kao što su C# i Java, te iako su funkcionalni i korisni razvojnim programerima ne moraju biti jasni ostalim poslovnim dionicima. (Martin, 2021)

Pomoću osnovna tri koraka TDD-a (ranije opisani; Red/Green/Refactor) može se definirati i BDD; u pristupu vođenom ponašanjem ovi se koraci ne primjenjuju za testiranje

svake klase ili metode u kodu, već za testiranje cjelokupne funkcionalnosti aplikacije. Upravo iz toga proizlaze sve prednosti korištenja ovoga pristupa u odnosu na pristup vođen testiranjem. U BDD-u specifikacije se izvršavaju u svim fazama razvoja, na početku projekta kako bi cijeli tim razumio zahtjeve, dok je aplikacija u razvoju kao smjernice za daljnju implementaciju, a u trenutku kada su svi testovi, to jest specifikacije, uspješno realizirani aplikacija se smatra dovršenom. (Martin, 2021)

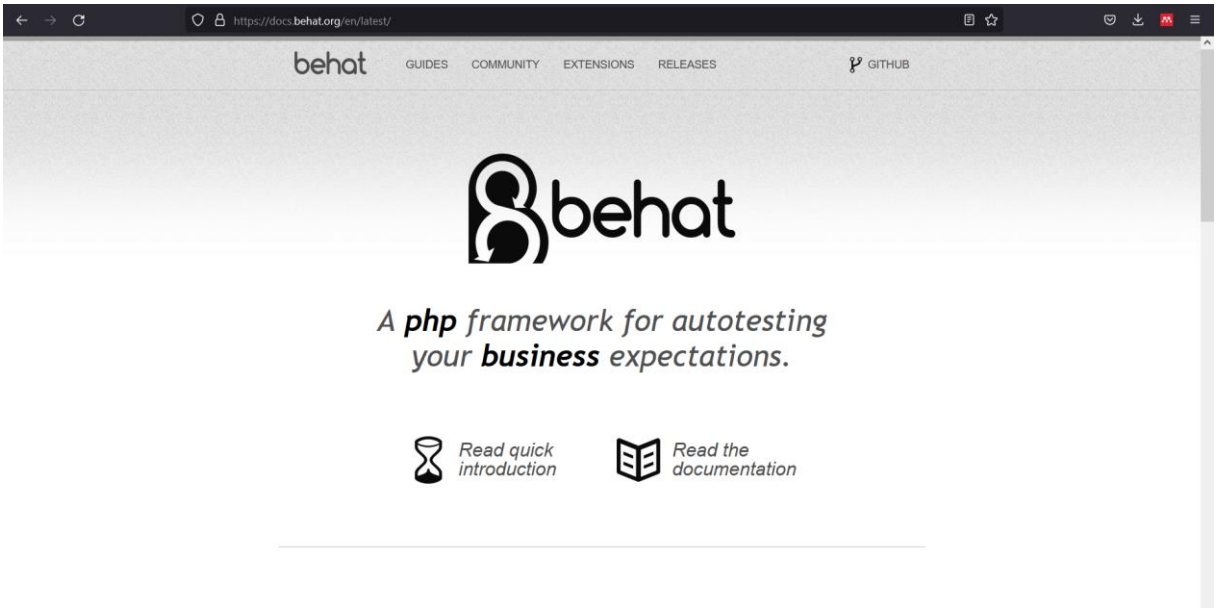
Prilikom odabira pristupa razvoju treba uzeti u obzir i tehničke aspekte. I TDD i BDD za uspješnu provedbu zahtijevaju određena znanja i vještine kako razvojnih programera tako i ostatka sudionika u procesu razvoja. Vrlo je bitno i iskustvo članova tima koje se mora razmotriti prije odabira pristupa razvoju. Osim toga, izbor između ova dva pristupa može ovisiti i o tome postoji li za određeni programski jezik odgovarajući okvir (eng. framework) za provedbu testova. (Martin, 2021)

4. Okviri za razvoj web aplikacija vođenih ponašanjem u PHP-u

Kako bi se uspješno koristio BDD, potrebno je koristiti razvojni okvir koji omogućava pisanje specifikacija, provođenje testiranja te implementaciju do sada opisanih karakteristika BDD-a. Za potrebe ovoga rada razviti će se aplikacija korištenjem PHP-a, a u ovome će se poglavlju opisati neki, često korišteni BDD okviri u PHP-u.

4.1. Behat

Na službenoj stranici behat.org dana je definicija ovoga okvira: „Behat je okvir otvorenog koda (eng. open source) za razvoj vođen ponašanjem u PHP-u . To je alat koji vam pomaže u isporuci softvera koji je važan korištenjem stalne komunikacije, namjernog otkrivanja i automatizacije testiranja.“ (*Behat*, bez dat.)



Slika 1: Behat (behat.org)

Behat razvojni okvir u potpunosti implementira BDD, te je korištenjem ovoga okvira moguće realizirati prednosti korištenja ovoga pristupa razvoju web aplikacija. Behat veliku važnost pridaje komunikaciji, te time u isto vrijeme pojednostavljuje korisničke zahtjeve te ih čini korisnijima. Ovaj je okvir izgrađen za PHP jezik, te je ukoliko se poznaju osnove PHP-a, jednostavan za korištenje. Uz to što ima brojne funkcionalnosti, gotovo svaka se može i proširiti jer postoje brojna proširenja (eng. extensions) za dodatno mijenjanje i unaprjeđivanje ovoga razvojnog okvira. (*Behat*, bez dat.)

Za pisanje BDD specifikacija Behat koristi uobičajeni BDD format scenarija, Given, When, Then zapis. Na sljedećem je primjeru koda sa službene stranice behat.org prikazana implementacija koraka scenarija koji opisuju funkcionalnosti košarice za neku web aplikaciju koja je e-trgovina. (*Behat*, bez dat.)

```
// features/bootstrap/FeatureContext.php

use Behat\Behat\Tester\Exception\PendingException;
use Behat\Behat\Context\SnippetAcceptingContext;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;

class FeatureContext implements SnippetAcceptingContext
{
    private $shelf;
    private $basket;

    public function __construct()
    {
        $this->shelf = new Shelf();
        $this->basket = new Basket($this->shelf);
    }
    /**
     * @Given there is a :product, which costs £:price
     */
    public function thereIsAWhichCostsPs($product, $price)
    {
        $this->shelf->setProductPrice($product, floatval($price));
    }
    /**
     * @When I add the :product to the basket
     */
    public function iAddTheToTheBasket($product)
    {
        $this->basket->addProduct($product);
    }
    /**
     * @Then I should have :count product(s) in the basket
     */
    public function iShouldHaveProductInTheBasket($count)
    {
        PHPUnit_Framework_Assert::assertCount(
            intval($count),
            $this->basket
        );
    }
    /**
     * @Then the overall basket price should be £:price
     */
    public function theOverallBasketPriceShouldBePs($price){
        PHPUnit_Framework_Assert::assertSame(
            floatval($price),
            $this->basket->getTotalPrice()
        );
    }
}
```

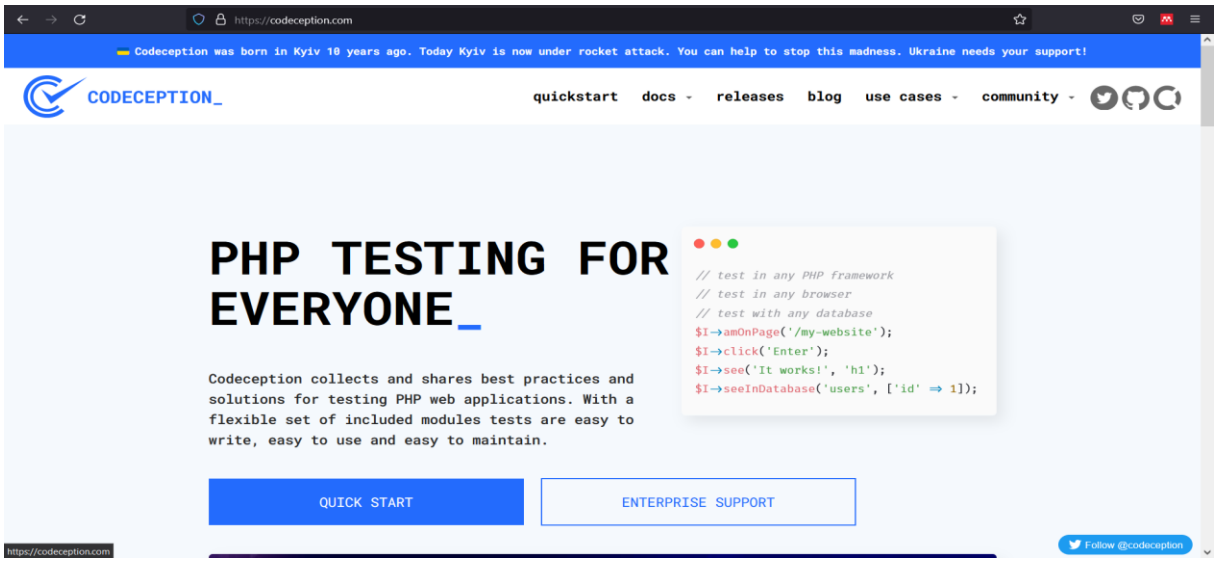
(*Behat*, bez dat.)

Na gornjem primjeru prije metoda definira se dio koda unutar znakova /** ... */. Ta posebna sintaksa u PHP-u naziva se doc-block i služi za pružanje dodatnih meta-informacija o klasama, metodama i funkcijama različitim PHP okvirima. U ovome slučaju Behat-u je kroz taj kod proslijeđena uputa da bi se ova metoda trebala izvršiti kada Behat vidi korak u scenarijima koji je u tom zadanom formatu. Osim toga Behat će prepoznati riječi koje su u scenarijima napisane unutar navodnih znakova kao argumente te će vrijednosti tih argumenata proslijediti metodi kao parametre. Jedna od prednosti korištenja ovoga okvira je i mogućnost Behat-a da sam generira isječke koda za određene korake. (*Behat*, bez dat.)

4.2. Codeception

Codeception je PHP razvojni okvir za provođenje BDD testova. Na web stanici codeception.com ovaj je okvir opisan na sljedeći način: „Codeception prikuplja i dijeli najbolje prakse i rješenja za testiranje PHP web aplikacija. S fleksibilnim skupom uključenih modula testove je lako napisati, koristiti i održavati.“ (*Codeception - PHP Testing Framework*, bez dat.)

Codeception omogućava provođenje testova usmjerenih na korisnika, koji se pišu u Given, When, Then formatu karakterističnom za BDD. Za testiranje aplikacija može se koristiti više različitih preglednika (Firefox, Chrome, Safari) ili emulator web preglednika (PhpBrowser). Codeception omogućava i testiranje unutar samog okvira, bez korištenja web preglednika, što u nekim situacijama olakšava i ubrzava testiranje. Provođenje jediničnih testova vrlo je jednostavno jer je Codeception izgrađen na PHPUnitu(alatu za jedinično testiranje koda u PHP-u) te može provoditi PHPUnit testove. Ovaj okvir podržava i spajanje na različite baze podataka (MySQL, PostgreSQL, MongoDB) te korištenje podataka iz baze unutar testova. (*Codeception - PHP Testing Framework*, bez dat.)



Slika 2: Codeception (codeception.com)

Codeception podržava jedinične testove, testove prihvatljivosti i funkcionalne testove (eng. functional test). Uz pomoć alata moguće je koristiti sve tri vrste testova, čime se mogu testirati svi najbitniji aspekti aplikacije. Ovaj se okvir sastoji od tri paketa (eng. suite) pomoću kojih je to realizirano; jedinični paket (eng. unit suite), funkcionalni paket (eng. functional suite) i paket prihvatljivosti (eng. acceptance suite) od kojih svaki sadrži svoju vrstu testova. (*Codeception - PHP Testing Framework*, bez dat.)

Testovi prihvatljivosti izvode se kako bi se provjerilo radi li web aplikacija, to jest ovi testovi osiguravaju da su korisnički zahtjevi uspješno implementirani. Sljedeći kod primjer je testa prihvatljivosti koji provjerava da li je ispravno izvedena prijava u aplikaciju. (*Codeception - PHP Testing Framework*, bez dat.)

```
$I->amOnPage('/');
$I->click(' Prijavite se ');
$I->submitForm('#signup', [
  'korisnicko_ime' => 'LaraTicic',
  'email' => 'lara@ticic.com'
]);
$I->see('Uspješno ste se prijavili!');
```

Primjer po uzoru na (*Codeception - PHP Testing Framework*, bez dat.)

Funkcionalni testovi provjeravaju ispravnost funkcionalnosti bez da se aplikacija pokreće u pregledniku. Vrlo su slični testovima prihvatljivosti jer isto provjeravaju da li su korisnički zahtjevi uspješno implementirani, razlika je u tome što se funkcionalni testovi izvršavaju unutar PHP-a; emulira se web zahtjev koji vraća HTML odgovor. Funkcionalni testovi omogućavaju i provjeru podataka u bazi, te provjeru odgovora. Sljedeći je kod primjer funkcionalnog testa u Codeceptionu, koji provjerava radi li prijava u sustav ispravno. (*Codeception - PHP Testing Framework*, bez dat.)

```
$I->amOnPage('/');
$I->click('Prijavite se');
$I->submitForm('#signup', ['korisnicko_ime' => 'LaraTicic', 'email' =>
'lara@ticic.com']);
$I->see('Uspješno ste se prijavili!');
$I->seeEmailIsSent('lara@ticic.com', 'Uspješno ste se registrirali!');
$I->seeInDatabase('users', ['email' => 'miles@davis.com']);
```

Primjer po uzoru na (*Codeception - PHP Testing Framework*, bez dat.)

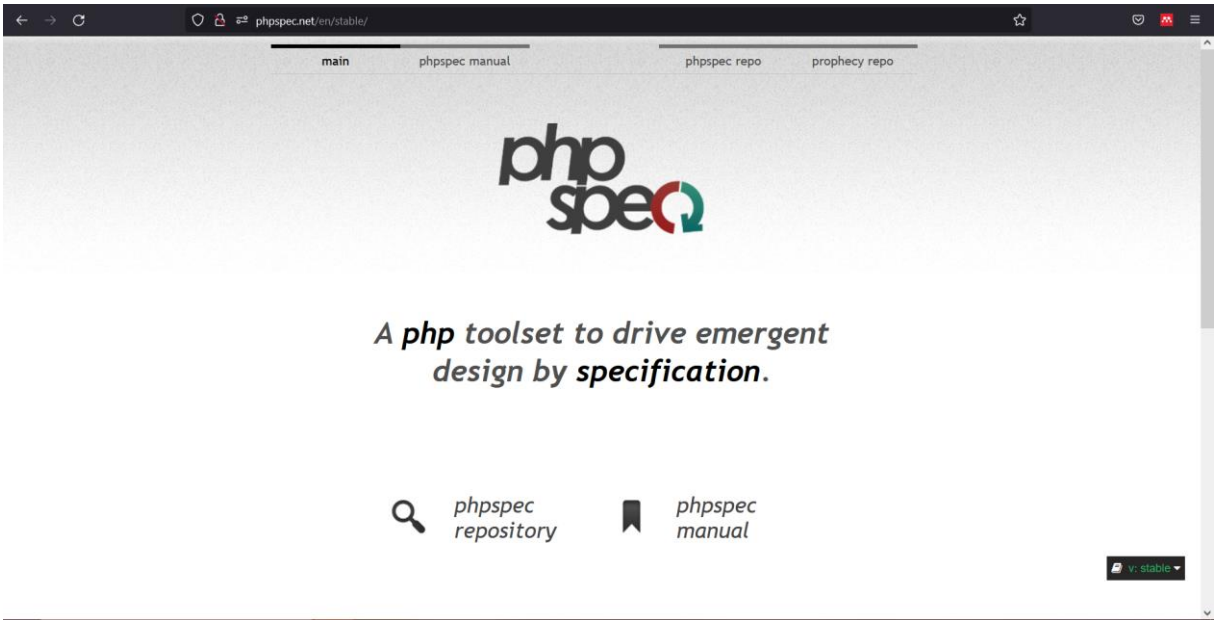
Jedinični testovi služe za provjeru manjih dijelova koda, a provođenje ovih testova u Codeception-u vrlo je jednostavno jer se unutar okvira mogu izvršavati standardni PHPUnit testovi. Sljedeći je kod primjer jediničnog testa, koji provjerava da li su ime i prezime korisnika jedinstveni u bazi podataka. (*Codeception - PHP Testing Framework*, bez dat.)

```
public function testSavingUser()
{
    $user = new User();
    $user->setName('Lara');
    $user->setSurname('Ticic');
    $user->save();
    $this->assertEquals('Lara Ticic', $user->getFullName());
    $this->tester->seeInDatabase('users', [
      'name' => 'Lara',
      'surname' => 'Ticic'
    ]);
}
```

Primjer po uzoru na (*Codeception - PHP Testing Framework*, bez dat.)

4.3. phpspec

phpspec je BDD alat za pisanje PHP koda. BDD pristup može se podijeliti na dvije razine u kojima se realizira, a koje su ranije objašnjene; specifikacije niže razine (eng. SpecBDD) i specifikacije niže razine (StoryBDD). Ovaj alat koristi se za realizaciju zahtjeva niže razine, tako da se prvo specificiraju (opišu) klase odnosno objekti, a zatim se realiziraju pisanjem koda. Nakon što kod uspješno realizira zahtjeve, on se po potrebi refaktorira. Ovaj je proces vrlo sličan ranije opisanom pristupu razvoju web aplikacija vođenom testiranjem, razlika je jedino u načinju pisanja samih testova, odnosno specifikacija, koje su ukoliko se koristi BDD pisane jezikom razumljivim svima. Ovom se tehnikom fokus stavlja na samu implementaciju, te zagovara pisanje koda u manjim cjelinama. (*Phpspec*, bez dat.)



Slika 3: phpspec (phpspec.net)

phpspec nije alat koji može u potpunosti realizirati BDD, te se zato koristi u kombinaciji s nekim drugim alatom ili okvirom (npr. sa ranije opisanim Behat-om) koji podržavaju izvođenje

specifikacija više razine. Na službenoj stranici phpspec.net ovaj je alat definiran kao : „Skup PHP alata za brzu realizaciju dizajna prema specifikacijama.“ (*Phpspec*, bez dat.)

Sljedeći primjer prikazuje specifikaciju u phpspec-u za potrebe izrade alata koji prevodi Markdown jezik (jednostavan jezik za formatiranje teksta) u HTML zapis. Ova specifikacija proširuje klasu ObjectBehavior koja omogućava pozivanje svih metoda novodefinirane kase i usporedbu dobivenih rezultata sa očekivanim rezultatima. U ovome primjeru specificira se da će objekt imati metodu „toHtml“ te se definira što bi ona trebala vratiti. Nakon pokretanja ovakve specifikacije, phpspec prepoznaje da ona opisuje klasu koja ne postoji, te će ponuditi kreiranje iste. Objekti se u phpspec-u sastoje od primjera (eng. examples) koji započinju izrazima „it_“ ili „its_“ (u donjem primjeru it_converts_plain_text_to_html_paragraphs()). (*Phpspec*, bez dat.)

```
<?php
namespace spec;
use PhpSpec\ObjectBehavior;

class MarkdownSpec extends ObjectBehavior
{
    function it_converts_plain_text_to_html_paragraphs()
    {
        $this->toHtml("Hi, there")->shouldReturn("<p>Hi, there</p>");
    }
}
```

(*Phpspec*, bez dat.)

Najvrjedniji aspekt korištenja ovoga alata je upravo specifikacija objekata, to jest realizacija specifikacija nižih razina na jednostavan i točan način, a koje osiguravaju funkcionalnost cijele web aplikacije. Za definiranje načina kako bi se određeni objekt trebao ponašati u phpspec-u se koriste „uparivači“ (eng. matchers).Oni služe kako bi se specificiralo ponašanje objekta, a ne samo kako bi se provjerili izlazi. Postoji mnogo vrsta „uparivača“ u phpspec-u, od kojih mnogi imaju različite varijacije koje olakšavaju pisanje čitljivih specifikacija. Omogućeno je i definiranje vlastitih prilagođenih „uparivača“ u konfiguraciji phpspec-a. (*Phpspec*, bez dat.)

„Uparivač identičnosti“ (eng. Identity matcher) služi za provjeru vraća li metoda točno određenu vrijednost, to jest uspoređuje vrijednosti pomoću operatora ===. Postoji više varijanti koje se razlikuju jedino u nazivima, kako bi specifikacije bile što čitljivije, a sljedeći primjer prikazuje četiri varijente ovoga „uparivača“(*Phpspec*, bez dat.)

```
<?php
namespace spec;
use PhpSpec\ObjectBehavior;

class MovieSpec extends ObjectBehavior
{
    function it_should_be_a_great_movie()
```

```
    {
        $this->getRating()->shouldBe(5);
        $this->getTitle()->shouldBeEqualTo("Star Wars");
        $this->getReleaseDate()->shouldReturn(233366400);
        $this->getDescription()->shouldEqual("Inexplicably popular
children's film");
    }
}
```

(*Phpspec*, bez dat.)

„Uparivač približnosti“ (eng. Approximately matcher) provjerava da li je vrijednost koju vraća metoda približno, s određenom preciznošću, jednaka danoj vrijednosti. Na sljedećem je primjeru „uparivač približnosti“, odnosno četiri varijante istog, koje se razlikuju isključivo u nazivima. (*Phpspec*, bez dat.)

```
<?php
namespace spec;
use PhpSpec\ObjectBehavior;
class MovieSpec extends ObjectBehavior
{
    function it_should_return_a_near_value(){
     $this->getRating()->shouldBeApproximately(1.444447777, 1.0e-9);
     $this->getRating()->shouldBeEqualToApproximately(1.444447777, 1.0e-9);
     $this->getRating()->shouldEqualApproximately(1.444447777, 1.0e-9);
     $this->getRating()->shouldReturnApproximately(1.444447777, 1.0e-9);
    }
}
```

(*Phpspec*, bez dat.)

„Uparivač podudaranja sadržaja“ (eng. StringContain matcher) služi za provjeru sadrži li niz znakova (eng. string) kojeg kao rezultat vraća neka metoda neki zadani niz. Primjer ovoga „uparivača“ prikazan je sljedećim kodom. (*Phpspec*, bez dat.)

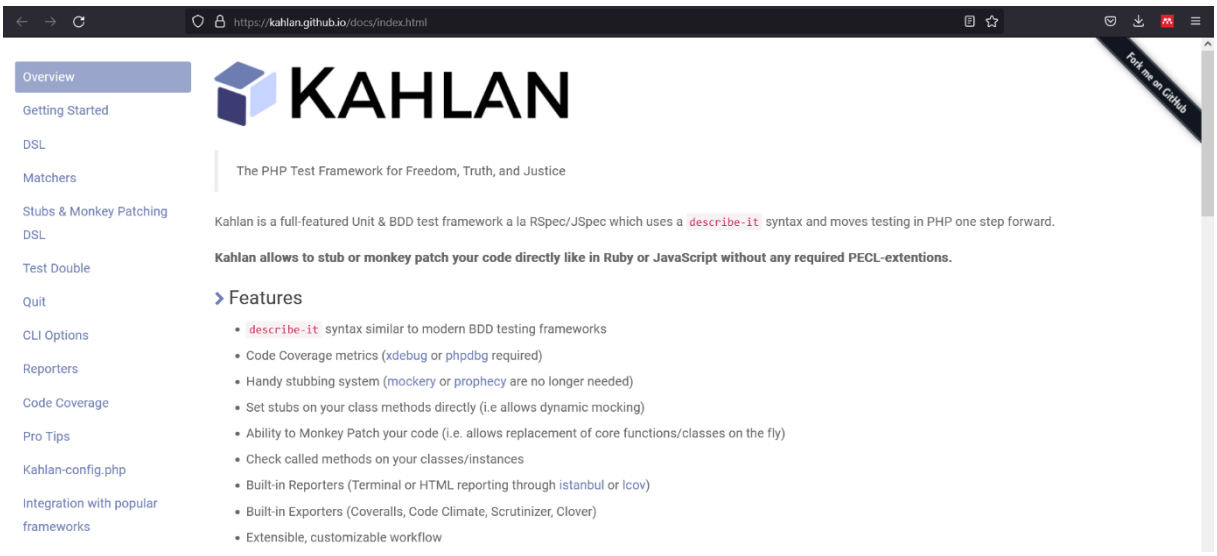
```
<?php
namespace spec;
use PhpSpec\ObjectBehavior;

class MovieSpec extends ObjectBehavior
{
    function it_should_have_a_title_that_contains_wizard()
    {
        $this->getTitle()->shouldContain('Wizard');
    }
}
```

(*Phpspec*, bez dat.)

4.4. Kahlan

Kahlan je okvir za provođenje jediničnih testova i BDD specifikacija koji omogućava testiranje cjelokupne web aplikacije, to jest provedbu ispitivanja od kraja do kraja. Koristi se vrlo razumljivom i opisnom sintaksom u stilu BDD-a. Na jednostavan način daje uvid u ispravnost koda, to jest određuje broj redaka koda koji su uspješno provjereni (eng. Code

Coverage metrics). Uz to je omogućeno provođenje vrlo jednostavnih provjera pokrenutih metoda na određenim klasama/instancama. (*Kahlan*, bez dat.)



Slika 4: Kahlan (kahlan.github.io/docs/index.html)

Sljedeći primjer prikazuje osnovnu sintaksu i organizaciju specifikacije. Kako je organizacija vrlo bitna kako bi specifikacije bile čitke te jednostavne za ažuriranje, Kahlan korištenjem vrlo jednostavne i opisne sintakse omogućava grupiranje testova u blokove; describe – sadrži sve specifikacije za određenu metodu te se uglavnom i naziva isto kao i ta metoda, context – služi za grupiranje testova za istu metodu prema tome na koji se slučaj upotrebe odnose te se uglavnom dodjeljuju opisni nazivi koji započinju s „when“ ili „with“, it – sadrži kod testa te se uglavnom opisuje jasnim i kraćim nazivima. (*Kahlan*, bez dat.)

```
describe("ToBe", function() {
    describe("::match()", function() {
        it("passes if true === true", function() {
            expect(true)->toBe(true);
        });
    });
});
```

(*Kahlan*, bez dat.)

Kahlan koristi ugrađene funkcije koje su vrlo korisne prilikom izvršavanja specifikacija a mogu se koristiti na bilo kojoj razini u sklopu describe ili context blokova:

- beforeAll funkcija pokreće se jednom unutar describe ili context bloka i to prije svih specifikacija sadržanih u tome bloku
- beforeEach funkcija pokreće se prije svake specifikacije iste razine
- afterEach funkcija pokreće se nakon svake specifikacije iste razine
- afterAll funkcija pokreće se jednom unutar describe ili context bloka i to nakon svih specifikacija sadržanih u tome bloku

(*Kahlan*, bez dat.)

Sljedeći primjer prikazuje korištenje beforeEach funkcije u sklopu describe funkcija različitih razina. (*Kahlan*, bez dat.)

```
describe(„Setup and Teardown“, function() {
    beforeEach(function() {
        $this->foo = 1;
    });

    describe(„Nested level“, function() {
        beforeEach(function() {
            $this->foo++;
        });

        it(„expects that the foo variable is equal to 2“, function() {
            expect($this->foo)->toBe(2);
        });
    });
});
```

(*Kahlan*, bez dat.)

U Kahlan-u postoji mnogo "uparivača", a vrijednosti koje oni vraćaju na vrlo se jednostavan način mogu provjeriti korištenjem funkcije expect. Očekivanja (eng. Expectations) se grade pomoću funkcije expect koja kao parametre prima određenu vrijednost te funkciju "uparivača". Očekivanja se ostvaruju isključivo unutar it blokova. Sljedeći kod primjeri su jednog pozitivnog, te jednog negativnog očekivanja, koje se ostvaruje korištenjem riječi not prije poziva "uparivača", a koristi se za negativnu procjenu vrijednosti koju "uparivač" vraća. (*Kahlan*, bez dat.)

```
describe("Positive Expectation", function() {
    it("expects that 5 > 4", function() {
        expect(5)->toBeGreaterThan(4);
    });
});

describe("Negative Expectation", function() {
    it("doesn't expect that 4 > 5", function() {
        expect(4)->not->toBeGreaterThan(5);
    });
});
```

(*Kahlan*, bez dat.)

Svi „uparivači“ u Kahlanu mogu se lančano povezati (eng. chain). Sljedeći je kod primjer jedne expect funkcije sa tri lančano povezana „uparivača“. (*Kahlan*, bez dat.)

```
it("can chain up a lots of matchers", function() {
   expect([1, 2, 3])->toBeA('array')->toBe([1, 2, 3])->toContain(1);
});
```

(*Kahlan*, bez dat.)

U sljedećem su kodu prikazana tri klasična „uparivača“ u Kahlan-u, koji služe za provođenje osnovnih provjera vrijednosti. (*Kahlan*, bez dat.)

toBe($expected)

```
it("passes if $actual === $expected", function() {
    expect(true)->toBe(true);
});
```

toEqual($expected)

```
it("passes if $actual == $expected", function() {
    expect(true)->toEqual(1);
});
```

toBeTruthy()

```
it("passes if $actual is truthy", function() {
    expect(1)->toBeTruthy();
});
```

(*Kahlan*, bez dat.)

5. Usporedba okvira

U prethodnom su poglavlju ukratko opisana četiri alata za izradu web aplikacija vođenih ponašanjem. U sljedećem će se poglavlju ukratko donijeti zaključak o opisanim alatima.

Pomoću Codeception okvira na vrlo se jednostavan i zanimljiv način može provesti BDD. Codeception omogućava provođenje tri vrste testova te je time omogućeno testiranje cjelokupne aplikacije. Svi testovi napisani su na vrlo jednostavan način te ne zahtijevaju predznanje u programiranju kako bi ih se razumjelo. (*Codeception - PHP Testing Framework*, bez dat.)

phpspec vrlo je koristan i zanimljiv alat koji služi za provođenje jediničnih testova u stilu BDD-a, može se reći da je to alat za dizajn specifikacija nižih razina to jest za opisivanje funkcionalnosti web aplikacije. Ovaj se alat ne koristi za definiranje specifikacija viših razina, ali može se koristiti u kombinaciji s drugim alatima kako bi se ostvario i taj dio BDD-a. (*Phpspec*, bez dat.)

Kahlan je koristan i jednostavan za korištenje te podržava provedbu razvoja web aplikacija vođenog ponašanjem. Ovaj okvir olakšava provođenje testova, to jest BDD specifikacija, te na zanimljiv način konkretizira korištenje BDD-a. (*Kahlan*, bez dat.)

Za potrebe praktičnog dijela ovoga rada izraditi će se aplikacija korištenjem Behat razvojnog okvira. Behat implementira cjelokupan postupak razvoja u BDD-u, jednostavan je za korištenje te u mnogim stvarima dodatno olakšava izradu samoga koda (npr. automatsko generiranje isječaka koda (eng. snippet)). Behat ima i najdetaljniju te najjasniju dokumentaciju, te je jedan od najčešće korištenih alata za testiranje u PHP-u, zbog čega će se praktični dio izraditi upravo u ovome alatu. (*Behat*, bez dat.)

6. Razvoj web aplikacije u odabranom okviru

6.1. Opis web aplikacije

Korištenjem razvojnog okvira Behat za potrebe ovoga rada razviti će se aplikacija za organizaciju događaja (koncerti, konferencije, predavanja i slično). Cilj aplikacije je promoviranje različitih događaja, omogućavanje jednostavnije rezervacije karata za pojedini događaj te dobivanje povratnih informacija u obliku ocjena i komentara od korisnika koji su određenom događaju prisustvovali. U nastavku su opisane uloge te su definirane funkcionalnosti web aplikacije koje će se implementirati, pri čemu uloge viših razina imaju pristup svim funkcionalnostima uloga nižih razina.

Neregistrirani korisnik može pristupiti početnoj stranici s pregledom svih događaja te osnovnih informacija o njima, na početnoj su stranici i poveznice na stranice za prijavu i registraciju. Može pristupiti stranici za prijavu te se upisom ispravnog korisničkog imena i lozinke prijaviti u aplikaciju. Može pristupiti i stranici za registraciju na kojoj je potrebno unijeti potrebne podatke za kreiranje novog korisničkog računa.

Registrirani korisnik, može pregledavati sve nadolazeće događaje te informacije o mjestu i vremenu događaja, izvođaču i slično. Može rezervirati kartu/karte ukoliko za određeni događaj ima dovoljno slobodnih mjesta, pri čemu se broj slobodnih mjesta smanjuje za 1/više a karta se pojavljuje u profilu korisnika pod "Moji događaji". Registrirani korisnik može i vidjeti sve vlastite prošle događaje uz mogućnost ostavljanja javne ocjene i recenzije. Može i pregledati sve prošle događaje te komentare i ocjene drugih korisnika za taj događaj.

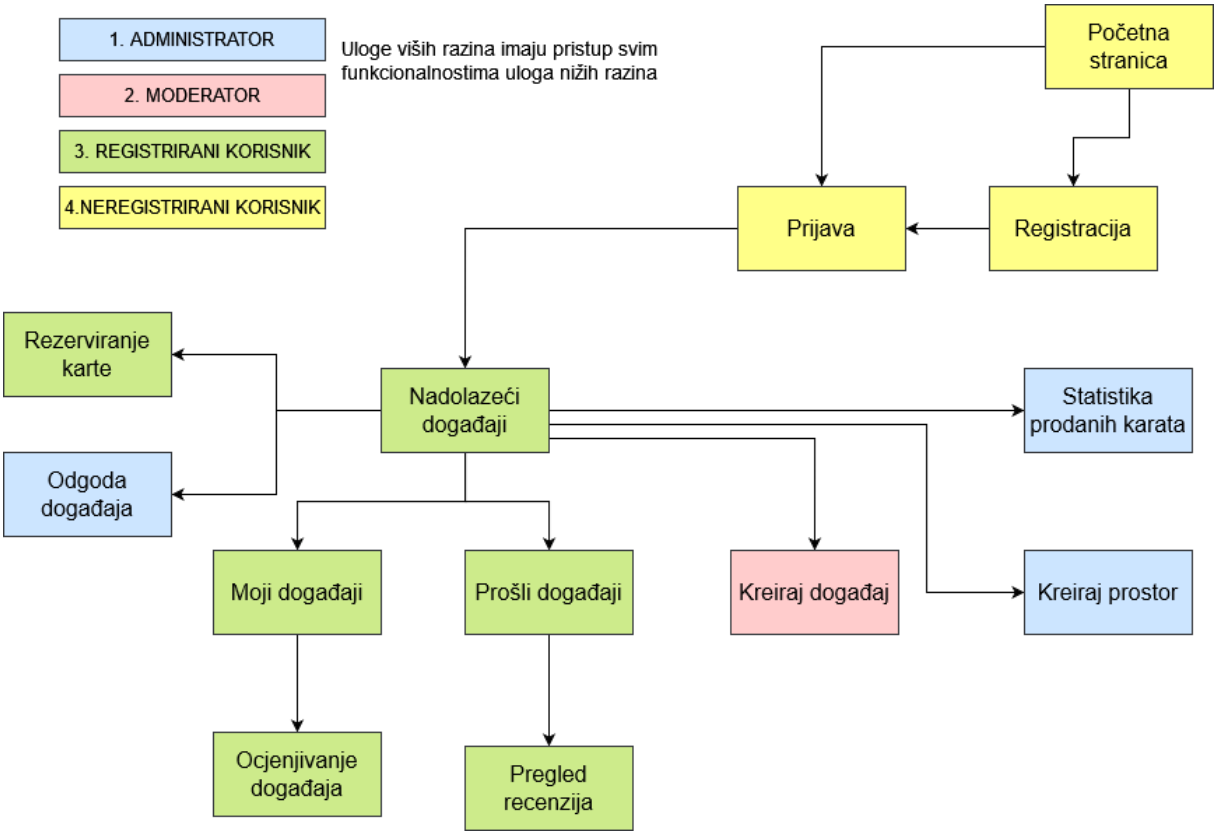
Moderator može kreirati novi događaj unosom datuma, vremena, izvođača, vrste događaja, opisa i slike te odabirom prostora. Ima uvid u broj preostalih dostupnih mjesta za nadolazeće događaje.

Administrator može dodati novi prostor (dvoranu ili slično) s informacijama o kapacitetu, adresom i opisom. Može odgoditi događaj pri čemu se svim korisnicima s rezervacijama za taj događaj pod „Moji događaji“ pojavljuju informacije o razlozima odgode te s novim datumom događaja. Administrator ima i uvid u statistiku, to jest u broj prodanih karata po događaju.

Za razvoj aplikacije koristiti će se PHP programski jezik, a za provođenje razvoja vođenog ponašanjem, a u svrhu testiranja gore navedenih funkcionalnosti, koristiti će se Behat okvir za provođenje testiranja. Baza za potrebe aplikacije izraditi će se korištenjem alata phpMyAdmin.

6.2. Realizacija funkcionalnosti aplikacije

Kako bi se ostvarili u prethodnom poglavlju opisani zahtjevi izgraditi će se web aplikacija čija je struktura prikazana navigacijskim dijagramom na sljedećoj slici.



Slika 5: Navigacijski dijagram (vlastita izrada)

Na početnoj stranici aplikacije prikazani su svi događaji sa pripadajućim informacijama te fotografijom, a sa početne stranice može se pristupiti stranicama za prijavu i registraciju. Na stranici za prijavu korisnici koji već imaju otvoren korisnički račun mogu se prijaviti u sustav ispravnim unosom vlastitog korisničkog imena i lozinke. Neregistrirani korisnici mogu napraviti novi korisnički račun na stranici za registraciju, unosom e-mail adrese, korisničkog imena te duplim unosom lozinke.

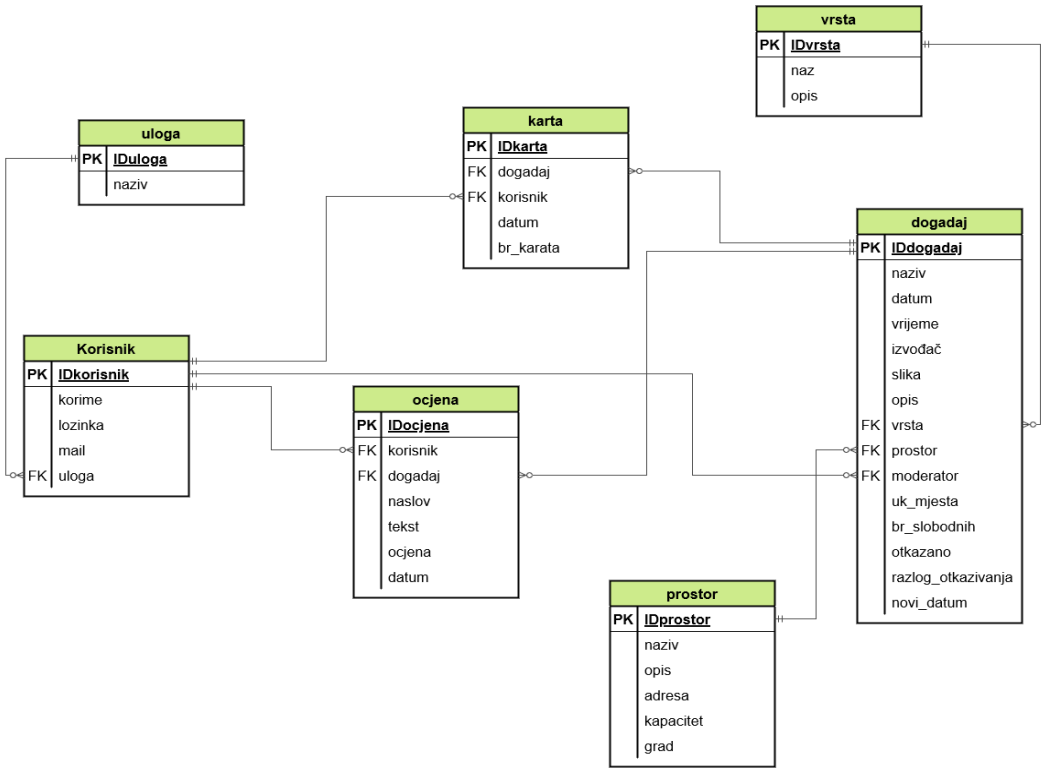
Uspješnom prijavom u sustav korisnika se prebacuje na stranicu s nadolazećim događajima. Odabirom određenog nadolazećeg događaja korisnika se prebacuje na stranicu za rezerviranje karata, na kojoj svaki korisnik može rezervirati od 0 do 20 karata, ukoliko karte za događaj nisu rasprodane. Na stranici sa nadolazećim događajima moderator aplikacije može vidjeti broj preostalih slobodnih mjesta za pojedini događaj, a administrator može pristupiti stranici za odgodu pojedinog događaja, uz mogućnost unošenja novog datuma te razloga odgode.

Registrirani korisnik na stranici sa vlastitim događajima može vidjeti relevantne informacije o nadolazećim događajima za koje je rezervirao karte, te može odabrati opciju ocjenjivanja prošlih događaja kojima je prisustvovao, pri čemu može unijeti komentar i ocjenu za pojedini događaj. Na stranici sa prošlim događajima svi su prošli događaji sa pripadajućom slikom, prosječnom ocjenom korisnika te poveznicom koja otvara stranicu sa komentarima svih korisnika na odabrani prošli događaj.

Stranici za dodavanje novog događaja mogu pristupiti korisnici s ulogama moderatora ili administratora, a na njoj je forma za unos informacija o novom događaju; naziva, izvođača, datuma i vremena događaja, opisa te odabir vrste događaja (koncert, konferencija, predavanje i slično) te mjesta održavanja događaja (dvorana, kazalište i slično).

Administrator aplikacije može pomoću forme na stranici za dodavanje novog prostora dodati novo mjesto održavanja događaja sa informacijama o adresi i kapacitetu. Na stranici sa statistikom administratoru su vidljive informacije o broju prodanih karata za pojedini događaj.

Kako bi se realizirala aplikacija sa do sada opisanim funkcionalnostima, korištenjem alata phpMyAdmin izrađena je baza podataka čiji je ERA model, izrađen u alatu draw.io, prikazan na sljedećoj slici.



Slika 6: ERA model (vlastita izrada)

6.3. Razvoj web aplikacije

Prvi korak u korištenju Behat okvira je opis željenih funkcionalnosti u .feature datotekama. To su datoteke koje sadrže scenarije koji opisuju funkcionalnosti određene stranice. Ovi se scenariji pišu u ranije spomenutom Given, When, Then formatu, te se definicijom istih na vrlo strukturiran ali jednostavan način specificira ponašanje dijelova aplikacije. Svaka se .feature datoteka sastoji od tekstualnog opisa funkcionalnosti (eng. feature) i jednog ili više scenarija (eng. scenario) koji kroz primjere opisuju kako bi funkcionalnost trebala raditi. Ispod je prikazan sadržaj jedne takve datoteke, registracija.feature, koja kroz tri scenarija definira funkcionalnu stranicu za registraciju.

```
#registracija.feature
Feature: functional sign up page

Scenario: Sign up - successful
    Given I am on "/Behat/registracija.php"
    When I fill in "korime" with: "mare"
    And I fill in "mail" with: "mare@mail.mail"
    And I fill in "lozinka" with: "lozy"
    And I fill in "lozinka2" with: "lozy"
    And I submitPost "registracija"
    Then I should see msg "Uspješna registracija! Možete se prijaviti u
sustav!"

Scenario: Sign up - unsuccessful - username already taken
    Given I am on "/Behat/registracija.php"
    When I fill in "korime" with: "lara"
    And I fill in "mail" with: "lara@mail.mail"
    And I fill in "lozinka" with: "lozy"
    And I fill in "lozinka2" with: "lozy"
    And I submitPost "registracija"
    Then I should see msg "Korisničko ime je zauzeto!"

Scenario: Sign up - unsuccessful - error in confirm password field
    Given I am on "/Behat/registracija.php"
    When I fill in "korime" with: "tome"
    And I fill in "mail" with: "maiil@mail.mail"
    And I fill in "lozinka" with: "lozzz"
    And I fill in "lozinka2" with: "passs"
    And I submitPost "registracija"
    Then I should see msg "Netočna ponovljena lozinka!"
```

Svaki se scenarij sastoji od proizvoljnog broja koraka, a svaki korak predstavlja određeno stanje ili akciju koji definiraju željeno ponašanje aplikacije. Koraci su napisani korištenjem strukturiranog jezika kako bi se osiguralo da svaki korak predstavlja poziv točno određenih metoda koje zatim obavljaju samo testiranje. Nakon što je za opis funkcionalnosti neke stranice web aplikacije izrađena .feature datoteka, sljedeći je korak pokretanje iste u Behat razvojnom okviru. Nakon prvog pokretanja Behat za svaki od ranije opisanih koraka

javlja da nisu definirani, te je zatim potrebno definirati koje funkcionalnosti to jest koji kod implementira korake iz .feature datoteke.

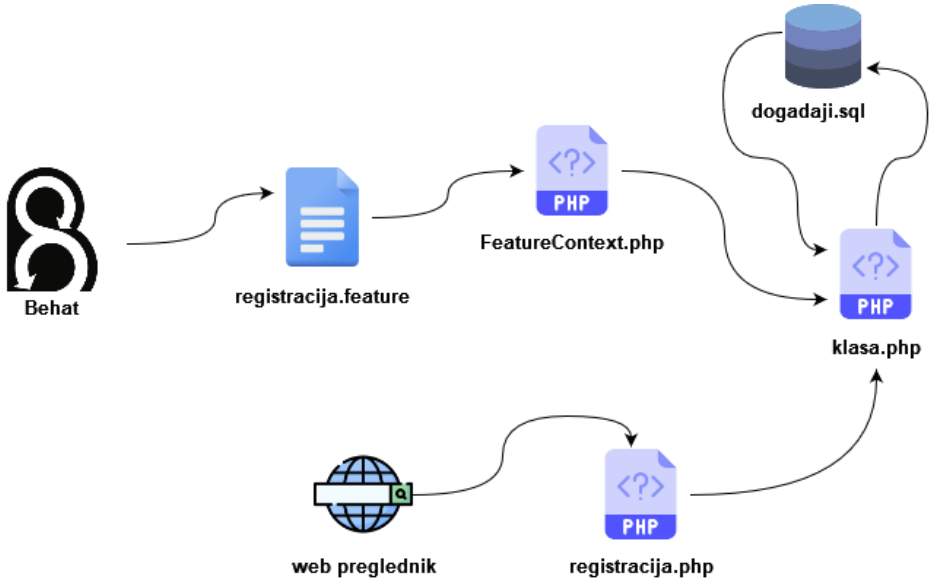
Kako bi se definirale funkcionalnosti koraka, Behat nakon neuspješnog pokušaja provedbe scenarija daje predloške koda (eng. snippet) kojima se definira određeni korak iz scenarija. Na ovaj način definirane funkcije zatim se dodaju u FeatureContext.php datoteku, u kojoj postaju metode FeatureContext klase koja se automatski pokreće svakim pokretanjem Behat testova to jest .feature datoteka. Kako bi FeatureContext klasa prepoznala koju je metodu potrebno pokrenuti kako bi se realizirao određeni korak iz ranije definiranog scenarija, svakoj se metodi prije nego što se ona definira unutar znakova /** ... */ prosljeđuje format koraka nakon kojeg se ta metoda mora pokrenuti.

```
     /**
     * @When I fill in :arg1 with: :arg2
     * @Given I fill in :arg1 with :arg2
     */
    public function iFillInWith($polje, $vrijednost)
    {
        $this->jedan->popuniPolje($polje, $vrijednost);
    }

    /**
    * @Then I should see msg :arg1
    */
    public function iShouldSeeMsg($arg1)
    {
        $korime = $_POST['korime'];
        $lozinka = $_POST['lozinka'];
        $mail = $_POST['mail'];
        $lozinka2 = $_POST['lozinka2'];

        $poruka = $this->kor->registracija($korime, $lozinka, $mail,
$lozinka2);
        PHPUnit_Framework_Assert::assertSame($arg1, $poruka);

        if ($arg1=="Uspješna registracija! Možete se prijaviti u sustav!")
$this->kor->registracijaBrisanje($korime);
}
```

U gornjem kodu prikazane su dvije metode iz FeatureContext klase: iFillInWith metoda koja implementira korak popunjavanja određenog polja za unos određenom vrijednosti, te iShouldSeeMsg metoda koja se poziva u zadnjem koraku sva tri scenarija iz datoteke registracija.feature, a koja implementira samu registraciju korisnika, to jest provjerava ispravnost popunjenosti polja forme za registraciju, vraća poruku o ispravnosti unesenih podataka te ukoliko su uneseni podatci ispravni unosi novoga korisnika u bazu. Ukoliko dođe do novog unosa u bazu, isti se odmah briše jer svrha ove metode nije implementiranje funkcionalnosti, nego testiranje. Metode iz FeatureContext klase pozivaju metode to jest kod same aplikacije, te ga izvršavaju u svrhu testiranja. Tek nakon definiranja ovih metoda započinje pisanje koda stranice s formom za registraciju, koje je sada lakše zbog toga što su

funkcionalnosti stranice već detaljno opisane, a implementacija funkcionalnosti biti će dovršena kada one budu zadovoljavale prethodno napisane testove.

Na sljedećoj je slici grafički prikazan postupak pokretanja testova za stranicu registracija.php. Pokretanjem Behat-a ovaj okvir traži scenarije u registracija.feature datoteci, zatim u FeatureContext klasi traži metode koje definiraju svaki pojedini korak iz scenarija. Kako bi se te metode izvršile one pozivaju kod same aplikacije to jest metode iz datoteke klasa.php koje komuniciraju s bazom te kojima je aplikacija ostvarena. Datoteka registracija.php predstavlja kod stranice za registraciju, koji zahtijeva metoda iz klase za realizaciju te koji se može pokrenuti u pregledniku. Na ovaj način Behat testira aplikaciju bez da joj pristupa preko web preglednika, umjesto toga testira kod klase koja implementira funkcionalnosti aplikacije.



Slika 7: proces pokretanja testova za stranicu za registriranje

Kako bi se pokrenuli Behat testovi za stranicu registracije, to jest proces prikazan na prethodnoj slici, potrebno je u naredbeni redak (eng. Command Prompt) unijeti sljedeću naredbu:

```
vendor\bin\behat features/registracija.feature
```

nakon čega Behat daje povratnu informaciju o tome koji su koraci te scenariji zadovoljeni.

Princip na kojemu se zasniva provođenje Behat testova sastoji se od više jednostavnih koraka, koji na logičan i precizan način definiraju i testiraju rad aplikacije. Prilikom pokretanja Behat izvršava testove zapisane u obliku scenarija tako što za svaki od koraka scenarija pronalazi kod koji definira kontekst koraka to jest što se određenim korakom događa i provjerava. Behat prepoznaje kojem koraku odgovara koji kod tako što kodu, odnosno metodama, unutar znakova /** ... */ prosljeđuje format koraka. Vrijednosti koje su u scenarijima

napisane u navodnicima (“ “) prosljeđuju se metodama kao argumenti. Osim samog formata, unutar znakova /** ... */ mogu se zapisati i opis i primjeri koraka za koje se određena metoda poziva. Iz na ovaj način definiranih metoda poziva se kod same aplikacije. Krajnji je cilj pristupa vođenog ponašanjem funkcionalna i korisna aplikacija, na sljedećoj je slici prikaz do sada opisane i definirane stranice za registraciju pokrenute u web pregledniku.

Slika 8:registracija.php u pregledniku (vlastita izrada)

Na sličan su način razvijeni svi testovi i stranice ove web aplikacije, sljedeći je primjer izrada stranice sa nadolazećim događajima. Na ovome je primjeru prikazano testiranje jedne vrlo bitne funkcionalnosti, a to je prikaz sadržaja određenom korisniku ovisno o njegovoj ulozi. Sljedeći kod prikazuje sadržaj nadolazeci.feature datoteke koja ispituje prikazuje li se korisnicima ispravan sadržaj. U ovome primjeru registrirani korisnik mora moći vidjeti poveznicu za rezerviranje karata, moderator uz to mora vidjeti koliko je za pojedini događaj preostalo slobodnih mjesta, a administrator uz to mora vidjeti poveznicu za odgađanje pojedinog događaja.

```
#nadolazeci.feature
Feature: functional page with information about all upcoming events

Scenario: Viewing basic content - admin
    Given I am on "/Behat/nadolazeci.php"
    Given the following users if I "lara" am "admin":
      | korime  | uloga     | IDuloga |
```

```
      | lara    | admin     | 1       |
      | bor     | moderator | 2       |
      | cvjetko | moderator | 2       |
      | ivan    | korisnik  | 3       |
      | karmen  | korisnik  | 3       |
    Then there should be visible "Rezerviraj"
    And there should be visible "Slobodnih mjesta:"
    And there should be visible "Odgađanje"

Scenario: Viewing basic content - moderator
    Given I am on "/Behat/nadolazeci.php"
    Given the following users if I "bor" am "moderator":
      | korime  | uloga     | IDuloga |
      | lara    | admin     | 1       |
      | bor     | moderator | 2       |
      | cvjetko | moderator | 2       |
      | ivan    | korisnik  | 3       |
      | karmen  | korisnik  | 3       |
    Then there should be visible "Rezerviraj"
    And there should be visible "Slobodnih mjesta:"

Scenario: Viewing basic content - korisnik
    Given I am on "/Behat/nadolazeci.php"
    Given the following users if I "ivan" am "korisnik":
      | korime  | uloga     | IDuloga |
      | lara    | admin     | 1       |
      | bor     | moderator | 2       |
      | cvjetko | moderator | 2       |
      | ivan    | korisnik  | 3       |
      | karmen  | korisnik  | 3       |
    Then there should be visible "Rezerviraj"
```

U nadolazeci.feature datoteci koracima se prosljeđuju vrijednosti u obliku tablice. Na ovaj se način testira ispravnost koda aplikacije bez dohvaćanja ili mijenjanja sadržaja iz baze. Umjesto s podatcima iz baze, testiranje se u ovome slučaju provodi s obzirom na vrijednosti direktno proslijeđene koraku ili koracima scenarija. Na taj se način uvelike ubrzava provođenje testova, pogotovo ako isto zahtijeva dohvaćanje podataka iz baza sa velikim brojem zapisa. Sljedeći kod dio je FeatureContext klase, a prikazuje dvije metode koje implementiraju rad gore prikazanih koraka.

```
    /**
     * @Given the following users if I :arg1 am :arg2:
     */
    //provjera ima li trazeni korisnik odredenu ulogu
    public function theFollowingUsersIfIAm($korime, $uloga, TableNode
$table)
    {
        //vrati multidimenzionalno polje
        $polje = $table->getColumnsHash();
        //broji elemente - br redova iz tablice
        $ukupno = count($polje);

        $uspjesno = false;
```

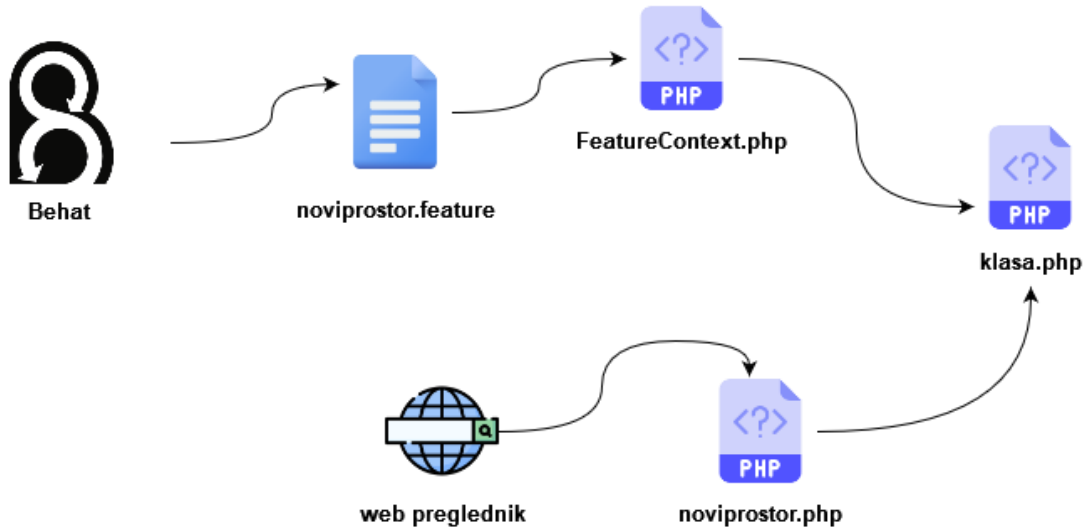
```
        for ($i = 0; $i < $ukupno; $i++) {
            if ($polje[$i]["korime"] == $korime) {
                if ($polje[$i]["uloga"] == $uloga) {
                    //echo "korisnicko ime: " . $korime . " uloga: " . $uloga;
                    $uspjesno = true;
                    $this->IDuloga = $polje[$i]["IDuloga"];
                    return true;
                }
            }
        }
        if ($uspjesno == false) throw new Exception("Nepostoji korisnik s
trazenim imenom i ulogom!");
    }
    /**
     * @Then there should be visible :arg1
     * @And there should be visible :arg1
     */
    //provjera kod nadolazecih dogadaja, koja uloga sto vidi
    public function thereShouldBeVisible($vidljivo)
    {
        $trenutnaUloga = $this->IDuloga;
        $ispis = $this->kor->ulogaSadrzaj($trenutnaUloga);
        //PHPUnit_Framework_Assert::assertSame($vidljivo, $ispis);
        //var_dump($ispis);
        $ispravan = false;
        $uk = count($ispis);
        for ($i = 0; $i < $uk; $i++) {
            if ($ispis[$i] == $vidljivo) {
                $ispravan = true;
                //echo $ispis[$i];
            }
        }
        return $ispravan;
    }
```

Kako bi se uspješno implementirala ranije opisana stranica nadolazećih događaja potrebno je provesti dva koraka u testiranju. Prvo je potrebno provjeriti koja je uloga trenutno prijavljenog korisnika; administrator, moderator ili korisnik aplikacije. Drugi je korak provjera prikazuje li se korisniku ispravan sadržaj s obzirom na njegovu ulogu. Metoda u FeatureContext klasi koja implementira provjeru uloge trenutnog korisnika je theFollowingUsersIfIAm metoda koja kao argumente prima korisničko ime trenutnog korisnika, ulogu koja ima ovlasti za pristup određenim dijelovima aplikacije te TableNode element, to jest tablicu sa zadanim setom podataka koja se u ovome slučaju koristi umjesto baze podataka. Metoda provjerava ulogu korisnika s korisničkim imenom koje joj je proslijeđeno te uspoređuje korisnikovu ulogu sa ulogom koja ima određene ovlasti u aplikaciji. Ukoliko postoji korisnik sa zadanim korisničkim imenom i ulogom izvršava se sljedeći korak, to jest poziva druga metoda zaslužna za provedbu ranije opisanih funkcionalnosti ove stranice; thereShouldBeVisible. Ova metoda ispituje ispisuje li se na stranici sadržaj koji bi trebao s obzirom na korisnikovu ulogu. Nakon što su izrađene metode u FeatureContext klasi, te kod koji predstavlja samu stranicu i

njezine funkcionalnosti, ako je u sustav prijavljen korisnik s ulogom administratora stranica s nadolazećim događajima pokrenuta u pregledniku izgleda kao što je prikazano na sljedećoj slici.

Slika 9: nadolazeci.php u pregledniku (vlastita izrada)

Na sljedećem će primjeru stranice za unos novoga prostora biti pokazano kako su testovi za funkcionalnost te stranice provedeni bez povezivanja na bazu, te bez da se koracima prosljeđuju podatci koji bi zamijenili bazu podataka. Na sljedećoj je slici grafički prikazan postupak pokretanja testova za stranicu noviprostor.php, koji u ovome slučaju ne uključuje komunikaciju sa bazom podataka.



Slika 10: proces pokretanja testova za stranicu za dodavanje novog prostora

Prvi korak u izradi stranice korištenjem Behat okvira je osmišljavanje scenarija za stranicu za dodavanje novoga prostora za održavanje događaja. Sadržaj noviprostor.feature datoteke prikazan je ispod.

```
#noviprostor.feature
Feature: functional page with information about all venues and form for
entering a new one

Scenario: Viewing basic content
    Given I am on "/Behat/noviprostor.php"
    Then I should see Unos novog prostora
    And I should see Popis prostora

Scenario: Entering new venue - unsuccessful
    Given I am on "/Behat/noviprostor.php"
    When I fill in "naziv" with: "prostor"
    And I fill in "opis" with: "opis"
    And I fill in "grad" with: "Gradić"
    And I submitPost "submit_obrazac"
    Then I should see noviprost "Molimo popunite sva polja!"

Scenario: Entering new venue - successful
    Given I am on "/Behat/noviprostor.php"
    When I fill in "naziv" with: "prostor"
    And I fill in "opis" with: "opis"
    And I fill in "adr" with: "adr 123"
    And I fill in "grad" with: "Gradić"
    And I fill in "kap" with: "234"
    And I submitPost "submit_obrazac"
    Then I should see noviprost "Uspješno unesen novi prostor!"
```

Tri su scenarija koja opisuju funkcionalnosti ove stranice. Prvi scenarij definira sadržaje koji moraju biti ispisani na stranici, a druga dva definiraju neuspješno te uspješno dodavanje novoga prostora. Stranica za unos novoga prostora sastojati će se od forme za unos novoga prostora, te tablice sa popisom svih raspoloživih prostora za održavanje događaja.

Prvi scenarij koji provjerava sadržaj stranice implementiran je pomoću metoda koje ne provjeravaju sadržaje baze, već provjeravaju statički dio web stranice, te uspoređuju tražene vrijednosti, koje su proslijeđene koracima u obliku nizova znakova u navodnim znakovima, sa tekstom koji je ispisan na stranici.Za implementaciju dva preostala scenarija koristi se više jednostavnih koraka čiji je cilj dovođenje sustava u željeno stanje prije provedbe same provjere funkcionalnosti. Metoda kojom je izveden posljednji korak ovih scenarija je iShouldSeeNoviprost prikazana slijedećim kodom. Ova metoda provjerava da li su uneseni svi potrebni podatci u formu za unos novoga prostora, jer je to uvjet pod kojim se vrši upis podataka u bazu. Na taj je način provjerena ispravnost funkcionalnosti dodavanja novog zapisa u bazu bez da se pristupa bazi.

```
     /**
      * @Then I should see noviprost :arg1
      */
    //provjera unosa novog prostora
    public function iShouldSeeNoviprost($arg1)
    {
        $naziv = (isset($_POST['naziv']) ? $_POST['naziv'] : "");
        $opis = (isset($_POST['opis']) ? $_POST['opis'] : "");
        $adr = (isset($_POST['adr']) ? $_POST['adr'] : "");
        $grad = (isset($_POST['grad']) ? $_POST['grad'] : "");
        $kap = (isset($_POST['kap']) ? $_POST['kap'] : "");

        $provjeraUnosa = $this->prostor->provjeraProstor($naziv, $opis,
$adr, $grad, $kap);

        if ($provjeraUnosa == 0) $provjera = "Molimo popunite sva polja!";

        else $provjera = "Uspješno unesen novi prostor!";

        PHPUnit_Framework_Assert::assertSame($arg1, $provjera);
    }
```

Stranica za dodavanje novoga prostora pokrenuta u pregledniku prikazana je na sljedećoj slici.

Slika 11: noviprostor.php u pregledniku (vlastita izrada)

7. Kritički osvrt na razvoj aplikacije

Primjena razvoja web aplikacija vođenih ponašanjem u osnovi je vrlo jednostavna i praktična. Određivanje funkcionalnosti prije same izrade aplikacije u obliku pisanja scenarija vrlo je intuitivan i efikasan način za osiguravanje korisne krajnje aplikacije. Definiranje scenarija uključuje i opis funkcionalnosti te zapis primjera što osigurava da se razvijaju isključivo dogovorene funkcionalnosti i to isključivo na unaprijed dogovoreni način, čime se minimalizira broj pogrešaka koje je potrebno ispravljati nakon što je aplikacija već dovršena. Automatizirani testovi uvelike ubrzavaju zadnje faze razvoja aplikacije, za testiranje funkcionalnosti dovoljno je samo pokrenuti testove, cijeli se proces zatim odvija automatski.

Glavni je problem i nedostatak razvoja web aplikacija vođenih ponašanjem to što on zahtijeva „naopaki“ pristup razvoju, na što se nije lagano naviknuti. Provođenje testova vrlo je bitan ako ne i najbitniji dio razvoja bilo kakvog softvera, ali potrebno je iskustvo i vrijeme za potpuno shvaćanje BDD pristupa testiranju, to jest definiranja testova dok se još ni nema što testirati.

Korištenje pristupa razvoju web aplikacija vođenih ponašanjem zahtijeva nešto više vremena u početnim fazama izrade aplikacije, no uzme li se u obzir koliko pojednostavljuje i ubrzava testiranje te to što osigurava korisne rezultate ne može se reći da je ovaj pristup dugotrajniji ili zahtjevniji od uobičajenih. Vrlo je bitna stavka kod ovoga pristupa razvoju dobro definirati korisničke zahtjeve te izraditi smislene i korisne scenarije bez kojih ovaj pristup razvoju uvelike gubi na uspješnosti. Behat okvir je jednostavan za korištenje te omogućava brzu i logičnu implementaciju BDD-a, te pomaže u izradi scenarija i metoda kojima su koraci scenarija realizirani.

Ovaj je pristup razvoju koristan prvenstveno kada na razvoju radi veći tim ljudi, te kada su korisnički zahtjevi vrlo specifični, no i iz jednostavnog primjera aplikacije i scenarija prikazanih u ovome radu može se reći da je pristup razvoju web aplikacija vođenih ponašanjem vrlo zanimljiv i koristan.

8. Zaključak

Razvoj web aplikacija složen je proces koji je u pravilu i vremenski i financijski ograničen, a provedba i uspješnost istoga ovisi kako o zahtjevima same web aplikacije tako i o timu koji ga provodi. Jasno je da je odabir pristupa razvoju vrlo bitan kako bi se ostvarili svi ciljevi te zadovoljile sve strane koje sudjeluju u razvoju. Razvoj web aplikacija vođenih ponašanjem u brojnim je slučajevima jako dobar izbor zato što najveću važnost pridaje korisnosti i funkcionalnosti aplikacije.

Svaki razvojni pristup ima svoje prednosti i mane, i niti jedan nije idealan za primjenu u svakoj situaciji. Uzmu li se u obzir prednosti i mane BDD-a, može se reći da je ovaj pristup vrlo uspješan i koristan, a mane su mu uglavnom tehničke prirode, najčešće su to ne educiran razvojni tim ili ne mogućnost komuniciranja sa strankama.

U usporedbi sa ostalim, u ovom radu spomenutim, razvojnim pristupima jasno je da je BDD nešto kompleksniji, ali i najpotpuniji pristup koji se provodi u svim fazama razvoja web aplikacija. Na sličnim se idejama temelji te na sličan način provodi i razvoj web aplikacija vođenih testiranjem, od kojega je i nastao BDD. Oba pristupa veliku važnost pridaju testiranju prije razvoja koda, a glavna je razlika između ovih pristupa to što BDD ide korak dalje, te osim testiranja programskog koda osigurava ostvarivanje poslovnih ciljeva.

Glavna posebnost, a ujedno i glavna prednost ovog pristupa je isticanje poslovnih ciljeva te fokusiranost na ostvarivanje istih u svim fazama razvoja. Može se zaključiti da je ovaj pristup kako u teoriji tako i u praktičnom provođenju vrlo zanimljiv, logičan i smislen.

Literatura

Beck, K. (2002). Test Driven Development: By Example 1st Edition. U *AddisonWesley Longman*.

Behat. (bez dat.). Preuzeto 22. srpanj 2022., od https://docs.behat.org/en/latest/

Codeception - PHP Testing framework. (bez dat.). Preuzeto 20. srpanj 2022., od https://codeception.com/

Dizdar, A. (2022, svibanj 29). *What is Penetration Testing? Process, Types, and Tools - Bright Security*. Preuzeto 11. rujan 2022., od https://brightsec.com/blog/penetration-testing/

Fowler, M. (2020, travanj 22). *DomainDrivenDesign*. Preuzeto 22. srpanj 2022., od https://martinfowler.com/bliki/DomainDrivenDesign.html

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). Refactoring Improving the Design of Existing Code - Fowler-Beck-Brant-Opdyke-Roberts. U *xtemp01*.

Hamilton, T. (2022, lipanj 25). *Web Application Testing: 8 Step Guide to Website Testing*. Preuzeto 22. srpanj 2022., od https://www.guru99.com/web-application-testing.html

Jazayeri, M. (2007). Some trends in Web application development. *FoSE 2007: Future of Software Engineering*, 199–213. Preuzeto 22. srpanj 2022., od https://doi.org/10.1109/FOSE.2007.26

Kahlan. (bez dat.). Preuzeto 21. srpanj 2022., od https://kahlan.github.io/docs/allow.html

Kenton, W. (2022, veljača 3). *Web 2.0 Definition*. Preuzeto 20. srpanj 2022., od https://www.investopedia.com/terms/w/web-20.asp

Martin, K. (2021). *The Agile Software Tester: Software testing in the agile world: Revision 7: Sv. Revision 7*.

O'Connor, M. (2020, lipanj 26). *LMS vs CMS: what are the differences and which do I need? - Synergy Learning*. Preuzeto 20. srpanj 2022., od https://synergy-learning.com/blog/lms-vs-cms-what-are-the-differences-and-which-do-i-need/

phpspec. (bez dat.). Preuzeto 21. srpanj 2022., od http://phpspec.net/en/stable/

Salehi, S. (2016). *Mastering Symfony : orchestrate the design, development, testing, and deployment of web applications with Symfony*.

Shahzad, F. (2017). Modern and Responsive Mobile-enabled Web Applications. *Procedia Computer Science*, *110*, 410–415. Preuzeto 22. srpanj 2022., od https://doi.org/10.1016/j.procs.2017.06.105

Smart, J. F. (2014). *BDD in action : Behavior-Driven Development for the whole software lifecycle.*

Software Development Approaches - AcqNotes. (2021, srpanj 16). Preuzeto 22. srpanj 2022., od https://acqnotes.com/acqnote/careerfields/software-development-approaches

Thanh, V. (2020, svibanj 22). *15 testing methods all developers should know | CircleCI*. Preuzeto 13. rujan 2022., https://circleci.com/blog/testing-methods-all-developers-should-know/

Web App Development: 5 relevant types & examples. (bez dat.). Preuzeto 26. lipanj 2022., od https://en.yeeply.com/blog/6-different-kinds-web-app-development/

What Is Acceptance Test Driven Development (ATDD). (bez dat.). Preuzeto 28. lipanj 2022., od https://www.digite.com/agile/acceptance-test-driven-development-atdd/

What is Data-Driven Web Design? - smartboost. (2020, srpanj 28). Preuzeto 11. rujan 2022., od https://smartboost.com/blog/what-is-data-driven-web-design/

Popis slika

Slika 1: Behat (behat.org) 18
Slika 2: Codeception (codeception.com) 20
Slika 3: phpspec (phpspec.net) 22
Slika 4: Kahlan (kahlan.github.io/docs/index.html) 25
Slika 5: Navigacijski dijagram (vlastita izrada) 30
Slika 6: ERA model (vlastita izrada) 31
Slika 7: proces pokretanja testova za stranicu za registriranje 34
Slika 8:registracija.php u pregledniku (vlastita izrada) 35
Slika 9: nadolazeci.php u pregledniku (vlastita izrada) 38
Slika 10: proces pokretanja testova za stranicu za dodavanje novog prostora 38
Slika 11: noviprostor.php u pregledniku (vlastita izrada) 40