Razvoj aplikacija za web vođen testiranjem

Polak, Aleksandra

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Aleksandra Polak Tomić

RAZVOJ APLIKACIJA ZA WEB VOĐEN TESTIRANJEM

DIPLOMSKI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Aleksandra Polak Tomić

Matični broj: 44457/15–R

Studij: *Baze podataka i baze znanja*

RAZVOJ APLIKACIJA ZA WEB VOĐEN TESTIRANJEM

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Stapić Zlatko

Varaždin, studeni 2020.

Aleksandra Polak Tomić

Izjava o izvornosti

Izjavljujem da je moj diplomski 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

Ovaj diplomski rad bavi se analizom najboljih praksi, tehnologija i alata koji se koriste u testiranjem vođenom razvoju aplikacija za web. Poseban naglasak u diplomskom radu odnosi se na aktivnosti osiguranja kvalitete web aplikacija.

Sistematizirani koncepti i proces razvoja primjenjeni su u razvoju web aplikacije za potrebe evidentiranja studenata na kolegiju tjelesne i zdravstvene kulture, te je sam proces razvoja jedne od funkcionalnosti prikazan kroz snimke zaslona.

Ključne riječi: web, razvoj, tdd, razvoj vođen testiranjem, analiza

Sadržaj

1. Uvod

Ako gledamo razna iskustva i mišljenja vidimo puno razilaženja oko pitanja donosi li razvoj aplikacije vođen testiranjem sa sobom više prednosti ili nedostataka. Dok jedni hvale i preporučaju razvoj aplikacija vođen testiranjem, drugi tvrde kako ne donosi mnogo profita. Danas je ovaj pristup vrlo raširen u poslovnom svijetu, a više o tome kada ga je poželjno primjeniti a kada ne, pisat ćemo u narednim poglavljima.

Kao što smo i prethodno naveli u samom sadržaju ovog diplomskog rada, tema kojom ćemo se baviti je razvoj aplikacija vođen testiranjem *(eng. Test Driven Development – kraće TDD)*. Kako sam se susrela sa informacijom o složenom postupku evidentiranja aktivnosti i teškom praćenju sudjelovanja studenata na raznim aktivnostima koje se nude u sklopu tjelesne i zdravstvene kulture na našem fakultetu, rodila se ideja o kreiranju web aplikacije koja će olakšati rad i praćenje cijelog procesa. Omogućit ćemo prijavu svim voditeljima dodatnih aktivnosti kako bi na licu mjesta mogli vršiti prijave studenata koji polaze njihove tečajeve. Time će se skratiti put informacija od voditelja do nastavnika i smanjiti dodatni posao nastavnika. Dosadašnje vođenje evidencija i pregleda stanja obavljenih obveza studenata korištenjem složene i nepregledne excel tablice zamijenit ćemo jednostavnim web sustavom.

Za razvoj aplikacije koristit ćemo PHP jezik koji slovi kao jedan od najboljih jezika namjenjenih razvoju web aplikacija. Kako je PHP vrlo raširen u praksi postoji i velika podrška alata i razvojnih okruženja namjenjenih PHP-u.

S obzirom na sve češću upotrebu TDD razvoja u praksi, primjenit ćemo tu tehniku na ovom projektu. Time ćemo vjerujem olakšati postupak testiranja tijekom razvoja i ograničiti razvoj na osnovne i bitne elemente. U drugom i trećem poglavlju reći ćemo nešto više o samom TDD-u, alatima i razvojnim okruženjima, te o čestim sigurnosnim napadima i kako ih spriječiti. U četvrtom poglavlju posvetit ćemo veću pozornost aktivnostima osiguranja kvalitete web aplikacija *(eng. Quality assurance activities – kraće QAA)*. U petom poglavlju upoznat ćemo se sa korisničkim zahtjevima i kreirati potrebne funkcionalnosti za izradu same aplikacije te ćemo izabrati jednu funkcionalnost na temelju koje ćemo kreirati šesto poglavlje i prikazati razvoj web aplikacije kroz TDD. Tu ćemo također prikazati u što većoj mjeri navedene aktivnosti osiguranja kvalitete web aplikacija na našem projektu.

2. Metode i tehnike rada

Prethodnim poglavljima naveli smo da je naša glavna tema razvoj web aplikacija vođen testiranjem. S obzirom da je ovo moj prvi susret sa TDD pristupom razvoja web aplikacija, izradi rada prethodilo je pretraživanje dostupne online literature koja je postala primarni izbor informacija. Najvećim dijelom u obradi teoretskog dijela rada korištena je stručna literatura, kao što su IEEE članci dugogodišnjih programera, ali i drugih autora koji pišu u sklopu edukacije, odnosno, pojašnjenja raznih tema koje su vezane za tvrtke u kojima rade želeći približiti svoja iskustva drugima. Time sam došla i do nekoliko preporuka kvalitetnih knjiga u kojima je pobliže pojašnjena i opisana tematika TDD razvoja, sigurnosti, alata i razvojnih okruženja.

Znamo da danas imamo širok izbor alata i tehnologija za izradu aplikacija, te je teško odlučiti i odabrati tehnologije koje ćemo primjeniti. Ranije smo spomenuli da je PHP najčešći izbor u izradi web aplikacija te ima veliku podršku raznih tehnologija i alata, osim toga, već sam upoznata sa sintaksom PHP jezika te ga smatram najboljim izborom za korištenje u ovom radu. Nastavno na izbor PHP-a kao jezika za programiranje, Alice Njenga (2018) iz Raygun tvrtke navodi i pojašnjava 10 najpopularnijih PHP razvojnih okruženja, među kojima su, Laravel, Codelgniter, Symfony, CakePHP, Yii, Zend Framework i drugi. Kako se prvi puta susrećem sa ovom tehnologijom, izabrala sam javno dostupan web okvir *(eng. open-source web framework)* pod nazivom Laravel[1]. Prema A. Njenga, Laravel je najbolji izbor za početnike u TDD razvoju, uz to što pojednostavljuje razvojni proces lakših zadataka, kao što su usmjeravanje, sesije, predmemoriranje i provjera autentičnosti *(eng. routing, sessions, caching and authentication)*, pogodan je i za kreiranje kompleksnih pozadinskih zahtjeva, bilo malih ili velikih. Osim što na službenim stranicama Laravel-a možemo pronaći detaljnu dokumentaciju za instalaciju i korištenje, nalazimo velik broj video tutorijala koji olakšavaju učenje i samo korištenje alata. Što se tiče odabira automatiziranih testnih okruženja, u Laravel je integriran alat PHPUnit koji nam omogućava upotrebu jediničnih testova, funkcionalnih testova i testova prihvatljivosti. Više o Laravelu možete pročitati na službenim stranicama: http://laravel.com/. Za razvoj aplikacije koristit ćemo besplatnu platformu za web riješenja razvijenu od strane

[1] Laravel je razvio Taylor Otwell s ciljem razvoja web aplikacija prateći model-pogled-kontroler (eng. model-view-controller – kraće MVC) arhitekturu. Smatra se jednim od najpopularnijih PHP razvojnih okruženja, a cijeli izvorni kôd Laravel-a nalazi se na GitHub-u pod MIT licencom. (Laravel, 2020)

„Apache Friends“ i integrirano razvojno okruženje (eng. Integrated development environment – kraće IDE) PHPStorm u koji ćemo instalirati Laravel.

3. Razvoj vođen testiranjem

Da bismo lakše shvatili temu krenut ćemo od definiranja pojma TDD razvoja web aplikacija. Proučit ćemo pravila i način primjene te prednosti i nedostatke razvoja vođenog testiranjem. Upoznat ćemo se sa vrstama testiranja te pogledati koji su to alati koji nam služe za jednostavniju provedbu samog testiranja. Također analizirat ćemo najbolje prakse za usvajanje razvoja vođenog testiranjem i na kraju, ne manje bitno, vidjet ćemo kada se može primjeniti TDD pristup.

3.1. Definicija

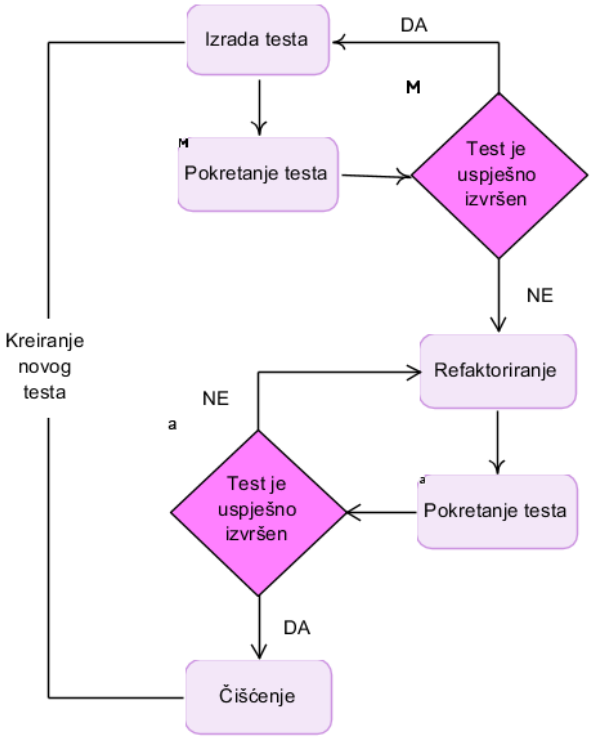
TDD je pristup razvoju softvera u kojem se test piše prije kôda. Jednom kada novi kôd zadovolji test, refaktoriran je do prihvatljivog standarda. TDD osigurava da je izvorni kôd u potpunosti jedinično testiran i da vodi do modularnog, fleksibilnog i proširivog kôda. Ovaj pristup fokusiran je na pisanje kôda u mjeri dovoljnoj da zadovolji test, čineći time dizajn jednostavnim i čistim (Techopedia, 2020).

3.2. Pravila i način primjene

Kada promatramo korake koje je potrebno provesti kako bismo programirali slijedeći TDD pristup, vidimo četiri glavna koraka koji se ponavljaju sve dok ne završimo sa razvojem aplikacije (*Laravel*, 2020):

1. Izrada testa,
2. Pokretanje testa, koji bi trebao biti neuspješan,
3. Korigiranje kôda tako da se test uspješno izvrši,
4. Ponovno pokretanje testa, koji bi sada trebao biti uspješan.

Sve dok je četvrti korak neuspješan,vraćamo se na treći korak i ponovno refaktoriramo kôd tako da se test uspješno izvrši. Jednom kada je četvrti korak uspješno proveden, vraćamo se na prvi korak TDD razvoja (*Laravel*, 2020). Slika 1 prikazuje grafički prikaz primjene ova četiri koraka u TDD razvoju.



Slika 1. Grafički prikaz TDD razvoja aplikacije (Izvor: Shilpa, 2020)

Shilpa (2020) povlači činjenicu da TDD ne govori o testiranju kojeg provode testeri, već programeri, naravno, riječ je o jediničnom testiranju. Slijedi da je prvi test koji se napiše, jedinični test. To ne znači da testeri nisu uključeni u TDD, već sudjeluju djeleći testne scenarije koji se sastoje od slučajeva graničnih vrijednosti, ispitnih slučajeva klase ekvivalencije, kritičnih poslovnih slučajeva, slučajeva funkcionalnosti sklonih pogreškama i slučajevima osiguranja razina, te surađuju sa programerima u implementaciji istih. Ako želimo implementirati TDD, tada testeri trebaju unaprijediti svoje analitičke i logičke vještine, razumjeti tehnički žargon kojeg koriste programeri. S druge strane, programeri moraju moći stati na mjesto testera i nastojati doći do sofisticiranijih scenarija koji će jedinično testiranje učiniti više robusnim i solidnim. U članku kojeg piše Shilpa za Software Testing Help organizaciju (2020), navodi korake TDD-a na sljedeći način (Shilpa, 2020):

- Jedinični test je prvi kreiran;
- Jedinični test se pokreče i završava neuspješno iz očitih razloga;
- Kôd je napisan (ili refaktoriran) samo da bi se test uspješno izvršio;
- Jedinični test se ponovno pokreče;
- Ako je test završio uspješno, prelazimo na sljedeći test, u suprotnom ispravljamo kôd tako da se test izvrši uspješno.

Beck (2002) je definirao dva jednostavna pravila primjene TDD-a:

- Napiši kôd samo onda kada je automatsko testiranje neuspješno
- Eliminiraj svako dupliciranje kôda koje pronađeš

Prema Beckovom iskustvu, dobar jedinični test mora sadržavati sljedeće karakteristike (Njenga, 2018):

- Test mora biti brz za izvođenje (kratko postavljanje testa, vrijeme izvođenja i prekida)
- Test se mora vrtjeti u izolaciji (programer mora biti u mogućnosti reorganizirati testove)
- Test mora koristiti podatke koji ga čine jednostavnim za čitanje i razumijevanje
- Test treba koristiti realne podatke kada je to potrebno
- Test mora biti uvijek jedan korak bliže cilju

Dakle, cilj nam je specificirati detaljan i izvediv dizajn za riješenje na pravovremenoj bazi. Pišemo jedan razvojni test koji je temeljen na jediničnom testiranju, te potom dovoljno kôda samo da bismo popunili test i da bi se on uspješno izvršio. Na taj način kreiramo jednostavan i čist kôd bez nepotrebnih funkcija i dodataka, a to je čest slučaj kod programera kada „u trenutku inspiracije“ kreiraju cijelu metodu.

3.3. Vrste testiranja

Kada gledamo sam koncept TDD-a, izgleda jednostavno, logično, ali kako implementirati i primjeniti TDD pristup u razvoju aplikacije? Postoji više vrsta testiranja i postavlja se pitanje kako znamo koja testiranja je potrebno primjeniti prilikom korištenja TDD pristupa?

Rahul Mody (2017) skreće nam pažnju na 14 različitih vrsta testiranja[2], od kojih su neki testovi napisani iz perspektive programera dok su ostali napisani iz perspektive klijenta odnosno krajnjeg korisnika. Iz perspektive razvojnog programera, TDD se može okončati kada razmišlja o tome kako odrediti svaku „jedinicu“ za testiranje i kako napisati testove koji pokrivaju „integraciju“ tih manjih jedinica. Iz navedenog, prema Modyu (2017), TDD zahtjeva namjanje dvije vrste testiranja, a to su jedinično i integracijsko testiranje, a Md Ataullah Khan (2017) u ovu skupinu dodaje funkcionalno testiranje i test prihvatljivosti.

Jedinično testiranje (*eng. Unit testing*) je najniži stupanj testiranja koji se može provesti. Uobičajeno je da se testiraju metode unutar klase. Također jedinični testovi ne

[2] http://www.aptest.com/testtypes.html (ApTest – specijalisti za testiranje softvera, 2013)

komuniciraju sa drugim klasama direktno već sa lažnim klasama (eng. mocks[3]). To čini jedinične testove izoliranima i jednostavnima za uklanjanje pogrešaka i refaktoriranje (Khan, 2017).

Integracijsko testiranje *(eng. Integration Testing*) obuhvaća više od jedne klase. Ovom vrstom testiranja testiramo integraciju između klasa, odnosno njihove ovisnosti. Integracijsko testiranje koristimo da bi testirali vraća li nam baza ispravne rezultate, da li vanjski API vraća ispravne podatke i slično. Testira stvarne klase i funkcionalnosti radije nego da koristi lažne instance. Integracijski testovi su značajno sporiji za izvedbu nego jedinični testovi jer komuniciraju sa bazom podataka i vanjskim poslužiteljima (Khan, 2017).

Funkcionalno testiranje (*eng. Functional Testing*) testira cijelu značajku koja može pozivati puno ovisnosti. Obično bismo testirali rutu za ispravan odgovor ili metodu Controller koja se odnosi na određenu značajku u aplikaciji. Ovi testovi su sporiji od integracijskih testova jer dodiruju puno više ovisnosti. Kao dio standardne strukture u alatima za razvoj možemo pronaći već generirani direktorij pod nazivom „Feature“ u kojem kreiramo funkcionalne testove (Khan, 2017).

Test prihvatljivosti (*eng. Acceptance Testing*) je najviši stupanj testiranja. On brine jedino da li značajke prolaze sa gledišta korisnika. Tijek testa prolazi kroz web stranicu klikčući i potvrđujući forme te očekuje ispravan rezultat. Za ovakvo testiranje koristimo alate poput Selenium-a (Khan, 2017).

3.4. Analiza najboljih praksi

Prema programerima XenonStack tvrtke (2019) najbolja praksa za usvajanje razvoja vođenog testiranjem je slijediti ove principe:

Autokarta (*eng. Road Map*) – slijediti crveno-zeleni pristup gradnje testa. Prvi korak je kreirati crveni test tako da napišemo minimalnu količinu kôda za pokretanje, potom nakon izlaganja problema vezanih uz kôd, potrebno je napraviti određene izmjene u kôdu i po potrebi ga refaktorirati sve dok se crveni test ne pretvori u zeleni test (Koutifaris, 2018).

- U crvenoj fazi moramo donijeti odluku kako će kôd biti korišten. Baziramo se na ono što nam treba u tom trenutku a ne na ono što bi nam moglo trebati. Također,

[3] *Eng. Mocking* – proces u kojem se kreira lažna instanca stvarne klase i testiramo ju. Radimo na taj način kako ne bi morali brinuti o stvarnoj funkcionalnosti vanjskih ovisnosti unutar klase.

nikako ne pišemo hrpu funkcija/klasa koje mislimo da ćemo trebati (Koutifaris, 2018).

- Zelena faza je najlakša jer pišemo produkcijski kôd. Ne smijemo raditi grešu i pisati cijele algoritme već dovoljno kôda da zadovoljimo test. Dozvoljeno je kršiti najbolju praksu i kreirati dupli kôd (ispravit ćemo to u fazi refaktoriranja). Ne pišemo cijeli kôd koji smo zamislili jer je jednostavan zadatak manje osjetljiv na pogreške i želimo minimalizirati greške (Koutifaris, 2018).
- Nakon završetka posljednje zelene faze dolazi faza refaktoriranja u kojoj je dopušteno mijenjati kôd sve dok su svi testovi zeleni. Mičemo dupli kôd, što je i jedina preporuka prema knjizi K. Becka. Trebamo imati na umu da se ne možemo preseliti na refaktoriranje sljedećeg testa sve dok nismo maknuli sav dupli kôd koji se tiče testa koji je trenutno u obradi (Koutifaris, 2018).

Implementacija (*eng. Implementation*) – važno je razdvojiti implementaciju izvornog kôda i testova. Obje grupe moraju imati vlastite direktorije (XenonStack, 2019). Testovima referenciramo izvorni kôd, a izvorni kôd ni na koji način nije ovisan o testovima.

Struktura (*eng. Structure*) – struktura za pisanje testova mora biti ispravna. Praksa je da se ime klase testa piše jednako kao i ime klase izvornog kôda, uz dodatak sufixa koji naglašava da je riječ o testu. Naprimjer, „Student“ – klasa izvornog kôda i „StudentTest“ – klasa testa. Isto vrijedi i za metode aplikacije, samo što metode testova dobivaju prefix. Naprimjer, „displayStudentName“ – metoda izvornog kôda i „testDisplayStudentName“ – metoda testa (XenonStack, 2019).

3.5. Alati

Kao što smo već ranije spomenuli, danas nam se pruža velik izbor alata i tehnologija kao pomoć u programiranju i testiranju. Kada prvi puta pristupamo TDD razvoju, teško je odlučiti koji alati su najbolji izbor, pa se oslanjamo na već ispitane i recenzirane alate.

Neke od najčešćih alata navodi i XenonStack (2019). To su: JUnit za jedinično testiranje, JMeter za ispitivanje opterećenja i preformansi, Mockito za ostalo API testiranje.

JUnit je razvojno okruženje za jedinično testiranje, dizajnirano za programski jezik Java. Jedinični testovi su najmanji elementi u procesu automatizacije ispitivanja. Uz njihovu pomoć programer može provjeriti poslovnu logiku bilo koje klase. JUnit igra vitalnu ulogu u TDD programiranju, a pripada obitelji razvojnih okvira za jedinično testiranje koja je zajednički poznata kao xUnit, a nastala je sa SUnit (XenonStack, 2019).

Apache JMeter može se koristiti za testiranje preformansi na statičkim i dinamičkim resursima, dinamičkim web aplikacijama, gdje se uglavnom upotrebljava za testiranje opterećenja ili preformansi). Koristi se i za simulaciju velikog opterećenja na poslužitelju, grupi poslužitelja, mreži ili objektu za testiranje njegove snage ili za analizu ukupnih performansi pod različitim vrstama opterečenja. Omogućava rad na različitim aplikacijama, poslužiteljima i tipovima protokola (XenonStack, 2019):

- Web – HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET);
- SOAP/REST Webservices;
- FTP;
- Database u JDBC;
- LDAP;
- Message-oriented middleware (MOM) u JMS;
- Mail – SMTP(S), POP3(S) i IMAP(S);
- Izvorne naredbe ili ljuske skripti;
- TCP;
- Java Objects.

Mockito je dizajniran kao razvojno okruženje otvorenog koda za Javu koji je dostupan pod MIT licencom. Mockito omogućuje programerima stvaranje i testiranje dvostukih objekata (lažnih objekata) u automatiziranim jeidničnim testovima za TDD razvoj. Jednostavno rečeno, Mockito je razvojno okruženje koje programeri koriste za učinkovito pisanje određenih vrsta testova (XenonStack, 2019).

Naravno, postoje mnogi drugi alati i razvojna okruženja, ovisno o programskom jeziku, programer može odabrati odgovarajući alat i razvojno okruženje za TDD (XenonStack, 2019).

Reprezentativna lista prema Ambleru (2020) sadrži alate prikazane u tablici 1. Alate smo grupirali u skupine koje sadrže: xUnit alate, alate za testiranje baza podataka, module, osnovne preglednike, knjižnice i biblioteke te ostale alate:

Tablica 1: Reprezentativna lista TDD alata

Naziv alata	Opis
xUnit alati → zajednički naziv za nekoliko razvojnih okvira za jedinično testiranje koji svoju strukturu i funkcionalnost izvode iz Smalltalk-ovog SUnit-a. SUnit je kreiran od strane Kent Beck-a 1998. godine, napisan je u visoko strukturiranom objektno orijentiranom stilu, koji je lako posuđivao suvremene jezike kao što su Java i C#. Kent Beck i Erich Gamma prenijeli su razvojno okruženje na Javu te je on stekao široku popularnost. Imena mnogih razvojnih okruženja su varijacije „SUnit"-a, obično zamjenjujući „S" prvim slovom ili slovima u imenu željenog jezika. („JUnit" za Javu, „Runit" za R i tako dalje) (*xUnit*, 2020).	
CppUTest	CppUTest je razvojni okvir za jedinično testiranje i stvaranje lažnih instanci. Namijenjen je za korištenje sa programskim jezicima C i C++. Može se koristiti na Linuxu i Windowsima. Link: http://cpputest.github.io/
DUnit (Delphi)	Omogućava programerima Delphija (object Pascal) primjenu TDD-a. Postao je standardni dio Delphi razvojnog okruženja od 2005 godine, a kreiran je na temelju JUnit. Nekoliko programera sada održava DUnit kao projekt na SourceForgeu. Link: https://sourceforge.net/projects/dunit/
JUnit	JUnit je kreiran od strane Kenta Beck i Ericha Gamma. To je jednostavno razvojno okruženje za kreiranje ponavljajućih testova, te pripada skupini xUnit arhitekture razvojnih okruženja za jedinično testiranje u programskom jeziku Java. Može se koristiti na svim platformama. Link: https://junit.org/junit4/
NUnit	NUnit je razvojno okruženje za jedinično testiranje zasnovan na .NET platformi. To je besplatan alat koji omogućuje ručno pisanje testinih skripti ali ne i automatskih. Radi na isti način kao što JUnit radi za Javu. Podržava

	testove vođene podacima koji se mogu izvoditi paralelno. Koristi Console Runner za učitavanje i izvršavanje testova. Link: https://nunit.org/
OUnit	OUnit je razvojno okruženje za jedinično testiranje za OCaml. Omogućava jednostavno kreiranje jediničnih testova za OCaml kôd. Razvoj je temeljen na HUnit, razvojnom okruženju za jedinično testiranje za Haskell. Sličan je JUnit i drugim xUnit razvojnim okruženjima za testiranje. Link: https://github.com/gildor478/ounit
PHPUnit	PHPUnit je programski orijentirano razvojno okruženje za testiranje za PHP. Pripada xUnit arhitekturi za razvojna okruženja za jedinično testiranje. Link: https://phpunit.de/
PyUnit (Python)	PyUnit je lagan način kreiranja programa za jedinično testiranje u Pythonu. Link: https://wiki.python.org/moin/PyUnit
Test::Unit (Ruby)	Test::Unit je razvojno okruženje za jedinično testiranje u Ruby-u. Pomaže u dizajniranju, uklanjanju pogrešaka i procjeni kôda olakšavajući pisanje i provođenje testova za kôd. Link: https://test-unit.github.io/
TestNG	Poput JUnit-a, TestNG je također okvir za automatsko testiranje otvorenog kôda za programski jezik Java. Ovaj alat je pod snažnim utjecajem JUnit-a i NUnit-a s istodobnim testiranjem i podrškom za napomene. TestNG podržava parametarsko, jedinično, funkcionalno i integracijsko testiranje. Link: https://testng.org/doc/index.html
VBUnit	VBUnit napisan je za podršku Visual Basic 6.0.. On je prvi član koji je podržao SuiteFixture Setup, a ujedno i onaj koji je uveo koncept naziva Testcase Class testnim uređajem. Nije besplatan, a druga mogućnost za

	VB i VBA programere je VB Lite Unit ili za .Net programere, NUnit, CsUnit i MbUnit. Link: http://xunitpatterns.com/VbUnit.html
XtUnit	XtUnit je proširenje za jedinično testiranje za razvojna okruženja temeljena na .Net-u. Koristi aplikacijski blok presretanja da bi omogućio vraćanje svih promjena koje su napravljene u bazi podataka tijekom jediničnih testova povezanih s podacima. Link: https://www.nuget.org/packages/XT-Unit.Unofficial/
xUnit.net	xUnit.net je besplatni alat za jedinično testiranje namjenjen za .NET razvojno okruženje i najnovija tehnologija za jedinično testiranje sa C#, F#, VB.NET i ostalih .NET jezika. Radi sa ReSharper-om, CodeRush-om, TestDriven.NET-om i Xamarin-om. Dio je .NET zaklade i djeluje prema njihovom kodeksu ponašanja. Link: https://xunit.net/
Googletest	Googletest je xUnit-ovo razvojno okruženje. Moguće ga je koristiti na sljedećim platformama: Linux, Mac OS X, Windows, Cygwin, MinGW, Windows Mobile, Symbian, PlatformIO. Osnovni zahtjevi za izgradnju i upotrebu GoogleTesta iz izvornog paketa su korištenje Bazel ili CMake sustava i C++ kompajlera. Bazel je sustav izrade koji GoogleTest interno koristi i testira, a CMake podržava zajednica. Link: https://github.com/google/googletest
Alati za testiranje baze podataka	
DBFit	DBFit je alat za kreiranje baze podataka TDD pristupom. Omogućava jednostavno jedinično i integracijsko testiranje kôda. Pruža podršku za Oracle, SQL Server, MySQL, DB2, PostgreSQL, HSQLDB i Derby. Testovi se mogu pokrenuti iz naredbenog retka bilo kojeg Java IDE ili CI alata za izgradnju.

	Link: https://dbfit.github.io/dbfit/
DBUnit	DBUnit, proširenje je JUnit-a (također se može koristiit s Antom) usmjereno na projekte vođene bazom podataka, koji između ostalog stavlja bazu podataka u poznato stanje između testnih pokretanja. To je izvrstan način da se izbjene bezbroj problema koji se mogu pojaviti kada jedan testni slučaj ošteti bazu podataka i uzrokuje da kasniji testovi ne uspiju ili pogoršaju štetu. Ima mogućnost izvoza/uvoza podataka baze u i iz XML skupova podataka. Od verzije 2.0, DBUnit može raditi s vrlo velikim skupovima podataka. Također, može pomoći u provjeri da li se podudaraju podaci baze podataka s očekivanim skupom vrijednosti. Link: https://www.dbunit.org/
Moduli	
DocTest (Python)	DocTest je modul uključen u standardnu knjižnicu programskog jezika Python koji omogućuje jednostavno generiranje testova na temelju izlaza iz standardne ljuske tumača Python, izrezanog i zaljepljenog u docstrings. Link: https://docs.python.org/2/library/doctest.html
Osnovni preglednici	
HTMLUnit	HTMLUnit je preglednik bez GUI-ja za Java programe, a koristi se kao osnovni „preglednik“ različitih alata otvorenog kôda, kao što su Canoo Web Test, JwebUnit, WebDriver, JSFUnit, WETATOR, Celerity, Spring Testing,... Pruža poseban način simulacije preglednika u svrhu testiranja. Modelira HTML dokumente i pruža API koji omogućuje pozivanje stranica, popunjavanje obrazaca, klikanje poveznica i tako dalje, baš kao što to radite u svom „normalnom“ pregledniku. Namijenjen je korištenju u drugim razvojnim okruženjima za testiranje, poput JUnit-a ili TestNG. Link: https://htmlunit.sourceforge.io/

HTTPUnit	HTTPUnit omogućava zaobilaženje pregledinika i pristup web mjestu iz programa. Napisan je u Javi, oponaša relevantne dijelove ponašanja preglednika, uključujući predaju obrasca, JavaScript, osnovnu http provjeru autentičnosti, kolačiće i automatsko preusmjeravanje stranica. Java testnom kodu omogućava pregled vraćenih stranica u obliku teksta, XML DOM-a ili spremnika obrazaca, tablice i poveznice. U kombinaciji sa razvojnim okruženjima kao što je JUnit prilično je jednostavno napisati testove koji vrlo brzo provjeravaju funkcioniranje web mjesta. Link: http://httpunit.sourceforge.net/
Knjižnice i biblioteke	
JMock	JMock je knjižnica koja podržava testni razvoj Java kôda s lažnim objektima. Lažni objekti pomažu u dizajniranju, testiranju i interakciji između oobjekata u programima. Biblioteka JMock: - Olakšava brzo i jednostavno definiranje lažnih objekata, tako da se ne narušava ritam programiranja - Omogućava precizno određivanje interakcija između vaših objekata smanjujući lomljivost testova - Dobro funkcionira sa značajkama automatskog dovršavanja i refaktoriranja vašeg IDE-a - Uključuje se u vaš omiljeni testni okvir Link: http://jmock.org/
Moq	Moq je knjižnica .NET razvojnog okruženja za stvaranje lažnih objekata. Koristi lambda izraze C# 3.0., koji se obično koriste u TDD-u. Link: https://github.com/moq/moq
NDbUnit	NDbUnit je .NET biblioteka za upravljanje stanjem baza podataka tijekom jediničnog testiranja. Posuđuje mnogo ideja iz DbUnit-a i čini ih dostupnima za .NET platformu, a napisan je u programskom jeziku C#.

	Podržava sljedeće poslužitelje baza podataka: Microsoft SQL Server 2005 2008, 2012, 2014; Microsoft SQL Server CE 2005 and 2008; Microsoft OleDB-supported databases; SQLLite; MySQL; PostgreSQL through 9.4; Oracle. Link: https://github.com/NDbUnit/NDbUnit
CUnit	CUnit je lagan sustav za pisanje, administriranje i izvođenje jediničnih testova u programskom jeziku C. Pruža C programerima osnovnu funkcionalnost za testiranje sa fleksibilnim raznim korisničkim sučeljima. CUnit je izgrađen kao statička knjižnica koja je povezana s korisničkim testnim kôdom. Koristi jednostavano razvojno okruženje za izgradnju testnih struktura i pruža bogat skup tvrdnji za testiranje uobičajenih tipova podataka. Uz to, osigurano je nekoliko različitih sučelja za pokretanje testova i izvještavanje o rezultatima. Može se koristiti sa Linux operativnim sustavom. Link: http://cunit.sourceforge.net
Ostali alati	
SimpleTest	SimpleTest je razvojno okruženje za jedinično tetiranje otvorenog kôda posvećen PHP programskom jeziku. To okruženje podržava SSL, obrasce, proxy-e i osnovnu provjeru autentičnosti. Ideja je da se uobičajeni ali nezgrapni PHP zadaci, poput prijave na web mjesto mogu lako testirati. Uključuje autorun.php.file za pretvaranje test slučajeva u izvršne testne skripte. Link: http://simpletest.sourceforge.net/
TestOoB (Python)	TestOoB je napredno razvojno okruženje za jedinične testove, kreiran za programski jezik Python. Jednostavno se integrira s postojećim PyUnit testnim paketima. Link: https://wiki.python.org/moin/TestOob

(Izvor: Ambler, 2020)

Prema Software Testing Help portalu koji se fokusira na testiranje softvera i analizu kvalitete (Software Testing Help [STH], 2020) i A. Monus (2020) slažu se mišljenja da su alati navedeni u tablici 2 najbolji alati za korištenje PHP jezika. Izdvojit ćemo one koji su primjenjivi kod korištenja TDD pristupa razvoja aplikacija, ostali koji se koriste prilikom primjene BDD tehnike su: Behat, Kahlan, Cucumber i PHPSpec.

Tablica 2: Lista najboljih alata i okvira za korištenje sa PHP jezikom

Naziv alata	Značajke
PHPUnit	- Besplatno razvojno okruženje za kreiranje jediničnih testova - Koristi se sa cmd-om - Moguće je proširiti testove ovisno o potrebama - Koristi tvrdnju kôda za testiranje ponašanja jedinice - Jednostavan i lagan okvir za jedinično testiranje - Dostupan na: https://phpunit.de/
StoryPlayer	- Besplatan alat za automatizaciju funkcionalnih i ne fukncionalnih zahtjeva - Koristi se za testiranje web-aplikaci-ja ili API-a - Koristi se za testiranje komponenti kao i za testiranje od kraja do kraja - Testiranje platformi od kraja do kraja - Dizajniran je za programere i testere - Može se koristiti i za testiranje kôda napisanog drugim jezicima osim PHP-a - Radi s web preglednicima i kombinacijama platformi - Može se proširiti vlastitim dodacima - * potrebno ga je instalirati sa composer-om - Dostupan na: https://datasift.github.io/storyplayer/
Codeception	- Besplatan alat za kreiranje testova prihvatljivosti, jediničnih testova i funkcionalnih testova - Jednostavan za korištenje i kompaktan - Radi bolje kada se integrira sa Selenium-om

	- Čini kôd jednostavnim za čitanje, pisanje i otklanjanje pogrešaka - Također se naziva i „razvoj vođen ponašanjem“ (eng. Behavior Driven Development – kraće BDD) - Dostupan na: https://codeception.com/install
Selenium i SeleniumHQ	- Besplatan alat za automatizaciju testova - Podržavaju ga svi operacijski sustavi - Regresijski testovi mogu biti automatizirani - Najšire korišten automatizirani alat - Ima svoj vlastiti API za web upravljačke programe - Koristi se u TDD-u - Dostupan na: https://www.selenium.dev/downloads/
Atoum	- Besplatan alat ako se koristi za osobnu upotrebu - Namijenjen za jedinično testiranje - Podržava davatelje podataka i automatsko pokretanje - Testni slučajevi se vrte paralelno - Jednostavna, lagana i fleksibilna struktura - Podržava značajne tvrdnje - Pomaže pisati lažne instance - Dostupan na: https://github.com/atoum/atoum
SimpleTest	- Besplatan alat - Koristi se za jedinično testiranje - Podržava najednostavniji HTML prikaz - Test slučajevi se automatski učitavaju - Upravlja kolačićima kada dohvaća razne stranice - Testni slučajevi prikazani su bez web preglednika - Podržava Proxy, SSI, obrasce i okvire i tako dalje - Dostupan na: https://github.com/simpletest/simpletest
Xdebuger	- Besplatno testno razvojno okruženje za razvoj i uklanjanje grešaka iz kôda - Kada se dogodi stanje pogreške, pokazuje spremljene korake - Koristi se za profiliranje iskoristivosti memorije - Koristi se za uklanjanje pogrešaka na daljinu

	- Omogućava funkcije za prijavu u različitim formatima - Dostupan na: https://xdebug.org/download
Peridot	- Koristi se za BDD i TDD testiranje - Brzo izvršavanje - Koristi opisnu sintaksu kao da koristi BDD testno razvojno okruženje koje pomaže razumjeti rečenice - Baziran je na SpecBDD i arhitekturi temeljenoj na događajima - Dostupan na: https://github.com/peridot-php/peridot

(Izvor: STH i Monus, 2020)

Prema listi najboljih alata i razvojnih okvira za korištenje sa PHP jezikom, vidimo da je PHPUnit najbolje ocijenjen. Slovi kao jednostavan i lagan za provedbu jediničnog testiranja, a na internetu možemo pronaći i dosta primjera sa pojašnjenjima o korištenju navedenog alata. Uza sve navedeno i odluku o izradi testova uz pomoć Laravel razvojnog okruženja (najboljeg izbora za početnike u TDD-u), već imamo uključen i alat PHPUnit koji ćemo sukladno tome i koristiti pri izradi jediničnih testova.

3.6. Prednosti razvoja vođenog testiranjem

XenonStack tim piše o alatima TDD-a i najboljim praksama (*eng. „Test Driven Development Tools and Agile Process“*) govori kako slijediti TDD pristup razvoja aplikacija znači programirati sa manje grešaka razvijajući veću kvalitetu softvera i fokusirati se na jednu funkcionalnost u određenom trenutku. Među ostalim, izdvaja i prednosti TDD razvoja (XenonStack, 2019):

- Omogućava promišljanje kroz zahtjeve ili dizajn prije razvoja funkcionalnog kôda;
- Omogućava programeru da gradi softver korak po korak, što nazivamo kodiranje u „malim koracima“;
- Produktivniji je u usporedbi sa pokušajem programiranja u globalu;
- Kada gledamo primjer gdje programer napiše određeni kôd, komplilira ga i testira, možda postoji mogućnost neuspjeha. U tom slučaju, jednostavno je pronaći i ispraviti pogreške ako je programer napisao nekoliko linija kôda, umjesto nekoliko tisuća linija.

U konačnici, TDD je vještina koju programeri usavršavaju. Ona štedi vrijeme koje se inače troši na prepravke kôda, te omogućava brže uočavanje grešaka i problema.

3.7. Nedostatci razvoja vođenog testiranjem

Jedan od glavnih nedostataka TDD razvoja je što ova tehnika zahtjeva puno znanja o novim praksama i tehnikama. Kao na primjer: eng. Unit testing, eng. mocks, eng. assertions i tako dalje. Također potrebno je više vremena unaprijed za razvoj, jer se testira u korak sa razvojem aplikacija (Khan, 2017).

Među ostale nedostatke možemo svrstati troškove implementiranja funkcionalnosti, održavanje testnog kôda, te činjenicu da previše TDD-a čini kôd kompliciranijim nego je to potrebno.

3.7.1. Troškovi implementiranja funkcionalnosti

Glavni nedostatak TDD-a je taj što nam je potrebno puno više vremena za implementaciju funkcionalnosti nego što nam je potrebno u slučaju standardnog ručnog testiranja. Troškovi ovise o nekoliko stvari među kojima su i sljedeći problemi (JRebel, 2016):

- Iskustvo TDD programera – manje iskustva povlači više troškova;
- Postojeća podrška kôdova za automatizirano testiranje – pokušaj automatiziranja testova na razvojnom okruženju koje nije namjenjeno za tu svrhu donjet će više troškova nego profita;
- Organizacijska podrška TDD-a – možda je jednostavno nagovoriti svoj odjel na korištenje TDD pristupa, no ako naš projekt ovisi o drugim odjelima tada možda nećemo iskoristiti sve prednosti TDD-a. Na primjer, ako nam je potrebna kopija komponenta odjela sa kojim surađujemo i svi njihovi testovi da bismo mogli pokrenuti integracijske testove, a oni nam ih ne žele ustupiti, tada nismo u mogućnosti provesti integracijske testove.

Kada je vrijeme na tržištu bitna stavka tada vrijedi razmotriti brzu izradu verzije bez automatskog testiranja. To je kažu „skliska cesta“ jer će kasnije promjene biti više podložne greškama što je teže i sporo zbog manjka regresijskih automatiziranih testova (JRebel, 2016).

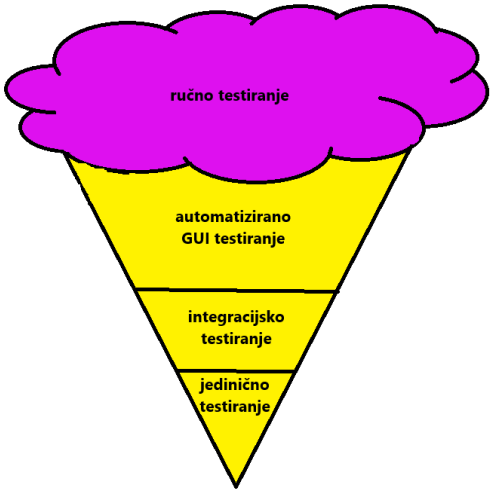
3.7.2. Testni kôd zahtjeva održavanje isto kao i razvojni kôd

Testni kôd raste zajedno sa produkcijskim kôdom, što je drugi problem TDD-a. Sve linije kôda zahtjevaju održavanje i to znači troškove. Testovi moraju biti modificirani jednako kao i produkcijski kôd (JRebel, 2016).

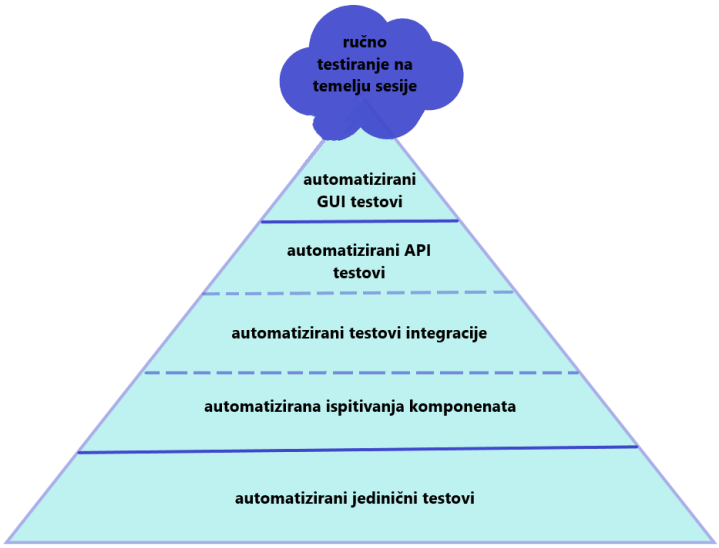
Navedeno postaje problem kada više testova izvršava istu liniju proizvodnog kôda. To se često događa u slučajevima kada imamo previše testova na visokoj razini. Testovi najviše

razine su testovi od kraja do kraja (eng. end-to-end test) koji započinju buđenjem GUI-a i kreću se skroz do razine gdje su pohranjeni podaci. No nemojmo misliti da samo testovi najviše razine mogu uzrokovati probleme (JRebel, 2016).

Testovi koji pokreću cijelu komponentu, primjerice web uslugu, također su visoko rangirani u usporedbi sa jediničnim testovima. Anti-uzorak sladoledne konzole, grafički prikazan na slici 2, lijepa je ilustracija kako TDD može otići u pogrešnom smjeru. Konus sladoleda nastaje kada preokrenemo idealnu piramidu za automatizaciju ispitivanja prikazanu na slici 3. Sladoled možemo napraviti tako što ćemo stvoriti više integracijskih testova od jediničnih i više GUI testova od integracijskih testova te još više ručnih regresijskih testova od automatiziranih GUI testova (JRebel, 2016).



Slika 2. Anti-uzorak sladoledne konzole za testiranje softvera (Prema: JRebel, 2016)



Slika 3. Idealna piramida za automatizaciju testova (Prema: JRebel, 2016)

3.7.3. Previše TDD-a čini kôd kompliciranijim nego što je potrebno

Treći problem odnosi se na previše TDD-a gdje testovi čine kôd više kompleksnim nego što je to potrebno. Jedinično testiranje MVC kontrolera nema smisla, to bi trebao biti integracijski test. Ne smijemo pokušavati eliminirati bazu podataka iz modela testiranja. Učitavanje baze traje samo 1 do 2 sekunde i možemo koristiti transakcijska tekstualna učvršćenja kako bi generirali različite scenarije (JRebel, 2016).

3.8. Kada primjeniti razvoj aplikacije vođen testiranjem i na što treba obratiti pažnju prilikom donošenja odluke?

TDD je odličan u detaljnoj specifikaciji i validaciji ali nije tako dobar u rješavanju većih problema kao što je cjelokupan dizajn, kako će korisnik koristiti sustav ili za dizajn korisničkog sučelja (*eng. User Interface – kraće UI)*. Prema Scott W. Ambleru (2020), direktoru Ambysoft Inc., bolje rješenje u tom području je razvoj vođen agilnim modelom (*eng. Agile Model-Driven Development – kraće AMDD*).

AMDD bi se trebao koristiti za kreiranje modela sa dionicima projekta kako bi pridonijeli istraživanju njihovih zahtjeva te kasnijem implementiranju tih zahtjeva u arhitekturalnom i dizajnerskom modelu. S druge strane, TDD bi trebao biti korišten kao ključan dio razvoja projekta kako bi kôd bio čist i funkcionalan (Ambler, 2020).

Primjena TDD pristupa ne preporuča se prilikom kreiranja grafičkog korisničkog sučelja jer košta više od ručnog testiranja. Iz tog razloga, bilo bi pragmatično automatizirati regresijske GUI testove nakon završetka razvojne faze. Generalno, automatizacija GUI testa je skupa prema sljedećim aspektima (JRebel, 2016):

- **Složenost** – broj mogućih korisničkih interakcija iz različitih aplikacijskih stanja raste eksponencijalno kako rastu GUI funkcionalnosti;
- **Verifikacija** – kako raste složenost postaje teže kreirati automatizirane testne skripte koje pokrivaju sve moguće slučajeve;
- **Promjene** – GUI se često mijenja tijekom razvoja, zbog čega održavanje testne skripte postaje teret.

Zbog ovih problema postavlja se pitanje: Kako možemo spojiti TDD i GUI razvoj aplikacija? Odgovor na to pitanje daje nam Kent Beck i predlaže kreiranje tankog sloja prikaza

zajedno sa temeljnim modelom iz kojih se razvija aplikacija vođena testiranjem, a iz ove ideje nastali su sljedeći softverski uzorci (JRebel, 2016):

- Presentation Model
- Model-View-ViewModel
- Model-View-Binder

Cilj ovih uzoraka je napraviti sloj prikaza što je moguće jednostavnije, tako da testiranje nije potrebno. Svi navedeni uzorci rade na sličan način. Prikaz ima temeljni model koji sadrži polja za svaki prikaz elementa ili stanja. Sloj prikaza delegira sve akcije na temeljni model, isto tako ako je potrebno ažurirati sloj prikaza, temeljni model to zahtjeva (JRebel, 2016).

Navedeni pristup pokriva većinu kôda sa automatskim testovima, jedino preostaje pitanje da li je stvarni prikaz ispravno integriran sa temeljnim modelom. Kako bismo to provjerili preporuča se nekoliko testova koji bude GUI, kao na primjer kroz web poslužitelj za web aplikacije. Ti se testovi ne moraju pisati na TDD način (JRebel, 2016).

TDD može dati povratnu reakciju kada okruženje nije primjereno ili nije ispravno korišteno. Ako planiramo koristiti TDD pristup potrebno je razmotriti sljedeće (JRebel, 2016):

- Koliki su troškovi implementiranja funkcionalnosti;
- Testni kôd zahtjeva održavanje isto kao i razvojni kôd;
- Previše TDD-a čini kôd kompliciranijim nego je potrebno.

Postoji sukob između dugoročnih i kratkoročnih ciljeva za timove za izradu softvera. Kratkoročno, najbrži način je razviti funkcionalnost, testirati je ručno, ispraviti programske pogreške, ponovno testirati i tako dalje, dok funkcionalnost ne bude spremna. Ne postoje argumenti da će TDD biti brži u kratkoročnim ciljevima. No, ako pogledamo dugoročne ciljeve neki problemi rastu sa ručnim testiranjem (JRebel, 2016):

- Vrijeme namjenjeno ručnom regresijskom ispitivanju raste u korak sa projektom iz čega proizlazi da su ciklusi otpuštanja duži;
- Može se zamrznuti vrijeme testiranja regresije kako bi se zamrznulo vrijeme ciklusa izdanja, ali tada sve više i više grešaka ide u proizvodnju;
- Kvaliteta kôda pati što čini dodavanje novih funkcionalnosti sporijim kako projekt raste.

Konačan rezultat korištenja AMDD-a i TDD-a je visokokvalitetan i funkcionalan sustav koji zadovoljava potrebe klijenata. Glavna prednost TDD-a je da omogućava razvoj softvera u malim koracima, što je produktivnije (Ambler, 2020).

Iz navedenog vidimo da se obavezno moramo fokusirati na to kako ćemo izgraditi čišći testni kôd a ne brži i izoliraniji.

4. Osiguranje kvalitete web aplikacije

Kao što nam je svima poznato, web stranice koje ne rade kako to očekujemo ili imaju puno grešaka u izvođenju, neželjenih preusmjeravanja i sličnih situacija, stvaraju odbojnost i rijetko kada im ponovno pristupamo ili ih imamo želju što manje koristiti. Da bismo spriječili takav utisak i stvorili što bolje korisničko iskustvo a time i zadovoljstvo korišenja našim web aplikacijama i stranicama primjenit ćemo postupak osiguranja kvalitete.

U ovom poglavlju posvetit ćemo više pažnje pojašnjenju osiguranja kvalitete web stranice i web aplikacije i pojasniti što to zapravo znači, zašto je ono važno, kako se provodi i koji alati nam mogu pomoći provesti osiguranje kvalitete što uspješnije i kvalitetnije. Pa krenimo od same definicije i pojašnjenja pojma.

4.1. Definicija i značenje osiguranja kvalitete

Osiguranje kvalitete (*eng. Quality Assurance – kraće QA*) je bilo koji sustavni postupak utvrđivanja ispunjava li proizvod ili usluga određene zahtjeve (Rouse i suradnici, 2019).

QA uspostavlja i održava set potreba za razvoj ili prozvodnju pouzdanog proizvoda. Trebala bi povećati korisničko povjerenje i tvrtkin kredibilitet, a isto tako poboljšava radni proces i učinak, te osigurava da tvrtka bude bolja u odnosu na konkurenciju. Koncept osiguranja kvalitete kao formalizirane prakse započeo je u proizvodnoj industriji i od tada se proširio na većinu industrija, uključujući razvoj softvera (Rouse i suradnici, 2019).

Internacionalna organizacija za standardizaciju (eng. International Organization Standardization – kraće ISO) je pokretačka snaga QA prakse i mapiranja procesa koji se koriste za provedbu QA-a. QA je često uparen s međunarodnim standardom ISO 9000, kojeg mnoge tvrtke koriste kako bi osigurale da njihov sustav osiguranja kvalitete bude na snazi i učinkovit (Rouse i suradnici, 2019).

4.2. Standardi osiguranja kvalitete

Rose u članku o osiguranju kvalitete govori kako se koncept QA javio još u srednjem dobu, a tijekom drugog svjetskog rata, kada se javlja potreba provjere velikog broja streljiva, QA dobiva još više na važnosti. ISO je otvoren u Ženevi 1947. godine, a svoj prvi standard na referencu temperature za industrijska mjerenja, objavljuje 1951 godine. ISO je postupno rastao i širio svoj opseg standarda. Serija ISO 9000 standarda objavljena je 1987. godine, gdje svaki

od 9000 brojeva nudi drugačiji standard ovisno o različitim scenarijima (Rouse i suradnici, 2019).

Govoreći o QA standardima, kao i većina njih, ažurirani i nadopunjavani su s vremenom jer trebaju ostati u korak sa današnjim poslovnim svijetom. Posljednji u svojoj seriji je ISO 9001:2015. Njegove smjernice uključuju jači fokus na klijenta, vrhunske prakse menadžmenta i kako one mogu promijeniti tvrtku, te održavanje prostora za kontiuniranim poboljšanjima. Ujedno sa općim poboljšanjima ISO 9001, ISO 9001:2015 uključuje poboljšanja njegove strukture i više informacija za donošenje odluka na temelju rizika (Rouse i suradnici, 2019).

4.3. Osiguranje kvalitete i kontrola kvalitete

Kada se govori o osiguranju kvalitete mnogi je miješaju sa kontrolom kvalitete (eng. quality control – kraće QC). Ova dva koncepta su slični ali su bitno različiti. Dok QA pruža smjernice koje su primjenjive gotovo svugdje, QC je proces čiji je fokus na proizvodnji, odnosno inspekciji proizvodnje. QA je bilo koji sistematski proces koji osigurava da proizvod zadovoljava specificirane zahtijeve, a QC ukazuje na druge probleme, kao što su individualne inspekcije ili defekti. U slučaju programskog razvoja, QA nastoji spriječiti greške u kôdu proizvoda, a QC provodi testiranje, riješava probleme i popravlja kôd (Rouse i suradnici, 2019).

Spomenute aktivnosti vezane uz osiguranje i kontrolu kvalitete mogu se primjeniti na bilo koji proizvod, a ne samo softver. Sukladno tome kada govorimo o osiguranju kvalitete programskog proizvoda, QA postaje osiguranje kvalitete softvera (eng. Software Quality Assurance – kraće SQA), a QC postaje testiranje softvera (eng. Software Testing). U tablici 3 prikazane su razlike između SQA i testiranja softvera (Guru99, 2020).

Tablica 3 : Pregled razlika između osiguranja kvalitete softvera i testiranja softvera

SQA	Testiranje softvera
Osiguranje kvalitete programa govori o inženjerskom procesu koji osigurava kvalitetu.	Testiranje softvera odnosi se na testiranje proizvoda i otkrivanje porblema prije nego se proizvod daje na korištenje.

Uključuje aktivnosti povezane s provedbom procesa, postupaka i standarda, na primjer, obuka za reviziju.	Uključuje aktivnosti u vezi s provjerom proizvoda, na primjer – pregledno ispitivanje.
Fokus je na procesu.	Fokus je na proizvodu.
Primjenjuje preventivne tehnike.	Primjenjuje korektivne tehnike.
Poduzima proaktivne mjere.	Poduzima reaktivne mjere.
Opseg SQA odnosi se na sve proizvode koje će organizacija stvoriti.	Opseg testiranja softvera odnosi se na određeni proizvod koji se ispituje.

(Izvor: *Guru99, 2020*)

Osiguranje kvalitete služi da bismo provjerili je li proizvod pogodan za korištenje, a da bismo to mogli, organizacije bi trebale imati procese i standarde koje mogu slijediti i koji moraju biti ažurirani na redovnoj bazi. Najvećim dijelom, koncentrira se na kvalitetu proizvoda ili servisa kojeg osiguravamo za korisnika tijekom ili nakon implementacije softvera (Guru99, 2020).

4.4. Zašto je osiguranje kvalitete važno za proces razvoja web aplikacije?

Da bi isporučili visoko kvalitetne proizvode, osiguranje kvalitete mora biti na mjestu. Držeći se strogih smjernica standardiziranog QA procesa, otkrit ćemo skrivene pogreške koje mogu prouzročiti višestruke probleme na stranici, oštetiti njenu funkcionalnost i potkopati korisničko iskustvo. Štoviše, osiguranje kvalitete može spriječiti postavljanje web aplikacije s lošim učinkom i spasiti ugled, vrijeme i novac organizacije (Prestianni, 2020). Ona osigurava da krajnji korisnik dobije funkcionalno sučelje i najbolje iskustvo prilikom korištenja web aplikacije ili stranice. Iako je osiguranje kvalitete preduvjet za svaki postupak koji želi pružiti visokokvalitetne proizvode, QA testiranje potrebno je provoditi rutinski čak i nakon završetka razvoja. To omogućava stalno lociranje i ispravljanje svih problema koji bi se mogli pojaviti na web stranici ili web aplikaciji. Berezhnoi (2019) opisuje dalekosežne blagodati rutinskog QA testiranja:

- QA osigurava da web aplikacija radi perfektno;
    - Prvo i osnovno, moramo potvrditi da nešto radi ispravno prije nego to postavimo dostupno svima. QA testeri koriste različitte alate za testiranje, kao što su virtualne mašine (eng Virtual Machines – kraće VMs), da bi simulirali različite internetske interakcije sa našom aplikacijom ili stranicom. Radeći na taj način, tester potvrđuje da li proizvod pruža sve očekivane funkcionalnosti.
- QA pomoći će u jačanju korisničkog iskustva (eng. user expirience – kraće UX);
    - QA mjeri upotrebljivost web aplikacije ili web stranice koja je direktno vezana sa korisničkim iskustvom. QA, posebno kada je potkrijepljen podacima o korisničkom testiranju, pomoći će u pronalasku načina za maksimalnu iskoristivost i posljedično optimalno korisničko iskustvo.
- QA jača brend tvrtke;
    - QA može ojačati brend tvrtke, no ako damo korisnicima proizvod koji ne radi kako treba ili čak ne radi uopće, tada se dovodimo do sljedećih situacija:
        - Korisnici će postati frustrirani sa našim proizvodom
        - Izgledati ćemo nesposobno ili ćemo raditi sa nesposobnim timom
        - Naš će proizvod biti ocijenjen kao loše kvalitetan i time bezvrijedan kupovanja i korištenja
        - Izgledati će kao da zapravo ne cijenimo ili ne marimo za svoj posao
        - Na poslijetku ćemo izgubiti povjerenje klijenata
- QA spriječava nastanak katastrofa.
    - Programeri su ljudi i samim time imaju predispozicije za pogreške. Nije neobično griješiti tokom programiranja. Također je moguće da napišu kôd koji ne radi kako je očekivano ili koji je nekompatibilan sa pojedinim tehnologijama koje se koriste u web aplikaciji ili na web stranici. Neke od tih pogrešaka mogu proći nezapaženo sve dok jednog dana ne prouzroče katastrofalni pad ili narušavanje sigurnosti i privatnosti. Stoga je pravilno testiranje preporučeno da bi pospješili otkrivanje nekih kritičnih pogrešaka koje mogu potencijalno prouzročiti probleme unutar naše web aplikacije ili web stranice.

QA provodi se neovisno o dizajnu i razvojnom procesu, a svrha joj je testirati funkcionalnosti web aplikacije. QA osigurava otkrivanje problema u dizajnu i razvojnih pogrešaka dok testira korisničko sučelje proizvoda i mjeri korisničko iskustvo. U mnogim slučajevima, korisnička lista osigurava da se ništa ne propusti tokom procesa osiguranja

kvalitete i da web aplikacija radi besprijekorno, ali unatoč tome potrebno je kontinuirano testirati i ažurirati web aplikaciju/stranicu i osigurati izuzetno iskustvo za korisnika (Prestianni, 2020).

4.5. Metode osiguranja kvalitete

Rouse i suradnici u svom blogu opisuju tri metode koje se mogu primjeniti prilikom osiguranja kvalitete proizvoda: ispitivanje kvarova, statistička kontrola procesa i ukupno upravljanje kvalitetom (Rouse i suradnici, 2019).

Ispitivanje kvarova (eng. Failure testing) je metoda koja kontinuirano provodi testiranje proizvoda kako bi utvrdila da li se proizvod „ruši“ ili ne radi. Za fizičke proizvode to može uključivati testiranje proizvoda pod utjecajem vrućine, pritiska ili vibracije, dok se za programske proizvode podrazumijeva testiranje ponašanja proizvoda prilikom visokog stupnja korištenja ili velikih količina protoka podataka (Rouse i suradnici, 2019).

Statistička kontrola procesa (eng. Statistical process control – kraće SPC) je metodologija bazirana na objektivnim podacima i analizi. Razvio ju je Walter Shewhart u Western Electric Company and Bell Telephone Laboratories između 1920. i 1930. Ova metoda koristi statističke metode kako bi upravljala i kontrolirala proizvodnju proizvoda (Rouse i suradnici, 2019).

Ukupno upravljanje kvalitetom (eng. Total quality management – kraće TQM) primjenjuje kvalitativne metode bazirane na kontinuiranom poboljšanju. TQM počiva na činjenicama, podacima i analizi za podršku planiranja proizvoda i pregled učinka (Rouse i suradnici, 2019).

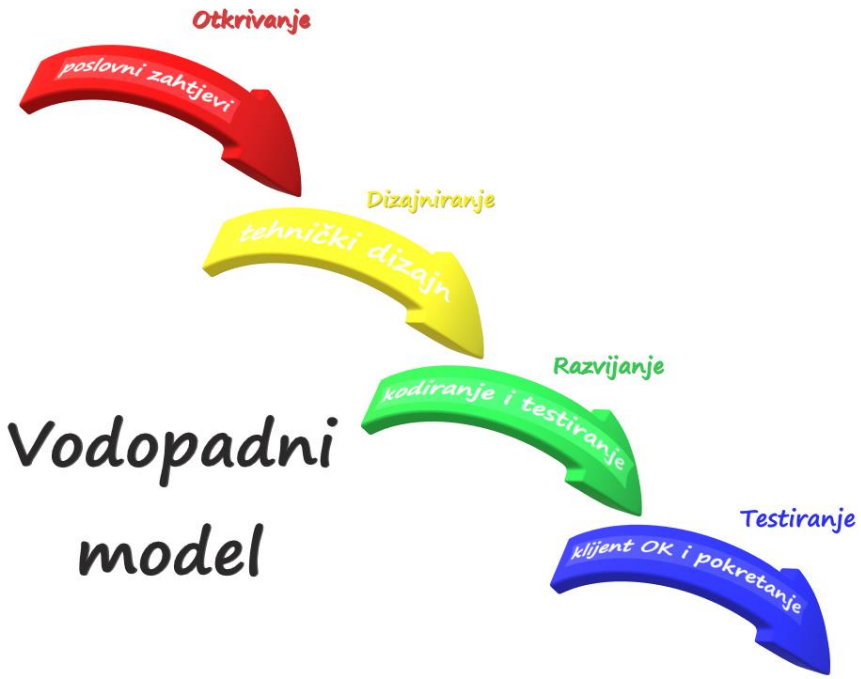
4.6. Osiguranje kvalitete softvera

SQA sistemski pronalazi uzorke i akcije potrebne za poboljšanje razvojnog ciklusa. Pronalaženje i ispravljanje grešaka u kôdu može donijeti nenamjerne posljedice, moguće je da se istovremeno ispravi jedna pogreška i time pokvari neka druga funkcionalnost. SQA postaje važan za developere kako bi izbjegli pogreške prije nego se dogode, ujedno smanjujući time vrijeme razvoja i troškove. No, unatoč SQA, ažuriranje softvera može „slomiti“ neke funkcionalnosti i prouzročiti probleme (Rouse i suradnici, 2019).

Postoje razne SQA strategije, poput integracije modela zrelosti sposobnosti (eng. Capability Maturity Model Integration – kraće CMMI). CMMI je SQA model usmjeren na poboljšanje performansi, radi na način da rangira razine zrelosti unutar područja organizacije i definira optimizaciju koja bi mogla donijeti poboljšanje. Metodologije razvoja softvera, koje

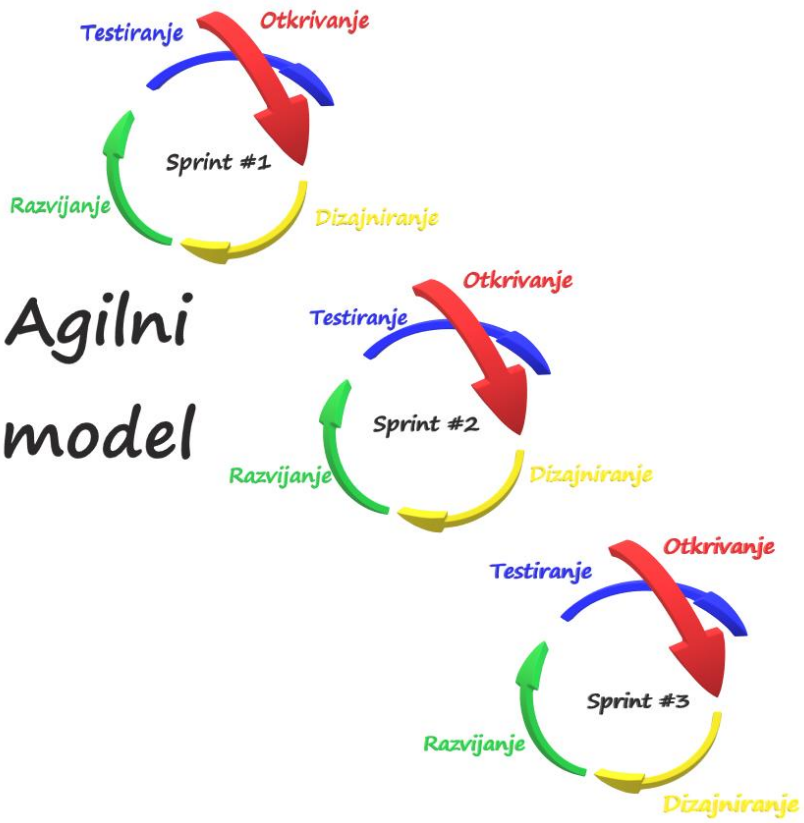
počivaju na SQA, napredovale su s vremenom. Tu su vodopadni model (eng Waterfall model), agilni model (eng. Agile model) i scrum model (Rouse i suradnici, 2019).

Vodopadni model je tradicionalni linijski pristup razvoja softvera. To je proces koji se odvija korak po korak a uključuje skupljanje zahtjeva, kreiranje dizajna, implementaciju koda, testiranje koda, ispravljanje pogrešaka i plasiranje proizvoda. Često se pokazao presporim, zbog čega su i kreirane alternativne metode (Rouse i suradnici, 2019). Vodopadni model prikazan je na slici 4.



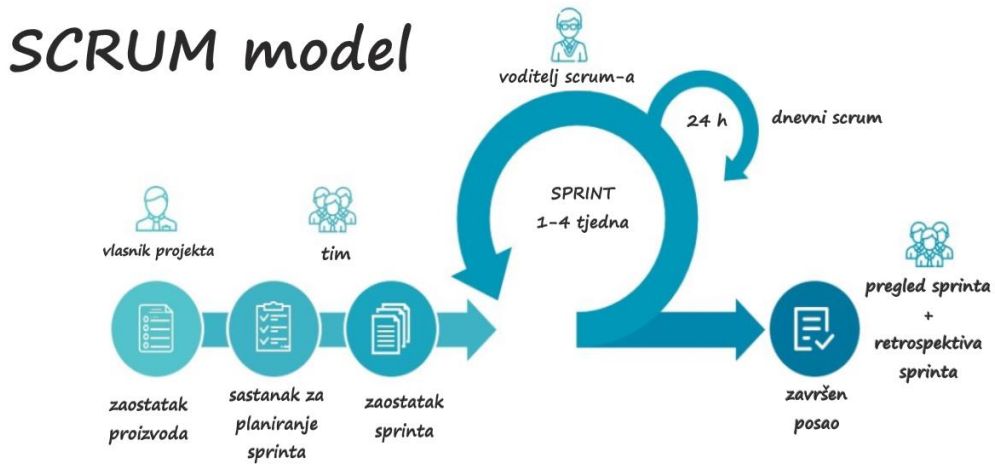
Slika 4: prikaz vodopadnog modela prema Software Testing Help (Prema: Software Testing Help, 2020)

Agilni model je timski orijentiran razvoj softvera gdje se svakom koraku procesa pristupa kroz sprintove. Vrlo je prilagodljiv, ali je manje predvidljiv jer se opseg projekta može lako promijeniti (Rouse i suradnici, 2019). Agilni model prikazan je na slici 5.



Slika 5: prikaz agilnog modela prema Software Testing Help (Prema: Software Testing Help, 2020)

Scrum model donosi kombinaciju vodopadnog i agilnog modela gdje su programeri podjeljeni u timove kako bi se bavili specifičnim zadacima, od kojih je svaki zadatak podijeljen u više sprintova (Rouse i suradnici, 2019). Scrum model prikazan je na slici 6.



Slika 6: prikaz scrum modela (Prema: *Allan Porras,* 2019)

Da bi se implementirao sustav QA-a, prvo je potrebno postaviti ciljeve za standarde. Potom razmotriti prednosti i mane pojedinog pristupa, kao što je maksimizacija učinkovitosti, reduciranje troškova i smanjenje pogrešaka. Menadžment mora biti voljan implementirati promjene u procesu i raditi zajedno kako bi podržao QA sustav i uspostavio standarde kvalitete. Dio karijera u SQA-u uključuje poslove poput SQA inženjera koji prate i testiraju softver kroz razvoj, SQA analitičara koji će kontrolirati implementaciju i praksu SQA kroz ciklus razvoja softvera; SQA automatizacije testova koji zahtjeva od pojedinca kreiranje programa za automatizaciju SQA procesa (Rouse i suradnici, 2019).

4.7. Kreiranje standardiziranog procesa osiguranje kvalitete i alati koji nam mogu pomoći u provedbi

Kako pomoću QA procesa provjeravamo stranicu ili aplikaciju i pokušavamo otkriti bilo kakvu manu koja se previdjela prilikom dizajniranja i razvoja, sam proces osiguranja kvalitete i testiranja postaju važan dio u procesu web razvoja (Berezhnoi, 2019). Da bismo kreirali standardizirani QA proces, moramo razumijeti različite faze QA testiranja. Prestianni (2020) ih dijeli na: testiranje funkcionalnosti (eng. functional testing), testiranje performansi (eng. performance testing), korisničko testiranje (eng. user testing), testiranje kompatibilnosti (eng. compatibility testing), testiranje sigurnosti (eng. security testing), testiranje optimizacije za tražilice (eng. Search Engine Optimization testing – kraće SEO testing) i pregled koda (eng. code review).

Tijekom **funkcionalnog testiranja** provjeravamo radi li stranica kako bi trebala raditi testirajući sve gumbiće, obrasce, menije, linkove i tako dalje. Osim toga, moramo biti sigurni da se svi podnosi obrasci usmjeravaju do imenovanih osoba i jesu li svi automatizirani odgovori ispravno konfigurirani (Prestianni, 2020).

Na **testiranje performansi** gledamo kao na jedan od najbitnijih koraka osiguranja kvalitete web stranica. Tokom ove faze testiramo: brzinu učitavanja stranice, temeljni vitalni skup za web (eng. Core Web Vitals[4]) i kako stranica reagira na pokušaje opterećenja. Na

[4] Eng. Core Web Vitals – obuhvaća set specifičnih faktora koje Google smatra važnim u ukupnom korisničkom iskustvu. Saastoji se od tri specifična mjerenja brzine stranice i interakcije korisnika: najveća sadržajna boja, prvo kašnjenje unosa i kumulativni pomak izgleda. Više o ovoj temi može se pročitati na sljedećem linku: https://backlinko.com/hub/seo/core-web-vitals

primjer, procjena učinka naglog povećanja broja posjetitelja web mjesta naziva se šiljastim testom (Prestianni, 2020).

Korisničko testiranje osigurava korisnicima da mogu intuitivno navigirati kroz web stranicu i pronaći informacije koje su im potrebne. Tokom testiranja, tražimo stvari kao što su hijerarhija sadržaja, funkcionalnosti pretraživanja sadržaja i konstrukciju menija (Prestianni, 2020).

Testiranjem kompatibilnosti procjenjujemo da li se naša stranica primjereno prikazuje na svim web poslužiteljima (Chrome, Safari, Firefox...) za desktop i mobilne uređaje, pritom osiguravajući da se tekst i slike poravnavaju i slažu ispravno prilikom različitih točaka prijeloma za različite veličine ekrana (Prestianni, 2020).

Faza **testiranja sigurnosti** omogućava nam da pronađemo i ispravimo ranjivosti stranice i zaštitimo stranicu od mogućih napada, pritom osiguravajući da naš odabrani sustav za upravljanje sadržajem (eng. content management system – kraće CMS) ima najnovija sigurnosna ažuriranja. Bilo da se radi o Drupalu, WordPressu ili nekom drugom sustavu. Kako se krećemo kroz ovu fazu, neka područja koja testiramo uključuju prijavljivanje u sustav, područja ograničenih ovlašenja korištenja, portale za plaćanje i integracije treće strane (Prestianni, 2020).

SEO testiranje testira recenzije da li je naša web stranica optimizirana za pronalaženje na tražilicama poput Googlea, Binga, Yahooa. Tokom ove faze tražimo tehničke i mrežne pogreške SEO-a na stranici, kao što su meta opisi, alt tagovi, naslovi stranica i SEO optimizirana URL struktura (Prestianni, 2020).

U fazi **pregleda koda** unutar QA procesa, proučavamo izvorni kôd stranice kako bi pronašli i ispravili bilo kakve pogreške koje su mogle promaknuti tokom inicijalne faze razvoja (Prestianni, 2020).

Da bi pomogli i pospješili provedbu QA procesa, pri tome ubrzali i što točnije proveli analize, postoje različi alati koje možemo koristiti u skeniranju i automatiziranim zadatcima. Prema Younes Rafieu (2017), jednom od mnogih autora koji redovito objavljuju na stranici SitePoint, središtu za web programere koji dijele svoju strast za izgradnju nevjerovatnih internet stvari, u tablici 4 donosimo nekoliko riječi o osam alata za osiguranje kvalitete PHP aplikacija. Post je objavljen još 2017 godine, ali i na novijim člancima drugih autora vidimo da su ti alati i dalje ostali među aktualnima za upotrebu.

Tablica 4: Popularni alati za provedbu osiguranja kvalitete proizvoda kreiranih u PHP-u

Naziv alata	Opis korištenja
PHPUnit	PHPUnit je okvir za testiranje PHP-a. Stvorio ga je Sebastian Bergman 2004. godine, a trenutno je u verziji 6 koja zahtijeva PHP 7. Dizajniran je na takav način da može brzo i jednostavno otkriti greške u kôdu. Kroz njegove specifične jedinice možete osigurati da je sve ispravno testirano, a testne slučajeve moguće je ponovno uportijebiti i zahtijevaju samo početno postavljanje [Acquaint Softtech, nema dat.].
Cucumber	Cucumber je okvir za stvaranje testova prihvatljivosti na osnovu specifikacija. Poznat je po opisno generiranim tekstovima koji se mogu čitati kao obični engleski. Službena PHP implementacija za Cucumber je Behat.
Atoum	Atoum je još jedan okvir za jedinično testiranje za PHP. To je samostalni paket koji se može instalirati putem GitHub-a, Composera ili putem izvršne datoteke PHAR. Atoumovi su testovi vrlo čitljivi s izražajnim imenima metoda i ulančavanjem.
Selenium	Selenium je alat za automatizirano testiranje preglednika (integracijsko testiranje i testiranje prihvaćanja). On transformira testove u naredbe API-ja preglednika i potvrđuje očekivane rezultate. Podržava većinu dostupnih preglednika. Selenium se može koristiti sa PHPUnit-om pomoću ekstenzije.
Dusk	Dusk je Laravelov alat namijenjen za automatizaciju preglednika. Može se koristiti samostalno (s Chrom-ovim driverom) ili sa Selenium-om. Ima jednostavan API i pokriva sve mogućnosti testiranja poput čekanja elemenata, prijenosa datoteka, upravljanja mišem i tako dalje.

Kahlan	Kahlan je cjeloviti testni okvir za jedinično testiranje i razvoj temeljen na ponašanju (eng. Behavior Driven Development – kraće BDD). Kahlan koristi opisnu sintaksu, prema kojoj je jako slčan Behatovim testovima.
PHP Testability	PHP Testability paket je alata za statičku analizu koji govori o problemima provjerljivosti u programu i generira detaljno izvješće. Paket trenutno nema označeno izdanje na koje se može osloniti, ali ga se može sigurno koristiti u razvoju. Instalacija je moguća putem programa Composer.
Continuius integration (CI) Services	Važan dio u isporuci kôda tijekom rada s timovima je mogućnost automatske provjere kôda prije nego što se spoji sa službenim repom projekta. Većina dostupnih usluga / alata za kontinuirano integriranje (eng. Continuius integration – kraće CI) servisa pruža mogućnost testiranja kôda na različitim platformama i konfiguracijama kako bi se osiguralo da je kôd siguran za spajanje. Postoji puno usluga koje nude dobre cjenovne razine, ali mogu se koristiti i alai otvorenog kôda: - PHPCI (besplatan) - TravisCI (besplatan za projekte otvorenog kôda) - SemaphoreCI (besplatan za projekte otvorenog kôda) - Jenkins

(Izvor: Younes Rafie, 2017)

Iz ove skupine alata za osiguranje kvalitete web aplikacija, koristit ćemo alat Dusk za automatizaciju preglednika, te prikazati na jednom dijelu aplikacije kako izgleda primjena navedenog alata. Dusk sam odabrala zato što je Laravelov alat i možemo ga instalirati u već postojeću integraciju Laravela u PHP Stormu. S obzirom na prvi doticaj sa QA, potrebni su mi alati koji imaju dobru podršku lako razumljivih primjera i pojašnjenja, a ovo je jedan od njih. PHPUnit se također nalazi u grupi alata za osiguranje kvalite aplikacije i njega sam već prethodno odabrala za kreiranje jediničnih i funkcionalnih testova.

4.8. Popis koraka koje je potrebno provesti u sklopu pojedinog testa

Da bismo kvalitetnije i jednostavnije proveli sve testove i da bismo bili sigurni da nismo ništa bitno ispustili i preskočili, potrebno je i preporuča se kreirati popis testova sa svim koracima koje ćemo provesti. Timothy Prestianni (2020) prikazuje primjer jedne takve liste. Elementi koje navodi prikazani su u tablici 5.

Većina ljudi razmišlja o provođenju osiguranja kvalitete web stranice prilikom lansiranja nove web stranice. Istina je da bi trebali kontinuirano pokretati svoju stranicu kroz testiranje. S vremenom će se na svakoj stranici, kako se dodaje sve više sadržaja, novih modula, te se mijenjaju algoritmi pretraživanja, razvijati i određeni problemi. Uvođenjem ritma stalnog testiranja omogućit ćemo brzo pronalaženje i ispravljanje svih problema jer se pojave mnogo prije nego što postanu problem za korisnike, klijente i tražilice (Prestianni, 2020).

Tablica 5: Lista testova i koraka za provođenje QA testiranja

Test	Korak
Testiranje funkcionalnosti	Funkcionira li web stranica kako je zamišljeno
	Provjeri sve padajuće stranice
	Provjeri gumbiće
	Provjeri sve menije
	Provjeri ima li prekinutih veza
	Provjeri da li postoje sintaksne pogreške
	Provjeri ima li pogreške boroj 404 (stranica nije pronađena, datoteka nije pronađena, poslužitelj nije pronađen)

	Ispravni izračuni transakcija
	Pretraživanje web mjesta radi ispravno
	Provjeri vanjske veze i PDF-ove otvorene na novim karticama
Testiranje performansi	Provjeri optimizaciju brzine web stranice
	Mobilna optimizacija, osigurajte da je aplikacija/stranica prilagođena za korištenje na mobilnim uređajima
	Kako se stranica ponaša prilikom: • Šiljci u prometu (testiranje otpornosti na stres) • Povećanje radnog opterećenja (ispitivanje opterećenja) • Uobičajeno opterećenje (ispitivanje stabilnosti) • Više korisničkih prijava istodobno (testiranje istodobnosti) • Povećanje volumena podataka baze podataka (ispitivanje volumena) • Kontinuirano povećanje opterećenja (ispitivanje izdržljivosti)
Testiranje korisnika	Testirati sve kontaktne obrasce
	Provjeri pravopisne i gramatičke pogreške
	Testiraj da li su tijekovi rada glatki i rade li ispravno
	Jednostavna navigacija i pronalaženje podataka

Testiranje kompatibilnosti	Testiraj web stranicu s različitim veličinama zaslona i razlučivosti zaslona na stolnim računalima, prijenosnim računalima, tabletima i pametnim telefonima
	Testirajte izgled i funkcionalnosti web stranice u više preglednika uključujući Internet Explorer, Firefox, Chrome i Safari
	Je li kreirana ikonica prečice (eng. favicon[5]) i prikazuje li se ispravno?
Testiranje sigurnosti	Provjeri koriste li sve stranice HTTPS protokol
	Provjeri jesu li korisnici preusmjereni na šifrirane SSL stranice
	Provjeri da se korisnici ne mogu prijaviti bez lozinke ili sa pogrešnom lozinkom
	Provjeri mogu li samo ovlašteni korisnici pristupiti određenim dijelovima web stranice.
	Potvrdi postojanje Captcha polja[6] na obrascima kako bi se ograničila neželjena pošta.
	Provjeri poštivanje zakonskih i regulatornih smjernica, npr. zaštita podataka i privatnosti.

[5] Favikon – poznat i kao ikonica prečice, ikonica web stranice, ikonica kratice ili ikonica oznake. To je datoteka povezana sa određenim web mjestom ili web stranicom koja sadrži jednu ili više ikonica koje su načešće veličine 16x16 piksela. (Izvor: Wikipedia, https://en.wikipedia.org/wiki/Favicon)

[6] Captcha - vrsta autentikacije "izazov-odgovor" koja se koristi u računarstvu da bi odredila je li korisnik čovjek ili računalo, s ciljem sprječavanja pristupa zlonamjernim računalnim programima (izvor: Wikipedia, https://hr.wikipedia.org/wiki/Captcha).

	Potvrdi da je onemogućeno pregledavanje direktorija.
	Simuliraj napad da provjeriš kako će se web stranica nositi s tim (ispitivanje penetracije)
SEO testiranje	URL-ovi prilagođeni SEO-u
	Provjeri nedostaju li naslovi stranica
	Provjeri nedostaju li oznake <h1>
	Provjeri nedostaju li oznake <h2> za podnaslove
	Provjeri jesu li opisi meta oznaka optimizirani
	Provjeri označavanje strukturiranih podataka
	Provjeri je li datoteka robots.txt dostupna
	Da li je Sitemap poslan u Googleovu konzolu za pretraživanje
	Provjeri imaju li sve slike alternativni tekst (eng. Alt text)
	Provjeri kanonske URL-ove, ako su potrebni.
	Provjeri „otvori označavanje podataka grafa“ (eng. Open Graph data markup)
	Provjeri da li je „301 – preusmjeravanje“ dovršeno
	Provjeri jesu li sve ikone društvenih mreža povezane sa ispravnim stranicama

	Instalirajte: Googlove analize, upravitelj oznaka i konzolu za praćenje (eng. Google Analytics, Tag Manager and Console for tracking)
Pregled kôda	Potvrdi da napisani kôd slijedi standarde
	Osiguraj da je sav kôd jednostavan za razumijevanje
	Ako je dokumentacija važan dio vaše inženjerske kulture, uključite popis u svoj pregled kôd

(Izvor: Timothi Prestianni, 2020)

Potrebno je napomenuti kako je popis prikazan u tablici 5 samo jedan primjer kako bi se olakšala i pospješila provedba osiguranja kvalitete našeg projekta. Svaki projekt će sukladno svojoj složenosti i funkcionalnostima imati više ili manje koraka koje je potrebno provesti.

4.9. Kako razvoj aplikacija vođen testiranjem utječe na osiguranje kvalitete?

James Shore (2014), voditelj bloga „Let's Code: Test-Driven JavaScript“ serije digitalnih zapisa usmjerenih na rigorozan i profesionalni web razvoj, u svom postu iz 2014. godine govori o spriječavanju pogrešaka tokom razvoja umjesto pronalaženja i ispravljanja pogrešaka kroz testove. Konačni rezultat je ukloniti potrebu za zasebnim QA korakom prije objavljivanja proizvoda. Ključni dio ovoga je razvoj vođen testiranjem.

Iako TDD nije savršena metoda razvoja, može pomoći programerima da otkriju kada su algoritam isprogramirali drugačije nego što su namjeravali. Također, dobar je za kreiranje robusnog niza regresijskih testova, ali postaje beskorisno kada programer pogrešno shvati što bi trebao raditi. TDD ne može uhvatiti netočne pretpostavke. Slijedeći model Briana Maricka, Shore (2014) smatra da su testeri tehnički istražitelji tima. Umjesto da budu štaka koja timu omogućava da napiše loš kôd, njihov je posao pomagati timu da shvati ono što oni ne znaju, ali trebaju znati.

Shore opisuje svojevrsni scenarij „savršenog svijeta“. Cilj je doći do točke kada QA ne treba testirati kôd prije nego što ga objavite. Da bi se to postiglo, tim mora stvoriti gotovo nula

nedostataka i to radeći dobar posao primjenom TDD-a na svim razinama razvoja, održavajući tehnički dug na niskoj razini i često refaktorirati, dobro komunicirati sa klijentom ili menadžmentom zaduženim za proizvod. Mnogi timovi nisu toliko dobri u agilnoj metodi, a ako to uključuje i vaš tim predlaže primjenu hibridnog pristupa. Potrebno je nastaviti testirati softver prije izdavanja ili na kraju svake iteracije dok radite timski kako biste poboljšali vještine sprečavanja kvarova. Kako se te vještine poboljšavaju, testeri će imati više vremena za ostale stvari. Pomoći će kupcima da definiraju što žele u smislu detaljnih poslovnih i nefunkcionalnih primjera. Također će tražiti sustavne mrtve točke, kako u pogledu nedostataka korisničkog iskustva, tako i temeljnih mogućnosti sustava. I na kraju, s vježbom ćete provoditi manje testiranja i dobiti više kvalitete (Shore, 2014).

4.10. Prednosti i nedostatci osiguranja kvalitete

Kvaliteta proizvoda i sevisa je ključni konkurentski diferencijator. Osiguranje kvalitete pomaže osigurati da organizacija kreira i isporuči proizvod bez pogrešaka i ujedno ispuni potrebe i očekivanja kupaca. Visokokvalitetni proizvodi naravno rezultiraju zadovoljnim klijentima, što može rezultirati lojalnošću kupaca, ponovljenom kupnjom, prodajom i zagovaranjem (Rouse i suradnici, 2019).

Osiguranje kvalitete može dovesti do smanjenja troškova koji proizlazi iz sprečavanja nedostataka na proizvodu. Ako je proizvod isporučen klijentu i otkrivena je pogreška, organizacija snosi troškove korisničke podrške, kao što je primanje izvještaja o pogrešci i riješavanja iste. To također zahtjeva troškove lociranja pogreške, troškove rada servisa i inženjera za ispravljanje pogreške, testiranje i validiranje ispravaka te troškove isporuke ažuriranog proizvoda nazad na tržište (Rouse i suradnici, 2019).

Osiguranje kvalitete zaista zahtjeva znatna ulaganja u ljude i procese. Ljudi moraju definirati tijek rada i nadzirati njgovu provedbu od strane QA tima. To može biti dugotrajan proces koji utječe na datum isporuke proizvoda. Uz nekoliko iznimki, nedostatak QA je više zahtjev za provedbom koraka koji se mora provesti kako bi se isporučio kvalitetan proizvod. Bez osiguranja kvalitete nastaju ozbiljniji nedostaci, poput grešaka u proizvodu i nezadovoljstva ili odbijanja proizvoda na tržištu (Rouse i suradnici, 2019).

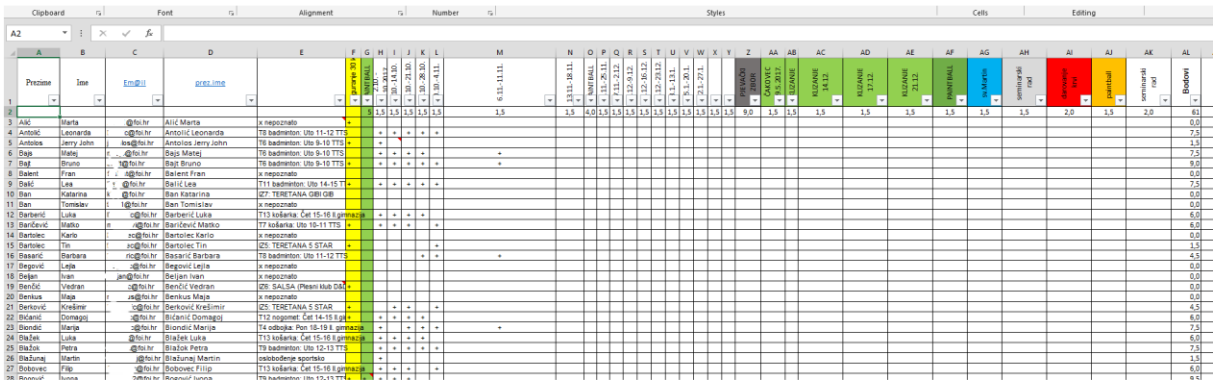
5. Projekt

U ovom poglavlju osvrnut ćemo se na praktični dio rada, sam projekt, pojasnit ćemo i prikazati izgled starog postupka evidentiranja studenata, popisati zahtjeve korisnika i sukladno tim zahtjevima osmisliti UML diagram klasa kako bismo jednostavnije prikazali funkcionalnosti koje je potrebno implementirati. U drugom dijelu ovog poglavlja pojasnit ćemo pobliže jednu funkcionalnost te prikazati cijeli proces razvoja i implementacije kroz TDD.

5.1. Opis projekta i obrada zahtjeva korisnika

Clj izrade projekta je pojednostavniti uvid u status pojedinog studenta i pregled obavljenih obveza, te olakšavanje vođenja evidencije studenata na satovima tjelesne i zdravstvene kulture na našem fakultetu

Dosadašnje evidencije vođene su putem excel tablice koja je vrlo nepregledna i velika. Iako je do neke mjere olakšano pretraživanje korištenjem raznih funkcija, stoji nepotrebno naknadno prepisivanje i evidentiranje dolazaka studenata koje se prethodno bilježi na papiru na pojedinim lokacijama i terminima. Na slici 7 vidimo prikaz trenutnog sustava evidentiranja studenata.



Slika 7: Prikaz tablične evidencije polazaka studenata u Microsoft Excel-u

Želimo omogućiti unos dolazaka na licu mjesta u bazu podataka, kako bi to bilo moguće, kreiramo web aplikaciju te će se moći kreirati korisnici koji će biti zaduženi za upis polaznika za svaku dodatnu aktivnost.

Nastavno na razgovor sa profesorom Vučićem i demonstratorima, te prema obrascima kako trenutno izgleda postojeće evidentiranje, slijedi tablica 6 sa popisom zahtjeva i funkcionalnosti koje je potrebno omogućiti u aplikaciji.

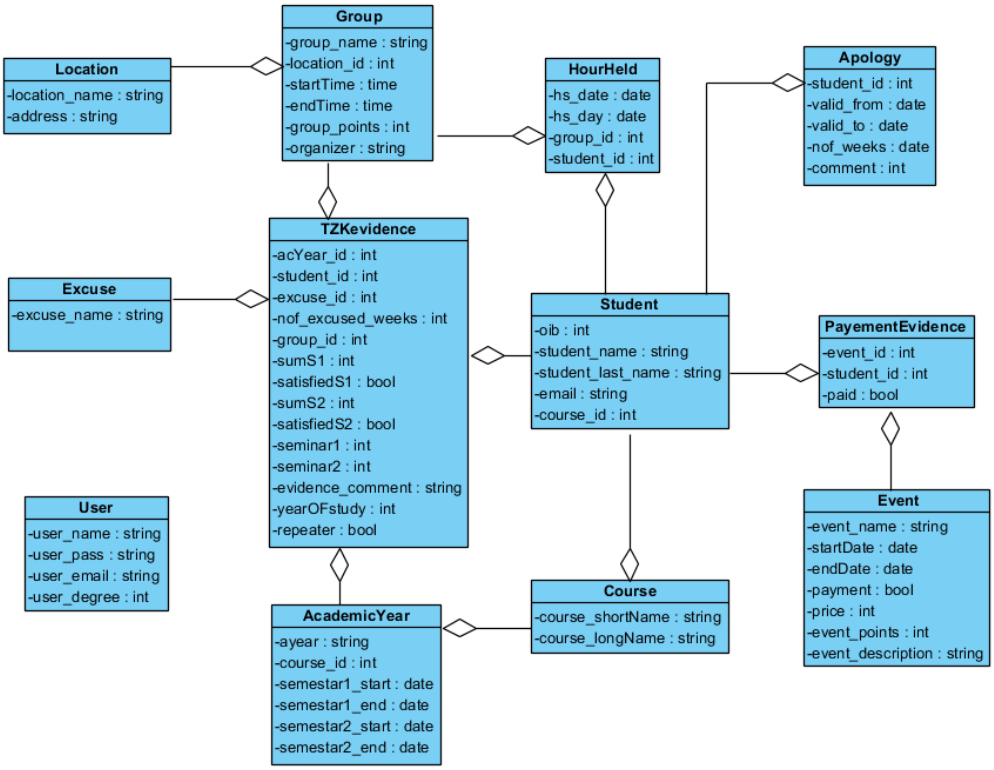
Tablica 6: Prikaz korisničkih zahtjeva i potrebnih funkcionalnosti

Tema	Zahtjevi	Funkcionalnosti temeljene na zahtjevima
Korisnici	Omogućiti pristup sustavu za profesore, demonstratore i instruktore dodatnih aktivnosti	• Prijava u aplikaciju • Kontrola prava pristupa
Studenti	Unos i pregled studenata	CRUD funkcije: • Kreiranje studenata • Pregled studenata • Ažuriranje studenata • Brisanje studenata
	Kreiranje i pregled evidencije studenata na kolegiju tjelesne i zdravstvene kulture prema akademskoj godini. Pregled statusa odrađenih sati u prvom i drugom semestru. Prikaz dodatnih bodova za sudjelovanje na raznim događajima. Bodovanje za prisustvovanje na više grupa u jednom danu: • ako je studentu zabilježeno prisustvovanje na dvije grupe čiji termini održavanja slijede jedan iza drugoga, prva grupa nosi puni broj bodova, a druga grupa nosi pola bodova. • ako je studentu zabilježeno prisustvovanje na dvije grupe između kojih postoji još jedan ili više termina tada se obje grupe boduju punim bodovima.	• CRUD funkcije za evidenciju studenata prema akademskoj godini • Sumiranje i prikaz svih dolazaka iz tablice odrađenih sati prema godini i semestru za koji se traži prikaz podataka • Sumiranje i prikaz dodatnih bodova iz tablice održanih događaja prema godini i semestru
	Upisom liječničkog opravdanja za osolobođenje od polaganja, automatski se treba pojaviti i obrazac za evidenciju ispričnice koja se tiče tog studenta.	• Automatska izrada novog zapisa ispričnice za studenta koji ima liječničko opravdanje za oslobođenje

Grupe	Unos, izmjena i pregled grupa: • Sportovi (nogomet, košarka, odbojka, badminton, stolni tenis, ...) • Dodatne aktivnosti u raznim klubovima (mačevanje, ples, ...)	CRUD funkcije: • Kreiranje grupa • Pregled grupa • Ažuriranje grupa • Brisanje grupa
	Omogućiti upis studenta u pojedinu sportsku grupu ili izvannastavnu aktivnost.	• Upis studena u grupu
Događaji	Unos, pregled i ažuriranje događaja: • Događaji održavani od strane fakulteta ili suradnika fakulteta (kao što su igre na dan fakulteta, „izleti“ u Zagreb na carting, paintball i slično)	CRUD funkcije: • Kreiranje događaja • Pregled događaja • Ažuriranje događaja • Brisanje događaja
	Evidencija plaćanja za one koji se prijave za pojedini događaj	Evidencija plaćanja • Kreiranje prijave • Pregled prijava • Ažuriranje uplata • Brisanje prijave
Raspored	Sastavljanje i pregled rasporeda	CRUD funkcije: • Kreiranje elemenata rasporeda • Pregled rasporeda • Ažuriranje rasporeda • Brisanje elemenata rasporeda
Ispričnice	Unos i pregled ispričnica za pojedinog studenta	CRUD funkcije • Kreiranje ispričnica • Pregled ispričnica • Ažuriranje ispričnica • Brisanje ispričnica
Održani sati	Omogućiti upis studenata koji su došli na pojedini sat	CRUD funkcije:

		• Kreiranje održanog sata • Pregled održanog sata • Ažuriranje održanog sata • Brisanje održanog sata
	Jedan student, u istu grupu može biti upisan samo jednom u danu.	• Kontrola upisa studenata u grupu održanih sati
Akademska godina	Omogućiti kreiranje nove akademske godine sa novim studentima, pri čemu je potrebno zadržati stanje protekle godine za postojeće studente i omogućiti kreiranje novih informacija za novu akademsku godinu. Svaki semestar je zasebno bodovan i nosi svoju ocijenu.	CRUD funkcije: • Kreiranje akademske godine • Pregled akademske godine • Aažuriranje akademske godine • Brisanje akademske godine

Nastavno na pregled korisničkih zahtjeva sastavili smo diagram klasa entitetnih klasa kako bismo olakšali daljnje planiranje i razvoj samih testova za konačni proizvod. Na slici 5 slijedi prikaz diagrama klasa, a potom i pojašnjenje korištenih klasa opisanih u tablici 7.



Slika 5: Diagram klasa

Tablica 7.: Pojašnjenje entiteta

Entitet	Opis
„TZKevidence“	„TZKevidence“ je glavna klasa, sadrži podatke o pojedinom studentu koji je upisao godinu tjelesnog u određenoj akademskoj godini (klasa „AcademicYear“). Također sadrži sve potrebne podatke i reference na pomoćne tablice koje su potrebne profesoru: da li student ima oslobođenje za obavljanje tjelesnog i koje je vrste to oslobođenje, broj opravdanih tjedana, grupa sporta ili dodatne aktivnosti kojoj pripada i koju godinu polazi, godina studija, da li je student ponavljač, broj bodova prikupljenih u prvom i drugom semestru na redovnim grupama i na posebno održanim i organiziranim događanjima od strane fakulteta. Tu također dolaze i dva atributa za seminar1 i seminar2 koji također nose dodatne bodove. Sukladno prikupljenom broju bodova u prvom/drugom semestru i broju ostvarenih bodova na dodatnim događanjima, te eventualno skupljenim bodovima na izradi seminara, kreira se oznaka da li je zadovoljio i položio pojedini semestar. Na posljetku slijedi i atribut za komentare, kao pomoć u bitnim bilješkama vezanim za pojedinog studenta.
„HourHeld“	„HourHeld“ je druga bitna klasa je klasa održanih sati i sudenata koji su prisustvovali pojedinom satu. Klasa sadrži sljedeće atribute: „hs_date“ – datum održanog sata, „hs_day“ – dan održanog sata, „group_id“ – vezu grupe za koju se održani sat upisuje, „student_id“ – veza na studenta koji je prisustvovao održanom satu.
„Apology“	„Apology“ je treća važna klasa za profesora, služi evidenciji ispričnica za pojedinog studenta i njihovo razdoblje važenja. Atributi koje ova klasa sadrži su: „student_id“, „valid_from“ – datum od kada vrijedi ispričnica, „valid_to“ – datum do kada ispričnica vrijedi, „nof_weeks“ – broj opravdanih tjedana koje ispričnica pokriva i „comment“.
„Group“	Klasa „Group“ sadrži popis sportskih grupa koje se održavaju od strane fakulteta i grupa održavanih u sportskim klubovima. Atributi

	„group_name“ – naziv grupe, „location_id“ – veza na lokaciju održavanja grupe, „startTime“ – vrijeme kada termin grupe počinje, „EndTime“ – vrijeme kada termin grupe završava, „group_points“ – broj bodova koji pojedina grupa nosi i „organizer“ – organizator odnosno nositelj grupe (to može biti fakultet ili vanjski suradnik).
„Location“	Klasa „Location“ sadrži lokacije za održavanje satova grupa. Sadrži dva atributa: „location_name“ – naziv institucije ili zgrade u kojoj se nalazi (II. Gimnazija, Plesni klub, itd) i „address“ – adresa održavanja.
„Excuse“	Klasa „Excuse“ sadrži vrstu opravdanja prema kojem je student oslobođen pohađanja satova tjelesne i zdravstvene kulture, bilo to na određeno razdoblje ili cijeli semestar.
„Student“	Klasa „Student“ sadrži osnovne podatke o studentu, a sastoji se od sljedećih atributa: „oib“ – osobni indentifikacijski broj, „student_name“ – ime studenta, „student_last_name“ – prezime studenta, „email“ – službeni FOI-ev e-mail, „course_id“ – smjer koji je student upisao.
„Course“	Klasa „Course“ sadrži puni naziv smjera i negovu kraticu, zapisane unutar atributa „course_longName“ i „course_shortName“.
„Event	Klasa „Event“ sadrži sve podatke o pojedinom događaju koji se organizira od strane fakulteta ili neke druge organizacije. Klasa se sastoji od sljedećih atributa: „event_name“ – naziv događaja, „startDate“ – datum kada događaj počinje, „endDate“ – datum kada događaj završava, „payement“ – da li se sudjelovanje na događaju plaća, „price“ – cijena sudjelovanja na događaju po studentu, „event_points“ – bodovi koje student dobiva ukoliko sudjeluje na događaju i „even_description“ – kratki opis događaja.
„PaymentEvidence“	Klasa „PayementEvidence“ sadrži evidenciju uplata studenata za održane dogđaje. Sastoji se od atrubuta: “event_id“ – veza na

	događaj, „student_id“ – veza na studenta koji će prisustvovati događaju, „paid“ – da li je primljena uplata za sudjelovanje.
„User“	Klasa „User“ sadrži podatke o registriranim korisnicima, sastoji se od sljedećih atributa: „user_name“ – korisničko ime, „user_pass“ – korisnička lozinka, user_email“ – e-mail korisnika, „user_degree“ – razina prava pristupa korisnika.
„AcademicYear“	Klasa „AcademicYear“ sadrži podatke o trajanju nastave pojedinog smjera u akademskoj godini, sastoji se od sljedećih atributa: „ayear“ – akademska godina, „course_id“ – smjer , „semestar1_start“ – datum početka prvog semestra, „semestar1_end“ – datum završetka prvog semestra, „semestar2_start“ – datum završetka drugog semestra, „semestar2_end“ – datum završetka drugog semestra.

5.2. Proces razvoja i implementacije funkcionalnosti kroz TDD pristup

Slijedeći praksu „autokarte“ kreirat ćemo sve potrebne testove u izradi spomenute web aplikacije. U narednom tekstu prikazat ćemo funkcionalni test za kreiranja studenta kroz sve korake razvoja. Ovaj postupak je skoro indentičan u testovima kreiranja događaja, evidencije plaćanja i svim ostalima koji se tiću provjere funkcionalnosti kreiranja. Kako u ovoj fazi razvoja nije bilo potrebe za spuštanjem na razinu jediničnog testiranja, prikazat ćemo i testiranje mogućnosti dodavanja evidencije plaćanja za pojedini događaj, gdje je bilo potrebno provjeriti ovu funkcionalnost i na dubljoj razini. Također nastavno na teorijski dio rada o osiguranju kvalitete, sastavili smo listu za provedbu osiguranja kvalitete, prikazanu tablicom 7.

Tablica 7: Lista koraka za provedbu QA nad funkcionalnosti prijave u aplikaciju

Test	Korak (napomena)
Funkcionalnost: prijava u aplikaciju	
Testiranje funkcionalnosti	Funkcionira li prikaz početne stranice aplikacije
	Da li linkovi na početnoj stranici vode na odgovarajuće stranice za prijavu i registraciju, a potom i stranice nakon prijave
	Provjeri postoje li sintaksne pogreške
	Provjeri ima li pogreške boroj 404 (stranica nije pronađena, datoteka nije pronađena, poslužitelj nije pronađen)
Testiranje sigurnosti	Provjeriti mogu li samo ovlašteni korisnici pristupiti aplikaciji i svim njenim stranicama
	Provjeri mogu li se korisnici prijaviti bez lozinke ili sa pogrešnom lozinkom
SEO testiranje	Provjeri nedostaju li naslovi stranica
	Provjeri nedostaju li oznake <h1>
	Provjeri nedostaju li oznake <h2>

Testiranje performansi, kompatibilnosti i testiranje korisnika su testovi koji će se provoditi završetkom većeg dijela funkcionalnosti projekta. Tada će se moći provjeriti jednostavnost pronalaženja informacija i navigiranja kroz sustav. Testiranje performansi, kompatibilnosti i testiranje korisnika su testovi koji će se provoditi završetkom većeg dijela funkcionalnosti projekta. Tada će se moći provjeriti jednostavnost pronalaženja informacija i navigiranja kroz sustav, ponoviti kontrola gramitčkih i pravopisnih pogrešaka, testirati optimizacija i brzina web stranice i drugo.

5.2.1. Fukncionalnost kreiranja studenata kroz TDD pristup

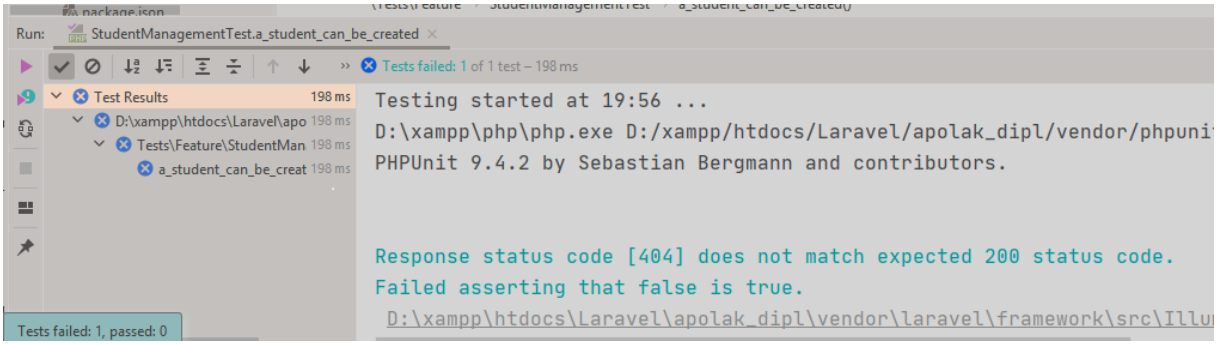
Test za kreiranje zapisa novog studenta u tablici `Student` provest ćemo kroz funkcionalni test. Kako nam i nalaže crveno-zeleni pristup kreirat ćemo dio po dio testne funkcije „`a_student_can_be_created`“ za kreiranje studenta.

Unutar `test/Feature` kreiramo `StudentManagementTest.php`. Ova klasa služi za testiranje svih osnovnih fukcija potrebnih za upravljanje tablicom `Student`. Prvi korak je kreirati osnovne dijelove testa. Šaljemo proizvoljne podatke putem POST metode u bazu i spremamo odgovor u varijablu `$response`, očekujemo da je slanje podataka provedeno uspješno i da nam pobrojavanje zapisa podataka u tablici `Student` vraća jedan redak zapisa.

```
class StudentManagementTest extends TestCase
{
    //CRUD testing
    /** @test */ //create
    public function a_student_can_be_created()
    {
        $response = $this->post('/students',[
            'oib' => '38120594636',
            'student_name' => 'Aleksandra',
            'student_last_name' => 'Polak Tomić',
            'email'=> 'apolak@foi.hr',
            'course_id' => '3',
        ]);
        $response -> assertOk();
        $this -> assertCount(1, Student::all());
    }
}
```

Isječak kôda 1: Kreiranje klase „StudentManagementTest“ i prvog testa „a_student_can_be_created“

Pokrenemo test i vidimo da test nije prošao. Prikaz rezultata provođenja testa prikazan je slikom 6. Umjesto očekivanog statusa 200 dobili smo 404.



Slika 6: prikaz rezultata provedbe neuspješno izvršenog testa (vraćen odgovor 404 umjesto očekivanog 200)

Kako je na slici 6 prikazan odgovor na test kojim je zapravo Laravel zapakirao stvarnu grešku. Prikaz pogreške 404 je dobar za odgovor koji je potrebno prikazati na web stranici, ali kada govorimo o testiranju tada moramo znati što se zapravo dogodilo. Da bismo omogućili prikaz stvarnih grešaka prilikom testiranja koristimo naredbu „*withoutExceptionHandling*“.

```
class StudentManagementTest extends TestCase
{
    //CRUD testing
    /** @test */ //create
    public function a_student_can_be_created()
    {
        $this->withoutExceptionHandling();
        $response = $this->post('/students',[
            'oib' => '38120594636',
            'student_name' => 'Aleksandra',
            'student_last_name' => 'Polak Tomić',
            'email'=> 'apolak@foi.hr',
            'course_id' => '3',
        ]);
        $response -> assertOk();
        $this -> assertCount(1, Student::all());
    }
```

Isječak kôda 2: test „a_student_can_be_created“ (dodavanje naredbe „withoutExceptionHandling“)

Kada smo unijeli izmjene pokrećemo ponovno test, te sada vidimo stvarni problem neuspješnog testa, nije moguće pronaći POST rutu za http://localhsot/student.

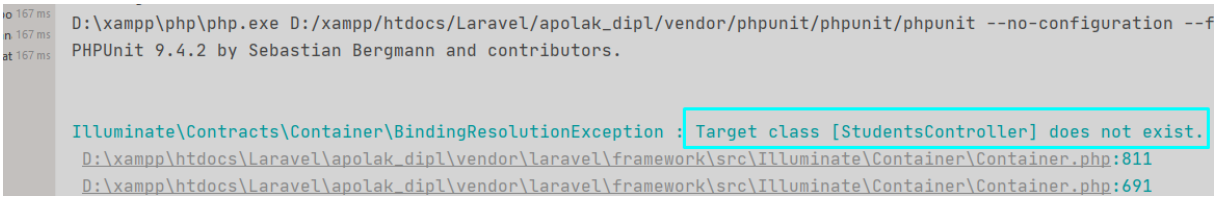
Slika 7: Prikaz rezultata provedbe neuspješno izvršenog testa (ne postoji POST ruta)

Iz rezultata provedbe testa vidimo da moramo dodati sljedeću rutu u routes/web.php klasu:

```
Route::post('/students', 'StudentsController@store');
```

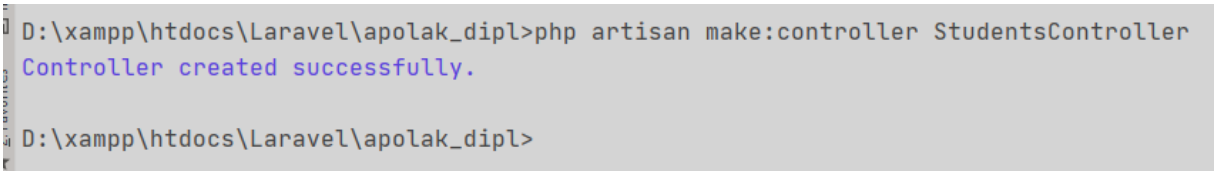
Isječak kôda 3: Dodavanje post rute u klasu „`web.php`“

Ponovno pokrećemo test i dobivamo grešku o nepostojanju klase StudentsController.



Slika 8: Prikaz rezultata provedbe neuspješno izvršenog testa (ne postoji klasa `StudentController`)

Ova pogreška je očekivana jer još nismo kreirali `StudentsController` klasu. Unutar terminala kreiramo kontroler na način prikazan na slici 9 i ponovno pokrećemo test te još jednom dobivamo istu pogrešku kao i u prethodnom rezultatu testiranja vidljivog na slici 8. Ne postoji klasa `StudentsControler`.



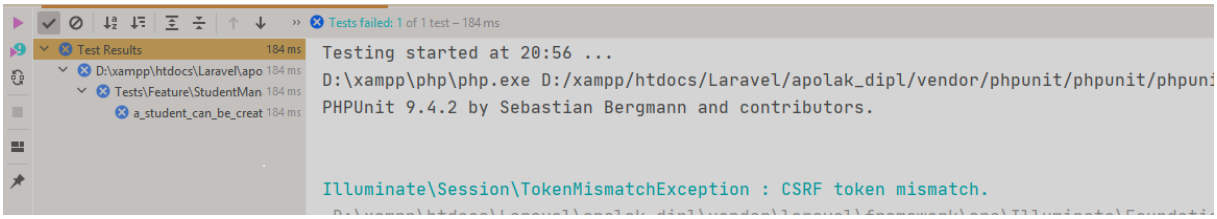
Slika 9: Kreiranje kontrolera `StudentsController` putem terminala

Nakon istraživanja otkrivamo da je novim ažuriranjem aplikacije PHPStorm promjenjeno korištenje rute za kontrolere, te je sada potrebno dodati punu rutu:

`action:'App\Http\Controllers\StudentsController@store'` ili napraviti izmjene u App/Providers/RouteServiceProvider.php i omogućiti upotrebu sljedeće linije:

```
// protected $namespace = 'App\\Http\\Controllers';
```

Upisana je cijela ruta do kontrolera i ponovnim pokretanjem testa slijedi nova greška, također nepoznata u prijašnjoj verzij PHPStorm-a: neusklađenost CSRF tokena, prikazana slikom 10.



Slika 10: Prikaz rezultata provedbe neuspješno izvršenog testa („CSRF token mismatch“)

Jedan od prijedloga rješavanja ovog problema bio je i provedba čišćenja predmemorije, što nije rezultiralo uspjehom u našem slučaju. Koji je razlog tome? Prema pojašnjenju u Laravelovoj dokumentaciji, upotrebom CSRF tokena omogućena je jednostavnija zaštita od napada krivotvorenja zahtjeva za više mjesta (eng. cross-site request forgery – kraće CSRF). Za svaku sesiju aktivnog korisnika Laravel automatski generira CSRF token, koji je korišten kako bi se potvrdilo da je autorizirani korisnik taj koji šalje zahtjev aplikaciji. Time nalaže da je prilikom svakog definiranja HTML forme u našoj aplikaciji potrebno uključiti sakriveni CSRF token kako bi se mogla provoditi validalidacija zahtjeva. Može se koristiit @csrf Blade direktiva za generiranje polja tokena:

```
<form method="POST" action="/profile">
    @csrf
    ...
</form>
```

Isječak kôda 4: primjer korištenja @csrf Blade direktive

Drugi prijedlog je onemogućavanje korištenja klase Middleware dok kreiramo testove i još ne postoje forme u koje bismo mogli uključiti CSRF tokene.

```
/** @test */ //create
public function a_student_can_be_created()
{
    $this->withoutMiddleware();
    (...)
}
```

Isječak kôda 5: test „`a_student_can_be_created`“ (dodavanje naredbe „`withoutMiddleware`“)

Zanemarivanjem korištenja Middleware klase dolazimo do očekivane pogreške. Nedostaje store metoda u `StudentsController` klasi (slika 11).

Slika 11: Prikaz rezultata provedbe neuspješno izvršenog testa (nedostaje metoda „`store`“)

Kreiramo store metodu unutar klase `StudentsController`:

```
    public function store()
    {


    }
```

Isječak kôda 6: Klasa `StudentController` (dodavanje metode „`store`“)

Pokrenemo test i vidimo rezultat prikazan na slici 12, klasa `Student` nije pronađena, logično, još je nismo kreirali.

```
Error : Class 'Tests\Feature\Student' not found
 D:\xampp\htdocs\Laravel\apolak_dipl\tests\Feature\StudentManagementTest.php:26
```

Slika 12: Prikaz rezultata provedbe neuspješno izvršenog testa (ne postoji klasa `Student`)

Pomoću terminala kreiramo model za Studenta (prikazano slikom 13), te istovremeno radimo migraciju kao pomoć u provedbi testiranja.

```
D:\xampp\htdocs\Laravel\apolak_dipl>php artisan make:model Student -m
Model created successfully.
Created Migration: 2020_11_12_205833_create_students_table

D:\xampp\htdocs\Laravel\apolak_dipl>
```

Slika 13: Kreiranje modela Student pomoću terminala

No, ponovnom provedbom testa vidimo da klasa i dalje nije pronađena i dobivamo rezultat provedbe testa kao i na slici 12, jer iako smo je kreirali još je nismo uključili u klasu. To ćemo ispraviti dodavanjem naredbe „`use App\Models\Student`“ u klasu `StudentManagementTest`.

```
namespace Tests\Feature;
use App\Models\Student;
use Tests\TestCase;
class StudentManagementTest extends TestCase
{
    (…)
}
```

Isječak kôda 7: Klasa `StudentManagementTest` (uključivanje klase `Student`)

Nakon što smo pokrenuli test „`a_student_can_be_created`“ dolazimo do nove pogreške prikazane na slici 14 , koja nam govori da ne postoji tablica student u našoj bazi. Taj problem rješavamo uključivanjem funkcije „`RefreshDatabase`“. Zašto to radimo? Netom prije pokretanja svakog testa potrebno je migrirati bazu podataka te nakon što se test završi srušiti stanje i vratiti prijašnje stanje baze. Na taj način uvijek imamo čistu bazu za testiranje.

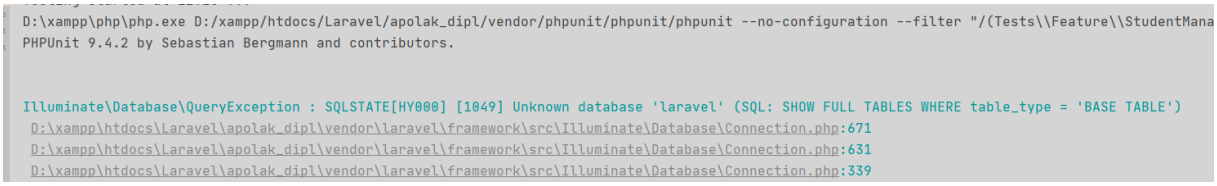
Slika 14: Prikaz rezultata provedbe neuspješno izvršenog testa (ne postoji veza na tablicu Student)

Uključujemo „`RefreshDatabase`“ u naš kôd i ponovno pokrećemo test:

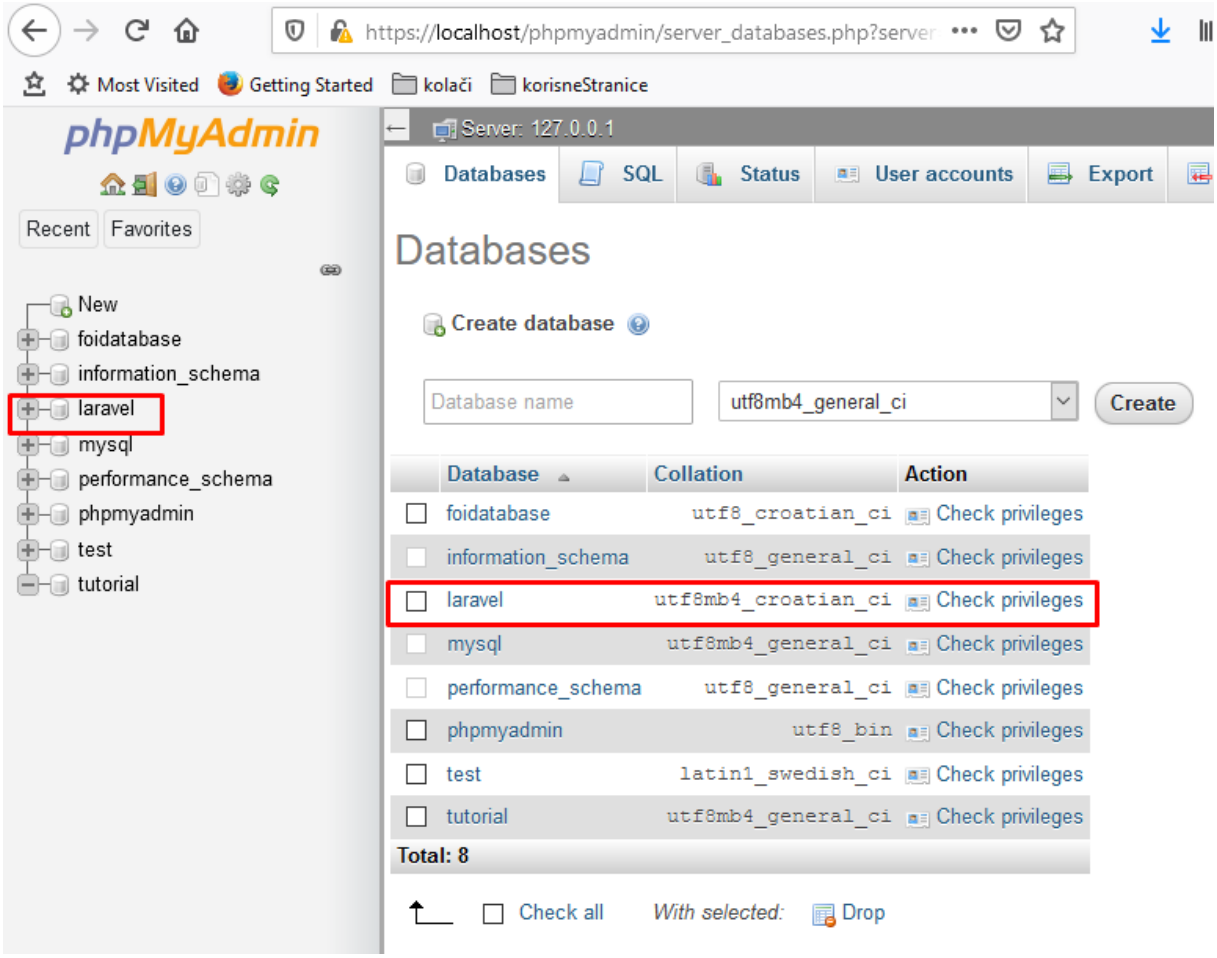
```
class StudentManagementTest extends TestCase
{
    use RefreshDatabase; (…) }
```

Isječak kôda 8: Klasa `StudentManagementTest` (uključivanje `RefreshDatabase`)

Rezultat provedbe testiranja prikazan je slikom 15. Ne postoji baza podataka naziva „laravel“. Otvaramo phpMyAdmin u pregledniku i kreiramo novu bazu podataka (slika 16), te odmah pomoću terminala radimo migraciju podataka u bazu, prikazano slikom 17.



Slika 15: Prikaz rezultata provedbe neuspješno izvršenog testa (ne postoji baza podataka laravel)

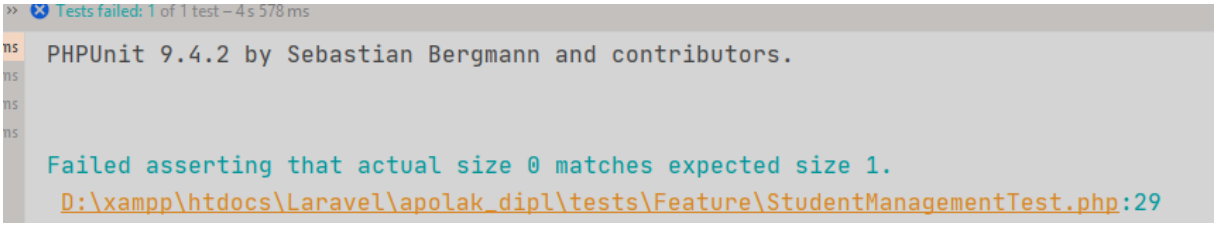


Slika 16: Kreirana nova baza podataka „laravel“

```
D:\xampp\htdocs\Laravel\apolak_dipl>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (827.49ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (409.98ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (491.27ms)
Migrating: 2020_11_12_205833_create_students_table
Migrated:  2020_11_12_205833_create_students_table (279.20ms)
```

Slika 17: Migriranje podataka u bazu podataka putem terminala

Pokrećemo test i nova pogreška koja se javlja prikazana je slikom 18. Vidimo da upisivanje u tablicu putem POST metode nije uspjelo i tablica je prazna, a mi u svom testu očekujemo rezultat sa jednim zapisom.



Slika 18: Prikaz rezultata provedbe neuspješno izvršenog testa (očekujemo 1 red dohvaćen iz tablice `Student`, a dohvaćeno je 0)

Da bismo ispravili pogrešku dopunimo funkciju store. Dohvaćamo tablicu `Student` i kreiramo zapis tako da svaki atribut punimo zahtjevanim oib-om, imenom i prezimenom studenta i tako dalje.

```
class StudentsController extends Controller
{
    public function store()
    {
        Student::create([
            'oib' => request('oib'),
            'student_name' => request('student_name'),
            'student_last_name' => request('student_last_name'),
            'email'=> request('email'),
            'course_id' => request('course_id'),
        ]);
    }
}
```

Isječak kôda 9: Klasa `StudentController` (dopunjavanje metode „`store`“)

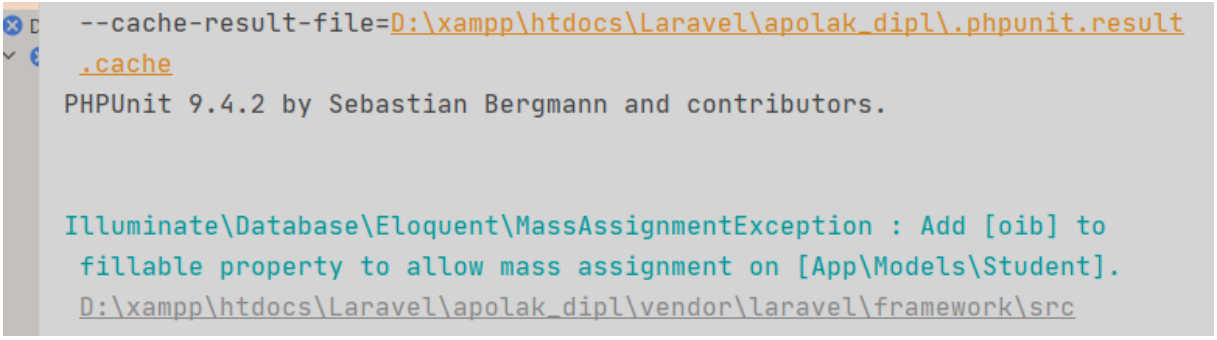
Pokrećemo test i vidimo da ponovno ne vidimo klasu `Student`. Dodajemo je na isti način kao i u klasi `StudentManagementTest`:

```
use App\Models\Student;
use Illuminate\Http\Request;
class StudentsController extends Controller
{
      (…)
}
```

Isječak kôda 10: Klasa `StudentController` (uključivanje klase `Student`)

Pokrenemo test i dobivamo novu pogrešku (slika 19). Potrebno je dodati „`oib`“ u svojstva koja mogu biti masovno ispunjena (eng. fillable property) u modelu `Student`.

Razlikujemo *fillable* i *guarded* svojstva. *Fillable* omogućava specificiranje polja koja mogu biti masovno dodijeljena u našem modelu. S druge strane, upravo suprotno, svojstvo *guarded* omogućava označavanje polja koja ne mogu biti masovno dodijeljena našem modelu. Svojstvo *fillable* je dobro iskoristiti u slučaju kada imamo 2-10 polja, ali kada je riječ o 20-50 polja među kojima ih mora biti nekolicina zaštićena, bolje je upotrijebiti svojstvo *guarded*. U svakom slučaju koristi se ili jedno ili drugo svojstvo, nikako oba.



Slika 19: prikaz rezultata provedbe neuspješno izvršenog testa (potrebno dodati „oib“ u fillable svojstva)

Uvođenjem svojstva guarded na sljedeći način, prelazimo na novu pogrešku (slika 20):

```
class Student extends Model
{
   protected $guarded;
}
```

Isječak kôda 11: Klasa StudentController (uključivanje klase Student)

Kako još nismo popunili migraciju za model Student, javljat će nam se redom greške za svaku kolonu koju još nismo dodali – oib, student_name, student_last_name, email, course_id (slike od 20 do 24). Dodavat ćemo ih postepeno u slučaju da se pojavi još koja pogreška u međukoracima, te na posljetku dobivamo sljedeći kod u CreateStudentsTable klasi:

```
class CreateStudentsTable extends Migration
{
    /**
     * Run the migration
     * @return void
     */
    public function up()
    {
        Schema::create('students', function (Blueprint $table) {
            $table->id();
            $table -> bigInteger('oib');
            $table -> string('student_name');
            $table -> string('student_last_name');
            $table -> string('email');
            $table -> integer('course_id');
            $table->timestamps();
        });
    }
```

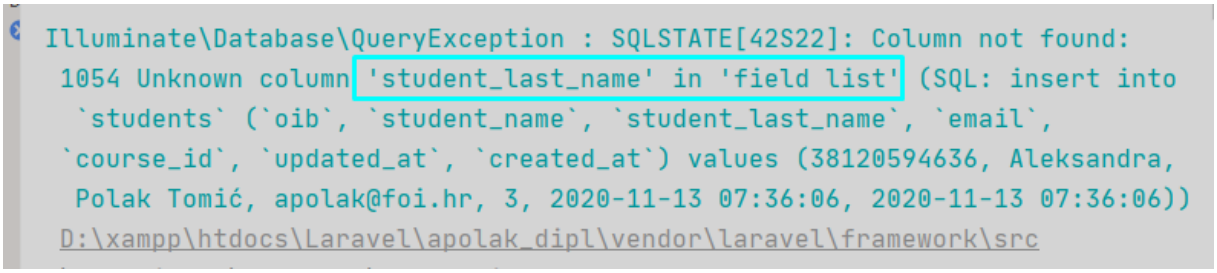
Isječak kôda 12: Klasa `CreateStudentsTable` (dodavanje atributa potrebnih klasi `Student`)

```
Illuminate\Database\QueryException : SQLSTATE[42S22]: Column not found:
 1054 Unknown column 'oib' in 'field list' (SQL: insert into `students`
 (`oib`, `student_name`, `student_last_name`, `email`, `course_id`,
 `updated_at`, `created_at`) values (38120594636, Aleksandra, Polak Tomić,
  apolak@foi.hr, 3, 2020-11-13 07:26:14, 2020-11-13 07:26:14))
 D:\xampp\htdocs\Laravel\apolak_dipl\vendor\laravel\framework\src
```

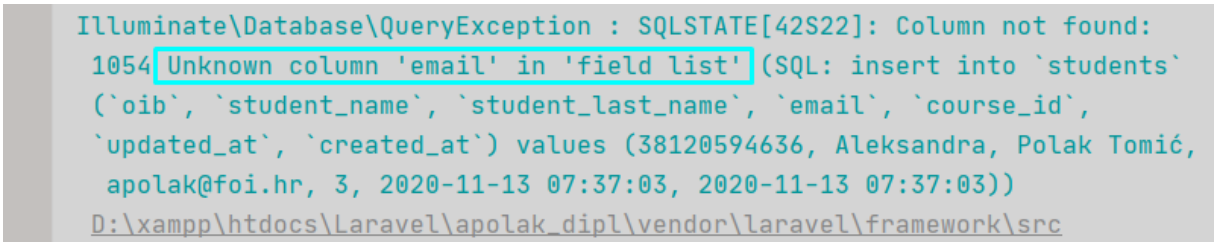
Slika 20: prikaz rezultata provedbe neuspješno izvršenog testa (potrebno dodati „oib“ u migraciju tablice `Student`)

```
Illuminate\Database\QueryException : SQLSTATE[42S22]: Column not found:
 1054 Unknown column 'student_name' in 'field list' (SQL: insert into
 `students` (`oib`, `student_name`, `student_last_name`, `email`,
 `course_id`, `updated_at`, `created_at`) values (38120594636, Aleksandra,
  Polak Tomić, apolak@foi.hr, 3, 2020-11-13 07:32:21, 2020-11-13 07:32:21))
```

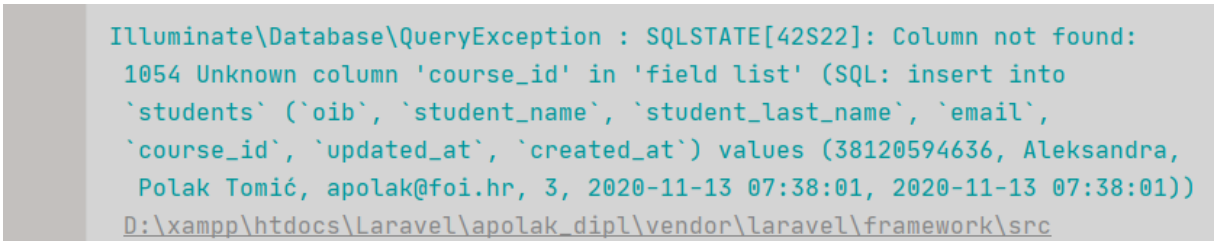
Slika 21: prikaz rezultata provedbe neuspješno izvršenog testa (potrebno dodati „`student_name`“ u migraciju tablice `Student`)



Slika 22: prikaz rezultata provedbe neuspješno izvršenog testa (potrebno dodati „`student_last_name`“ u migraciju tablice `Student`)

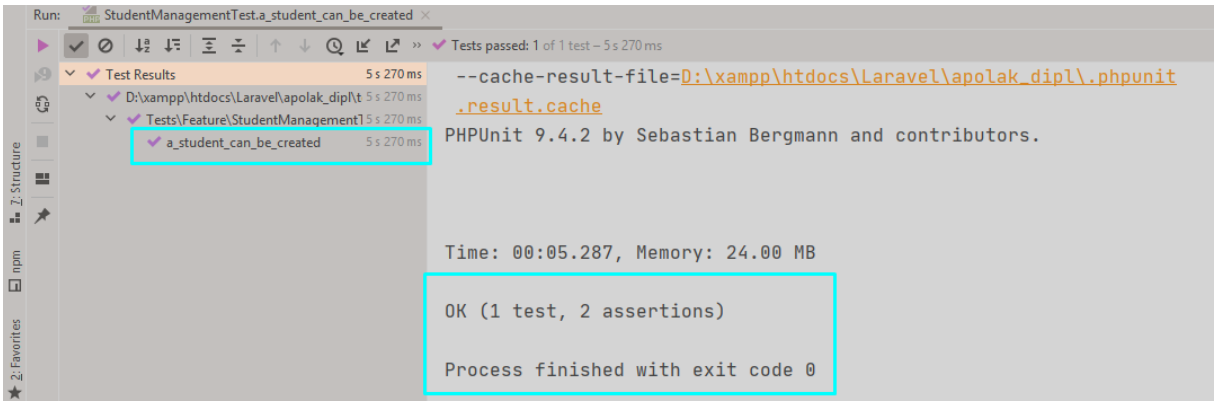


Slika 23: prikaz rezultata provedbe neuspješno izvršenog testa (potrebno dodati „`email`“ u migraciju tablice `Student`)



Slika 24: prikaz rezultata provedbe neuspješno izvršenog testa (potrebno dodati „`course_id`“ u migraciju tablice `Student`)

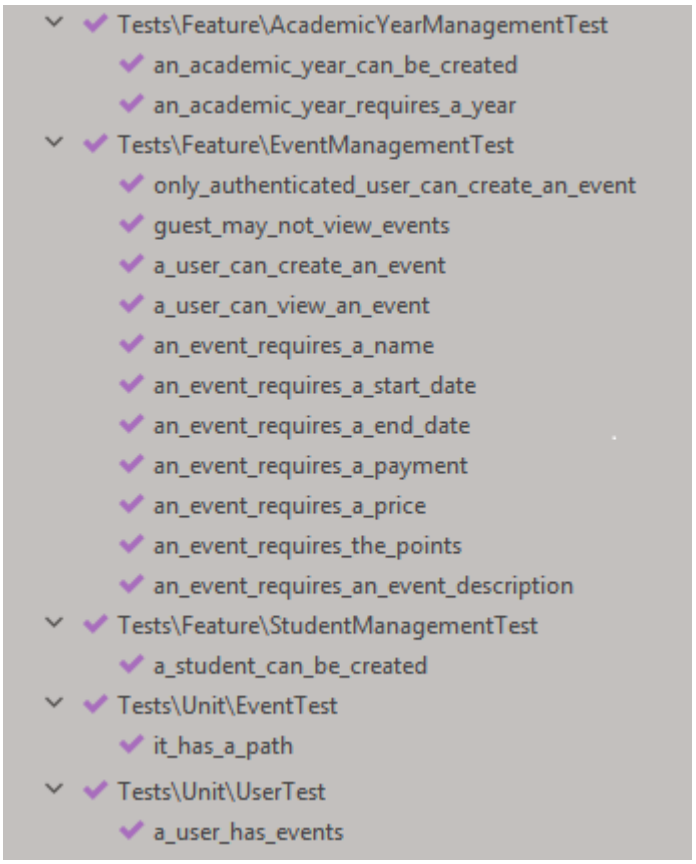
Uključivanjem i zadnje kolone u migraciju podataka tablice Student. Dolazimo do posljednjeg pokretanja testa koji nam vraća uspješan rezultat.



Slika 25: uspješno proveden test „`a_student_can_be_created`“

5.2.2. Funkcionalnost kreiranja evidencije plaćanaj za pojedini događaj

Prije nego što nastavimo sa sljedećim testom, nakon što završimo zadnji korak i dovedemo trenutni test u zelenu fazu, provedemo testiranje za sve do sada kreirane testove.



Slika 26: Provjera prolaze li svi prethodni testovi nakon kreiranja novog testa

Vidimo da se svi testovi izvršavaju uspješno i možemo provesti brisanje komentara i suvišnog kôda. Nakon uređivanja kôda vidimo kako u većini testova prvo osiguravamo da nešto želimo napraviti kao prijavljeni korisnik. Da bismo skratili pisanje i da bi sam kôd bio jasniji za čitanje izdvojit ćemo

„`$this->actingAs(User::factory()->create());`“ u zasebnu metodu `singIn` u klasi `TestCase` na sljedeći način:

```php
<?php

namespace Tests;

use App\Models\User;

use Illuminate\Foundation\Testing\TestCase as BaseTestCase;


abstract class TestCase extends BaseTestCase

{

    use CreatesApplication;

    protected function signIn($user = null)

    {

        return $this->actingAs($user ?: User::factory()->create());

    }

}
```

Isječak kôda 13: Izdvajanje zasebne metode za ponašanje aktivnog korisnika

Pozviom metode `signIn`() bez prosljeđivanja varijable, kreirat ćemo novog korisnika, a ako pozovemo metodu i prosljedimo joj varijablu, tada će ona dohvatiti postojećeg korisnika sa pripadajućim id-om. Sada možemo uvesti korekcije u kôd i provjeriti radi li sve i dalje kako treba.

```
(...)

        /** @test */

          public function a_user_can_view_an_event(){

              $this->signIn();

              $event = Event::factory()->create();

                  $this->get($event->path())

                       ->assertSee($event->event_name)

                       ->assertSee($event->startDate)

                       ->assertSee($event->endDate)

                       ->assertSee($event->payment)

                       ->assertSee($event->price)

                       ->assertSee($event->event_points)
```

```
                    ->assertSee($event->event_description);
        }

    /** @test */
     public function an_event_requires_a_name(){
          $this->signIn();
          $attributes = Event::factory()->raw(['event_name' => '',]);
          $this -> post('/events',$attributes)
               -> assertSessionMissing('event_name');
     }

     /** @test */
     public function an_event_requires_a_start_date(){
          $this->signIn();
          $attributes = Event::factory()->raw(['startDate' => '',]);
          $this->post('/events',$attributes)
               ->assertSessionMissing('startDate');
     }

     /** @test */
     public function an_event_requires_a_end_date(){
         $this->signIn();
          $attributes = Event::factory()->raw(['endDate' => '',]);
          $this->post('/events',$attributes)
              ->assertSessionMissing('endDate');
     }
(...)
```

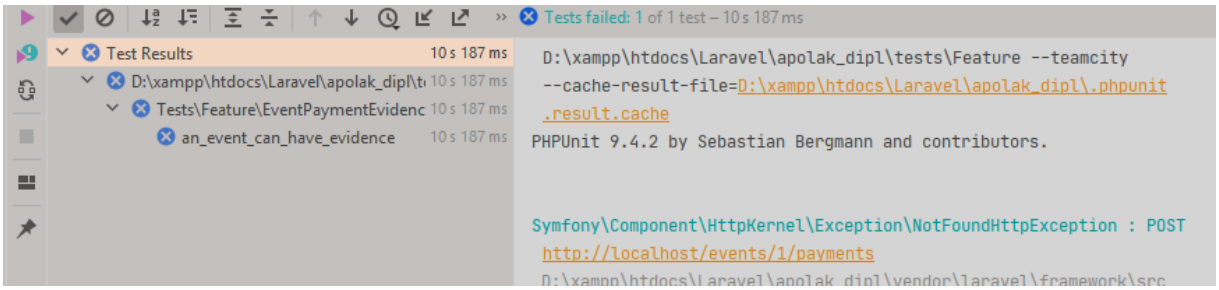
Isječak kôda 14: Refaktoriranje kôda (zamjena linije metodom)

Ponovnom provjerom vidimo da su svi testovi izvršeni uspješno, te možemo prijeći na sljedeći test.

```
public function an_event_can_have_payment_evidence()
    {
        $this-> withoutExceptionHandling();
        $this->signIn();

        $event = Event::factory()->create(['owner_id' => auth()->id()]);
        $this->post($event->path().'/payments', ['student_id' => 1]);

        $this->get($event->path())
            ->assertSee('1');
    }
```

Isječak kôda 15: Klasa `EventPaymentEvidenceTest` (kreiranje testa „`an_event_can_have_payment_evidence`“

Pokretanjem testa dobivamo poruku o nepostojanju zadane rute „`POST`“ (slika 27.) nakon čega dodajemo rutu u `web.php` klasu (isječak koda 16).



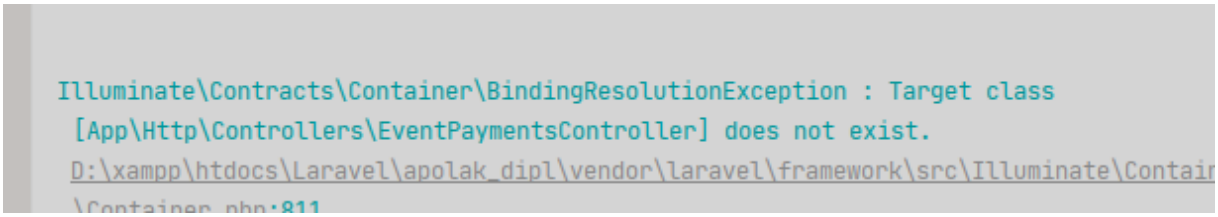
Slika 27. Nedostaje `POST` ruta za http:://localhost/events/1/payments

```
Route::post('/events/{event}/payments',
        'App\Http\Controllers\EventPaymentsController@store');
```

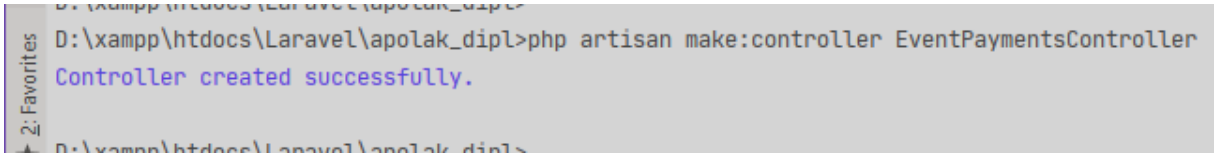
Isječak koda 16: Dodavanje `post rute` u `web.php`

Kao i u prethodnom testiranju kreiranja studenata slijedi da ne postoji klasa `EventPaymentController` (slika 28) koju potom kreiramo pomoću terminala i naredbe `php`

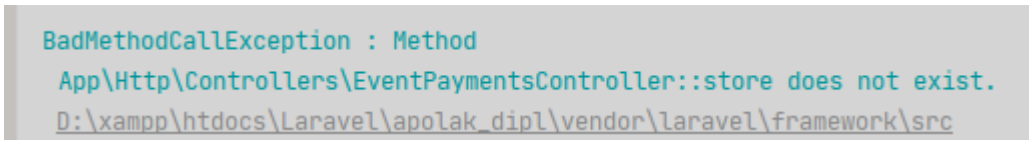
`artisan make:controller EventPaymentController` (slika 29). Nakon toga slijedi greška o nepostojanju `store` metode u kreiranom kontroleru (slika 30).



Slika 28: Ne postoji klasa `EventPaymentController`



Slika 29: Kreiranje klase `EventPaymentController` pomoću terminala

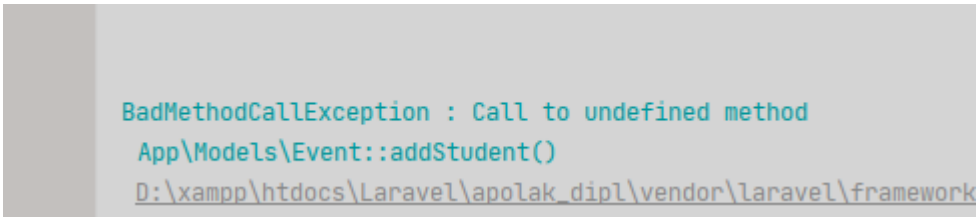


Slika 30: Ne postoji metoda `store` u klasi `EventPaymentController`

Dodavanjem metode `store` u klasu `EventPaymentsControler` ponovno se vraćamo na pogrešku o nepostojanju.metode `addStudent()`.

```
public function store(Event $event){

}
```

Isječak koda 16: Dodavanje metode `store()` u klasu `EventPaymentController`



Slika 31: Ne postoji metoda `addStudent()`

Prije nego nastavimo, vidimo da bi trebali provjeriti može li događaj dodati studenta u evidenciju plaćanaja, skukladno tome kreirajmo jedinični test "`it_can_add_a_payment_evidence`". Test je prilazan isječkom koda 17. Također dobivamo grešku jednaku kao na slici 31 ne postoji `addStudent()` metoda..

```
    /** @test  */
    public function it_can_add_a_payment_evidence()
    {
        $this->withoutExceptionHandling();
        $event = Event::factory()->create();
        $payment = $event->addStudent('1');
        $this->assertCount(1,$event->payments);
        $this->assertTrue($event->payments->contains($payment));
        $this->assertEquals(0, PaymentEvidence::first()->paid);
        $this->assertEquals(
                    $event->id, PaymentEvidence::first()->event_id);
    }
```

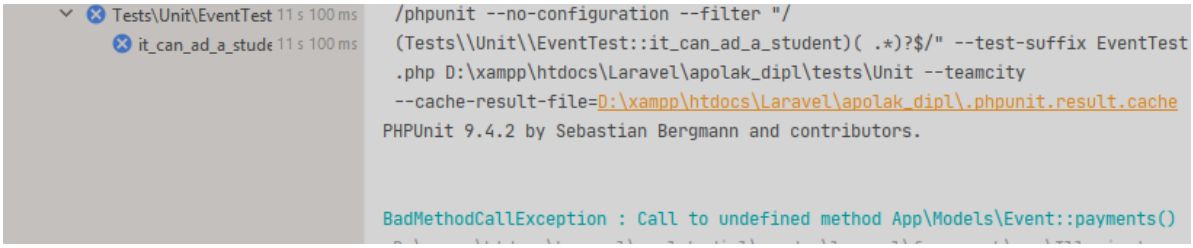
Isječak kôda 17 : jedninični test „it_can_add_a_payment_evidence“ u klasi EventTest

Metoda `addStudent()` implementirana je u klasu `Event`, jer na postojeći događaj dodajemo odabranog studenta.

```
     public function addStudent($student_id){
           return $this->payments()
                       ->create(compact('student_id'));
        }
```

Isječak kôda 18: Klasa `Event` (kreiranje metoe `addStudent()`)

Pokretanjem jediničnog testa vdimo da nam još nedostaje elokventna veza „`payments()`“, prikaz pogreške vidljiv je na slici 32. Navedenu vezu također kreiramo unutar klase za `Event` kako bi stvorili vezu `hasMany` (isječak kôda 19). Jedan događaj može imati više prijavljenih sudionika.

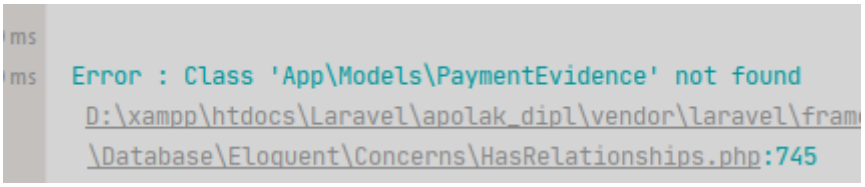


Slika 32: Klasa EventTest, test „it_can_add_a_student“ (ne postoji elokventna veza `payments()`)

```
public function payments(){

    return $this->hasMany(PaymentEvidence::class);

}
```

Isječak kôda 19: Klasa `Event` (kreiranje metode `payments()`)

Kreiranjem metode payments() sustav nam vraća novu pogrešku, nedostata klase `PaymentEvidence` (slika 33). Traženi model kreiramo pomožu terminala i artisanove naredbe „`php artisan make:model PaymentEvidence -m`“ čime ujedno kreiramo i migraciju za naveden model (slika 34).

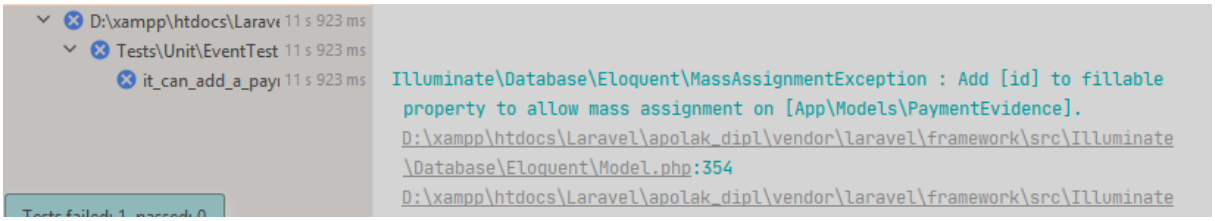


Slika 33: Ne postoji model `PaymentEvidence`

```
D:\xampp\htdocs\Laravel\apolak_dipl>php artisan make:model PaymentEvidence -m
Model created successfully.
Created Migration: 2020_11_23_195543_create_payment_evidence_table

D:\xampp\htdocs\Laravel\apolak_dipl>
```

Slika 34: Kreiranje modela `PymentEvidence`

Pokrenemo test i dobivamo novu pogrešku (slika 35). Potrebno je dodati „`id`“ u svojstva koja mogu biti masovno ispunjena u modelu `PaymentEvidence`. Ispravak pogreške vidljiv je u isječku kôda 20.

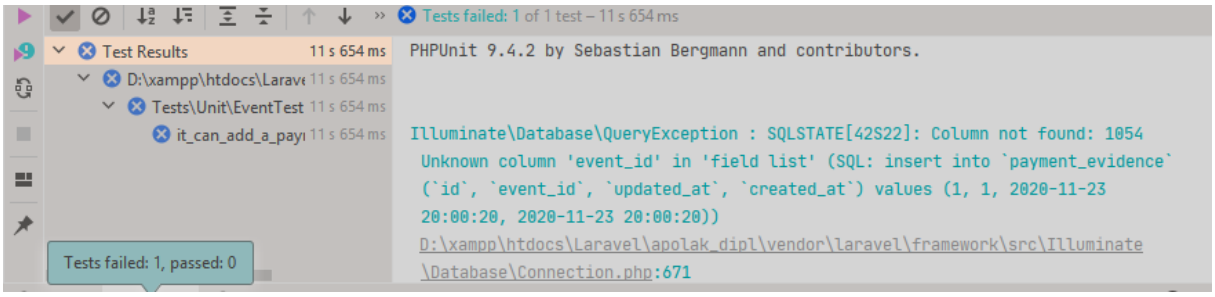


Slika 35: prikaz rezultata provedbe neuspješno izvršenog testa (potrebno dodati „`id`“ u fillable svojstva klase `PaymentEvidence`)

```
class PaymentEvidence extends Model

{

    use HasFactory;

    protected $guarded= [];

}
```

Isječak koda 20: Klasa `PaymentEvidence` (dodavanje `guarded` svojstva)

S obzirom da smo kreirali model `PaymentEvidence`, no još nismo dopunili shemu za kreiranje tablice javljena je pogreška o ne postojanju atributa „`event_id`“ (slika 36).

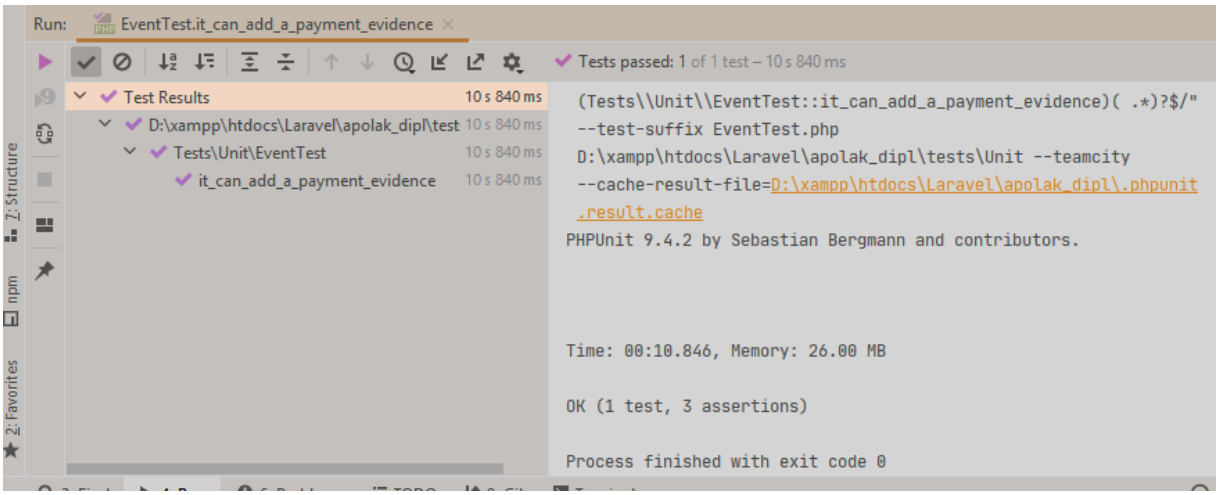


Slika 36: prikaz rezultata provedbe neuspješno izvršenog testa (nedostaje atribut „`event_id`“ u tablici `payment_evidence`)

Sljedeći isječak kôda prikazuje popunjavanje sheme za kreiranje tablice `payment_evidence` u klasi `CreatePaymentEvidenceTable`. Pokrenemo test „`it_can_add_a_payment_evidence`“ i vidimo da je uspješno izvršen (slika 37).

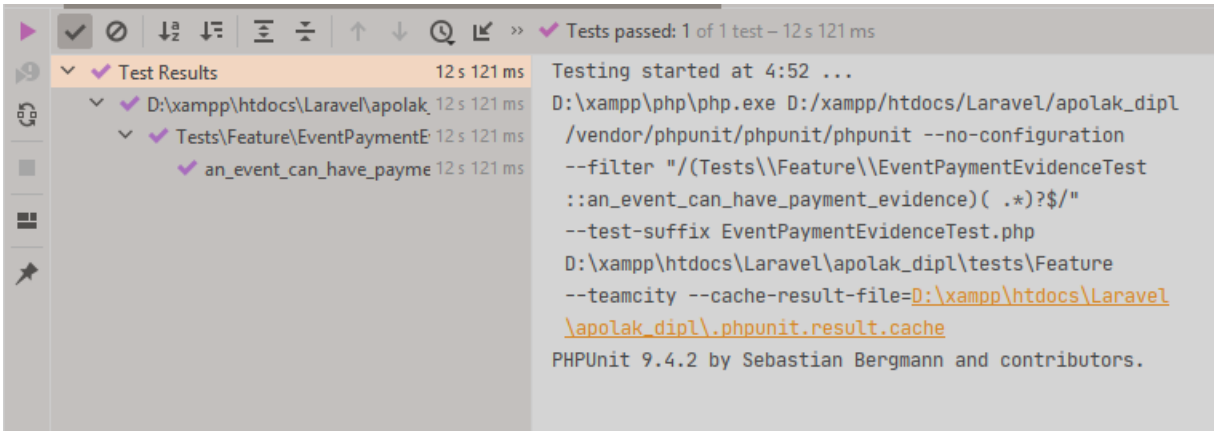
```
public function up()
  {
      Schema::create('payment_evidence', function (Blueprint $table) {
          $table->increments('id');
          $table->unsignedBigInteger('event_id');
          $table->unsignedBigInteger('student_id');
          $table->boolean('paid')->default(false);
          $table->timestamps();
      });
  }
```

Isječak kôda 21: popunjavanje funkcije za kreiranje modela PaymentEvidence

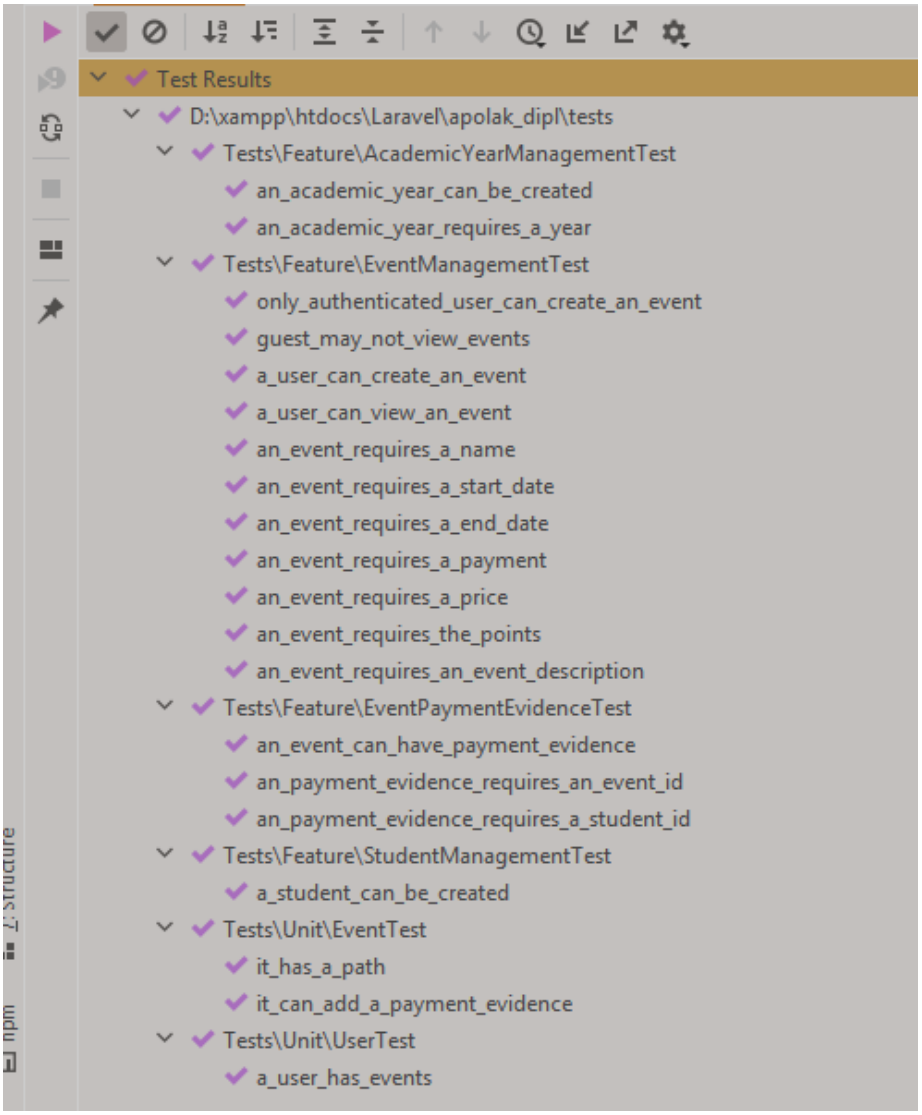


Slika 37: Uspješno proveden test „`it_can_add_a_payment_evidence`“ (klasa `EventTest` za jedinično testiranje)

Nakon uspješno provedenog testiranja jediničnog testa vračamo se na funkcionalni test „`an_event_can_have_payment_evidence`“. Pokretanjem testa dobivamo rezultat o uspješnom provođenju (slika 38), možemo očistiti kôd i nastaviti dalje jer vidimo da se svi testovi i dalje uspješno provode (slika 39).

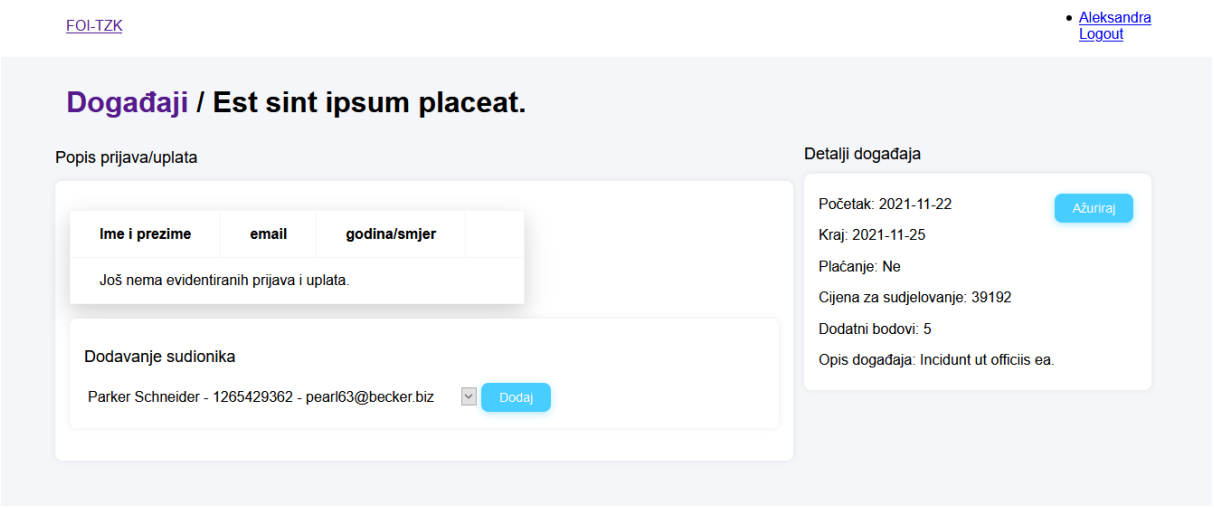


Slika 38: Uspješno proveden test „`an_event_can_have_payment_evidence`“ (klasa `EventPaymentEvidence` za funkcionalno testiranje)

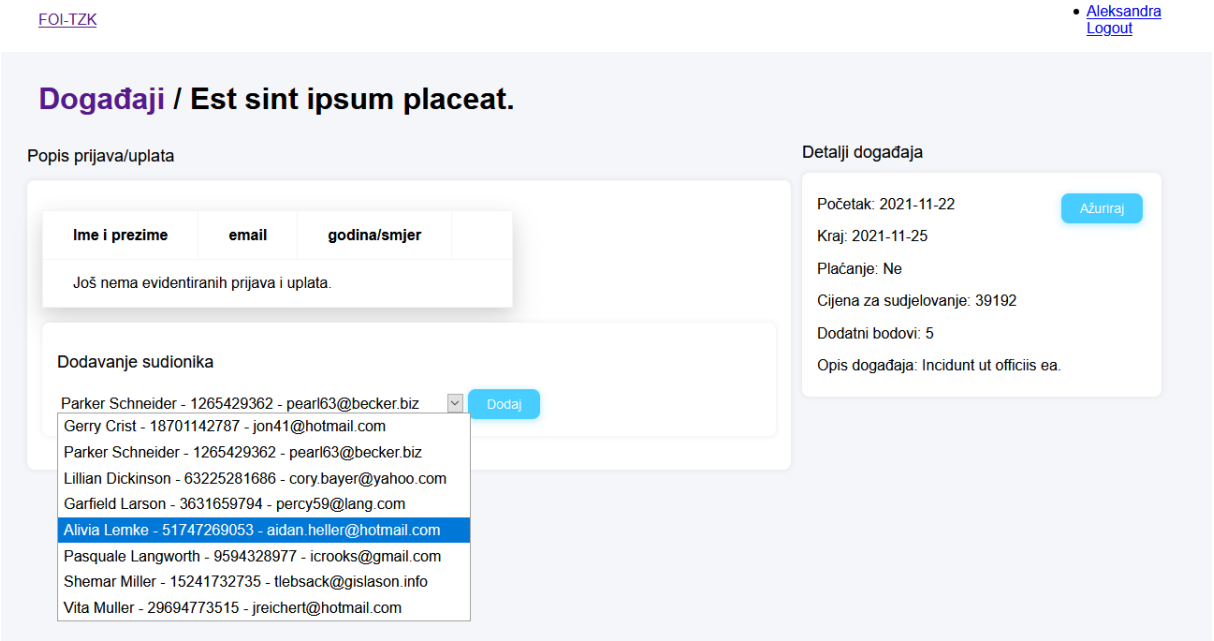


Slika 39: Uspješno provedeni svi trenutno kreirani testovi

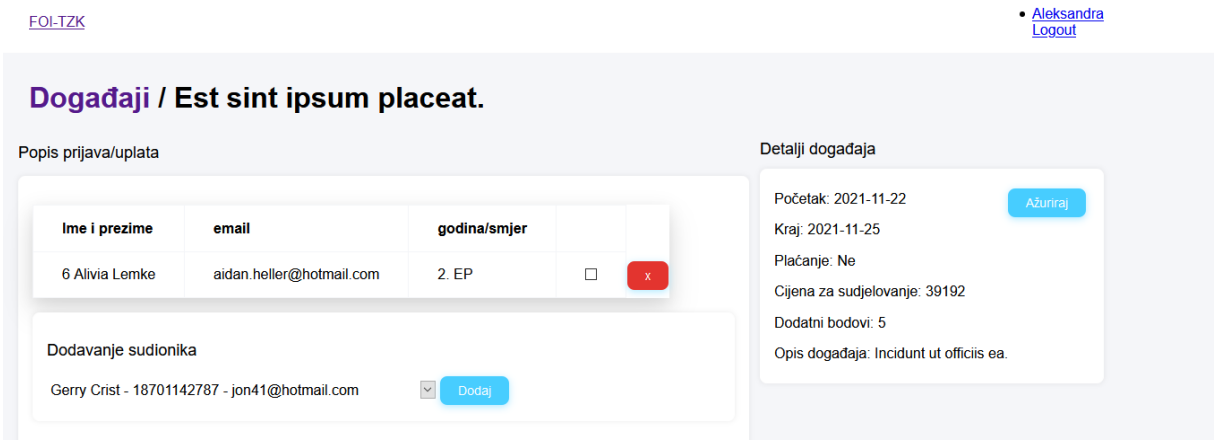
Nakon provedenih testova kreirane su stranice za preglednik koje će nam omogućiti kreiranje događaja, pregled detalja događaja i dodavanje polaznika pojedinom događaju. Na slici 40 vidljiv je prikaz stranice sa detaljima o pojedinom događaju. Ona sadrži tablicu sa prikazom popisa prijava/uplata. Vidimo da je trenutno prazna jer još nismo dodali niti jednog polaznika. U donjem dijelu prozora vidimo polje za dodavanje studenata. Kada odaberemo jednog studenta (prikazano slikom 41) i odaberemo gumb dodaj, student će biti dodan u gornji popis polaznika, te će biti omogućeno mijenjanje statusa o obavljenoj uplati za sudjelovanje na događaju (slika 42).



Slika 40: Stranica za prikaz detalja o događaju



Slika 41: Stranica za prikaz detalja o događaju (odabir studenta koji će sudjelovati na događaju)



Slika 42: Stranica za prikaz detalja o događaju (student je dodan u područje prikaza popisa prijavljenih studenata)

5.2.3. Osiguranje kvalitete za funkcionalnost prijave korisnika

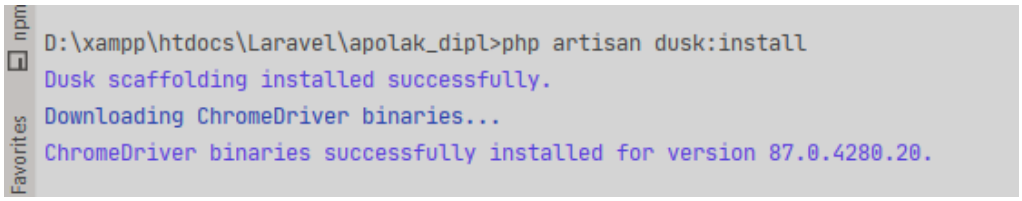
Prema prethodno kreiranoj tablici (Tablica 7) vršimo provjere i bilježimo zapažanja, daljnje zadatke, te označavamo korake koji su uspješno/neuspješno provedeni.

Da bismo bili u mogućnosti provesti testiranja preglednika instalirali smo Laravel Dusk okvir za testiranje preglednika. Detaljna dokumentacija, upute o instalaciji i korištenju dostupni su na službenoj stranici Laravela[7].

U terminalu PHPStorm IDE razvojnog okruženja unosimo naredbu za laravel/dusk ovisnost za naš projekt: „`composer require --dev laravel/dusk`". Nakon uspješno provedenog prvog koraka instalacije prikazanog na slici 43, koji je instalirao Dusk paket u naš project, slijedi upotreba artisanove naredbe za instalaciju Duska: "`php artisan dusk:install`". Ovim korakom kreirana je "`Browser`" datoteka unutar postojeće datoteke "`tests`". Tu će se spremati svi testovi kreirani pomoću Dusk okvira. Korak instalacije prikazan je na slici 44.

[7] https://laravel.com/docs/8.x/dusk#installation

Slika 43: Prvi korak instalacije Laravel Dusk alata



Slika 44: Drugi korak instalacije Laravel Dusk alata

Nakon instalacije alata možemo kreirati testove. U terminalu putem naredbe "`php artisan dusk:make Register`" kreiramo test za provjeru preglednika stranice za pristup aplikaciji. Unutar klase `Register` kreirali smo test "`RegisterTest`" kojim provjeravamo da li funkcionira stranica za registraciju i da li nas vraća na početnu stranicu. Test je prikazan isječkom koda 22. Nakon njega slijedi test "`not_register_user_can_not_login`" koji provjerava da korisnik koji se nije prethodno registrirao na sustav i ne postoji u bazi podataka nije u mogućnosti prijaviti se u sustav (isječak koda 23). Test "`guest_can_not_register_with_used_email`" provjerava može li se više korisnika registrirati sa istim emailom (isječak koda 24). I na posljetku test "`not_register_user_can_not_acces_to_events`" provjerava da li je uključeno

preusmjeravanje ili neregistrirani korisnik može vidjeti stranice aplikacije, u ovom slučaju, testiramo pristup stranici događaja (isječak koda 25).

```
        public function RegisterTest() {
            $this->browse(function (Browser $browser) {
                $browser->visit('/register')
                    ->assertSee('Registracija')
                    ->type('name', 'vanja')
                    ->type('email', 'vanja@foi.hr')
                    ->type('password', 'password')
                    ->type('password_confirmation', 'password')
                    ->press('Register')
                    ->assertPathIs('/dashboard')
                    ->logout();
            });
        }
```

Isječak koda 22: Klasa `Register` (dusk test "`RegisterTest`")

```
    /** @test */
        public function not_register_user_can_not_login() {
            $this->browse(function (Browser $browser) {
                $browser->visit('login')
                    ->assertSee('Prijava')
                    ->type('email', 'ana@foi.hr')
                    ->type('password', 'password')
                    ->press('Login')
                    ->assertSee('These credentials do not match our
records.');
            });
        }
```

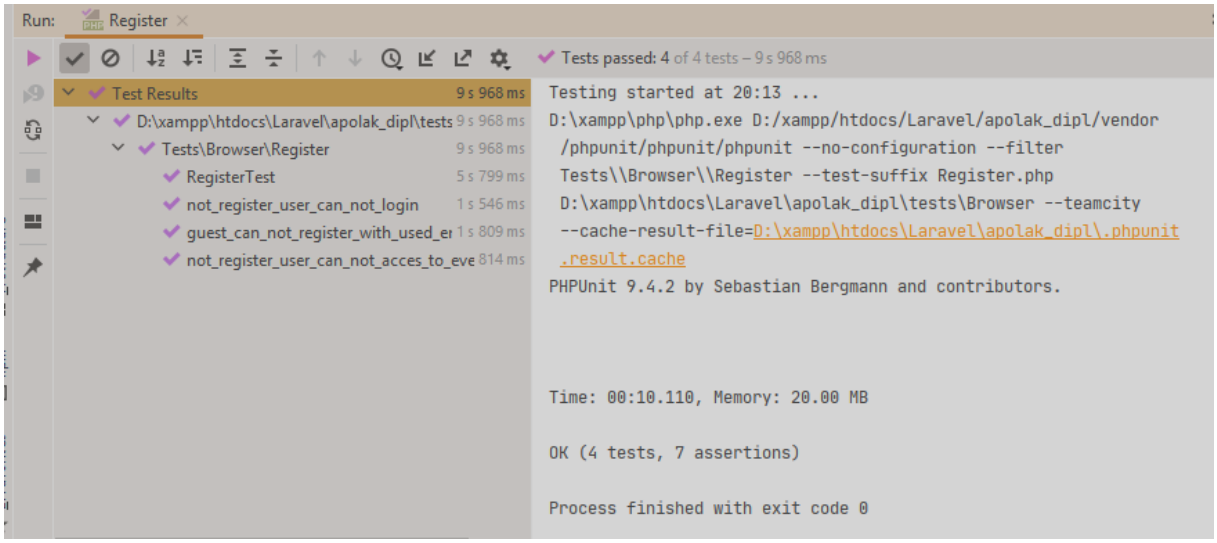
Isječak koda 23: Klasa `Register` (dusk test "`not_register_user_can_not_login`")

```
    /** @test */
        public function guest_can_not_register_with_used_email(){
            $this->browse(function (Browser $browser) {
                $browser->visit('/register')
                    ->assertSee('Registracija')
                    ->type('name', 'Marin')
                    ->type('email', 'marko@foi.hr')
                    ->type('password', 'password')
                    ->type('password_confirmation', 'password')
                    ->press('Register')
                    ->assertSee('The email has already been taken.');
            });
        }
```

Isječak koda 24: Klasa `Register` (dusk test "`guest_can_not_register_with_used_email`")

```
    /** @test */
        public function not_register_user_can_not_acces_to_events(){
            $this->browse(function (Browser $browser) {
                $browser->visit('events')
                    ->waitForLocation('/login')
                    ->assertPathIs('/login')
                    ;
            });
        }
```

Isječak koda 25: Klasa `Register` (dusk test "`not_register_user_can_not_acces_to_events`")



Slika 45: prikaz uspješno provedenih dusk testova za provjeru funkcionalnosti prijave u aplikaciju

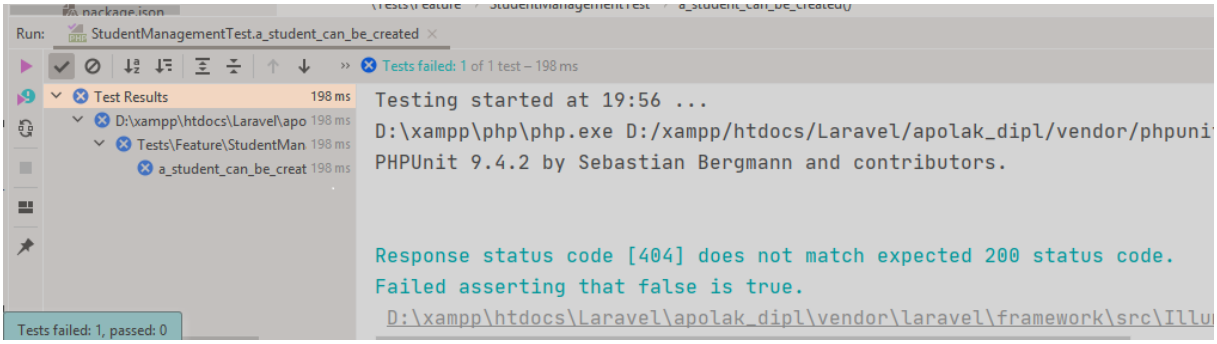
Nastavno na osiguranje kavalitete za funkcionalnost prijave u apliakciju te kontrolu linkova za login, registraciju i veze na postojeće stranice. Slijedi tablica 8 sa uspješno provedenim navedenim koracima kontrole.

Tablica 8: Rezultati provođenja QA nad funkcionalnosti prijave u aplikaciju

Test	Korak (napomena)	Rezultat
Funkcionalnost: prijava u aplikaciju		
Testiranje funkcionalnosti	Funkcionira li prikaz početne stranice aplikacije	√ da
	Da li linkovi na početnoj stranici vode na odgovarajuće stranice za prijavu i registraciju, a potom i stranice nakon prijave	√ da
	Provjeri postoje li sintaksne pogreške	√ ne
	Provjeri ima li pogreške boroj 404 (stranica nije pronađena, datoteka nije pronađena, poslužitelj nije pronađen)	√ ne
Testiranje sigurnosti	Provjeriti mogu li samo ovlašteni korisnici pristupiti aplikaciji i svim njenim stranicama	√ da

	Provjeri mogu li se korisnici prijaviti bez lozinke ili sa pogrešnom lozinkom	√ ne
SEO testiranje	Provjeri nedostaju li naslovi stranica	√ ne
	Provjeri nedostaju li oznake <h1>	√ ne
	Provjeri nedostaju li oznake <h2>	√ ne

Među ostalim oblicima rada na osiguranju kvalitete projekta nalaze se komunikacija i konzultacije sa profesorom i demonstratorima



o potrebama koje su se javljale kroz rad sa studentima i vođenjem evidencija njihovog prisustvovanja nastavi i raznim izvan nastavnim aktivnostima.

6. Zaključak

Vjerujem da smo uspjeli jasno prikazati i sažeti pojam TDD-a i osiguranja kvalitete programskog proizvoda. Sabrana je lista aktualnih alata koji nam mogu pomoći u primjeni istog uz korištenje PHP jezika koji i danas prednjači kao jezik za razvoj web stranica.

Kada sumiramo sve što smo opisali u prethodnim poglavljima i prikazali u praktičnom dijelu rada, sa sigurnošću možemo reći da pisanje sigurnog kôda možemo više pripisati stanju uma nego vještini. Uvijek je potrebno biti u toku sa najnovijim sigurnosnim ranjivostima, prijetnjama i rješenjima. Čitati blogove i tehničke novosti, te naravno konstantno primjenjivati nova saznanja u kôdu kojeg kreiramo. Prilikom pisanja testova, potrebo je zapitati se, kako naš kôd može biti izložen, te sukladno tome poduzeti odgovarajuće mjere. Iako je vrijeme programiranja duže u početku zbog kreiranja testova, u konačnici testovi će nam skratiti vrijeme refaktoriranja i ispravljanja grešaka, te omogućiti bržu isporuku proizvoda klijentima. Najviše vremena uzima učenje i savladavanje same tehnike rada, no s vremenom to postaje jednostavnije i brže, olakšano je održavanje jednostavnosti projekta i ujedno štedimo vrijeme potrebno za programiranje i razvoj.

Osiguranje kvalitete je ključno i ne bi se smjelo izbjegavati tijekom razvojnog ciklusa web stranice ili aplikacije. Osigurava da naš proizvod pruži vrhunske funkcionalnosti i performanse. Također, svakako bi bilo idealno kada bi tvrtke imale zaseban tim koji je zadužen za QA i radi „ruku pod ruku“ sa dizajnerima, programerima i najbitnije, klijentima. Osiguranje kvalitete potrebno je provoditi neovisno o dizajnu i razvojnom procesu. Taj proces će otkriti greške u dizajnu i razvojne pogreške dok testira korisničko sučelje proizvoda i mjeri korisničko iskustvo. U mnogim slučajevima, korisnička lista osigurava da se ništa ne propusti tokom procesa osiguranja kvalitete i da web aplikacija radi besprijekorno nakon postavljanja.

S obzirom na to da sam i sama početnik u TDD-u i QA-u mnogo vremena je potrošeno na učenje i traženje odgovarajućih alata i okvira za razvoj. Sukladno tome kreiran je jedan dio aplikacije za prikaz postupka primjene TDD-a i provedbe osiguranja kvalitete za isti dio projekta. Aplikacija je razvijana u PHPStorm IDE okruženju uz instalaciju vrlo popularnog Laravel alata, koji je ujedno i najbolji izbor za početnike u programiranju TDD tehnikom. Za provođenje jediničnih i funkcionalnih testova tu je PHPUnit razvojni okvir koji je instaliran zajedno sa Laravelom na razvojno okruženje. Sukladno korištenju Laravela kao razvojnog okvira za primjenu TDD razvoja, tu je i Laravel Dusk alat koji nam pomaže u provedbi osiguranja kvalitete prilikom razvoja aplikacije. Kao podršku za korištenje Apache servera instalirana je XAMPP platforma za web rješenja razvijena od strane „Apache Friends“.

Najveći dio edukacije odrađen je na službenim stranicama Laracasta gdje se redovno dopunjava baza sa novim edukacijskim video materijalima od strane Jeffreya Waya (kreatora Laravela) i njegovih suradnika. Svakako smatram da ovo nije kraj učenju i radujem se napretku i proširenju stečenih znanja.

Popis literature

[1] Acquaint Softtech (bez dat.) *5 PHP Quality Assurance Tools You Must Check Out*. Preuzeto 24.10.2020. s https://www.acquaintsoft.com/5-php-quality-assurance-tools-must-check/

[2] Ambler, S. W. (bez dat.). *Introduction to Test Driven Development (TDD)*. Preuzeto 06.04.2020. s http://agiledata.org/essays/tdd.html#WhatIsTDD

[3] Ataullah Khan, Md (01.12.2017). *PHP Test Driven Development Part 1: Introduction*. Preuzeto 16.07.2020. s https://hackernoon.com/php-test-driven-development-part-1-introduction-5483362d79b5

[4] Beck Kent (2002). *Test-Driven Development By Example*. Three Rivers Institute, draft 14.07.2002.

[5] Berezhnoi, Roman (23.01.2019.). *What is QA and why quality assurance is important in web development proces?* Preuzeto 18.10.2020. s https://medium.com/@f5studio/what-is-qa-and-why-quality-assurance-is-important-in-web-development-process-f17ae9c59de7

[6] Guru99 (2020). *What is Quality Assurance(QA)? Process, Methods, Example*. Preuzeto 28.10.2020. s https://www.guru99.com/all-about-quality-assurance.html?fbclid=IwAR3x68xbK1RNzYt2HPrW7-zF1JITCSGqgOQKmZSPiW6I2_tFgoJJW_flU-E

[7] JRebel (27.04.2016.). *When to Use Test Driven Development*. Preuzeto 16.07.2020. s https://www.jrebel.com/blog/when-to-use-test-driven-development

[8] Khatavkar, Viraj (23.02.2017.). *Laravel Dusk – Intuitive and Easy Browser Testing for All*. Preuzeto 27.10.2020. s https://www.sitepoint.com/laravel-dusk-intuitive-and-easy-browser-testing-for-all/

[9] Koutifaris, Andrea (02.07.2018.). *Test Driven Development: what it is, and what it is not.* Preuzeto 15.07.2020. s https://www.freecodecamp.org/news/test-driven-development-what-it-is-and-what-it-is-not-41fa6bca02a2/

[10] *Laravel* (13. 05. 2020.). U Wikipedia. Preuzeto 11.09.2020. s https://hr.wikipedia.org/wiki/Laravel

[11] Mody Rahul (28.04.2017.). *Test Driven Development — Breaking Down Unit & Integration Tests.* Preuzeto: 16.07.2020. s https://medium.com/@RahulTMody/test-driven-development-breaking-down-unit-integration-tests-d4a723817419

[12] Monus, Anna (23.06. 2020.). *9 Best Automated Testing Frameworks For PHP*. Preuzeto 16.07.2020. s https://www.hongkiat.com/blog/automated-php-test/

[13] Njenga, Alice (21.11.2018.). *Top PHP frameworks.* Preuzeto 11.09.2020. s https://raygun.com/blog/top-php-frameworks/

[14] Porras, Allan (11.11.2019.). *Scrum Methodology for Digital Product Development.* Preuzeto 04.11.2020. s https://blog.4geeks.io/scrum-for-digital-product-development/

[15] Prestianni, Timothy (23. 07. 2020.). Website quality assurance. Preuzeto 18.10.2020. s https://www.unleashed-technologies.com/blog/website-quality-assurance?fbclid=IwAR2175MeU2OomRIyOLKUzFYof1jCVTebKrCmI-5ngjo242mehnNWk5g0eqI

[16] Rouse, Margaret i suradnici, TechTarget (2019.). *quality assurance (QA)*. Preuzeto 18.10.2020. s https://searchsoftwarequality.techtarget.com/definition/quality-assurance*Test Driven Development Tools and Agile Process* (25.11.2019.). Preuzeto 12.07.2020. s https://www.xenonstack.com/insights/what-is-test-driven-development/

[17] Shore, James (03.02.2014.). *How does TDD change QA?* Preuzeto 19.10.2020. s https://www.letscodejavascript.com/v3/blog/2014/02/how_does_tdd_change_qa

[18] Shilpa (1.09.2020.). *How The Testers Are Invoved in TDD, BDD & ATDD Techniques?* Preuzeto 1.11.2020. s https://www.softwaretestinghelp.com/testers-in-tdd-bdd-atdd-techniques/

[19] Software Testing Help (13.09.2020.). Agile Vs Waterfall: Which Is The Best Methodology For Your Project? Preuzeto 04.11.2020. s https://www.softwaretestinghelp.com/agile-vs-waterfall/

[20] Software Testing help (30.06.2020.). *Top 10 Popular PHP Testing Frameworks and Tools*. Preuzeto 18.07.2020. s https://www.softwaretestinghelp.com/php-testing-framework-tools/

[21] Techopedia Inc. (2020) *Test driven development - TDD*. Preuzeto 22.09.2020. s https://www.techopedia.com/definition/25850/test-driven-development-tdd

[22] *Test Driven Development Tools and Agile* Process (25.11.2019.). Preuzeto 12.07.2020. s https://www.xenonstack.com/insights/what-is-test-driven-development/

[23] *xUnit* (11.05.2020.). U Wikipedia. Preuzeto 06.11.2020. s https://en.wikipedia.org/wiki/XUnit#xUnit_architecture

[24] Younes, Rafie (30.06.2017.). *8 Must Have PHP Quality Assurance Tools*. Preuzeto 25.10.2020. s https://www.sitepoint.com/8-must-have-php-quality-assurance-tools/

[25] Alati za instalaciju dostupni su na sljedećim linkovima:

- Composer: https://getcomposer.org/
- PHPStorm: https://www.jetbrains.com/phpstorm/
- XAMPP: https://www.apachefriends.org/download.html

Popis slika

Popis tablica

Popis isječaka iz kôda