Winter 2-2021

CIKLI I JETËS SË ZHVILLIMIT TË SOFTUERIT

Entor Arifi
University for Business and Technology - UBT

Programi për Shkenca Kompjuterike dhe Inxhinierisë

CIKLI I JETËS SË ZHVILLIMIT TË SOFTUERIT

Shkalla Bachelor

Entor Arifi

Shkurt / 2021
Prishtinë

Programi për Shkenca Kompjuterike dhe Inxhinierisë

Punim Diplome
Viti akademik 2014 – 2015

Entor Arifi

CIKLI I JETËS SË ZHVILLIMIT TË SOFTUERIT

Mentori: Msc. Medina Shamolli

Shkurt / 2021

Ky punim është përpiluar dhe dorëzuar në përmbushjen e kërkesave të pjesshme
për Shkallën Bachelor

ABSTRAKTI

Zhvillimi i një sistemi softuerik të sukseshëm i cili i përgjigjet standardeve të kualitetit nuk është një proces i thjeshtë. Përkundrazi, planifkim dhe përkushtim i madh kërkohet për realizimin e një projekti të tillë. Një pjesë e madhe e produkteve softuerike dështojnë për arsye të keq menaxhimit dhe planifikimit jo të duhur. Ky hulumtim shqyrton ndikimin që metodologjitë si SDLC kanë në zhvillimin e produkteve softuerike. Një pjesë e hulumtimit ka të bëjë me shqyrtimin e fazave të Ciklit të Jetës së Zhvillimit të Softuerit, renditjen dhe implementimin e saktë të tyre. Pjesa tjetër ka të bëjë me një produkt softuerik i ndërtuar me API Platform, i cili është krijuar për të provuar aspektin teorik të këtij hulumtimi. Gjetjet e këtij hulumtimi konfirmojnë se SDLC dhe implementimi i saktë i fazave të saj ka ndikim të drejtpërdrejt në kualtietin dhe suksesin e një produkti softuerik. Përzgjedhja e teknologjive për zhvillimin e atij softueri është një faktor tjetër me ndikim. Si përfundim i këtij hulumtimi del se ekipet e zhvillimit të softuerit duhet t'i kushtojnë rëndësi të veçantë implementimit të një metodologjie siq është SDLC në mënyrë që produkti i tyre të jetë i suksesshëm.

MIRËNJOHJE/FALENDERIME

Do të doja që të shprehja një falemnderim dhe një mirënjohje të thellë fillimisht familjes time të cilës i detyrohem shumë për fillimin dhe finalizimin me sukses të këtij udhëtimi, sa të vështirë aq edhe të bukur.

Gjithashtu një falemnderim i veçantë shkon për mentoren time, Msc. Medina Shamolli, për ndihmën dhe mbështetjen e çmuar që më ofroi përgjatë gjithë punës sime dhe për kontributin e saj në finalizimin e punimit tim të diplomës.

Falemnderim tjetër shkon për miqtë e mi, profesorët, dhe UBT-në për mbështetjen e vazhdueshme që më kanë dhënë gjithmonë.

PËRMBAJTJA

LISTA E FIGURAVE

FJALORI I TERMAVE

API – Application Programming Interface
CD – Continous Delivery
CI – Continous Integration
CRUD - Create Retrieve Update Delete
CSS – Cascading Style Sheets
CSV – Comma Seperated Values
DMBS – Database Management System
DOM – Document Object Model
ECMAScript – European Computer Manufacturer's Association Script
HAL - Hypertext Application Language
HTML – Hyper Text Markup Language
IDE – Integrated Development Environment
JS – JavaScript
JSON – JavaScript Object Notation
JSON-LD - JavaScript Object Notation for Linked Data
JSX – JavaScript XML
ORM – Object Relational Mapping
PHP – Personal Home Page
RDF – Resource Description Framework
REST API - Representational State Transfer
SDLC – Software Development Life Cycle
XML – Extensible Markup Language
YAML - YAML Ain't Markup Language

1. HYRJE

Cikli i Jetës së Zhvillimit të Softuerit (SDLC) është një proces me anë të të cilit zhvillohen produkte softuerike. Thënë shkurt, Cikli i Jetës së Zhvillimit të Softuerit është një sistem që zhvillohet hap pas hapi dhe shërben si urë lidhëse mes kompanive të zhvillimit të softuerit dhe klientëve. Me anë të këtij sistemi bëhet shëndrrimi i ideve në softuerë me kualitet të lartë, në kohë sa më të shkurtër, dhe çmim sa më të ulët [1].

Historia e SDLC-së filloi në të njejten kohë me historinë e Zhvillimit të Softuerit që mori hov në vitet 1950. Atë kohë, termet si "framework" dhe "agile" ende nuk ishin të përhapura në këtë fushë. Cikli i Jetës së Zhvillimit të Softuerit (SDLC) nuk është veçse një term i zbukuruar në botën e menaxhmentit, ky sistem u zhvillua për shkak të nevojës për komunikim dhe standardizim gjatë zhvillimit të projekteve softuerike [1].

Në saje të progresit që ka ndodhur gjatë viteve në industrinë e programimit dhe zhvillimit të softuerit, edhe modelet e SDLC-së kanë evoluar me kohë. Disa nga modelet me më ndikim janë: "Structured Programming" (1950), "Iterative & Incremental" (1970), "Prototyping" (1980), "Spiral" dhe "V-Model" (1980), "Scrum" (1995), "Agile Methods" (1990 – 2000), si dhe metodologjitë "Agile" bashkëkohore. Sot, ISO/IEC/IEEE 12207 është një standard kushtuar proceseve të zhvillimit të softuerit që synon të definojë të gjithë fazat dhe mekanizmat për zhvillimin dhe mirëmbajtjen e sistemeve softuerike [1].

Ky proces është bërë pjesë e pothuajse të gjitha kompanive që merren me zhvillim të softuerëve. Me përparimin e teknologjisë në shkallë të madhe, njëtrajtësisht është rritur edhe varësia që korporatat, bizneset dhe institucionet e ndryshme kanë në sistemet apo aplikacionet e tyre softuerike. Për këtë arsye, kuptimi, implementimi dhe përshtatja e saktë e SDLC është shumë e rëndësishme për secilën kompani që merret me zhvillim e ketyre softuerëve.

Zhvillimi i softuerit është një proces i gjatë dhe i komplikuar, andaj ekipi zhvillues përveç aftësive të shkëlqyera programuese duhet të kuptoj rrënjësisht edhe fazat e SDLC në menyrë që rezultati përfundimtar të jetë një produkt me kualitet të lartë. Fazat e SDLC janë: planifikimi, caktimi i kërkesave, dizajni dhe protipi, implementimi, testimi, dhe mirëmbajtja**.** Jo gjithmonë është e nevojshme që të kalohet nëpër të gjitha fazat e lartcekura. Secilit ekip apo secilës kompani i lejohet që të përshtat procesin e SDLC kundrejt nevojave të tyre.

Në një sondazh të bërë nga Innotas, 55% e të anketuarëve treguan se projekti i tyre dështoi për arsye të kohës së pamjaftueshme, stafit të pakët, dhe buxhetit të limituar. Arsyeja kryesore për

këto dështime është planifikimi jo i duhur. Më saktësisht, përjashtimi i rrethanave të paparashikueshme është e vetmja arsye. Shumë kompani e anashkalojnë analizën e realizueshmërisë dhe nuk i kushtojnë rëndësi të mjaftueshme aspekteve logjistike. Kjo shkakton pritje joreale që doemos shkaktojnë stres dhe zhgënjim dhe çojnë projektin drejt dështimit [2].

Për të ilustruar dhe përshkruar procesin e zhvillimit të një softueri sipas këtyre fazave, unë kam krijuar një softuer për menaxhimin e klinikave. Softueri është ndërtuar duke përdorur teknologji moderne si API Platform e cila në vete ngërthen vegla dhe standarde moderne për zhvillimin e web-aplikacioneve për të cilat do të flasim më detajisht në vijim. Ndërkaq, ndërfaqja (front-end) i këtij aplikacioni është zhvilluar me ReactJS, pjesa e procesimit të të dhënave dhe logjikës (back-end) me Symfony ndërsa si bazë për menaxhimin e të dhënave është përdorur MySQL.

2. SHQYRTIMI I LITERATURËS

Procesi i zhvillimit të softuerit, që ndryshe njihet edhe si metodologji, model ose cikël i jetës, është një sistem i cili përdoret për të strukturar, planifikuar dhe drejtuar procesin e zhvillimit të sistemeve softuerike. Një numër i madh i këtyre sistemve janë krijuar gjatë viteve ku secili ka përparësitë dhe mangësite e veta. Ekzistojnë çasje të ndryshme të zhvillimit të softuerit: disa prej tyre janë më të strukturuara dhe të orientuara në pjesën e inxhinieringut, ndërsa disa tjera kanë një çasje graduale ku softueri ndërtohet duke u përmirësuar gjatë gjithë kohës. Një metodologji e zhvillimit nuk është e përshtatshme për të gjitha projektet. Secila prej metodologjive përshtatet në projekte specifike, pra, duke pasur parasysh aspektin teknik, organizativ dhe ekipor [3].

Metodologjitë e ndryshme dallojnë nga njëra tjetra, dhe për këtë arsye çasja më e mirë për zgjidhjen e një problemi që ka të bëjë me ndërtim të softuerit varet nga lloji i problemit. Nëse problemi është i kuptueshëm nga i gjithë ekipi dhe puna mund të planifikohet paraprakisht, modeli "ujëvarë" është më i përshtatshëm. Në anën tjetër, nëse problemi është më i veçantë, pra, diçka që ekipi zhvillues nuk ka rastisur më parë, një çasje "graduale" (shkallë – shkallë) do të ishte më e përshtatshme.

2.1 Cikli i Jetës së Zhvillimit të Softuerit

SDLC përbëhet prej disa fazave të cilat janë qartë të definuara, të ndara nga njëra tjetra dhe kanë qëllim të caktuar. Ky proces përdoret prej ekipeve të zhvillimit të sofuterit për të planifikuar, dizajnuar, testuar dhe dorëzuar sisteme softuerike. Qëllimi i SDLC është që të nxjerr si rezultat përfundimtar një softuer të një kualiteti shumë të lartë i cili i përgjigjet kërkesave të klientit duke i respektuar kufzimet e buxhetit dhe kohës [4].

Sistemet softuerike janë produkte të ndërlikuara ku jo rrallë herë përvec zhvillimit të moduleve të ndryshme, duhet të përshtaten module që janë zhvilluar nga palë të treta. Për të trajtuar varësitë dhe komplikimet e ndryshme të cilat shfaqen gjatë këtij procesi, janë krijuar disa metodologji të SDLC. Disa prej këtyre metodologjive janë: Agile, Lean, Waterfall, Iterative, Spiral, DevOps etj [5].

Të gjitha metodologjitë e zhvillimit të softuerit në thelb përcjellin fazat e SDLC mirëpo mënyra e implementimit dallon për metodologji të ndryshme. Si shembull marrim Waterfall dhe SCRUM. Në metodologjinë Waterfall, secila kërkesë përpunohet dhe pastaj shtohet në listën e

kërkesave. Në vazhdim, këto kërkesa implementohen së bashku gjatë një periudhe tre deri në nëntë muaj, kështu duke kaluar nëpër të gjitha fazat e SDLC. Për dallim nga Waterfall, në SCRUM një rast përdorimi (use case) përpunohet në një user story ku kalon nëpër të gjitha fazat gjatë një periudhe që zgjat dy deri katër javë. Kjo periudhë ndryshe njihet edhe si "sprint". Këto dy metodologji edhe pse dallojnë nga njëra tjetra për nga mënyra e implementimit, të dyja përshkojnë të gjitha fazat e SDLC ku pas përfundimit zakonisht fillon cikli i ri [5].

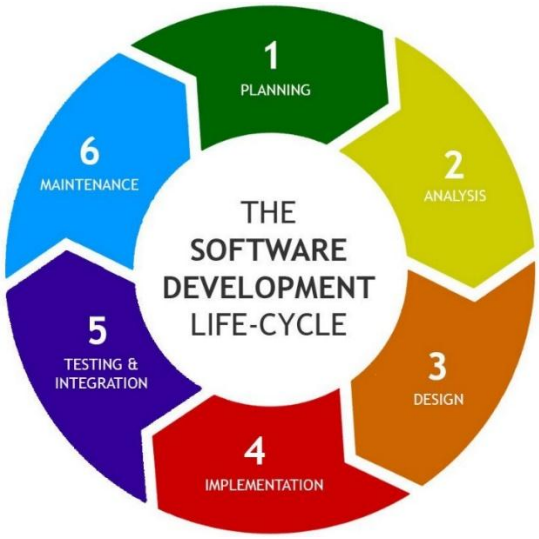

Figura 1. Fazat e ciklit të jetës së zhvillimit të softuerit [5]

Figura e mësipërme (Figura 1) ilustron fazat e SDLC-së, të shënuara me numra sipas rendit të zbatimit në cikël. Fazat zbatohen në këtë radhë: planifikimi, caktimi i kërkesave, dizajni dhe protoipi, implementimi, testimi dhe mirëmbajtja. Ky cikël mund të përseritet një ose më shumë herë, në varësi të metodologjisë së përdorur.

2.2 Fazat e SDLC

Cikli i Jetës së Zhvillimit të Softuerit përbëhet prej shtatë fazave ku secila prej tyre ka funksion të veçantë dhe përdoren për të treguar se në cilën pikë gjendet projekti në kohë të caktuar, si dhe cili është hapi vijues në këtë cikël. Këto faza janë: planifikimi, caktimi i kërkesave, dizajni dhe prototipi, implementimi, testimi, publikimi dhe në fund, mirëmbajtja [6]. Në raste të caktuara, sidomos kur produkti që planifikohet të zhvillohet është i madh, fazat e lartëcekura jo rallë herë ndahen në njësi më të vogla për ta bërë punën më lehtë të menaxhueshme [7].

Secila prej këtyre fazave si rezultat përfundimtar prodhon një artefakt që mund të jetë dokument, diagram ose softuer. Ky prodhim i fazës paraprake i jepet si lëndë hyrëse fazës vijuese [8].

2.2.1 Planifikimi

Në fazën e planifikimit bëhet vlerësimi i qëllimit dhe objektivave të projektit. Kjo përfshinë llogaritjen e çmimit të punës dhe materielve tjera, caktimi i afateve kohore për objektiva të ndryshëm si dhe krijimi i ekipit punues dhe strukturës udhëheqëse. Parashikimi i rreziqeve dhe krijimi i procedurave të parandalimit të rreziqeve është një hap tjetër i rëndësishëm që ndodh në këtë fazë [7].

Gjatë planifikimit merren parasysh mendimet dhe pikëpamjet e palëve të interesit. Palë të interesit janë ato të cilët përfitojnë prej projektit në fjalë. Gjatë kësaj faze është mire të grumbullohen informata dhe pikëpamje prej zhvilluesëve, eskpertëve të çështjes në fjalë, shitësve, blerësve të mundshëm, etj [7].

Këtu duhet të përcaktohet saktësisht fushëveprimi dhe qëllimi i projektit në mënyrë që të krijohet një parafytyrim i saktë i projektit. Kjo ndihmon ekipin për të krijuar softuerin në mënyrë sa më efektive, me shpenzime minimale, si dhe cakton kufijtë e projektit duke mos lejuar që të zgjerohet apo të ndryshojë nga qëllimi fillestar [7].

2.2.2 Caktimi i kërkesave

Procesi i caktimi të kërkesave është një hap shumë i ndijshëm dhe i rëndësishëm që cakton rrjedhën e projektit. Në këtë fazë bëhet edhe caktimi i mekanizmave që garantojnë kualitetin e softuerit si dhe projektit në përgjithësi. Për këtë arsye, ky hap zakonisht udhëhiqet nga antarët më me përvojë të ekipit. Këtu, ekipi duhet të bëj grumbullimin e hollësishëm dhe të saktë të kërkesave të projektit. Ky hap ndihmon që të caktohet afati kohor për kryerjen e objektivave të projektit [9].

2.2.3 Dizajni dhe Prototipi

Në fazën e dizajnit projektohet softueri në fjalë. Si pjesë e dizajnit caktohet arkitektura e softuerit, mënyra e komunikimit të shërbimeve mes vete, metodologjitë e zhvillimit, sistemi i verzionimit të kodit, strategjitë e publikimit të ndryshimeve si dhe shumë aspekte tjera. Këtu

caktohen edhe platformat për të cilat do të zhvillohet softueri dhe platformat prej nga mund të çaset përdoruesi. Në këtë hap bëhet edhe zgjedhja e gjuhës programuese ku duhet patur parasysh platformat e zgjedhura pasi që këto dyja janë të lidhura ngushtë me njëra tjetrën [7].

Siguria është një aspekt tjetër i rëndësishem i një softueri, andaj gjatë fazës së dizajnit caktohen mënyrat më të sigurta për ruajtjen e të dhënave, bëhet kriptimi i komunikimeve, zgjedhet algoritmi më efikas dhe i sigurtë për ruajtjen e fjalëkalimeve (hashing algorithm), etj [7].

Te ndërfaqja e përdoruesit shqyrtohen mënyrat e ndryshme se si përdoruesi ndërvepron me softuerin dhe mënyrën se si softueri reagon ndaj veprimeve të përdoruesit [7].

Jo rallë herë pjesë e fazës se dizajnit është edhe prototipi. Protipi është një version minimal i softuerit në fjalë i cili përmban karakteristikat më të rëndësishme. Prototipi përdoret që të tregoj se si duket dhe si funksionon softueri në fjalë. Ky verzion përdoret për të marrë mendime dhe kritika nga palët e interesit herët në fazën e zhvillimit për shkak që është shumë më pak e kushtueshme që këto ndryshime bëhen në këtë fazë [7].

2.2.4 Implementimi

Në këtë fazë bëhet zhvillimi i softuerit duke u bazuar në kërkesat e grumbulluara në fazat e mëhershme. Në mënyrë që ekipi zhvillues ta ketë sa më të lehtë zhvillimin e softuerit, është shumë e rëndësishme që kërkesat të jenë të organizuara dhe dokumentuara mirë. Zhvilluesit njëashtu duhet ndjekin udhëzimet dhe standardet e vendosura nga kompania gjatë zhvillimit të softuerit [10].

Gjatë kësaj faze bëhet edhe dokumentimi i softuerit. Dokumentimi mund të shprehet në formën e një udhëzuesi për përdoruesin i cili përmban në vete procedura të hollësishme mbi përdorimin e atij softueri. Formë tjetër e dokumentimit mund të jenë edhe komentet në kodin burimor të cilat vendosen nga zhvilluesit për të shpjeguar pjesë të caktuara të kodit [7].

2.2.5 Testimi

Softueri duhet testuar hollësisht para se të bëhet i çashëm për publikun. Procesi i testimit garanton se secili modul funksionon ashtu si është paraparë. Njëashtu, pjesë të ndryshme të softuerit duhet të testohen për përpuethshmëri, pra, të sigurohet se funksionojnë mirë së bashku.

Testimi zakonisht bëhet në një ambient të veçantë ku inxhierët e specializuar sigurohen që çdo modul funksionon si duhet. Përveç testimit manual, një pjesë e madhe e testimit mund të automatizohet duke përdorur vegla dhe sisteme të posaçme.

Faza e testimit në shkallë të madhe ndihmon në lokalizimin dhe largimin e gabimeve para se softueri të publikohet. Kjo nënkupton që softueri do të ketë një kualitet të lartë, rrjedhimisht një numër më të madh të përdoruesëve [7].

2.2.6 Publikimi

Pas fazës së testimit ku sigurohet që të gjitha gabimet e gjetura janë rregulluar, fillon faza e publikimit. Gjatë kësaj faze, në përputhje më udhëzimet e udhëheqësit të projektit, softueri publikohet, kështu duke e bërë të çasshëm për përdoruesit [9]. Fillimisht softueri duhet të publikohet në një ambient testues në mënyrë që të sigurohet që nuk ka probleme në këtë fushë. Nëse gjithcka shkon në rregull, vetëm atëherë softueri bëhet i çashem për përdoruesit [11].

Procesi i publikimit të softuerit mund të jetë tejet i ndërlikuar dhe zakonisht ky proces automatizohet [7]. Automatizimi i procesit të publikimit të softuerit e bën të mundur lëvizjen e kodit softuerik, më saktësisht produktit softuerik, mes ambienteve testuese dhe atyre reale duke përdorur procese të automatizuara. Këto procese të përsëritshme dhe të besueshme mundësojnë zbatimin e ndryshimeve më shpesh dhe më shpejtë dhe në të njejtën kohë zvoglojnë nevojën për ndëryrhje nga njerëzit [12].

2.2.7 Mirëmbajtja

Pasi që softueri është publikuar, ekipi zhvillues përqëndrohet në mirëmbajtje. Në këtë fazë, zhvilluesit duhet të jenë të gatshëm t’i përgjigjen kërkesave për përmirësim, rregullim të gabimeve dhe krijimin e moduleve shtesë. Këto kërkesa merren prej palëve të ndryshme dhe është detyrë e ekipit udhëheqës të caktoj se cilat prej këtyrë kërkesave do të miratohen dhe zbatohen. Kjo është faza e fundit e SDLC [13].

2.3 API Platforma

API Platforma është një "full stack framework" i fuqishëm dhe i lehtë për t'u përdorur. Ky "framework" i kushtohet projekteve të bazuar në API të cilat implementojnë arkitekturën Jamstack. API Platforma ka librarinë e saj bazë të shkruar në PHP me anë të së cilës mund të krijojmë projekte të bazuar në API të cilat mbështesin standarde moderne si JSON-LD, Hydra dhe OpenAPI [14].

API Platforma gjithashtu ofron vegla të shkruara në JavaScript me anë të cilave mund të krijojmë aplikacione në web dhe në mobile duke përdorur teknologjitë më të famshme për "front-end". Këto vegla procesojnë dokumentimin e API-t (Hydra, OpenAPI) për të gjeneruar faqe dinamike dhe interaktive duke u bazuar në entitetet që kemi krijuar. Përveç kësaj, API platforma është e pajisur edhe me konfigurime për Docker dhe Kubernetes të cilat mundësojnë publikimin e shpejtë të aplikacionit në internet [14].

API Platforma përbëhet nga këta komponentë:

- **Core Library**. Është libraria bazë me anë të së cilës mund të krijojmë REST API projekte të cilat mbështesin standarde moderne si JSON-LD, Hydra, HAL, Swagger/OpenAPI, XML, JSON, CSV dhe YAML. Mund të përdoret së bashku me Symfony (mënyra e preferuar) ose veçmas [14].
- **Schema Generator**. Është një vegël me anë të së cilës mund të gjenerojmë klasat e nevojshme në PHP nga një fjalor RDF siç janë Schema.org dhe ActivityStreams. Të gjitha klasët e gjeneruara përcjellin standardet më të fundit të programimit siç janë: enkapsulimi, përdorimi i saktë i klasave konkrete dhe abstrakte, përdorimi i "interfaces". Përveç kësaj, Schema Generator gjeneron dokumentimin e nevojshëm, bën lidhjen me bazën e të dhënave përmes Doctrine ORM, bën validimin e të dhënave, etj [14].
- **Admin.** Është një vegël për krijimin e panelit të administrimit me anë të të cilit mund të menaxhojmë të gjithë entitetet e krijuara si dhe të kryejm veprime mbi to (CRUD). Ky panel i adminstrimit fuqizohet nga React Admin, një librari e ndërtuar me React e cila ndërton një ndërfaqe tejet interaktive dhe të lehtë për t'u përdorur. Në mënyrë që kjo të funksionoj, API i krijuar duhet t'i përgjigjet standardit Hydra.

Vlenë të ceket se secila pjesë e React Admin mund të ndryshohet nga programeri, sipas nevojës [14].

- **Client Generator.** Është një vegël për krijimin e skeletit të një "front-end" projekti. Me anë të Client Generator në mund të krijojmë "Single Page Applications" në web dhe aplikacione mobile të cilat kosumojnë çdo API projekt i cili është ndërtuar duke përcjellur formatin Hydra. Me Client Generator mund të krijojmë SPA me cilën do nga këto teknologji: Next.js, Nuxt.js, Quasar, Vuetify, React & Redux, React Native, Vue.js [14].

Në vazhdim do të flasim për disa nga konceptet dhe termat e përmendur më lartë.

2.3.1 REST API

REST API, ndryshe i njohur edhe si RESTful API, është një ndërfaqe e programimit të aplikacioneve (API) e cila ndjek strukturën e arkitekturës REST dhe mundëson komunikimin me shërbime të webit të cilat ndjekin po të njejtën arkitekturë. REST u krijua nga Roy Fielding dhe nënkupton "representational state transfer" e që në shqip përkthehet në "transferimi i gjendjes përfaqsuese" [15].

Një API është një grumbull i përkufizimeve dhe protokoleve të cilat përdoren për ndërtimin e aplikacioneve softuerike. Mund të themi se API është një kontratë mes një entiteti që ofron të dhëna (ofruesi) dhe një entiteti tjetër që merr ato të dhëna (marrësi). Kjo kontratë përcakton edhe përmbajtjen që marrësi (thirrja) i kërkon ofruesit (përgjigja). Me fjalë të tjera, nëse qëllimi ynë është që të marrim të dhëna ose të ekzekutojmë një funksion përmes një sitemi kompjuterik, një API kontratë duhet të pranohet nga të dyja anët në mënyrë që palet të kuptohen me njëra tjetrën [15].

API ndryshe mund të cilësohet edhe si ndërmjetësues mes përdoruesëve dhe shërbimeve në web apo sistemeve kompjuterike. Një përparësi tjetër e API është se përdoruesi nuk ka nevojë të kuptoj mënyren se si procesohen dhe përpunohen të dhënat që ata marrin [15].

2.3.2 Open API

Specifikimi OpenAPI (i njohur më parë si Swagger) përcakton një standard për zhvillimin e RESTful API pa marrë parasysh se cilën gjuhë programuese e përdorim. Zhvillimi i Swagger

(tani OpenAPI) filloi në vitin 2010 nga Tony Tam që në atë kohë punonte për kompaninë Wordnik. Përparësia kryesore e një standardi të tillë është se të gjithë shfrytëzuesit e atij shërbimi (zhvilluesit, përdoruesit si dhe mikroshërbimet e tjera) janë në gjendje ta kuptojnë dhe të ndërveprojnë me shërbimin në fjalë. Specifikimi OpenAPI (OAS) gjithashtu ofron vegla për gjenerimin e kodit për klient (front-end) dhe server (back-end), vegla për gjenerimin e dokumentimit si dhe vegla për testim. Një RESTful API i cili zbaton specifikimin e OpenAPI mund të ndërtohet duke përdorur formatin JSON ose YAML [16].

2.3.3 Hydra

Hydra është një standard i cili synon të thejshtoj zhvillimin e API-ve që përfshijnë tekst, media dhe zë, me të cilët mund të ndërveprojnë zhvilluesit, përdoruesit e thjeshtë dhe sistemet komjuterike. Komponentët kryesorë të Hydra janë Hydra Core Vocabulary dhe JSON-LD [17].

Fjalori themelor i Hydra-s (Hydra Core Vocabulary) përcakton një grumbull konceptesh themelore të cilat i tregojnë kompjuterëve se si të ndërveprojnë me një API. Meqenëse të dhënat që ofron API mund të lexohen nga sistemet kompjuterike, me anë të këtij fjalori bëhet i mundur krijimi i aplikacioneve të përgjithshme "front-end" (client) të cilat shfrytëzojnë API-në në fjalë. Fjalori themelor i Hydra-s fuqizohet edhe më tej duke përdorur fragmente të dhënash të lidhura (Linked Data Fragments). Këto fragmente janë një grumbull i përkufizimeve që mundësojnë nxjerrjen në mënyrë efikase të të dhënave nga një API [18].

2.3.4 JSON-LD

JSON-LD është format për strukturimin e të dhënave i cili përdoret për komunikim mes klientëve dhe serverëve (client-server). Duke përcaktuar një grumbull të koncepteve të cilat përdoren zakonisht gjatë zhvillimit të API-ve, JSON-LD mund të përdoret si bazë për ndërtimin e shërbimeve në web të cilat janë të shkallëzueshme dhe lehtë mirëmbahen. Një nga idetë kryesore të cdo formati RDF, duke përfshirë edhe JSON-LD është strukturimi i vlerave sipas formateve të standardizuara siç është schema.org [17].

2.4 JavaScript

JavaScript, e njohur ndryshe edhe si JS është një gjuhë programuese që zbaton standardin e definuar nga ECMA [19]. Së bashku me HTML dhe CSS, JavaScript është një prej tekonologjive kryesore për zhvillimin në web [20]. JavaScript mundëson krijimin e faqeve interaktive dhe është pjesë thelbësore e thuajse secilës faqe në internet. Shumica e faqeve të internetit e përdorin JavaScripit-in për të programuar ndërfaqen (front-end), ose ndryshe pjesën me të cilën përdoruesi ndërvepron drejtpërdrejtë [21]. Të gjithë shfletuesit e internetit bashkëkohor përmbajnë në vete një "motor" të dedikuar për ekzekutimin e kodit të JavaScript-it.

Në vitin 1995, Brendan Eich, një inxhinier i softuerit i cili në atë kohë punonte për kompaninë Netscape Communications Corporation, u ngarkua me zhvillimin e një gjuhe programuese si pjesë e versionit 2.0 Beta të shfletuesit "Navigator". Gjuha programuese në fjalë duhej të ishte e lehtë por njëashtu e fuqishme në mënyrë që të mundësonte programimin e faqeve interaktive. Duke i pasur parasysh këto kufizime, Brendan Eich u inspirua nga sintaktsa e gjuhës programuese Java në mënyrë që progamerët ta kishin më të lehtë kalimin në JavaScript pasi që në atë kohe Java ishte një ndër gjuhët më të përhapura. Ai gjithashtu mori konceptet e programimit të orientuar në objekte nga gjuha programuese Self si dhe pjesë tjera nga C, Smalltalk, Lisp dhe Scheme. Pasi që afati kohor i caktuar nga kompania ishte shumë i shkurtër, ai ndërtoi versionin e parë të JavaScript-it për vetëm 10 ditë [22].

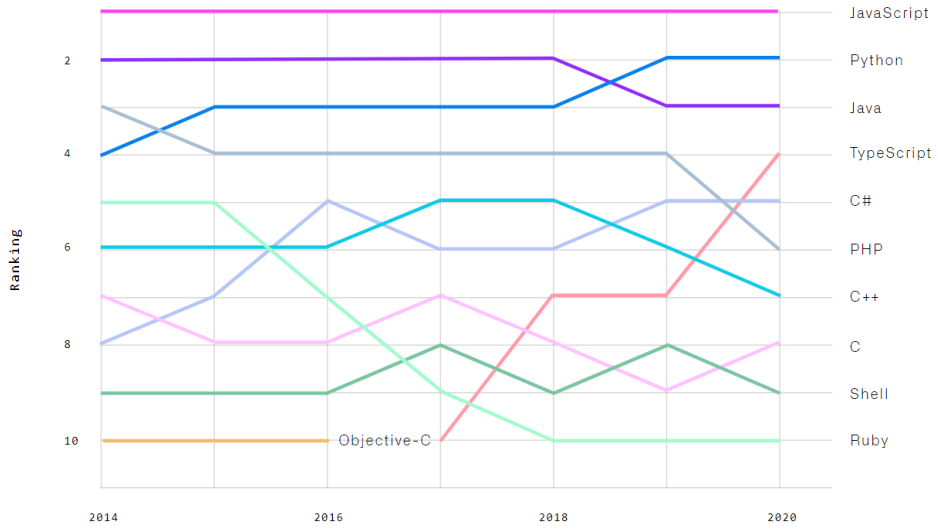

Figura 2. Renditja e gjuhëve programuese sipas përdorimit [18]

Tabela e mësipërme (Figura 2) tregon renditjen e gjuhëve programuese sipas përdorimit, nga viti 2014 deri në fund të vitit 2020. Këtu vrojtojmë se JavaScript mban vendin e parë për vitin e gjashtë me rradhë, duke lënë pas Python dhe Java. Si faktor për rritjen e shpejtë të JavaScript janë edhe "frameworks" të shumta që janë krijuar për të lehtësuar zhvillimin e aplikacioneve. Një "framework" është një platformë për zhvillimin e programeve softuerike e cila ofron një bazë mbi të cilën zhvilluesit ndërtojnë programet softuerike. Një "framework" përmban klasa, funksione dhe vegla të gatshme që mundësojnë kryerjen e punëve të ndryshme në mënyrë që zhvilluesi të mos i krijoj ato vegla nga fillimi sa herë që fillon një aplikacion të ri. Kjo ndikon në përshpejtimin e procesit të zhvillimit të softuerit [23].

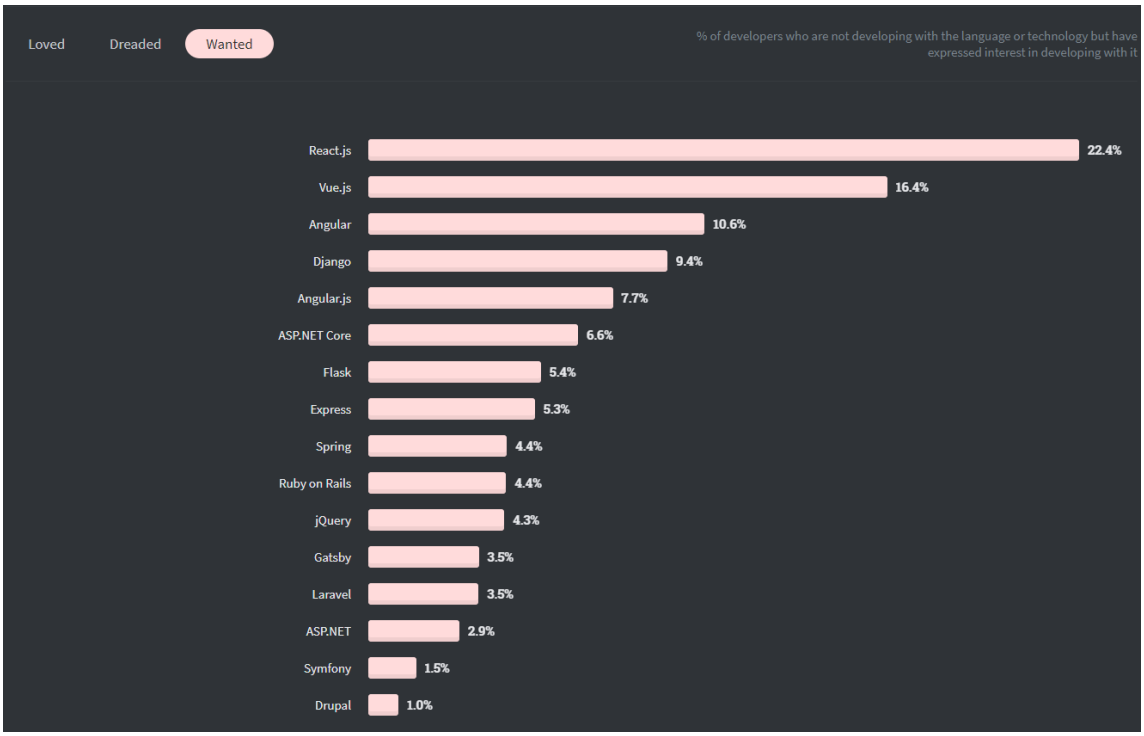

Figura 3. Renditja e frameworks të ndryshme në bazë të kërkesës, sipas sondazhit të StackOverflow [24]

Në vitin 2020, JavaScript ishte gjuha më e përdorur për vitin e tetë me rradhë, sipas sondazhit të StackOverflow. Me përhapjen e JavaScript është rritur edhe numri i "front-end frameworks", prej të cilave më të përhapurat janë: React, Angular, Ember.js, Preact, Svelte dhe Vue.js. Sipas rezultateve të sondazhit të StackOverflow, "framworks" më të përdorur janë React, Angular dhe Vue.js, ku sipas programerëve, React dhe Vue.js janë "frameworks" më të dashura dhe të kërkuara [24]. (Figura 3)

2.4.1 TypeScript

Faktor tejet me ndikim për rritjen e përdorimit të JavaScript në projekte të ndërmarrjeve të mëdha është TypeScript. TypeScript është një gjuhë programuese e ndërtuar mbi bazat e JavaScript-it që zgjeron sintaksën ekzistuese dhe ofron mundësinë e caktimit të tipeve të të dhënave në mënyre statike [25]. E krijuar nga Microsoft, kjo gjuhë është dizajnuar për zhvillimin e softuerëve të mëdhenjë. I gjithë kodi i zhvilluar në TypeScript në fund përpilohet në JavaScript, rrjedhimisht, kjo nënkupton që secili program i zhvilluar në JavaScript është i vlefshëm edhe në TypeScript [26]. Perveç tjerash, kjo gjuhë ofron mbështetje për "definition files" të cilat përmbajnë informata në lidhje me tipet e përdorura në projekt. Kjo karakteristikë është e ngjajshme me "header files" në C dhe C++ të cilat përdoren për të përshkruar strukturën e objektve.

TypeScript është një gjuhë relativisht e re e cila u publikua në Nëntor të vitit 2012, në versionin 0.8. Një ndër përsonat që dha më së shumti kontribut për krijimin e TypeScript është Anders Hejlsberg, arkitekti kryesor i C# si dhe krijuesi i gjuhës Delphi dhe Turbo Pascal. Pak kohë pasi u publikua, Miguel de Icaza, një programer i famshëm Meksikan e cilësoi TypeScript si një gjuhë të shkëlqyer mirëpo në të njejtën kohë kritikoi mbështetjen e pakët që kishin ofruar IDE-t e asaj kohe. Në vitin 2013, shumë IDE dhe përpunues të tekstit filluan të ofrojnë mbështetje për programim me TypeScript. Në vazhdim, versioni 0.8 u pasua nga një version më i ri, versioni 0.9 i cili ofronte mbështetje për "generics". Më pastaj, në vitin 2014, versioni 0.9 u pasua nga versioni 1.0. Në vitin 2014, ekipi i TypeScript publikoi përpiluesin e ri i cili do të përmirësonte përformancen për 5 herë. Në të njejtën kohë, kodi burimor i TypeScript u bart nga CodePlex në Github [27].

Figura 4. Renditja e gjuheve programuese ne bazë të pëlqimit sipas sondazhit të StackOverflow [24]

Nga figura e mësipërme (Figura 4) mund të shohim se TypeScript është një nga gjuhët më të dashura për programeret ku renditet e dyta pas Rust. Ky është një tregues tjetër për ndikimin e TypeScript në rritjen e përdorimit të JavaScript-it dhe "frameworks" të saj.

2.4.2 Përdorimi i React për zhvillim të aplikacioneve

React është një librari e JavaScript-it më kod të hapur, e krijuar për zhvillimin e web faqeve interaktive. React u krijua nga Jordan Walke, një inxhinier i softuerit në Facebook dhe në fillimet e saj u quajt FaxJS [28]. Zhvillimi i React u ndikua nga XHP, një librari me komponentë të HTML për gjuhën programuese PHP. Në vitin 2011 u përdor për të fuqizuar ballinën e Facebook ndërsa në vitin 2012 u përdor për të ndërtuar ndërfaqen e Instagram [29]. React aktualisht mirëmbahet nga Facebook dhe një komunitet i përberë nga zhvillues dhe kompani të ndryshme.

Përmes React, zhvilluesit mund të ndërtojne web aplikacione të mëdha të cilat mund të ndryshojnë ndërfaqen e përdoruesit pa e rifreskuar faqen. Qëllimi kryesor i React është shpejtësia, shkallëzimi dhe thjeshtësia [30]. Në thelb, të gjithë aplikacionet e React janë komponente. Një komponent është një modul i mbyllur i cili si rezultat jep dicka që mund të shfaqet te përdoruesi. Të gjitha këto komponente janë të ripërdorshme. Me anë të React ne mund të ndërtojmë elemente të ndryshme të ndërfaqes, si për shembull një buton ose një fushë për futjen e të dhënave. Komponentët mund të përmbajnë edhe komponentë të tjerë. Thënë gjerësisht, gjatë ndërtimit të aplikacioneve me React, ne ndërtojmë komponente të cilat në vete përmbajnë elemente të ndërfaqes. Këto komponente organizohen në mënyrë hirearkike dhe definojnë strukturën e aplikacionit [31].

Në vazhdim do të shqyrtojmë disa nga karakteristikat e veçanta të React.

2.4.2.1 React Native

React Native është një JavaScript "framework" për krijimin e aplikacioneve mobile për iOS dhe Android. Është e bazuar në React, mirëpo për dallim nga React i cili për qëllim ka ndërtimin e web faqeve interaktive, React Native përqëndrohet në ndërtimin e aplikacioneve mobile. Me fjalë të tjera, zhvilluesit e web-it tani mund të ndërtojnë aplikacione mobile të cilat duken dhe funksionojnë si të ishin ndërtuar në platformat e tyre përkatese, iOS dhe Android. E gjitha kjo duke përdorur JavaScript, një gjuhë shumë të dashur dhe të përdorur nga programerët.

Perveç kësaj, i gjithë kodi që shkruhet në React Native funksionon për të dy platformat, Android dhe iOS, gjë që e bën shumë të lehtë dhe të shpejtë zhvillimin në të dy platformat njëkohësisht [32].

2.4.2.2 Virtual DOM

DOM, ose ndryshe Document Object Model përfaqson ndërfaqen e përdoruesit, të shprehur si një objekt në JavaScript. Sa herë që gjendja e aplikacionit ndryshon, DOM-i përditësohet për të reflektuar atë ndryshim. Rrjedhimisht, sa më shpesh e ndryshojmë gjendjen e aplikacionit aq më e keqe është performanca. Kjo ndodhë për arsye se të gjitha ato ndryshime duhet të procesohen dhe të ri-shfaqen te përdoruesi. Sa më shumë elemente të ndërfaqes gjenden në DOM, aq më i ngadaltë është procesi i përditësimit [33].

Këtu hyn në punë koncepti i DOM-it virtual i cili është është një përfaqsim virtual i DOM-it të vertetë. Sa herë që gjendja e aplikacionit ndryshon, DOM-i virtual përditësohet në vend të DOM-it të vertetë. Pas shtimit të elementeve të reja në ndërfaqen e përdoruesit, krijohet DOM-i virtual. Ky objekt përfaqsohet nga një strukturë e të dhënave të llojit pemë. Secili element i DOM-it të vertetë përfaqësohet nga një nyje në këtë pemë. Nëse gjendja e cilitdo prej këtyre elementeve ndyrshon, një DOM virtual, i përfaqsuar nga një objekt i llojit pemë krijohet. Më pastaj, këto dy pemë krahasohen mes vete për ndryshime. Pasi që procesi i krahasimit ka përfunduar, DOM-i virtual kalkulon mënyrën më efikase për zbatimin e këtyre ndryshimeve në DOM-in e vertetë. Ky proces garanton se një numër minimal i veprimeve bëhet ne DOM-in e vërtetë, kështu duke e përmirësuar përformancën në masë të madhe [33].

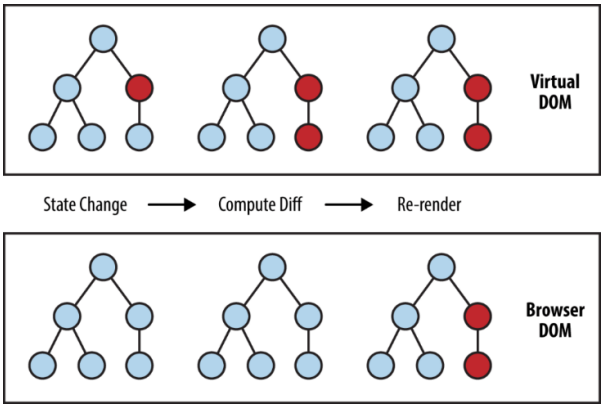

Figura 5. Procesi i përditësimit të ndryshimeve në ndërfaqe duke përdorur DOM-in virtual

Qarqet me ngjyrë të kuqe përfaqësojnë nyjet që kanë ndryshuar. Këto nyje përfaqësojnë elementet e ndërfaqes gjendja e të cilave ka ndryshuar. Këtu bëhet krahasimi për ndryshime mes DOM-it virtual aktual dhe atij paraprak. Nyja e cila ka pësuar ndryshim i bashkangjitet DOM-it të vertetë së bashku me të gjithë nën-nyjet e saj. Në këtë mënyrë përditësohet DOM-i i vertetë duke përdorur DOM-in virtual [33].

2.4.2.3 Thjeshtësia

React është shumë i thjeshtë për t'u kuptuar. Mënyra se si React funksionon me anë të komponentëve, cikli i jetës i definuar mirë, dokumentimi i mirë dhe përdorimi i JavaScript-it të zakonshëm e bën React-in shumë të lehtë për ta mësuar dhe përdorur. React përdor një sintaksë të veçantë të quajtur JSX e cila lejon përzierjen e HTML me JavaScript gjë që e zvoglon barrieren mes logjikës dhe prezantimit. Programeri nuk është i detyruar të përdor JSX gjatë zhvillimit të aplikacioneve me React. Zhvillimi i kodit mund të bëhet edhe pa JSX, mirëpo JSX lehtëson në masë të madhe punën me komponentë [30].

2.4.2.4 I lehtë për t'u mësuar

Secili zhvillues më njohuri bazike të programimit është në gjendje të kuptoj kodin e zhvilluar me React ndërsa "frameworks" si Angular dhe Ember trajtohen si gjuhë programuese të fushave të veçanta, kështu duke lënë të nënkuptohet se janë më të vështira për t'u mësuar në krahasim me React [30].

2.4.2.5 Testueshmëria

Aplikacionet e zhvilluara me React janë relativisht të lehta për t'u testuar. Pjesa e "pamjeve" në React mund të trajtohet si një funksion. Në këtë mënyrë, ne mund të manipulojmë gjendjen e një React komponenti për të parë se si reagon ai komponent ndaj gjendjes së ndryshuar në varësi të të dhënave hyrëse [30]. Egzistojnë disa mënyra për të testuar komponentë në React. Thënë gjerësisht, këto mënyra ndahen në dy kategori kryesore:

- Testimi individual i komponentëve në një ambient testues
- Testimi i tërë aplikacionit në një ambient real që ndryshe njihet edhe si testimi "end-to-end" [34]

2.5 CI/CD dhe veglat për publikim

Integrimi i vazhdueshëm (Continuous Integration – CI) dhe dorëzimi i vazhdueshëm (Continous Delivery – CD) mishërojnë një kulture, një grumbull principesh të veprimit, si dhe një numër të praktikave që i mundësojnë një ekipi zhvillues të dorëzojnë ndryshime të kodit më sigurtë dhe më shpesh. Ky zbatim ndryshe njihet edhe si "CI/CD pipeline". CI/CD është një prej praktikave më të mira që një ekip zhvillues mund të zbatojë. Njëashtu, është një praktike e mirë e metodologjisë Agile për arsye se i mundëson ekipeve të zhvillimit të përqëndrohen në arritjen e afatave, përmirësimin e kualitetit të kodit dhe sigurisë, në vend se të merren me hapat e dorëzimit dhe integrimit pasi që ato tani më janë automatizuar [35].

Në vazhdim, ne do të flasim për proceset e CI/CD si dhe disa nga veglat/platformat e ndërlidhura.

2.5.1 Integrimi i vazhdueshëm (Continous Integration)

Ekipet e zhvillimit që parktikojnë integrimin e vazhdueshëm bëjnë ndryshime të shpeshta në degën kryesore të një projekti. Këto ndryshime, para se t'i bashkangjiten degës kryesore, pra para se të bëhen pjesë e kodit burimor nga i cili ndërtohet softueri për t'u përdorur nga klientet, validohen dhe testohen automatikisht. Duke e bërë këtë, ekipet e zhvillimit shmangin problemet me integrim që mund të ndodhin si pasojë e testimit jo të duhur, mospërputhjes së kodit nga puna e zhvilluesëve të ndryshëm, etj. Integrimi i vazhdueshëm i jep rëndësi të veçantë automatizimit të testeve për t'u siguruar që aplikacioni do të vazhdojë të funksionoj si duhet edhe pas ndryshimeve të reja që i bashkangjiten kodit burimor [36].

2.5.2 Dorëzimi i vazhdueshëm (Continous Delivery)

Dorëzimi i vazhdueshëm është një zgjatim i integrimit të vazhdueshëm për arsye se të gjitha ndryshimet që ndodhin në kod vendosen në një ambient testues ose real, në mënyrë automatike, pas fazës së integrimit të vazhdueshëm. Praktikisht, në mund të fillojmë procesin e lëshimit të versionit të ri të aplikacionit më një klik të një butoni pasi që procesi i integrimit dhe i dorëzimit tashmë është automatizuar. Me anë të dorëzimit të vazhdueshëm, ne mund të lëshojmë versione të reja të aplikacionit në baza ditore, javore, dy-javore ose çfardo periudhe tjetër. Mirëpo,

është praktikë e mirë që ndryshimet të lëshohen sa më shpejtë, në tufa më të vogla, në mënyrë që t'a kemi më të lehtë zgjidhjen e problemve në rast të ndonjë gabimi eventual [36].

2.5.3 Publikimi i vazhdueshëm (Continous Deployment)

Publikimi i vazhdueshëm shkon një hap më para nga dorëzimi i vazhdueshëm. Me këtë hap, çdo ndryshim që kalon të gjitha fazat paraprake lëshohet te klientët. I gjithë ky proces është i automatizuar dhe nuk ka nevojë për ndërhyrjen e njerzëve. Në rast se një ose më shumë teste nuk kanë kaluar, procesi i publikimit dështon. Publikimi i vazhdueshëm është një mënyrë shumë e mirë e përshpejtimit të procesit së marrjes së reagimeve nga klientët (përdoruesit) pasi që çdo ndryshim publikohet për pak minuta [36].

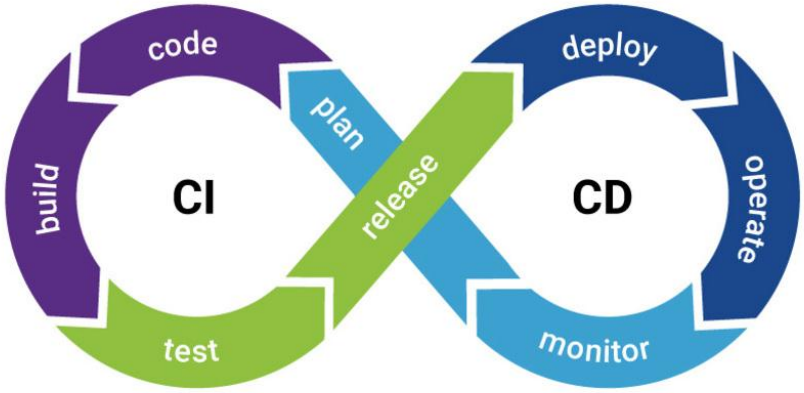

Figura 6. Cikli i CI/CD [37]

Imazhi i mësipërm paraqet ciklin e integrimit dhe dorëzimit të vazhdueshëm. Këtu vërjemë se pjesa e integrimit është përgjegjëse për kompilimin dhe testimin e kodit. Në rast se integrimi është kryer me sukses, pjesa e dorëzimit merret me lëshimin/publikimin e atyre ndryshimeve si dhe është përgjegjëse për vëzhgim të vazhdueshëm për ndonjë gabim eventual.

2.5.4 GitHub Actions

Veprimet (actions) në GitHub janë detyra (tasks) me anë të të cilave ne mund të krijojmë procedura (jobs) që rregullojnë rrjedhën e punës në një projekt. Ne mund të krijojmë detyra (actions), ose të përdorim dhe ndryshojmë detyra (actions) të krijuara nga komuniteti i GitHub. Ne

krijojmë detyra (actions) duke shkruar kod i cili ndërvepron me kodin burimor (repository) në çfardo lloj mënyre që na nevojitet. Këtu përfshihen edhe API-t publike të shërbimeve të palëve të treta. Për shembull, një detyrë (action) mund të publikoj module të NPM, të dërgoj SMS kur ndodhin ngjarje të caktuara, të publikoj versionin e ri të softuerit, etj. Detyrat (actions) mund të startohen edhe në kontenirë të Docker-it [38].

2.5.5 Docker

Docker është një produkt i tipit PaaS (platform as a service) i cili përdor virtualizimin në nivel të sistemit operativ për të krijuar mjedise për zhvillim të softuerit të quajtura kontenierë (containers). Kontenierët janë të izoluar nga njëri tjetri, kanë softuerin e vetë, si dhe librari dhe konfigurime të veçanta. Komunikimi mes kontenierëve bëhet përmes kanaleve të veçanta [39]. Për dallim nga makinat virtuale, Docker shfrytëzon më pak resurse pasi që të gjithë kontenierët shfrytëzojnë kernelin e sistemit operativ [40].

Ndërtimi i aplikacioneve sot do të thotë më shumë se vetëm zhvillimi i kodit burimor. Shumë gjuhë programuese, "frameworks", arkitektura, dhe ndërfaqe të ndryshme mes veglave për secilen fazë të ciklit të jetës së softuerit, krijonë ndërlikime të shumta. Docker thjeshton dhe përshpejton rrjedhën e punës, njëkohësisht i mundëson zhvilluesëve përdorimin e veglave, "frameworks", dhe mjediseve të ndryshme në bazë të nevojave për secilin projekt [41].

Docker përshtatet mirë me proceset e Integrimit dhe Dorëzimit të vazhdueshëm. Me anë të GitHub actions ne mund të krijojmë procedura për ndërtimin (build) e aplikacionit sa herë që bëhen ndryshime në kod. Ky proces krijon një imazh të ri në Docker i cili është i çasshëm në atë mjedis. Me anë të imazheve të Docker-it ekipet mund të ndërtojnë, shpërndajnë, dhe dorëzojnë aplikacionin e tyre në mënyre efikase [42].

2.6 Arkitektura e softuerit

Arkitektura softuerike e një sistemi përshkruan organzimin ose strukturen e një sistemi, dhe tregon se si sillet ai sistem. Një sistem paraqet një grumbull të komponentëve që kryejnë një funksion ose një grumbull funksionesh. Me fjalë të tjera, arkitektura softuerike e një sistemi ofron një themel mbi të cilën softueri mund të ndërtohet. Një varg i shqyrtimeve dhe i vendimeve ndikojnë drejtpërdrejtë në kualitet, performancë, mirëmbajtje, dhe në vetë funksionimin e duhur të sistemit. Nëse nuk marrim parasysh problemet e zakonshme që mund të ndodhin dhe pasojat afatgjate që ato probleme sjellin, ne vëmë në rrezik vetë sistemin [43].

Ekzistojnë një numër i modeleve dhe principeve të arkitekturës të cilat shpeshë përdoren në sisteme moderne. Këtyre iu referohemi si stile të arkitekturës. Një sistem nuk kufizohet në një arkitekturë të vetme. Zakonisht, një kombinim i modeleve përdoret për të ndërtuar një sistem bashkëkohor [43].

2.6.1 Arkitektura Jamstack

Arkitektura Jamstack është krijuar për t'a bërë web-in më të shpejtë, më të sigurtë, dhe më të lehtë për t'u shkallëzuar. Jamstack është ndërtuar mbi shumë vegla dhe rrjedha të punës të cilat zhvilluest i pëlqejnë dhe të cilat ofrojnë një nivel të lartë të produktivitetit. Principet themelore të "pre-rendering" dhe shkëputjes mes moduleve mes vete (decoupling), mundësojnë që aplikacionet të dorëzohen në mënyrë efikase dhe të qëndrueshme. Ky koncept është i bazuar në arkitekturën me të cilën një webfaqe mund të shërbehet te klienti në mënyrë statike, duke ofruar HTML statik, ku përmbajtja mund të ndryshohet me anë të veglave të ndryshme në JavaScript. Ndryshe, fjala JAM do të thotë "Javascript APIs Markup" [44].

Me Jamstack, i gjithë "front-end"-i është i para-ndërtuar (pre-built), i shprehur në formë të faqeve statike dhe aseteve. Si rezultat i këtij procesi, këto faqe mund të shërbehen nga një CDN (Content Delivery Network), kështu duke ulur koston, kompleksitetin dhe rrezikun që serverët tradicional sjellin. Vegla si Gatsby, Hugo, Jekyll, Eleventy, NextJS dhe shumë tjera janë krijuar për këtë qëllim [44].

2.6.1.1 Siguria

Jamstack heq nga përdorimi shumë pjesë lëvizëse dhe serverë nga infrastruktura e aplikacionit. Kjo do të thotë se mundësia për sulme është më e vogel. Përdorimi i faqeve statike dhe aseteve, si pjesë e një strukture të para-ndërtuar mundëson që skedarët në sistem të jenë "read-only" (vetëm të lexueshme) duke zvogluar edhe më tej mundësitë për sulme [44].

2.6.1.2 Shkallëzueshmëria

Modelet e arkitekturave softuerike e trajtojnë trafikun e ngarkuar në server duke vendosur logjikë shtesë e cila vë në përdorim shtresa për "caching", për resurset dhe faqet më të vizituara. Me Jamstack, nuk ka nevojë të zbatohet logjikë shtesë pasi që këto faqe mund të sherbehen nga CDN [44].

2.6.1.3 Performanca

Shpejtësia më të cilën hapen faqet ka ndikim të drejtpërdrejtë në eksperiencën e përdoruesit (user experience). Për shkak që faqet e aplikacioneve të ndërtuara me Jamstack para-ndërtohen, nënkupton që shpejtësia e hapjes së faqeve është shumë e madhe. Pasi që të gjitha këto faqe janë të çasshme në një CDN ne nuk kemi nevojë të shpenzojmë shume kohë dhe para për infrastrukturë shtëse meqenëse performanca garantohet nga ai CDN [44].

2.6.1.4 Mirëmbajtja

Pasi që i gjithë aplikacioni është i natyrës statike, nevoja për mirëmbatje është shumë e vogël. Puna më e mundimshme bëhet gjatë gjenerimit të faqeve statike. Pasi që i gjithë aplikacioni vendoset në një CDN, ne nuk kemi nevojë për serverë të cilet duhen mirëmbajtur dhe përditësuar [44].

2.6.1.5 Eskperienca e zhvilluesit

Jamstack mund të ndërtohet me vegla të ndryshme dhe nuk varet nga teknologji ose "framework" pak të njohura, kështu duke krijuar një ambient atraktiv për ekipet e zhvillimit të softuerit [44].

3. DEKLARIMI I PROBLEMIT

Në një sondazh të bërë në vitin 2013 nga Innotas, një kompani për menaxhimin e portfolios në cloud, tregoi se 50 përqind e bizneseve të cilat morrën pjesë në sondazh, kanë pasur të paktën një projekt që ka dështuar brenda 12 muajve të kaluar. Tre vite më vonë ky numër ishte rritur; sondazhi i bërë nga po e njejta kompani në të cilën morrën pjesë 162 profesionistë të fushës së tekonologjisë informative, nga muaji Janar deri në Mars të 2015, tregoi se 55 përqind e pjesëmarrësve kanë qenë pjesë e një projekti i cili kishte dështuar [45].

Mos implementimi, implementimi i pjeshëm ose jo i saktë i praktikave të menaxhimit të projektit dhe Ciklit të Jetës së Zhvillimit të Softuerit gjatë zhvillimit të projekteve softuerike shkakton pasoja. Duke marrë parasysh që sistemet softuerike janë bërë pjesë thelbësore e jetës dhe ekonomisë tonë, sa herë që një projekt softuerik dështon dëmet janë të mëdha, e sidomos ato ekonomike. Më të prekurit në këtë mes janë kompanitë e zhvillimit të softuerit, palët e interesit dhe personat e prekur nga sistemet në fjalë.

Pjesa praktike e këtij hulumtimi do të zhvillohet në formën e një softueri duke përodorur arkitekturë dhe teknologji moderne. Disa nga gjuhët dhe veglat që do të përdoren janë: API Platform, React, Open API, Docker, Git etj.

Disa nga pyetjet të cilat do të gjejnë përgjigje në këtë punim janë:

1. Si të mbledhim kërkesat?
2. Cfarë arkitekture duhet përdorur për zhvillimin e një softueri modern i cili lehtë mund të zgjerohet?
3. Si bëhet dokumentimi i një softueri?
4. Si të ndërtojmë një ambient për zhvillimin e softuerit i cili është lehtë i çasshëm?
5. Si të përdorim vegla dhe metoda të ndryshme për organizimin e punës në ekip?

Qëllimi i këtij projekti është që të trajtoj dhe të shtjelloj implementimin e saktë të Ciklit të Jetës së Zhvillimit të Softuerit duke tentuar të rris shkallën e suksesit të produkteve soufterike. Ky punim bazohet në hulumtim të aplikuar i cili përdor metoda shpjeguese për të shtjelluar temën në fjalë.

4. METODOLOGJIA

Për shkak të natyrës së temës të këtij punimi, pjesa më e madhe e hulumtimit është bazuar në internet. Hulumtimi në internet është një metodë e hulumtimit e cila përfshinë mbledhjen e informacionit nga interneti. Me zhvillimin e shpejtë të internetit, teknikat tradicionale të hulumtimeve po gjejnë më pak zbatim, kështu duke i dhënë përparësi hulumtimit në internet. Sondazhet në internet kanë më shumë ndikimin se sa ato tradicionale për arsye të kostos më të vogël dhe lehtësisë së çasjes që i ofrohet të anketuarëve. Shkalla e përgjigjeve në hulumtimet që bëhen në internet është shumë më e madhe se sa ato tradicionale për shkak se të anketuarit janë të sigurt se identiteti i tyre do të mbrohet [46].

Më saktësisht, në këtë punim është përdorur Metodologjia Kualitative e bazuar në Internet. Artikuj, hulumtime, raporte, eksperimente dhe gjetje të ndryshme nga kompani dhe profesionistë të fushës janë përdorur si të dhëna për paraqitjen, shtjellimin dhe eksplorimin e temës. Përveç hulumtimit në internet, një sistem softuerik është zhvilluar si eskperiment praktik i cili implementon format, parktikat, metodologjitë dhe gjetjet tjera për zhvillimin adekuat të një sistemi softuerik. Ky softuer shërben si mbështetje praktike për gjetjet e këtij hulumtimi dhe mund të konsiderohet rezultat i këtij hulumtimi.

Softueri në fjalë është ndërtuar duke përcjellur metodologjitë dhe arkitëkturën e shtjelluar në këtë punim. Në vijim janë listuar arkitektura, gjuhët, veglat dhe teknologjitë përdorura për këtë shembull.

- **Gjuhë:** PHP, JavaScript
- **Ariktektura:** Jamstack, RESTful
- **“Frameworks”:** API Platform, Symfony, React, React Admin
- **Specifikime/Standarde:** JSON-LD, Hydra, HAL, Swagger/OpenAPI
- **Formate:** YAML, JSON
- **Kontroll i versionit:** Git, GitHub
- **CI/CD:** GitHub Actions, Docker
- **Të tjera:** DigitalOcean Cloud (Ubuntu 20.04 LTS), Nginx, Node, NPM, Postman, JSON Web Tokens

5. REZULTATET

5.1 Ndikimi i SDLC-së në një projekt softuerik

Nga hulumtimet e bëra në këtë punim, duke marrë parasysh një numër të madh të raporteve mbi shkallën e suksesit së projekteve softuerike, del se shumica e projekteve softuerike dështojnë për shkak të implementimit jo të saktë të SDLC. Sipas hulumtimeve, për nga aspekti teknik, fazat më kritike të kësaj metodologjie janë faza e planifkimit dhe caktimit të kërkesave. Ndryshimi i fushëveprimit të projektit, mos përcaktimi i saktë i kërkesave (duke mos iu përshtatur interesave të klientit) ose caktimi i kërkesave në mënyrë të pjesshme duke lënë vend për interpretim, shkakton probleme zinxhirore në fazat tjera. Në faza më të vonshme të projektit, të gjithë modulet e zhvilluara konform kërkesave jo të sakta, pas rishikimit nga palët e interesit duhet të përmirësohen ose në rastin më të keq të ribëhen. Ky problem drejtpërdrejt reflekton në tejkalim të buxhetit dhe afateve kohore, kështu duke çuar projektin drejt dështimit.

5.2 Përdorimi i API Platform për ndërtimin e një produkti softuerik

Për fazën e implementimit të SDLC, unë kam zgjedhur API Platform si bazë për ndërtimin e këtij softueri. Gjatë zhvillimit të softuerit kam hasur gjetje që mund të konsiderohen si përparësi si dhe në gjetje tjera që mund të konsiderohen si mangësi.

Disa nga përparësitë:

- Shkurton kohën e zhvillimit të softuerit
- Dokumentim i mirë
- Gjenerimi automatik i panelit të administrimit, dokumentimit dhe ndërfaqes së klientit
- Përdorimi i standardeve më të fundit të WEB
- Ofron vegla të ndryshme për lëshim/publikim (CI/CD) të lehtë
- Përdor në vete një "framework" të dëshmuar, Symfony

Disa nga mangësitë:

- Relativisht e re në treg, numër jo i madh i zhvilluesëve dhe i kompanive që e përkrahin
- Vështirë për t'u mësuar
- Krijimi i funksionalitetit të veçantë (custom) i cili del nga rrjedha e "framework" pak i vështirë

```
/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 * @ApiResource(
 *     normalizationContext={"groups"={"read"}},
 *     denormalizationContext={"groups"={"write"}},
 * )
 * @ApiFilter(SearchFilter::class, properties={"id": "exact", "firstName": "parti
al", "lastName": "partial", "email": "partial"})
 * @ApiFilter(OrderFilter::class, properties={"id"})
 * @UniqueEntity("email")
 */
class User implements UserInterface
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     * @Groups({"read", "write"})
     * @Assert\NotBlank
     */
    private $firstName;

    /**
     * @ORM\Column(type="string", length=255)
     * @Groups({"read", "write"})
     * @Assert\NotBlank
     */
    private $lastName;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     * @Groups({"read", "write"})
     * @Assert\NotBlank
     */
    private $email;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];
```

Figura 7. Një pjesë e shkëputur nga klasa User

Për dallim nga "frameworks" tradicional ku çdo hap duhet bërë manualisht, API Platform me anë të "Annotations" thjeshton skajshmërisht procesin e krijimit dhe integrimit të një entiteti të ri në aplikacion. Për ndërtimin e një entiteti si RESTful API, i cili përmban të gjithë operacionet CRUD, e gjitha çfarë duhet bëjmë është të krijojmë një klasë të re dhe të përdorim "Annotations" për të përshkruar karakteristikat dhe sjelljen e atij entiteti. API Platforma ofron një funksionalitet jashtëzakonisht të mirë për krijimin dhe përshtatjen e sjelljes së këtyre entiteve. Me anë të "Annotations" në mund të kontrollojmë të gjitha aspektet e një entieti duke filluar nga lidhja me bazën e të dhënave dhe konfigurimeve përcjellëse (@ORM), caktimin e çasjes në nivel të fushave

ose në nivel entieti me anë të grupeve (@Groups), validimi i fushave (@Assert), krijimi i filterave për kërkim dhe sortim (@ApiFilter) si dhe shumë aspekte të tjera. (Figura 7)

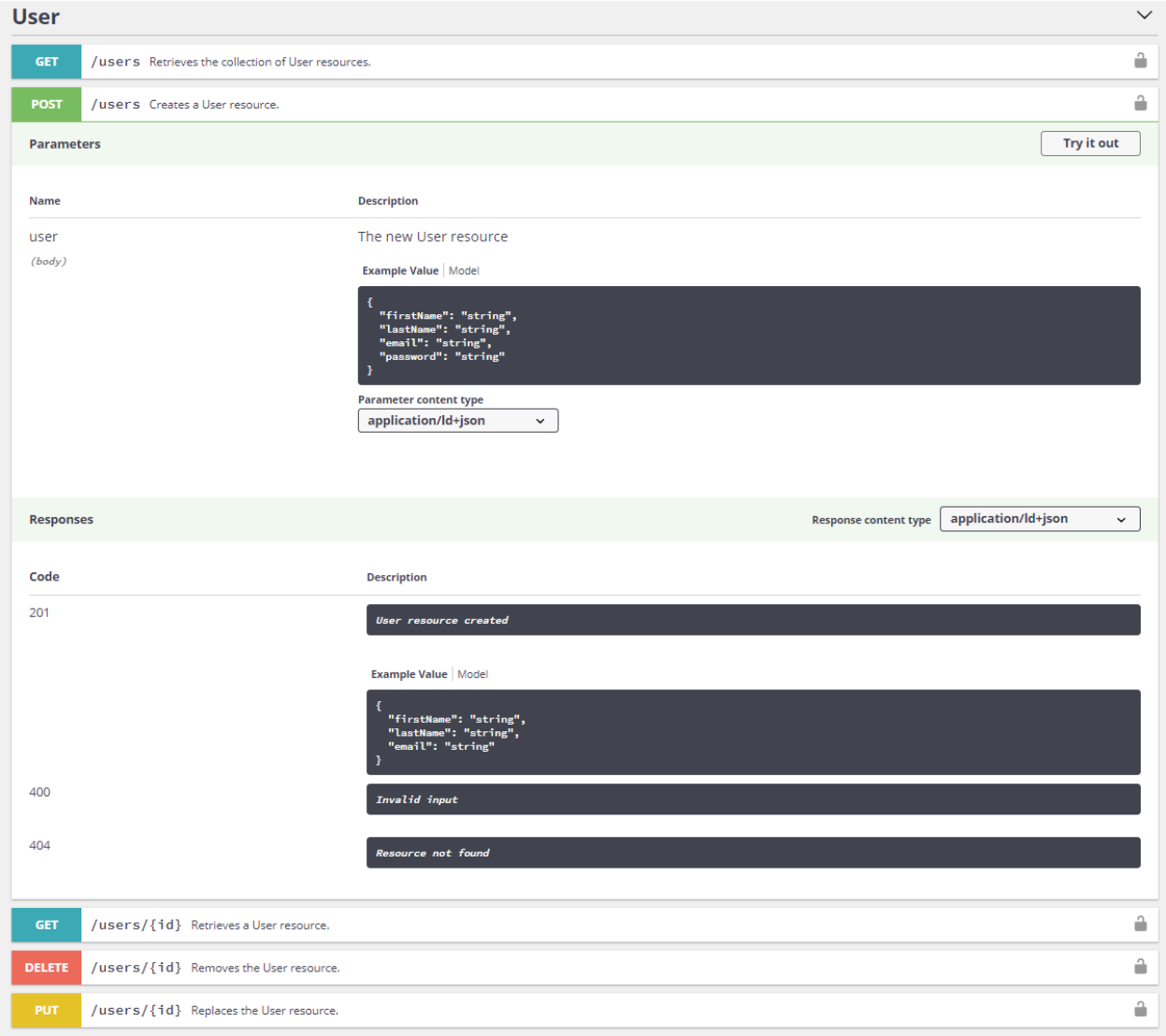

Figura 8. Dokumentimi i OpenAPI për entitetin User i gjeneruar nga API Platform

Dokumentimi dhe mirëmbajtja e një projekti softuerik është një hap shumë i rëndësishëm. Në mënyrë që një RESTful API të përdoret dhe të funksionalizohet, konsumuesi (klienti) duhet të dijë së cilët "endpoints" janë të çasshëm, cilat operacione janë të lejuara dhe cilat janë vlerat hyrëse dhe dalësë që nevojiten. API Platforma gjeneron dokumentimin e komplet REST API aplikacionit në formation e OpenAPI (Swagger). Këtë e bënë duke e lexuar kodin dhe direktivat (annotations) që zhvilluesit i deklarojnë gjatë ndërtimit të një klase. Dokumentimi gjenerohet në mënyrë

automatike (edhe futja e “endpoints” në mënyrë manual është e mundshme) dhë përditësimet ndodhin në kohë reale. Përveç kësaj, në mundë të përdorim ndërfaqen e dokumentimit për të testuar CRUD operactionet e ndryshme në të gjitha entitetet që kemi krijuar. (Figura 8)

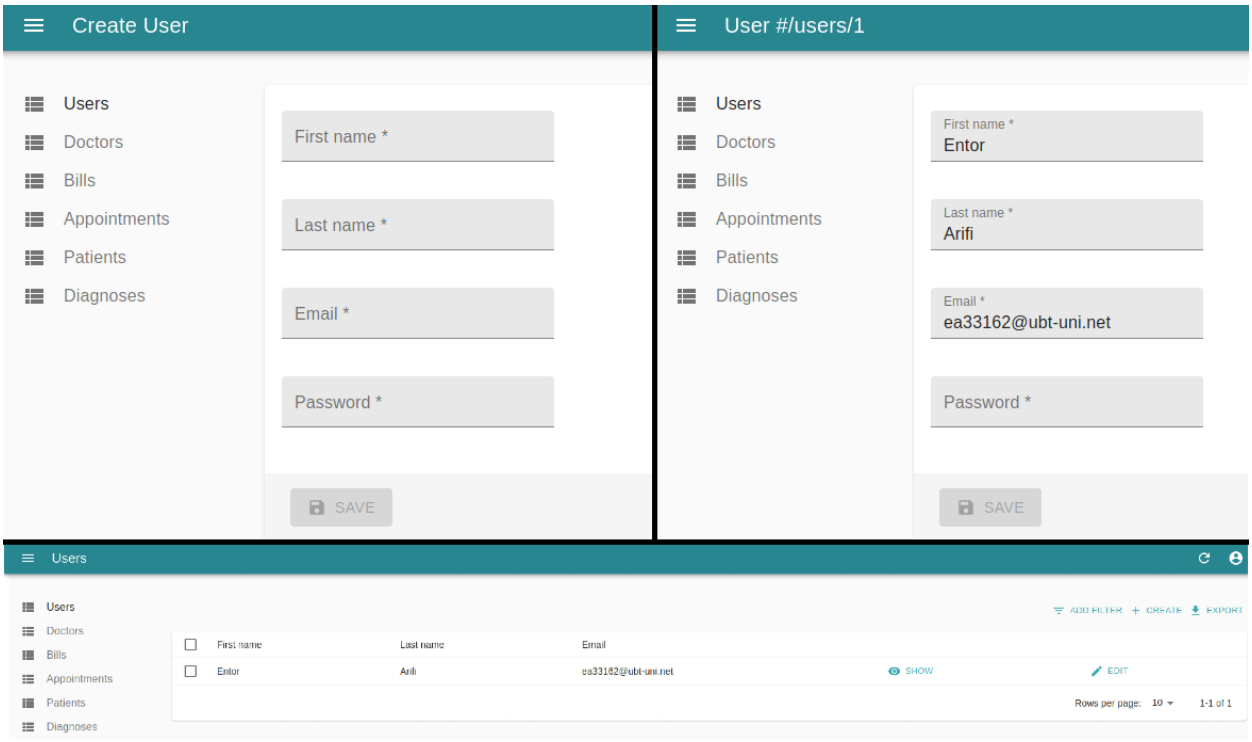

Figura 9. Operacionet mbi entitetin User të pasqyruara në panelin e administrimit

Pasi që kemi ndërtuar entitetin si një klasë në PHP, ndërtimi i ndërfaqes për menaxhimin e këtij entieti nuk kërkon punë shtesë. API Platform ofron komponentin e administrimit, i ndërtuar përmes React Admin i cili reflekton të gjitha ndryshimet që janë deklaruar në klasën përkatëse. Të gjitha fushat, lidhjet me enitetet tjera, validimiet, filtrimet, sortimet dhe konfigurimet e tjera automatikisht shfaqen në ndërfaqen e administrimit. E gjitha kjo mundësohet përmes standardit Hydra që API Platform gjeneron. Nëse është e nevojshme, paneli i administrimit mund të ndryshohet sipas nevojës. (Figura 9)

```
/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20190922171031 extends AbstractMigration
{
    public function getDescription() : string
    {
        return '';
    }

    public function up(Schema $schema) : void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() !== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('CREATE TABLE user (id INT AUTO_INCREMENT NOT NULL, email VARCHAR(180) NOT NULL, roles JSON NOT NULL, password VARCHAR(255) NOT NULL, UNIQUE INDEX UNIQ_8D93D649E7927C74 (email), PRIMARY KEY(id)) DEFAULT CHARACTER SET UTF8 COLLATE UTF8_unicode_ci ENGINE = InnoDB');
    }

    public function down(Schema $schema) : void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() !== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE user');
    }
}
```

Figura 10. Migration i gjeneruar në mënyrë automatike për entitetin User

Transformimi i një entiteti i shprehur si klasë (PHP) në instruksione që një DBMS i kupton (SQL) bëhet përmes “migrations”. DoctrineBundle është një modul i Symfony, rrjedhimisht edhe i API Platform, i cili mundëson përkthimin e një entiteti në tabela dhe relacione, në bazën e të dhënave. I gjithë kodi në entitetin në fjalë përkthehet në një “migration file” i shprehur përmes një klase PHP që deklaron instruksione në SQL se si duhet të krijohet tabela në fjalë. Ky modul është shumë i vlefshem pasi që mundëson verzionimin e tabelave dhe fushave në bazën e të dhënave. Në rast se diçka shkon keq pasi që kemi publikuar versionin e ri të softuerit, në mund të përdorim “migration rollback” për të kthyer gjendjen e bazës së të dhënave në versionin paraprak. (Figura 10)

```
name: CI

on:
  push:
    branches: [ release ]
  workflow_dispatch:

jobs:
  build_admin:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@master

      - name: Archive the admin component
        run: |
          cd admin &&
          npm install &&
          npm run build &&
          zip -r build.zip build

      - name: Copy the admin component to server
        uses: appleboy/scp-action@master
        with:
          host: ${{ secrets.SSH_SERVER }}
          username: ${{ secrets.SSH_USERNAME }}
          key: ${{ secrets.SSH_KEY }}
          source: "./admin/build.zip"
          target: "~/var/www/clinic-management-system/admin/build.zip"

      - name: Deploy changes to server
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.SSH_SERVER }}
          username: ${{ secrets.SSH_USERNAME }}
          key: ${{ secrets.SSH_KEY }}
          script: |
            cd /var/www/clinic-management-system &&
            git clean -fd &&
            git reset origin/master --hard &&
            git fetch &&
            git pull &&
            cd api &&
            composer install &&
            php bin/console doctrine:migrations:migrate --no-interaction --
allow-no-migration &&
            php bin/console c:c &&
            cd ../admin &&
            rm -rf build &&
            unzip build.zip
```

Figura 11. Procesi i CI/CD i shprehur në formatin YAML

Pasi që grumbulli i ndryshimeve është testuar nga ekipi për garantimin e kualitetit, është koha që këto ndryshime të jenë të çasshme për përdoruesit. Rëndom, për të kryer këtë proces, një varg i hapave të njëpasnjëshëm duhet të ekzekutohen me rradhë për të garantuar integrimin e kodit të ri me atë ekzistues. GitHub, si dhe shumica e operatorëve të ngjajshëm (BitBucket, GitLab, etj.) ofrojnë vegla të veçanta të cilat mundësojnë që këta hapa të ekzekutohen automatikisht sa herë që një ngjarje (push, pull request, etj.) ndodhë në depon e kodit (repository). Me anë të GitHub Actions ne mund të caktojmë hapat e nevojshëm për integrimin dhe publikimin e këtyre ndryshimeve në ambiente testuese dhe/ose reale (production). Figura e mësiperme (Figura 11) paraqet një varg hapash të shprehur në formë të komandave të cilat përdoren për të publikuar versionin e ri të softuerit sa herë që ka ndryshime në degën *release*. Ky proces ndryshe njihet si CI/CD.

6. DISKUTIME DHE PËRFUNDIME

Qëllimi i këtij hulimtimi ka qenë shqyrtimi i Ciklit të Jetës së Zhvillimit të Softuerit, implementimi i saktë i saj dhe ndikimi që ka mbi procesin e ndërtimit të një sistemi softuerik. Si pjesë e këtij hulutimi, për të plotësuar pjesën teorike, unë kam ndërduar një softuer prototip për menaxhimin e klinikave.

Gjatë këtij studimi unë kam rënë në përfundim se ndikimi që Cikli i Jetës së Zhvillimit të Softuerit ka mbi produkte softuerike është shumë i madh. Të gjitha fazat janë të rëndësishme, mirëpo është shumë e rëndësishme që planifikimi, caktimi i fushëveprimit, caktimi i kërkesave dhe përzgjedhja e teknologjive të bëhet konform natyrës së projektit, palëve të interesit dhe përbërjes së ekipit zhvilues, respektivisht. Caktimi i metodologjisë (Agile, Waterfall, Prototype, etj.) njëashtu duhet bërë konform këtyre parametrave.

Përdorimi i API platform për produkte softuerike moderne del të ketë qenë një zgjidhje e mirë. Shfrytëzimi i arkitekturave, standardeve, gjuhëve, veglave dhe komunikimit modern është një përparësi që API platform ka ndaj platformave tjera të ngjashme. Nëse këto standarde implementohen saktë, koha e zhvillimit të një softueri zvogëlohet në masë të madhe pasi që punët e zakonshme siq janë krijimi i CRUD operacionve, krijimi i admin panelit për menaxhimin e entiteve, krijimi dhe mirëmbajtja e dokumentimit kryhen në mënyre automatike nga vetë "framework-u". Përveç këtyre, API Platform së bashku me Symfony ofrojnë një mori të moduleve dhe veglave tjera të hapura për përdorim.

Unë rekomandoj që të gjitha ekipet zhvilluese që si për qëllim kanë të hedhin në treg një produkt softuerik të sukseshem i cili i përmbahet limiteve buxhetore dhe kohore, t'i kushtojnë rëndësi të madhe implementimit të saktë të SDLC-së.

7. REFERENCAT

[1] Tara, "Software Development Life Cycle (SDLC): The Guide," 16 April 2020. [Online]. Available: https://blog.tara.ai/software-development-life-cycle/.

[2] The Scalers, "Four Major Reasons Why Software Projects Fail," [Online]. Available: https://thescalers.com/reasons-software-projects-fail/. [Accessed 10 02 2021].

[3] L. V. Knight, T. A. Setinbach and V. Kellen, System Development Methodologies for Web-Enabled E-Business: A Customization Framework.

[4] D. Howe, "Systems Development Life Cycle," FOLDOC, 12 11 2013. [Online]. Available: https://foldoc.org/Systems+Development+Life+Cycle.

[5] R. Half, "6 Basic SDLC Methodologies: Which One is Best?," 24 05 2019. [Online]. Available: https://www.roberthalf.com/blog/salaries-and-skills/6-basic-sdlc-methodologies-which-one-is-best.

[6] GoodFirms, "What is SDLC?," 12 2020. [Online]. Available: https://www.goodfirms.co/glossary/sdlc/.

[7] G. Jetvic, "What is SDLC?," Phoenixnap, 19 05 2019. [Online]. Available: https://phoenixnap.com/blog/software-development-life-cycle.

[8] D. Swersky, "The SDLC: 7 phases, popular models, benefits & more," Raygun, 31 05 2018. [Online]. Available: https://raygun.com/blog/software-development-life-cycle/.

[9] Guru99, "SDLC: Phases & Models of Software Development Life Cycle," 12 2020. [Online]. Available: https://www.guru99.com/software-development-life-cycle-tutorial.html.

[10] Tutorialspoint, "SDLC Overview," 12 2020. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm.

[11] N. Sethi, "Software Development Life Cycle (SDLC) – Importance, Various Phases & Explanation," 12 2020. [Online]. Available: https://electricalfundablog.com/software-development-life-cycle-sdlc/.

[12] RedHat, "What is deployment automation?," [Online]. Available: https://www.redhat.com/en/topics/automation/what-is-deployment-automation. [Accessed 28 01 2020].

[13] ProductPlan, "What is the Software Development Life Cycle.," 2020. [Online]. Available: https://www.productplan.com/learn/software-development-lifecycle/.

[14] K. Dunglas, "API Platform," 28 01 2021. [Online]. Available: https://api-platform.com/.

[15] RedHat, "What is a REST API?," [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Accessed 28 01 2020].

[16] A. Lavasani, "RESTful APIs: Tutorial of OpenAPI Specification," [Online]. Available: https://medium.com/@amirm.lavasani/restful-apis-tutorial-of-openapi-specification-eeada0e3901d. [Accessed 28 01 2020].

[17] M. Lanthaler, "Hydra: Hypermedia-Driven Web APIs," [Online]. Available: https://www.markus-lanthaler.com/hydra/. [Accessed 28 01 2020].

[18] HydraCG, "Hydra: Hypermedia-Driven Web APIs," 16 07 2014. [Online]. Available: https://github.com/HydraCG/Specifications.

[19] ECMA International, "ECMAScript® 2021 Language Specification," 21 01 2021. [Online]. Available: https://tc39.es/ecma262/#sec-overview.

[20] D. Flanagan, JavaScript - The definitive guide (6 ed.), O'REILLY, 2011.

[21] W3Techs, "Usage statistics of JavaScript as client-side programming language on websites," [Online]. Available: https://w3techs.com/technologies/details/cp-javascript/.

[22] R. D. Caballar, "Behind the Rise of JavaScript Front-end Frameworks," 05 05 2020. [Online]. Available: https://www.welcometothejungle.com/en/articles/btc-javascript-frontend-frameworks.

[23] TechTerms, "Framework," 07 03 2013. [Online]. Available: https://techterms.com/definition/framework.

[24] StackOverflow, "2020 Developer Survey," 12 2020. [Online]. Available: https://insights.stackoverflow.com/survey/2020#most-popular-technologies.

[25] Microsoft, "TypeScript: Typed JavaScript at Any Scale," 12 2020. [Online]. Available: https://www.typescriptlang.org/.

[26] P. Bright, "Microsoft TypeScript: the JavaScript we need, or a solution looking for a problem?," 3 10 2012. [Online]. Available: https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/.

[27] Cleverism, "TypeScript," [Online]. Available: https://www.cleverism.com/skills-and-tools/typescript/. [Accessed 2021 01 25].

[28] J. Walke, "FaxJS," [Online]. Available: https://github.com/jordwalke/FaxJs. [Accessed 28 01 2021].

[29] F. Hámori, "The History of React.js on a Timeline," [Online]. Available: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/. [Accessed 28 01 2021].

[30] N. Pandit, "What and Why React.js," 23 01 2021. [Online]. Available: https://www.c-sharpcorner.com/article/what-and-why-reactjs/.

[31] A. Lerner, "What is React?," FullStack React, [Online]. Available: https://www.newline.co/fullstack-react/30-days-of-react/day-1/. [Accessed 28 01 2021].

[32] O'Reilly, "Chapter 1. What Is React Native?," [Online]. Available: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html. [Accessed 28 01 2021].

[33] Ravichandran, Adhithi, "React Virtual DOM Explained in Simple English," 14 06 2019. [Online]. Available: https://adhithiravi.medium.com/react-virtual-dom-explained-in-simple-english-fc2d0b277bc5.

[34] Facebook Inc., "Testing Overview," 28 01 2020. [Online]. Available: https://reactjs.org/docs/testing.html.

[35] I. Sacolick, "What is CI/CD? Continuous integration and continuous delivery explained," 17 01 2020. [Online]. Available: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html.

[36] S. Pittet, «What are the differences between CI, CD, and CD?,» [Në linjë]. Available: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment. [Qasja 28 01 2021].

[37] S. Swaminathan, "23 CI/CD Tools," 20 11 2020. [Online]. Available: https://journeyofquality.com/2020/11/20/23-ci-cd-tools/.

[38] GitHub, "About actions," [Online]. Available: https://docs.github.com/en/actions/creating-actions/about-actions. [Accessed 28 01 2020].

[39] Docker, "What is a Container?," [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 28 01 2021].

[40] Schlosser, Hartmut, "Docker vs Virtual Machine: Where are the differences?," [Online]. Available: https://devopscon.io/blog/docker/docker-vs-virtual-machine-where-are-the-differences/. [Accessed 28 01 2021].

[41] Docker, "Why Develop with Docker," [Online]. Available: https://www.docker.com/why-docker. [Accessed 28 01 2021].

[42] Kovuru, Vittal, "How and Why You Need to Use Dockers In CI & CD," 09 05 2020. [Online]. Available: https://www.cigniti.com/blog/need-use-dockers-ci-cd/.

[43] Synopsys, "Software Architecture & Software Security Design," [Online]. Available: https://www.synopsys.com/glossary/what-is-software-architecture.html. [Accessed 28 01 2021].

[44] Jamstack, "Jamstack," [Online]. Available: https://jamstack.org/. [Accessed 12 02 2021].

[45] Florentine, Sharon, "IT project success rates finally improving," 27 02 2017. [Online]. Available: https://www.cio.com/article/3174516/it-project-success-rates-finally-improving.html.

[46] Questionpro, "Online research- Definition, Methods, Types and Execution," [Online]. Available: https://www.questionpro.com/blog/execute-online-research/. [Accessed 13 02 2021].

[47] Arkbauer, "Software development life-cycle (SDLC)," 12 2020. [Online]. Available: https://arkbauer.com/blog/software-development-life-cycle-sdlc/.

[48] GitHub, "The 2020 State of the Octo-Verse," 12 2020. [Online]. Available: https://octoverse.github.com/#top-languages.

[49] S. Matteson, 26 01 2018. [Online]. Available: https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/.