

УНИВЕРЗИТЕТ У БЕОГРАДУ  
МАТЕМАТИЧКИ ФАКУЛТЕТ



Мирослав Ђ. Мишљеновић

АНАЛИЗА СИСТЕМА ЗА НЕПРЕКИДНУ  
ИНТЕГРАЦИЈУ И НЕПРЕКИДНУ  
ИСПОРУКУ

мастер рад

Београд, 2021.

**Ментор:**

др Владимир ФИЛИПОВИЋ, редовни професор  
Универзитет у Београду, Математички факултет

**Чланови комисије:**

др Саша МАЛКОВ, ванредни професор  
Универзитет у Београду, Математички факултет

др Александар КАРТЕЉ, доцент  
Универзитет у Београду, Математички факултет

**Датум одбране:** септембар 2021.

*Велику захвалност гућујем ментору,  
професору др Владимиру Филиповићу,  
на мноћобројним корисним саветима.*

*Захваљујем се и члановима комисије  
професору др Саши Малкову  
и доценту др Александру Картељу  
који су допринели да рад буде квалитетнији.*

*Такође, захваљујем се колеги Младену Ристићу  
који је ставио на располагање продукциони сервер  
и тиме омоћућио да се овај рад заврши.*

*Овај рад посвећујем породици  
за љубав и подршку  
током студија и писања рада.*

**Наслов мастер рада:** Анализа система за непрекидну интеграцију и непрекидну испоруку

**Резиме:** Данас је уобичајено да се у развоју апликација ангажује велики број програмера и тимова, јер се сложеност мери десетинама човек/година. Због тога се јавља потреба да се аутоматизују процеси који се јављају при развоју тих сложених апликација.

У овом раду, направљени су токови непрекидне интеграције и непрекидне испоруке, помоћу три популарна алата - Jenkins, GitHub Actions и GitLab CI/CD. Токови се аутоматски активирају након било које промене изворног кода на складишту и обухватају процесе превођења, тестирања јединичним и интеграционим тестовима, као и испоруке на удаљени продукциони сервер.

Имплементација и анализа токова је показала да су алати GitHub Actions и GitLab CI/CD приближно једнаких перформанси, док се Jenkins концептуално разликује, али је ипак сличних перформанси и могућности. Такође, нове верзије су доступне у све краћим временским интервалима, а то има за последицу да неке функционалности брзо постају превазиђене, па о томе треба водити рачуна током животног века апликације.

**Кључне речи:** непрекидна интеграција, непрекидна испорука, јединични тестови, интеграциони тестови, Jenkins, GitHub Actions, GitLab CI/CD

# Садржај

<b>1</b>	<b>Увод</b>	<b>1</b>
<b>2</b>	<b>Апликација Currency Explorer</b>	<b>3</b>
2.1	Развој вођен тестовима . . . . .	3
2.2	Развој са и без тестова . . . . .	4
2.3	Опис апликације . . . . .	5
<b>3</b>	<b>Непрекидна интеграција/испука</b>	<b>8</b>
3.1	CI/CD процес . . . . .	8
3.2	Непрекидна интеграција . . . . .	9
3.3	Непрекидна испука . . . . .	9
3.4	Модерни поглед на CI/CD проблематику . . . . .	11
<b>4</b>	<b>Алати за CI/CD</b>	<b>13</b>
4.1	Jenkins . . . . .	13
4.2	GitHub Actions . . . . .	17
4.3	GitLab CI/CD . . . . .	22
<b>5</b>	<b>Рад на токовима</b>	<b>29</b>
5.1	Имплементација токова . . . . .	29
5.2	Поређење алата GitHub и GitLab . . . . .	31
5.3	Поређење GitHub Actions/GitLab CICD и Jenkins-а . . . . .	33
<b>6</b>	<b>Закључак</b>	<b>35</b>
<b>7</b>	<b>Додаци и прилози</b>	<b>37</b>
	<b>Библиографија</b>	<b>43</b>

# Глава 1

## Увод

Предмет мастер рада је поређење три система за CI/CD (енг. continuous integration/continuous deployment), на примеру развијене апликације Currency Explorer.

Данас се развијају сложене апликације уз помоћ већег броја програмера<sup>1</sup>, често распоређених у тимове, евентуално географски удаљене. Такав је и случај са веб апликацијама.

Прављење извршних верзија је сложен процес који захтева писање програмског кода, тестова, проверу квалитета итд. У том циљу, постоји више алата помоћу којих се аутоматизује поступак превођења, тестирања и испоруке програмског кода.

Најпознатији алати отвореног кода су Jenkins, GitHub Actions, GitLab CI/CD, који омогућују аутоматизацију процеса од написаног кода апликације, до извршног кода, најчешће на веб платформи. Коришћењем ових алата лакше се проналазе и отклањају грешке у целом процесу, убрзава и олакшава комуникација међу програмерима и цео ток развоја ставља на располагање свим учесницима.

У поглављу **Архитектура апликације** детаљно је описана апликација Currency Explorer и функционалности Exchange Rate и Investment Consulting, као и методе за предвиђање курса валута. За ту апликацију су касније развијани CI/CD токови. Дискутовани су и јединични и интеграциони тестови, као и развоји апликација вођени са или без тестова.

---

<sup>1</sup>Термин *програмер* ће се користити за лица која обављају послове развоја софтвера (енг. developer), контроле квалитета (енг. QA - quality assurance), администрације базе података, веб дизајнера итд.

Поглавље **Непрекидна интеграција/испурука** обухвата исрпно описана својства непрекидне интеграције и непрекидне испоруке, уз осврт на најпопуларније произвођаче присутне данас.

У поглављу **Алати за CI/CD** су описана три изабрана алата Jenkins, GitHub Actions и GitLab CI/CD. Сваки од алата је детаљно описан, уз исрпно објашњење како је сваки од њих имплементиран за дату апликацију. Ово поглавље садржи доста илустрација, како самог графичког интерфејса појединачних алата и њихових корака, тако и критичних делова кода.

Поглавље **Рад на токовима** описује имплементацију самих токова, са акцентом на продукционо окружење. Дискутована су одређена ограничења и проблеми који су се јављали током имплементације. Након тога, упоређени су алати које нуде GitHub и GitLab, као и обједињено поређење ова два алата насупротив Jenkins-а.

Поглавље **Закључак** садржи преглед урађених делова апликације, као и потенцијалне активности за развој непрекидне интеграције и непрекидне испоруке у будућности.

## Глава 2

# Апликација Currency Explorer

За потребе рада је развијена веб апликација Currency Explorer. Коришћена је C# ASP.NET Core платформа, у развојном окружењу Visual Studio 2019. У односу на уобичајену MVC архитектуру, апликација је рефакторисана тако што је додат сервисни слој, што је олакшало тестирања.

Апликација је дизајнирана у складу са методологијом вођеном тестовима (енг. Test-driven development[1]) и налази се на јавном складишту (енг. repository) GitHub [2].

### 2.1 Развој вођен тестовима

Раније се софтвер писао тако што су се директно преносили захтеви из пројекта у извршни кôд. Данас је често присутна техника развоја вођеног тестовима, где се прво пишу тестови које мора да задовољи апликација, а затим се програмира и сама апликација. Дакле, полази се од тога да су сви тестови неуспешни, јер у почетку нема програмског кôда, а затим се писањем програмског кôда задовољавају тестови.

Постоје две врсте тестова, јединични (енг. unit tests) и интеграциони (енг. integration tests).

#### Јединични тестови

Ова врста тестова је карактеристична по томе што тестира мање делове кôда, пише се велики број ових тестова и брзо се извршавају. Брзина извршавања се постиже тиме што се приликом тестирања не користи ниједан спољни



елемент (база, систем датотека,...), такозвана изолација, као и могућност да се тестови извршавају паралелно.

Постоји више пакета који омогућују писање јединичних тестова, као што су xUnit, jUnit, nUnit, CUnit, COBOLUnit, RUnit... Очигледно, односе се на различите програмске језике. За апликацију Currency Explorer развијани су тестови помоћу xUnit.net [3] пакета.

### Интеграциони тестови

Интеграциони тестови покривају веће делове кода апликације, често и читаве подсистеме и њихову интеракцију. И овде постоји више пакета за ту намену (Selenium, Cucumber, Robot Framework,...), а за апликацију је коришћен Selenium [4]. Ови тестови захтевају знатно дуже време извршавања него код јединичних тестова.

## 2.2 Развој са и без тестова

Поставља се питање сврсисходности употребе развоја вођеног тестовима. У књизи [5], аутор наводи пример велике софтверске компаније, са више од 1000 програмера, која је желела да почне са имплементирањем јединичних тестова на постојећој великој апликацији. Формирали су два тима, подједнаког квалитета; један је радио на традиционалан начин без тестова, а други применом јединичних тестова. Захтевало се да оба тима испоруче сличне функционалности.

Мерење перформанси развијеног софтвера је урађено по три критеријума:

- време развоја појединачне фазе,
- укупно време до испоруке апликације,
- број грешака пријављен од стране клијента.

У табели 2.1 су приказана времена појединих фаза.

Дакле, укупно време испоруке је мање код приступа са тестовима него без тестова. Иако се сматра да развој вођен тестовима траје дуже због писања тестова, видимо да је то случај само у првој фази. Али, основна предност је та што се добија квалитетнија апликација за краће време и са много мањим бројем испоручених грешака.

Фаза	Тим без тестова		Тим са тестовима	
Имплементација (кодирање)	7 дана		14 дана	
Интеграција	7 дана		2 дана	
Тестирање и исправљање грешака	Тестирање:	3 дана	Тестирање:	3 дана
	Исправљање:	3 дана	Исправљање:	1 дан
	Тестирање:	3 дана	Тестирање:	1 дан
	Исправљање:	2 дана	Исправљање:	1 дан
	Тестирање:	1 дан	Тестирање:	1 дан
	Укупно:	12 дана	Укупно:	7 дана
Укупно време испоруке	26 дана		23 дана	
Број грешака након испоруке	71		11	

Табела 2.1: Перформансе са и без тестова

## 2.3 Опис апликације

На насловној страни апликације, приказује се списак вредности валута у односу на евро на дан приступа страници. Подаци су са сајта Exchange Rates API [6] и памте се у бази података.

Приликом приступа апликацији се проверава постојање вредности валута на текући дан; уколико подаци не постоје, база се ажурира недостајућим вредностима. Валуте се чувају у бази од почетног датума 1. марта 2021. године.

Потом се захтева регистрација или логовање корисника, коришћењем електронске поште и лозинке. Лозинка мора садржати сложенију комбинацију знакова. Након логовања, кориснику су на располагању две функционалности:

- Exchange Rate,
- Investment Consulting.

Exchange Rate функционалност омогућује кориснику преглед свих валута за жељени датум. Подаци се не приказују уколико је жељени датум пре почетног датума или након датума приступа апликацији.

Investment Consulting функционалност предвиђена је за датуме из будућности и омогућује задавање жељених података. Бирају се две валуте, почетни и крајњи датум, као и метода предвиђања. Потом се генерише и приказује график.

График је направљен уз помоћ библиотеке FusionCharts [7]. Могуће су различите врсте графика, а за потребе апликације изабран је график са тачкама (енг. dot chart), јер омогућује приказ две валуте истовремено. На x-оси графика су приказани датуми из задатог интервала, а на у-осама су приказане вредности две изабране валуте, по одговарајућим скалама.

Вертикална црвена линија означава датум приступа апликацији.

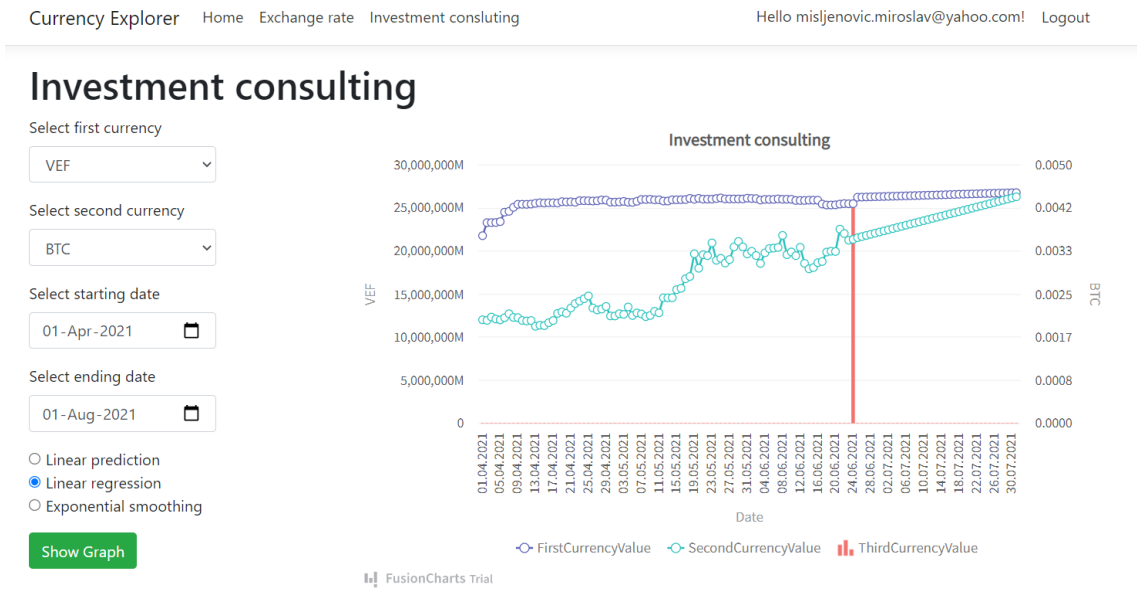
Померањем миша по графику се активира вертикална сива линија, уз коју су приказане вредности валута и датум који одговара положају миша. Такође, могућ је приказ само једне од валута.

### Предвиђање валута

За предвиђање курсева изабраних валута у будућности, користе се три методе:

1. линеарно предвиђање,
2. линеарна регресија,
3. експоненцијално изглађивање (енг. exponential smoothing), са жељеним параметром  $\alpha$ .

Како је реч о екстраполацији, мора се водити рачуна да датум у будућности не буде много удаљен од текућег датума. Уколико постоји потреба, једноставно је проширити предвиђања валута другим алгоритмима.



Слика 2.1: График предвиђања линеарном регресијом за венецуелански боливар и биткоин

На слици 2.1 је приказана функционалност Investment Consulting, за изабране валуте, датуме и методу предвиђања. Ове валуте су изабране због њихове варијабилности у временском периоду од интереса.

# Глава 3

## Непрекидна интеграција/испорука

Модерни развој софтверских апликација подразумева извршавање низа корака, којима се долази до продукционог кода. Од велике важности је да ти кораци буду аутоматизовани, чиме се смањује време испоруке апликације, а такође и побољшава интеракција међу учесницима на развоју апликације. Цео процес се уобичајено назива ток (енг. pipeline), а поједини кораци се називају фазе (енг. stages).

### 3.1 CI/CD процес

Већина модерних апликација захтева развијање кода помоћу различитих платформи и алата, па тимови програмера траже механизме за интеграцију и проверу промена. CI/CD је таква метода, за честу испоруку софтвера, којом се уводи аутоматизација у све кораке развоја апликације.

Особине које чине добар CI/CD ток су брзина, поузданост и прецизност. Постојање CI/CD процеса има и додатне позитивне ефекте:

- програмери су фокусирани на писање кода и праћење својстава апликације;
- тим за проверу квалитета има увид у најсвежију и све остале верзије апликације;
- освежавање апликације није стресно;

- измене кода, тестова и испорука су расположиве у сваком тренутку;
- повратак на претходне верзије је рутински;
- омогућена је брза повратна информација.

Интеграција новог кода може довести до такозваног интеграционог пакла (енг. integration hell), што се у великој мери разрешава CI/CD приступом, тј. применом непрекидне интеграције и непрекидне испоруке. Овај приступ се заснива на ставу да је лакше развијати апликацију у честим и малим корацима, него свим променама одједном. Тако, Амазон избацује промене у просеку сваких 11.2 секунде. Откад су увели такву праксу, добили су смањење времена за исправљање грешака од 90% [8] .

### 3.2 Непрекидна интеграција

Циљ непрекидне интеграције је да обезбеди конзистентан и аутоматизован процес изградње и тестирања апликација. Уколико је процес такав, тимови програмера чешће могу да обједине кораке развоја апликације и тиме остваре бољу међусобну сарадњу и реализују квалитетнији софтвер.

Приликом развоја софтвера више програмера раде истовремено на различитим функционалностима. Уколико се очекује да њихов рад буде обједињен у договореном тренутку (енг. merge day) то може бити тешко изводљиво и временски захтевно. Посебно, потешкоће настају ако програмери истовремено мењају исти део апликације.

Непрекидна интеграција помаже програмерима да чешће обједињују промене на складишту. Потом се аутоматски покрећу превођење и тестирање апликације јединичним и интеграционим тестовима, како би се утврдило да апликација функционише исправно. Уколико је дошло до грешке, програмери се обавештавају о томе, да би исправили грешку што пре.

### 3.3 Непрекидна испорука

У пракси се упоредо користе два енглеска термина која се односе на непрекидну испоруку. Један је непрекидно испоручивање (енг. continuous delivery), а други непрекидно смештање (енг. continuous deployment). Први термин се

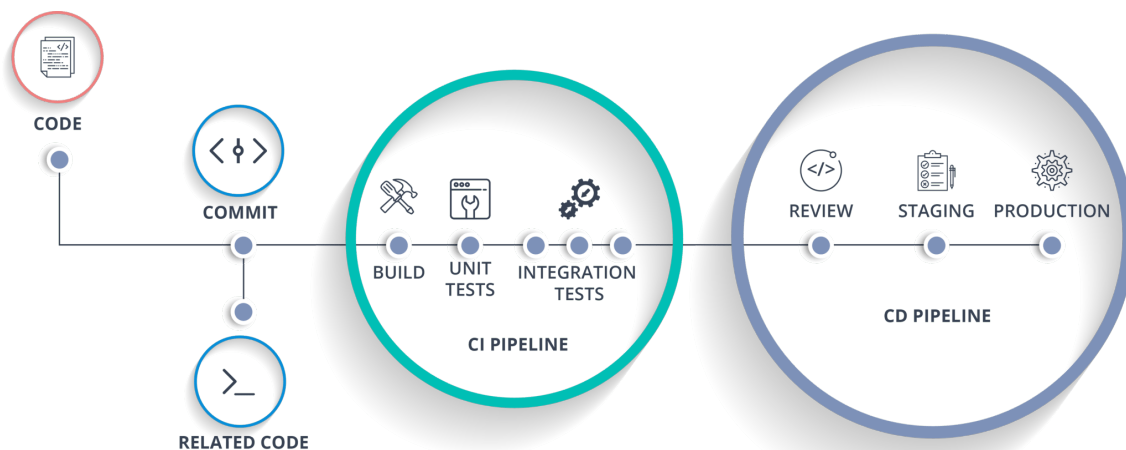
односи на испоручивање у окружење где ће кôд бити додатно обрађен, на пример, на окружење за проверу квалитета и сигурности или на продукционо окружење, док се други термин користи када се подразумева само смештање на продукционо окружење. Мала је разлика у практичном значењу ове две врсте, па ћемо користити заједнички назив непрекидна испорука.

Након што је готова фаза непрекидне интеграције и реализовано превосијање и тестирање, апликација је спремна за испоруку. Уобичајено постоји више окружења за испоруку, на пример бета окружење које се користи интерно или продукционо окружење за крајње кориснике.

Честа пракса је да је тренутак покретања испоруке диктиран од стране програмера, а не корисника. То може бити лоше решење због, на пример, опасности освежавања софтвера у критичном тренутку. У таквим ситуацијама, боље решење је да корисник може да дефинише тренутак покретања испоруке.

Испорука је често временски захтевна, што може да буде неповољно за кориснике апликације. Стога, испорука се често врши у тренуцима мале оптерећености система.

Непрекидна испорука се реализује за кратко време од тренутка измене кôда и тиме се омогућује да се брзо добијају повратне информације од корисника.



Слика 3.1: График тока непрекидне интеграције и непрекидне испоруке

Од многих расположивих CI/CD токова, на слици 3.1 је приказан онај који највише одговара реализованом у овој апликацији.

## 3.4 Модерни поглед на CI/CD проблематику

Данас су у широкој употреби алати за CI/CD, који имају базични део који је бесплатан, а додатна напредна својства се доплаћују:

- Jenkins,
- Travis CI,
- Circle CI,
- Team City,
- GitLab CI/CD,
- GitHub Actions,
- BitBucket Pipelines & Bamboo.

Ова проблематика је почела да се развија када је Мартин Фаулер 2006. године објавио рад [9] у коме је заговарао да би код који се развија требало стално интегрисати у развојни циклус и покретати тестове аутоматски, како би се брзо уочиле грешке и смањио њихов број.

Овакви алати су се развијали почев од 2008. године, пратећи развој алата Hudson CI, који је 2011. наставио свој развој под именом Jenkins. Јако повећано интересовање за CI/CD је резултовало куповином GitHub-а од стране Microsoft-а 2018. године за 7.5 милијарди долара. Овај успешни пословни потез довео је до значајног проширења заједнице корисника. Ове године, Microsoft је објавио да се број активних организација које користе GitHub повећао за 70% у претходних 12 месеци.

Приликом избора алата, требало би водити рачуна о многим елементима, како би крајњи резултат рада одговарао захтевима. Неки од њих биће дискутовани у наставку.

**Hosting** - Неки алати користе облак (енг. cloud), имплементирани на страни провајдера и захтевају мало посла око почетног конфигурисања. На супрот њих, постоје алати који се могу имплементирати на сопственом локалном серверу, чиме се постиже већа флексибилност и скалабилност.



**Интеграција и софтверска подршка** - Проблем који треба размотрити је како се одређени алат интегрише са другим софтверским алатима који се користе у целокупном развоју пројекта. На пример, пројекат може користити алат за праћење софтвера (Jira), алате за статичку анализу, алате за изградњу/превођење (Make, Shell, Ant, Maven, Gradle), као и систем за контролу верзија (Git, Subversion, BitBucket).

**Употребљивост** - Неки послови се могу урадити брже коришћењем алата код којих је једноставано и интуитивно графичко окружење (енг. GUI - graphical user interface) и позитивно корисничко искуство (енг. UX - user experience).

**Подршка контејнера** - Уколико су подржани алати за управљање контејнерима, као што су Kubernetes и Docker, лако се CI алати повезују са окружењем пројекта.

**Доступност библиотека кода** - Пожељно је да постоји што више јавно доступног отвореног кода, библиотека, додатака, и сл. Неки од њих могу бити и комерцијални. Поред олакшаног развоја апликација, отворени код олакшава процес образовања, увидом у програмску реализацију.

# Глава 4

## Алати за CI/CD

### 4.1 Jenkins



Jenkins [10] је бесплатан сервер за аутоматизацију развоја софтвера (превођење, тестирање, испорука) и подржава непрекидну интеграцију и непрекидну испоруку. Такође, подржава и системе за контролу верзија (енг. version control system) као што су Git, CVS,

Subversion, Mercurial и сл., и извршавање скриптова.

Jenkins се у почетку звао Hudson, али је променио име услед спора са компанијом Oracle. Спор је разрешен 2011. године када је професионална заједница великом већином одлучила да се формира Jenkins project. У то време, Hudson/Jenkins је прихваћен као најбољи избор међу серверима за превођење као што су Cruise Control и други сервери отвореног кода. На конференцији JavaOne, одржаној маја 2008. године, овај софтвер је освојио специјалну награду у категорији Developer Solutions [11].

До тада развијени софтвер (Hudson) су преузели (енг. fork<sup>1</sup>) и Oracle и Jenkins. Oracle је овај софтвер проследио фондацији Eclipse. Jenkins је наставио са успешним развојем и 2019. године је на GitHub-у имао 667 пројеката и око 2200 јавних складишта.

Прве верзије Jenkins-а су писане за програмски језик Java, а касније се омогућила примена и других програмских језика. У даљем тексту ће бити

---

<sup>1</sup>Овај термин се користи у ситуацијама када се копира отворени код, како би се даље развијао и ставио другима на располагање.

описани делови Jenkins-а који су коришћени у овом раду.

Након инсталације овог софтвера, креира се нови пројекат и најчешће коришћен је пројекат слободног стила (енг. freestyle project); често се примењују и пројекти тока (енг. pipeline project). У овом раду је коришћен пројекат слободног стила под називом CurrencyExplorer, доступан на адреси <http://liss.matf.bg.ac.rs:8080/>.

Након што се креира пројекат, потребно је подешавати конфигурације како самог система, тако и појединачног пројекта.

### Конфигурација Jenkins-а

Након бирања опције **Подеси Jenkins** (енг. manage Jenkins) подешавање се највећим делом спроводи кроз три подсистема:

- конфигурација,
- општи алати (енг. global tools),
- додаци (енг. manage plugins).

**Конфигурација** обухвата прецизирање разних опција, као што су нпр. локација самог система, окидачи веб кљка (енг. generic web hook), испорука путем SSH или FTP протокола, обавештења електронском поштом и др.

**Општи алати** подразумевају уношење путања до локација извршних датотека на рачунару на коме је Jenkins покренут као сервис. Наведимо неке од њих: Git, MSBuild, MSTest, VSTest. Те датотеке ће се користити за реализацију CI/CD тока.

**Додаци** омогућују проширивање функционалности самог система и прилагођавање програмском језику у коме је писана апликација. Саставни део чини претрага додатака за инсталирање и/или њихово унапређивање.

### Конфигурација пројекта

За пројекат који се развија, на располагању је опција **Подешавања** (енг. configure). Опције које се подешавају су:

- општа подешавања (енг. general),
- рад са изворним кодом (енг. source code management),

- израда окидача (енг. build triggers),
- израда окружења (енг. build environment),
- израда тока (енг. build),
- завршне активности (енг. post-build actions).

**Општа подешавања** омогућују прецизирање описа пројекта, брисање претходних токова, блокирање рада док се пре/пост активности не реализују, локација жељеног радног простора и сл.

**Рад са изворним кодом** дозвољава везивање са спољним складиштем. У раду је коришћен Git, па је било потребно подесити путању до складишта и гране која се прави.

**Израда окидача** је корисна у процесу аутоматизације, јер омогућује периодична освежавања коришћењем хронометра. Једно од њих је синхронизација помоћу опције *Провера разлика* (енг. poll SCM) радног простора и складишта. Напреднији начин је постављање веб куке на складиште, која ће сигнализирати Jenkins-у да је дошло до промена. Друга корисна опција окидача је могућност периодичног аутоматског превођења.

**Израда окружења** дозвољава брисање радног простора пре покретања тока, слање фајлова и извршавање команди путем SSH протокола пре и после покретања тока, укључивање променљивих окружења у ток, и сл.

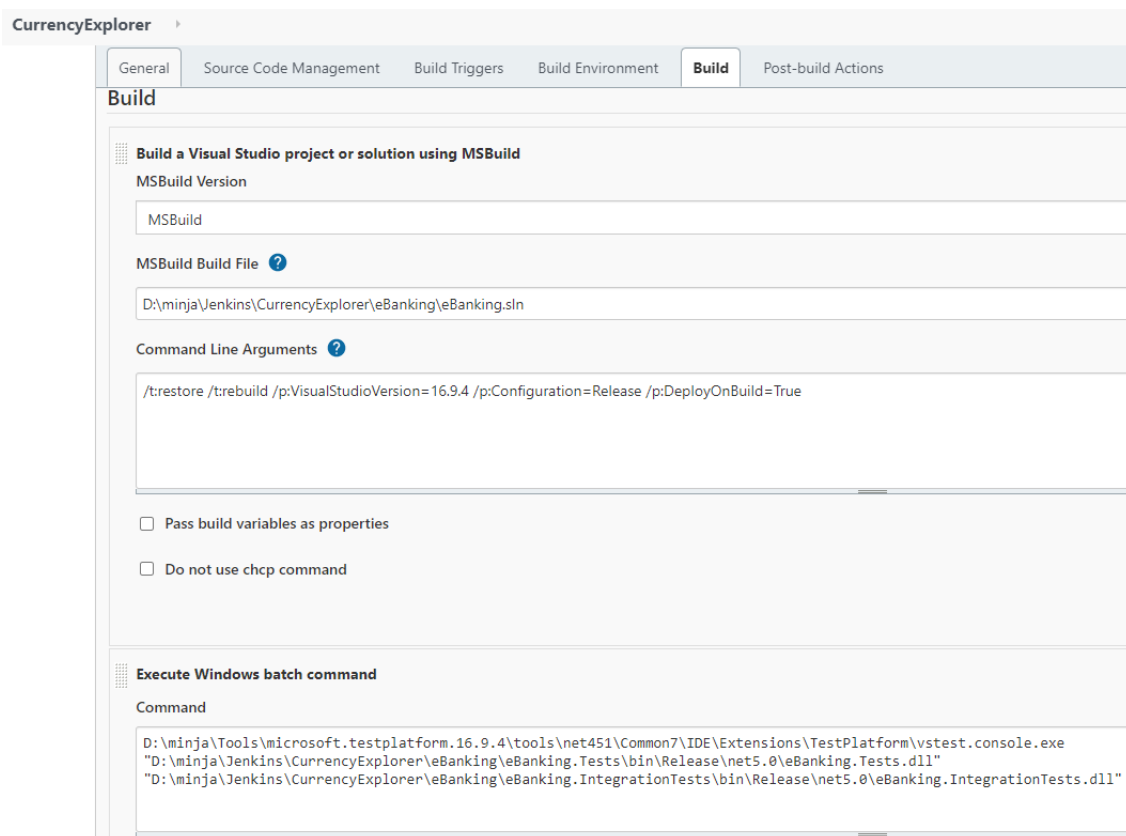
**Израда тока** подразумева прављење највећег дела CI/CD процеса. Наводе се кораци аутоматизације:

1. инсталација или освежавање NuGet пакета,
2. превођење пројекта помоћу MSBuild-а (евентуално уз параметре),
3. покретање јединичних тестова помоћу MSTest алата,
4. покретање интеграционих тестова помоћу VSTest алата,
5. извршавање shell или batch команди.

**Завршне активности** се односе на комплетирање резултата претходних корака, који се могу архивирати и/или послати путем SSH/FTP протокола. Такође, може се генерисати извештај о тестирању, обрисати радни простор, послати електронско обавештење.

## Имплементација

На слици 4.1 приказан је део израде тока који се односи на превођење и тестирање кода.

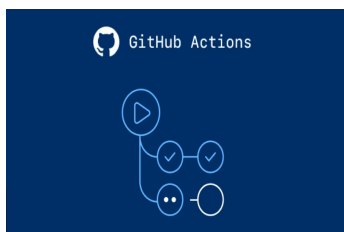


Слика 4.1: Израда тока

Наведена је датотека која се преводи, са параметрима командне линије; параметар `/t:restore` се односи на инсталацију и/или освежавање NuGet пакета, а параметар `/p:DeployOnBuild=True` означава да се након превођења прави директоријум за испоруку.

Тестирање се извршава једном конзолном командом. У њој се наводи путања до VSTest извршне датотеке, као и одговарајуће датотеке јединичних и интеграционих тестова.

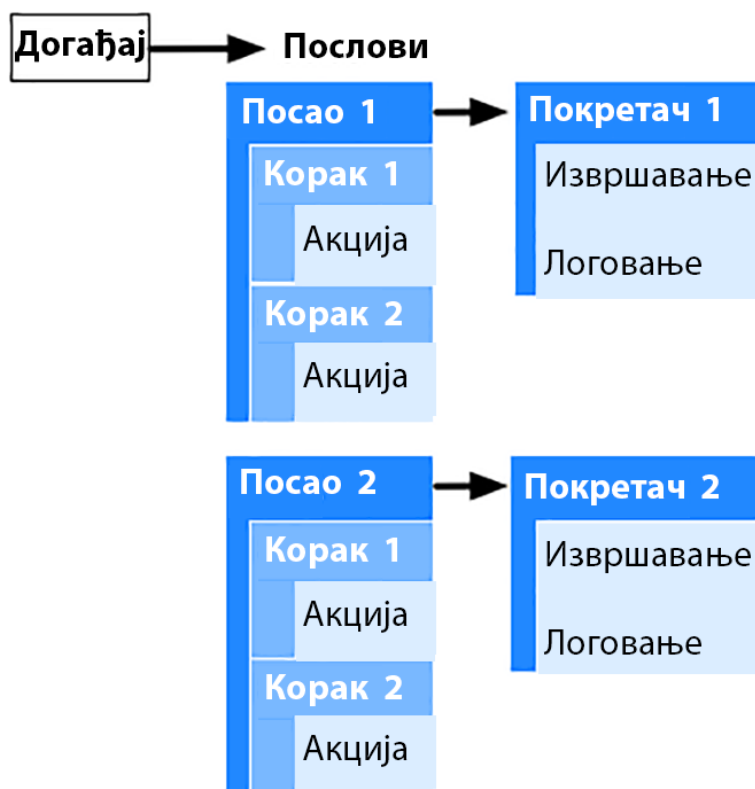
## 4.2 GitHub Actions



GitHub [12] се у почетку користио углавном само као складиште. Временом су се додавали други сервиси међу којима је и подршка за CI/CD - Github Actions, који омогућује аутоматизацију тока, превођења, тестирања и испоруке, без коришћења алата других произвођача. Овај сервис је вођен

догађајима (енг. event-driven), што значи да се процес покреће након што се деси неки од предефинисаних догађаја. На пример, сваки пут када се неки садржај смести у складиште, аутоматски се извршава жељена акција.

На слици 4.2 је дат детаљан приказ корака аутоматизације.



Слика 4.2: GitHub Actions дизајн

Догађај активира радни ток, који се састоји од послова. Сваки посао садржи кораке којима се дефинише редослед акција. Послу одговарају покретачи (енг. runner), тј. сервери на којима је GitHub Actions покретачки софтвер инсталиран.

**Покретачи** постоје за Ubuntu Linux, Microsoft Windows и macOS и сваки се извршава у виртуалном окружењу. Они чекају послове, извршавају их један по један и визуелно извештавају о напретку и резултатима.

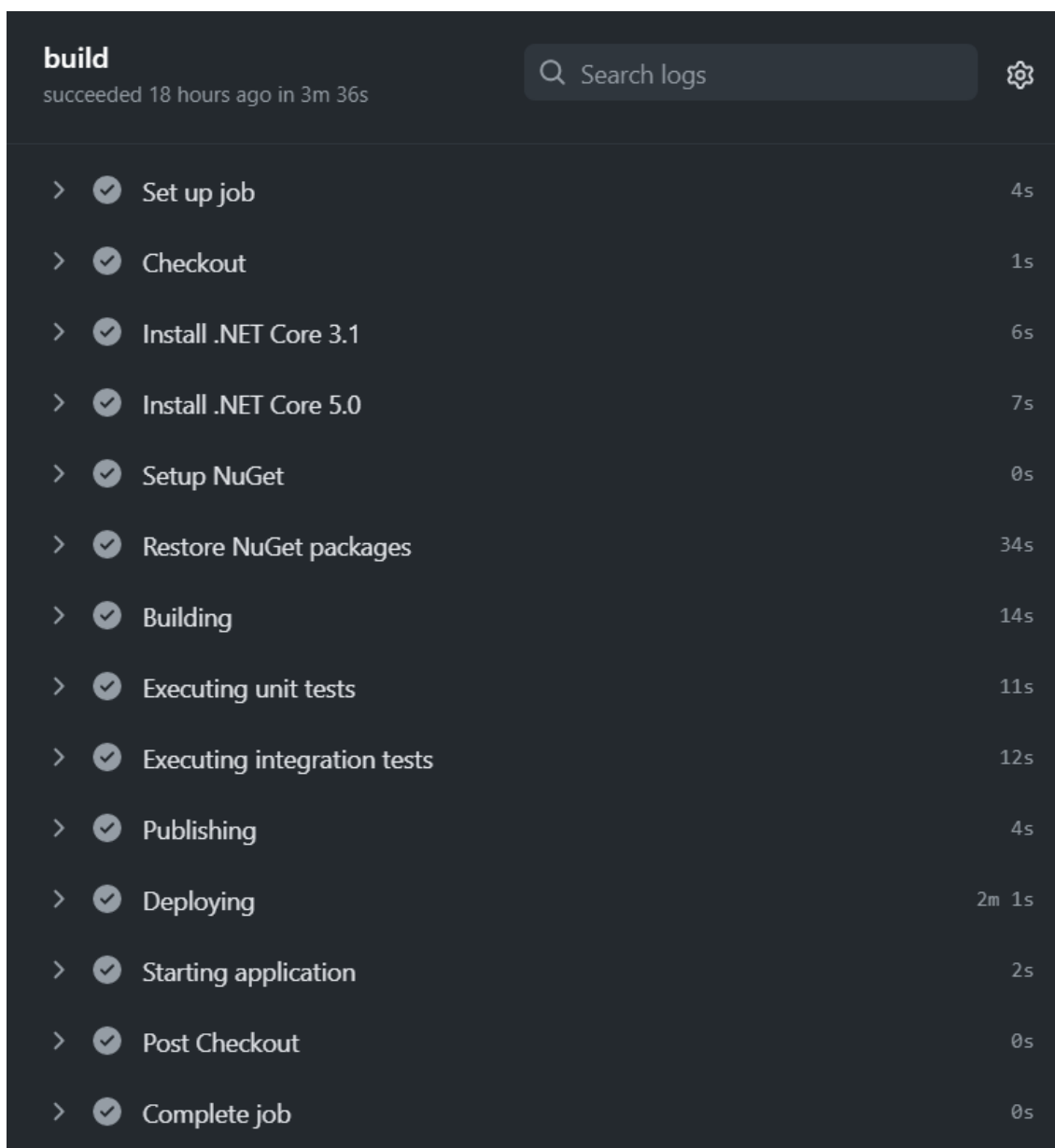
**Посао** чини скуп корака који се секвенцијално извршавају на покретачу. Ако ток садржи више послова, подразумева се да се они извршавају паралелно. Могуће је и секвенцијално извршавање послова у оквиру радног тока уколико су послови међусобно зависни. Нпр. ако имамо два посла од којих је један превођење, а други тестирање, други не треба извршити ако први посао није успешно завршен; препознају се у радном току службеном речју *needs*.

**Корак** неког посла је команда, која може бити акција или shell/batch команда. Сваки корак посла се извршава у оквиру истог покретача, чиме се омогућава размена података међу корацима.

**Акције** чине независне команде које се комбинују у кораке. Акције су најмањи градивни блок радног тока. Могу се креирати сопствене акције или искористити акције које су доступне у Marketplace-у GitHub заједнице. Оне се препознају у радном току кључном речју *uses* и припадним параметрима.

### Имплементација

Формирање CI/CD радног тока спроводи се писањем конфигурационе `.yaml` датотеке, која се на складишту налази у директоријуму `.github/workflows`. Након постављања ове датотеке у складиште, у одељку Actions се може пратити ток извршавања појединих послова и појединачних корака. На слици 4.3 је дат графички приказ свих корака успешног радног тока.



Слика 4.3: GitHub Actions радни ток

Почетни корак односи се на иницијално подешавање окружења; доступни су подаци о покретачу, оперативном систему покретача, припремају се директоријум радног тока и све потребне акције које ће се користити у корацима. Акције су дефинисане на јавно доступним складиштима и допремају се у текући покретач.



Други корак односи се на будућу синхронизацију складишта на GitHub-у и складишта на покретачу. Прво се обрише претходни директоријум радног тока, а затим се у нови допреми актуелни садржај GitHub складишта.

Потом се инсталирају две верзије подршке за dotnet SDK (3.1 и 5.0). Ово је потребно јер је за саму апликацију потребна верзија 3.1, док се за јединичне и интеграционе тестове користи верзија 5.0.

Следећа два корака односе се на NuGet пакете. У првом се допрема последња верзија nuget.exe датотеке, а у наредном се помоћу те датотеке допремају и инсталирају сви потребни NuGet пакети. Ово се уобичајено ради помоћу наредбе dotnet restore, међутим неке зависности између појединих пакета/датотека су доводиле до неуспеха, а алтернативно решење помоћу nuget.exe датотеке је довело до успешног завршетка овог корака.

Наредни корак се односи на превођење пројеката апликације и тестова, наредбом dotnet build. У извештају се може видети да се за саму апликацију користила верзија dotnet 3.1, а за пројекте јединичних и интеграционих тестова верзије dotnet 5.0. Резултат рада је да је све преведено без грешака и упозорења.

У наредна два корака се прелази у фазу тестирања. Извршава се наредба dotnet test за одговарајуће .dll датотеке. За интеграционе тестове је потребно усагласити верзије Google Chrome претраживача, Chromedriver-а и NuGet пакета. Да би овај корак био успешно завршен, било је потребно додати следеће аргументе Chromedriver-а: no-sandbox, disable-dev-shm-usage и headless.

Након што су готове фазе превођења и тестирања, на реду је фаза испоруке на продукционо окружење, што је приказано на слици 4.4.

```
- name: Publishing
  run: |
    cd eBanking
    ls
    dotnet publish
    cd ./eBanking/bin/Debug/netcoreapp3.1/publish/
    rm appsettings.json
    rm -r cs de es fr it ja ko pl ru zh-Hans zh-Hant tr pt-BR
    ls

- name: Deploying
  uses: garygrossgarten/github-action-scp@v0.7.3
  with:
    local: './eBanking/eBanking/bin/Debug/netcoreapp3.1/publish/'
    remote: '/home/miki/GithubActionsPublish'
    host: ${ secrets.SSH_HOST }
    username: ${ secrets.SSH_USERNAME }
    password: ${ secrets.SSH_PASSWORD }

- name: Starting application
  uses: garygrossgarten/github-action-ssh@v0.6.3
  with:
    command: |
      screen -ls
      cd /home/miki/GithubActionsPublish
      screen -X -S gitHubActions quit
      screen -dmS gitHubActions dotnet eBanking.dll
      screen -ls
    host: ${ secrets.SSH_HOST }
    username: ${ secrets.SSH_USERNAME }
    password: ${ secrets.SSH_PASSWORD }
```

---

Слика 4.4: Продукциони део GitHub .yaml датотеке

Корак почиње командом `dotnet publish` којом се креира садржај за продукцију. Овде су избачене датотеке и директоријуми који нису неопходни за даљи рад.

Следећи корак се односи на испоруку на удаљени сервер тј. у продук-

ционо окружење. Да би се то безбедно урадило, коришћене су могућности GitHub-а. У одељку Settings се налази опција Secrets у којој се дефинишу сви подаци који не би требало да буду јавно видљиви (адреса удаљеног сервера, корисничко име, шифра итд.). Пренос је урађен акцијом са складишта [13], постављањем тајни и путања до одговарајућих директоријума на покретачу и удаљеном серверу, помоћу SSH протокола.

Сада је апликација спремна за покретање или освежавање уколико је већ била у продукцији. Помоћу наредбе `screen -dmS gitHubActions dotnet eBanking.dll`, направљен је нови процес који покреће апликацију и ради у позадини; дато му је име `gitHubActions`, како би се пре освежавања угасио тај конкретан процес.

Последња два корака се односе на завршетак рада покретача и брисање непотребних датотека, директоријума и процеса.

### 4.3 GitLab CI/CD



GitLab [14] је основан 2011. године и заснован је на раду два украјинска програмера, Дмитрија Запорожеца и Валерија Сизова. То је веб оријентисан алат који подржава рад са Git складиштем, Wiki стране, праћење задатака (енг. *issue-tracking*) и непрекидну интеграцију и непрекидну испоруку. Кôд апликације је иницијално писан у програмском језику Ruby, а касније су неки делови поново написани у програмском језику Go. Основни циљ је био да се подржи тимски развој софтвера и управљање извршним кôдом. Временом се апликација развијала тако да обухвати цео животни век развоја софтвера, од планирања до имплементације, изградње, верификације, тестирања сигурности, испоруке и надгледања.

Компанија GitLab има интересантан пословни модел. С једне стране, корисницима је на располагању *freemium* модел коришћења апликације. Термин *freemium* је настао од две енглеске речи (*free* и *premium*) и односи се на стратегију плаћања по којој се основни скуп сервиса добија бесплатно, а додатни сервис/својства се наплаћују. С друге стране, GitLab је 2020. године проглашен за највећу светску компанију која у потпуности ради на даљину; притом, има 1200 запослених у преко 65 земаља и региона. Интересантно је да не-

мају централну управу, као ни пословне просторије било где у свету; сваки запослени ради даљински. GitLab је остварио раст од 50 пута у 4 године и тренутно вреди 2.75 милијарди долара.

GitLab је у раду [15] спровео испитивање са 3000 запослених у разним индустријама, на различитим позицијама и различитим географским локацијама, који су радили на даљину у периоду од 30. јануара до 10. фебруара 2020. године. Направљен је исцрпан извештај о главним мотивима за такав рад, и код запослених и код послодаваца, и томе како рад на даљину мења друштво.

GitLab апликација омогућује високу скалабилност и може радити како на локалном рачунару, тако и на облаку. Такође, дозвољена је величина складишта до 10 гигабајта. Неки од значајних корисника су компаније Alibaba, IBM, SpaceX, GNOME.

### Имплементација

Корисно својство које нуди GitLab је синхронизација (енг. mirroring) кода са GitHub складиштем. Ова опција се може аутоматски активирати уколико је дошло до промене садржаја на неком од складишта. На слици 4.5 је приказана опција којом се то реализује; наводи се адреса другог складишта, смер у ком се спроводи синхронизација, метода аутентификације и лозинка.

### Mirroring repositories

Collapse

Set up your project to automatically push and/or pull changes to/from another repository. Branches, tags, and commits will be synced automatically. [How do I mirror repositories?](#)

#### Git repository URL

Input the remote repository URL

- The repository must be accessible over `http://`, `https://`, `ssh://` or `git://`.
- When using the `http://` or `https://` protocols, provide the exact URL to the repository. HTTP redirects will not be followed.
- Include the username in the URL if required: `https://username@gitlab.company.com/group/project.git`.
- The update action will time out after 180 minutes. For big repositories, use a clone/push combination.
- Git LFS objects will be synced if LFS is [enabled for the project](#). Push mirrors will **not** sync LFS objects over SSH.
- In case of pull mirroring, your user will be the author of all events in the activity feed that are the result of an update, like new branches being created or new commits being pushed to existing branches.

#### Mirror direction

Push

#### Authentication method

Password

#### Password

Слика 4.5: GitLab синхронизација складишта

GitLab омогућује рад са специфичним и дељеним покретачима. У секцији Подешавања -> CI/CD, налази се опција Покретачи. На сликама 4.6 и 4.7 приказане су ове две врсте покретача, број расположивих покретача обе врсте са одговарајућим идентификационим бројем и таговима, дугме за приказ инсталације специфичног покретача и дугме за поновно генерисање регистрационог токена.

## Runners

Collapse

Runners are processes that pick up and execute CI/CD jobs for GitLab. [How do I configure runners?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine. Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

### Specific runners

These runners are specific to this project.

#### Set up a specific runner automatically

Register a runner on a Kubernetes cluster. [Learn more.](#)

1. Click the button below.
2. Select an existing Kubernetes cluster or create a new one.
3. From the Kubernetes cluster details view, applications list, install GitLab Runner.

Install GitLab Runner on Kubernetes

### Shared runners

These runners are shared across this GitLab instance.

[Shared Runners on GitLab.com](#) run in **autoscale mode** and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 400 CI minutes per month per group for private projects. Read about all [GitLab.com plans](#).

Enable shared runners for this project



Available shared runners: 15

Слика 4.6: GitLab покретачи

#### Set up a specific runner manually

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:  
<https://gitlab.com/>

And this registration token:

S\_UJEEAh9Hz\_\_7zV8RU5

Reset registration token

Show Runner installation instructions

### Available specific runners

#7513156 (skwwGzc8)

euve267394

testing

Remove runner

### Available shared runners: 15

#2072938 (D4kd9MvC)

gitlab-docker-shared-runners-manager-02

gitlab-org-docker

#1506020 (Hs8mheX5)

windows-shared-runners-manager-1

shared-windows windows windows-1809

#1506021 (6QgxEPvR)

windows-shared-runners-manager-2

shared-windows windows windows-1809

#157328 (1d6b581d)

gitlab-shared-runners-manager-3.gitlab.com

gitlab-org

#2072991 (pVR9XBDq)

gitlab-docker-shared-runners-manager-04

gitlab-org-docker

Слика 4.7: GitLab специфични и подељени покретачи

За успешно спровођење читавог тока, било је потребно инсталирати покретач на удаљеном серверу, на Ubuntu оперативном систему, архитектуре amd64. На том покретачу су инсталиране две верзије подршке за dotnet SDK 3.1 и 5.0 и усаглашене верзије Google Chrome претраживача, Chromedriver-a и NuGet пакета како би се формирало окружење за интеграционе тестове.

Радни ток непрекидне интеграције и непрекидне испоруке развијене апликације се описује .yaml датотеком која је приказана на слици 4.8.

```
stages:
  - build
  - test
  - publish

Building:
stage: build
script:
  - "cd eBanking"
  - "dotnet build"
tags:
  - testing

Testing:
stage: test
script:
  - "cd eBanking/eBanking.Tests"
  - "dotnet test"
  - "cd ../eBanking.IntegrationTests"
  - "dotnet test"
tags:
  - testing

Publishing:
stage: publish
script:
  - "cd eBanking"
  - "dotnet publish"
  - "screen -ls"
  - "screen -X -S gitLab quit"
  - "rm /home/gitlab-runner/builds/skwwGzc8/0/miroslav-misljenovic/eBanking/eBanking/eBanking/bin/Debug/netcoreapp3.1/publish/appsettings.json"
  - "cp /home/gitlab-runner/appsettings.json /home/gitlab-runner/builds/skwwGzc8/0/miroslav-misljenovic/eBanking/eBanking/eBanking/bin/Debug/netcoreapp3.1/publish"
  - "cd /home/gitlab-runner/builds/skwwGzc8/0/miroslav-misljenovic/eBanking/eBanking/eBanking/bin/Debug/netcoreapp3.1/publish"
  - "screen -dmS gitLab dotnet eBanking.dll --urls=http://*:5005/"
tags:
  - testing
```

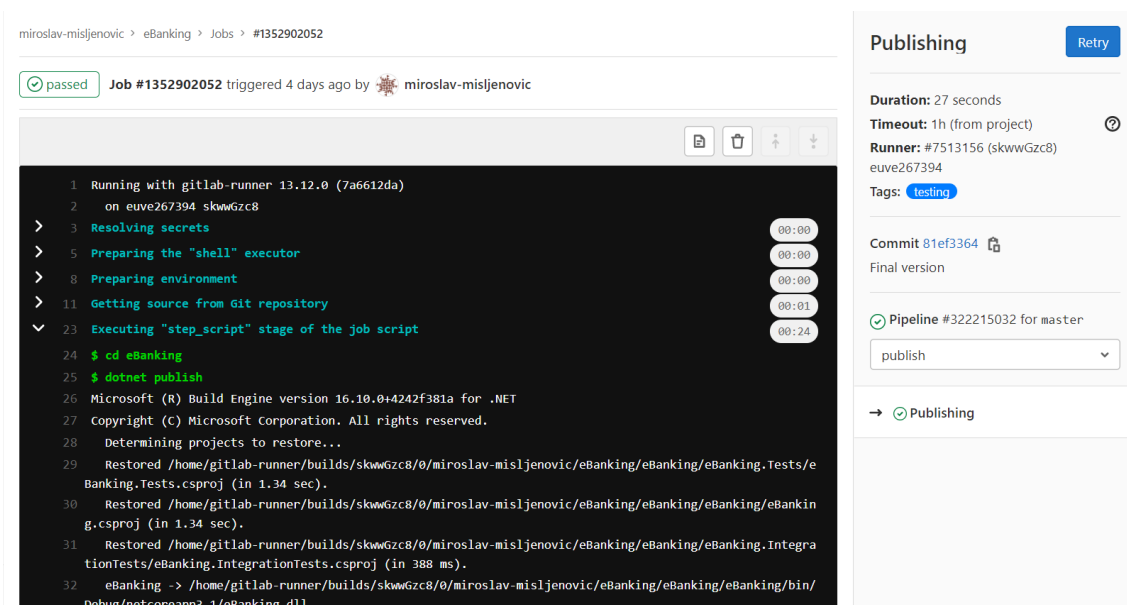
Слика 4.8: GitLab .yaml датотека

У почетку се наводе фазе које се секвенцијално извршавају. На следећу фазу се прелази само ако је претходна успешно завршена. На тај начин се брзо открива где је дошло до грешке и програмеру шаље порука (кроз апликацију, електронском поштом и сл.) о неуспеху. Свака фаза садржи одговарајућу ознаку и скрипт део у коме се наводе жељене наредбе. Такође, свака фаза се извршава на специфичном покретачу и то се постиже навођењем тага изабраног покретача.

Прве две фазе, превођења и тестирања, се извршавају позиционирањем у одговарајући директоријум и навођењем dotnet build, односно dotnet test наредбе. Трећа фаза је фаза испоруке која се спроводи наредбом dotnet publish. Наредбом „screen -X -S gitLab quit” се поништава претходна продукциона верзија, копира се конфигурациона „appsettings.json” датотека, позиционира у продукциони директоријум и наредбом „screen -dmS gitLab dotnet eBanking.dll –urls=http://\*:5005/” покреће нова инстанца апликације, доступна на адреси

## ГЛАВА 4. АЛАТИ ЗА CI/CD

<http://62.75.156.53:5005/>. На слици 4.9 је приказан први део посла који се обавља у трећој фази.




Слика 4.9: GitLab испорука

На сликама 4.10 и 4.11 се види да је цео ток успешно реализован, као и увид у сваку појединачну фазу.


Status	Pipeline	Triggerer	Commit	Stages	Duration
passed	#322215032 latest	miroslav-misljenovic	master -> 81ef3364 Final version	✓ ✓ ✓	00:01:34 4 days ago
passed	#322214150	miroslav-misljenovic	master -> 54e6338d Update .gitlab-ci.yml file	✓ ✓ ✓	00:01:30 4 days ago


Слика 4.10: GitLab радни ток






passed Pipeline #322215032 triggered 4 days ago by  miroslav-misljenovic

## Final version







 3 jobs for `master` in 1 minute and 34 seconds (queued for 11 seconds)

 `latest`

 `81ef3364` 

 No related merge requests found.

Pipeline Needs Jobs 3 Tests 0

Build	Test	Publish
 Building 	 Testing 	 Publishing 

Слика 4.11: GitLab послови

# Глава 5

## Рад на токовима

### 5.1 Имплементација токова

Иницијална замисао овог мастер рада била је да се анализирају функционалности и имплементирају токови CI/CD система Jenkins, GitHub Actions и GitLab CI/CD за апликацију Currency Explorer, као и да се изврши упоредна анализа остварених времена, перформанси и скалабилности ова три система.

На располагању је био рачунар ЛИСС Математичког факултета, доступан на веб адреси `www.liss.matf.bg.ac.rs`, на оперативном систему Windows 10.

Прво је развијан ток помоћу алата Jenkins. Извршена је конфигурација читавог пројекта, убачени неопходни додаци, подешене путање до сервиса и извршних датотека. Такође, направљена је и SQLServer база података са подацима о валутама и попуњена миграцијом са локалног рачунара. Тиме је део непрекидне интеграције, који обухвата превођење и тестирање, био успешно окончан. Реализовано је да део непрекидне испоруке буде лоциран на том рачунару и пуштен у продукцију помоћу програма IIS Manager, на адреси `www.liss.matf.bg.ac.rs:5000`. Дакле, испорука није била на удаљено, већ локално окружење.

Следећи ток је развијан помоћу алата GitHub Actions. Део непрекидне интеграције је успешно реализован и требало је извршити испоруку на удаљени рачунар ЛИСС. Тада је дошло до проблема са везивањем на ЛИСС, јер је одбијана конекција. Покушан је пренос података за продукционо окружење, помоћу протокола SSH и FTP. Иако су урађени сви неопходни кораци, конекција није успевала. Показало се да је ипак била потребна нека верзија оперативног система Windows Server. Због тога је обезбеђено да се испорука

изврши на други сервер. Реч је о серверу на оперативном систему Ubuntu 16.04.6 LTS. Испорука путем FTP протокола, као и покретање апликације је успешно реализовано и доступно на ИП адреси `http://62.75.156.53:5000`. Нагласио бих да је ток реализован на дељеном покретачу.

Алат GitLab CI/CD је коришћен за прављење трећег тока. Са развијањем се почело на дељеном покретачу, док се нису појавили проблеми са интеграционим тестирањем. Наиме, верзије претраживача Google Chrome и Chromedriver-а нису биле усклађене са оним верзијама потребним за апликацију. Покушај да се одговарајуће верзије инсталирају на дељеном покретачу се завршио неуспешно, јер нису биле дозвољене одређене наредбе из терминала, којима би се постигло усаглашавање верзија. Да би се тај проблем разрешио, активиран је специфични покретач, на Ubuntu серверу који је служио као продукционо окружење и за претходни алат. На њему је успешно спроведено превођење, тестирање, прављење продукционе верзије и пуштање апликације у продукцију на адреси `http://62.75.156.53:5005`.

Пошто су три тока реализована на три различита сервера (ЛИСС, дељени сервер алата GitHub Actions и Ubuntu сервер) нема превише смисла упоређивати њихова времена рада, перформансе и скалабилност, како због различите хардверске конфигурације, тако и због различитих капацитета комуникационих канала. Ипак, како би се стекао општи утисак о временима извршавања појединачних фаза, дата је табела 5.1 у којој су упоређена времена ових фаза за GitHub Actions и GitLab CI/CD.

	GitHub Actions	GitLab CI/CD
Превођење	14 секунди	23 секунде
Тестирање	23 секунде	42 секунде
Испорука	121 секунда	
Продукција	6 секунди	27 секунди

Табела 5.1: Времена појединачних фаза токова

## 5.2 Поређење алата GitHub и GitLab

Постоји неколико значајних разлика између алата GitHub и GitLab, како у области непрекидне интеграције и непрекидне испоруке, тако и у осталим, па ће у наставку оне бити дискутоване[16]. У прошлости је приметно да су се одређена својства која су била доступна само у комерцијалним верзијама, због конкуренције и доминантног положаја на тржишту, постепено уграђивала и у бесплатне верзије. Приказаћемо неке од њих:

- складишта,
- непрекидна интеграција/испука,
- животни век развоја софтвера,
- документација за улазне операције,
- заједница.

**Складишта** су раније била велика предност GitLab-а, који је бесплатно омогућавао неограничен број приватних складишта. Од јануара 2019. године, GitHub је омогућио приватна складишта за тимове са максимум три члана. GitLab је у овој функционалности у благој предности, јер је одувек имао могућности неограниченог броја јавних складишта и неограниченог броја чланова по пројекту, док је ту функционалност GitHub омогућио почевши од 14. априла 2020. године. Што се капацитета тиче, GitLab дозвољава укупно 10 гигабајта по налогу, док GitHub нуди 500 мегабајта по складишту.

**Непрекидна интеграција и непрекидна испорука** је била велика предност GitLab-а, јер је користио уграђен CI/CD радни оквир и то бесплатно и то је био један од главних разлога зашто су се програмери одлучивали за (прелазак на) GitLab. Али, у међувремену, GitHub се приближио конкуренцији када је кроз алат Github Actions такође уградио сопствени CI/CD слој. Дакле, више није било потребе за интегрисањем посебног алата за CI/CD, типа Jenkins или CircleCI, како би се омогућила функционалност непрекидне интеграције и непрекидне испоруке, премда је та могућност још увек на располагању. Ипак, пошто је GitLab дуго година нудио CI/CD радни оквир, заузео је позицију једног од најбољих алата у тој области, ако не и најбољи.

Донедавно је GitLab био у великој предности, али га је новембра 2019. сустигао GitHub увођењем алата Actions. Чини се да је од тада GitHub привукао много корисника, јер је од јануара 2020. године направљено више од 40 милиона нових налога. Компаративна предност Github Actions-а је могућност матричног изграђивања (енг. matrix build). Ради се о својству које омогућује да се апликација паралелно тестира за рад на различитим оперативним системима. Због тога што је релативно нов алат, Github Actions је прошао кроз многе актуелне оптимизације. Можемо закључити да су ова два алата данас приближно истих перформанси.

**Животни век развоја софтвера**, који потиче од скраћенице SDLC (енг. Software Development LifeCycle), је својство система где је GitLab у предности у односу на GitHub. Иако и GitHub нуди много сервиса, GitLab покрива све делове, од иницијалне идеје до извршног пројекта. Једна примедба на GitLab је његов кориснички интерфејс који је недовољно обрађен. Иако GitLab нуди најшири спектар алата из ове области, много корисника је на форумима имало примедбе о непотпуним компонентама појединих алата, тј. нису сви алати фино подешени (енг. fine-tuned).

Насупрот томе, GitHub-ов кориснички интерфејс је много интуитивнији и структурно боље прилагођен кориснику. Иако не нуди баш све алате за животни век развоја софтвера, нема бојазни да постоје непотпуно завршене компоненте.

Дакле, избор је остављен програмеру - да ли да користи интуитивно окружење (и тиме изабере GitHub) или да користи окружење које садржи све потребне компоненте (иако недовољно дорађене) за целокупан SDLC на једном месту (и тиме се одлучи за GitLab).

**Документација за улазне операције** је код GitLab-а написана корак по корак и чини да се цео процес уради без много залагања. С друге стране, GitHub нема тако детаљну документацију; доступан је алат GitHub Importer који у потпуности аутоматизује улазне операције. Када је реч о излазним операцијама, оне нису толико добро документоване.

**Заједница** је нешто по чему се GitHub веома истиче. Када једном приступите овој платформи, постајете део јаке заједнице програмера отвореног кода, са преко 40 милиона чланова. Многи чланови помажу једни другима, тако што сугестијама или оптимизацијом доприносе побољшању кода. Такође, може се добити искрена и квалитетна повратна информација. Ово је

најпопуларнија платформа и заједница програмера.

GitLab такође има квалитетну заједницу програмера. Организују се догађаји на којима се повезују програмери који доприносе отвореном коду других. Један од таквих је Општи месец волонтера (енг. Global Volunteer Month), који се одржава сваког децембра и тада се подржавају сви волонтери да развијају софтвер за локалне или друге друштвено одговорне заједнице.

### О верзијама алата

Познато је да постоји традиција GitLab-а да сваког 22. у месецу објављује нове верзије, у којима су додата нова својства и исправљене грешке и недостаци претходних верзија. Сличне механизме имају и други произвођачи, само је временски интервал дужи. На први поглед, ово изгледа повољно, али временом праћење тих измена постаје врло сложено. Такође, нека својства се избацују, па треба извршити одговарајуће измене у коду. Тако, на пример, Jenkins за неке додатке има дефинисано време до када ће они важити (докле ће их подржавати и одржавати) и унапред упозорава програмера о том ограничењу.

Један од начина да се помогне у развоју апликације је да се фиксирају бројеви верзија које апликација користи. Јасно, тиме се делимично нарушава квалитет и отежава имплементација будућих верзија, које могу бити оптимизоване и прилагођене новим захтевима.

## 5.3 Поређење GitHub Actions/GitLab CI/CD и Jenkins-а

У претходном делу смо видели да су GitHub Actions и GitLab CI/CD засновани на сличним принципима и да су мале разлике у реализацији. Због тога ћемо у даљем тексту приказати разлику између Jenkins-а и њих. То се може видети у табели 5.2. Приказана су карактеристична својства, код којих се може видети да су или подједнако добро реализована или је знатна разлика у реализацији.

Својство	Jenkins	GitHub/GitLab
Паралелизам	Дозвољава се паралелно превођење, али сви кораци деле исто окружење, тако да може доћи до проблема приликом дељења заједничких ресурса, као што је, на пример, систем датотека.	Дозвољава се матрична изградња, често и за различите оперативне системе истовремено.
Градивни елемент	Додаци (енг. Plugin)	Акције/Контејнери
Подршка контејнеризације	Делимично подржава рад са контејнерима. Подразумевано, сва изградња се извршава у истом окружењу на истом серверу, што може довести до већег броја проблема и генерално није добра пракса.	У потпуности подржавају рад са контејнерима.
Подршка за управљање члановима/пројектима и додељивање рола/дозвола	У пракси, подразумева се да је једна особа одговорна за цео поступак конфигурације и управљања пројектом. Недостају својства тимске сарадње, која су доступна у сличним системима. Није омогућен лак начин да се одреди ко је одговоран за неуспешан корак изградње, тестирања или испоруке, чак иако придружени систем за контролу верзија садржи ту информацију.	Лако се проверава који члан (групе) је урадио активацију појединачног корака, опционо и на којој грани. У подешавањима се могу дефинисати роле или дозволе/ограничења.
Начин рада	Хостован у локалу	SaaS (eng. Software-As-A-Service)
Изградња токова	Омогућује богату подршку за изградњу токова и њихову визуелизацију, било кроз конфигурациони Jenkinsfile или кроз веб интерфејс.	Графички приказ је изузетно интуитиван и садржајан; приказан је у одељцима Workflows (GitHub Actions) односно Pipelines (GitLab CI/CD)
Заједница корисника	Захваљујући великој популарности, доступан је велики избор додатака, на адреси <a href="https://plugins.jenkins.io/">https://plugins.jenkins.io/</a> .	Доступан је велики број јавно доступних акција за GitHub Actions односно контејнера за GitLab CI/CD.
Интегрисање са осталим алатима	Омогућена интеграција са разним алатима за комуникацију, за контролу верзија, (нпр. Slack, Github), као и додаци за обавештења путем електронске поште.	Омогућена интеграција са токовима других произвођача (AWS, Azure, Zeit, Kubernetes и многи други).

Табела 5.2: Поређење GitHub Actions/GitLab CICD и Jenkins-a

Посебно треба истаћи својство матричне изградње GitHub Actions-a и GitLab CICD-a за више оперативних система истовремено, чиме се омогућује прецизирање конкретних верзија оперативних система и постиже високи паралелизам у развоју и значајна уштеда времена. Ово својство је изузетно популарно, па и Jenkins развија такву функционалност и омогућује све више додатака у том правцу.

## Глава 6

# Закључак

Основни циљ овог рада је да се покаже цео поступак развоја једне веб апликације и аутоматизације процеса непрекидне интеграције и непрекидне испоруке. Дакле, идеја је била да се ”једним кликом” покрене низ корака којима се изворни кôд преводи, анализира јединичним и интеграционим тестовима и испоручи на продукционо окружење; додатно, преузимају се и инсталирају неопходни пакети и додаци како би читав процес био успешно спроведен.

Од свих алата за непрекидну интеграцију и непрекидну испоруку, одлучено је да се токови спроведу уз помоћ три најпопуларнија алата Jenkins, GitHub Actions и GitLab CI/CD, користећи њихове бесплатно доступне верзије.

Применом ових алата, уочено је да су GitHub Actions и GitLab CI/CD слично концептуално осмишљени, док се Jenkins у многим елементима доста разликује. То је последица тога што је Jenkins присутан доста дуже и да се проширивао и прилагођавао новим програмским језицима углавном кроз обогаћивање библиотекâ додатака. Такође, савремена пракса да се цео изворни кôд чува на складишту, није била толико заступљена у време када је Jenkins достигао врхунац популарности, па је користио системе за контролу верзија других произвођача, док су касније GitHub и GitLab природно омогућили својим корисницима систем аутоматизације процеса непрекидне интеграције и непрекидне испоруке. Стога је одлучено да GitHub Actions и GitLab CI/CD буду упоређени међусобно, а затим обједињено упоредимо са Jenkins-ом.



Резултат рада је да се цео CI/CD ток, за апликацију Currency Explorer, успешно спровео и реализовао, помоћу алата Jenkins, на продукционом окружењу Windows оперативног система, док се, помоћу алата GitHub Actions и GitLab CI/CD, ток такође успешно изградио и испоручио на продукционо окружење Ubuntu оперативног система.

Уобичајено се као основна мера поређења користи време извршавања тока. Упркос проблемима са одређивањем времена, показује се да је време приближно једнако за сваки од ових алата. Надам се да се не би добила значајнија временска разлика и у случајевима сложенијих апликација.

Интересантно би било дефинисање додатних интеграционих тестова, нарочито на апликацији са више функционалности, који би могли довести до још једноставнијег проналажења грешака и повећане поузданости апликације. Такође, уколико је апликација значајно већег обима, таква да је за њен развој потребан тим програмера, било би омогућено извршити поређење у којој мери је подржана сарадња чланова тима.

Постоје и други алати за овакву аутоматизацију, па би упознавање са њима и поређење са ова три алата додатно олакшало избор најбољег алата за дату апликацију. Критеријуми могу бити време извршавања, доступни ресурси, дељени или локални покретачи, једноставност коришћења алата и сл.

# Глава 7

## Додаци и прилози

У овом поглављу су дати интересантни делови кода по избору аутора. Изабрани су:

- експоненцијално изглађивање,
- контролер Investment,
- јединични тестови за рад са стринговима,
- интеграциони xUnit тест,
- интеграциони Selenium тестови.

**Експоненцијално изглађивање** је једна од три методе предвиђања курса валута (преостале две су познате методе линеарног предвиђања и линеарне регресије). Она је мање позната, али даје квалитетне оцене, које зависе од вредности изабраног параметра  $\alpha$  из интервала  $[0, 1]$ . Број тачака који се узима у разматрање се одређује тако да се узима најмањи број  $k$ , за који важи  $(1 - \alpha)^k < 10^{-5}$ . Очигледно, број тачака који се узима је променљив и зависи од параметра  $\alpha$ . Сâмо израчунавање нове предвиђене вредности се остварује по следећој формули:

$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1 - \alpha)y_{T-1} + \alpha(1 - \alpha)^2 y_{T-2} + \dots$$

где су  $y_T, y_{T-1}, y_{T-2}, \dots$  познате (или оцењене) вредности.

На слици 7.1 је приказан део кода којим се ова метода имплементира. Пошто се на графикону приказују две валуте, у коду постоје два низа вредности. За формирање низа  $\alpha$  параметара се позива засебна функција. Вредности које се уписују на графикон су помножене са сто због лепшег приказа; ово је битно за валуте чије су вредности курсава приближно једнаке, док за оне са великом разликом вредности нема много ефекта.

```

139     if (investmentChart.RadioResponse.Equals("ExponentialSmoothing"))
140     {
141         TimeSpan span = investmentChart.EndingDate.Subtract(DateTime.Now);
142
143         List<double> parameters = AlphaParameters(investmentChart.Alpha);
144         int countParameters = parameters.Count();
145         if (countParameters > data1.Count())
146         {
147             countParameters = data1.Count();
148         }
149
150         var values1 = data1.TakeLast(countParameters).Reverse().ToList();
151         var values2 = data2.TakeLast(countParameters).Reverse().ToList();
152         for (int i = 1; i < span.Days; i++)
153         {
154             double firstValue = 0;
155             double secondValue = 0;
156             for (int j = 1; j < countParameters; j++)
157             {
158                 firstValue += parameters.ElementAt(j - 1) * values1[j - 1].Rate;
159                 secondValue += parameters.ElementAt(j - 1) * values2[j - 1].Rate;
160             }
161             values1.Insert(0, new BusinessModels.CurrencyRateHistory { Rate = firstValue });
162             values2.Insert(0, new BusinessModels.CurrencyRateHistory { Rate = secondValue });
163
164             _data.Add(new ChartDataValues()
165             {
166                 Date = data1[previousValues - 1].Date.AddDays(i).ToString("dd.MM.yyyy"),
167                 FirstCurrencyValue = firstValue * 100,
168                 SecondCurrencyValue = secondValue * 100
169             });
170         }

```

Слика 7.1: Експоненцијално изглађивање

**Контролер Investment** користи методе сервиса `CurrencyRateService` и `ChartService`, како би прилагодио и приказао жељени графикон са претходним и израчунатим вредностима. На слици 7.2 је приказано формирање иницијалног графикона, који узима валуте Венецуелански боливар и наш динар, вредности курса валута из претходних 70 дана и захтев за нових 30 вредности помоћу линеарне регресије.

```

15 public InvestmentController(IChartService chartService, ICurrencyRateService currencyRateService)
16 {
17     _chartService = chartService ?? throw new ArgumentException(nameof(chartService));
18     _currencyRateService = currencyRateService ?? throw new ArgumentException(nameof(currencyRateService));
19 }
20 public void PopulateCurrencies()
21 {
22     var currencies = _currencyRateService.GetCurrencyList();
23     ViewData["Currencies"] = new SelectList(currencies.OrderBy(x => x.Name), "Id", "Name");
24     ViewData["Today"] = DateTime.Today.ToString("yyyy-MM-dd");
25 }
26 public IActionResult Index()
27 {
28     PopulateCurrencies();
29     var currencies = _currencyRateService.GetCurrencyList();
30     int currency1 = currencies.First(a => a.Name == "VEF").Id;
31     int currency2 = currencies.First(a => a.Name == "RSD").Id;
32
33     var investmentChart = new InvestmentChart
34     {
35         FirstCurrency = currency1,
36         SecondCurrency = currency2,
37         StartingDate = DateTime.Now.AddDays(-70),
38         EndingDate = DateTime.Now.AddDays(30),
39         RadioResponse = "LinearRegression",
40         Alpha = 0.5
41     };
42     ViewData["ChartJson"] = _chartService.PrepareInvestmentChart(investmentChart);
43     return View(investmentChart);
44 }

```

Слика 7.2: Реализација контролера Investment

Јединични тестови за рад са стринговима, приказани на слици 7.3, проверавају да ли се добијају тачне вредности или одговарајући изузеци приликом операција са стринговима. Тестирају се:

- тачан датум,
- неисправна преступна година,
- погрешан датум,
- празан стринг.

```
9 [Fact]
0 references
10 public void TestValidString()
11 {
12     DateService ds = new DateService();
13     DateTime dt = ds.StringToDateTime("1993-06-02");
14     Assert.Equal(new DateTime(1993, 06, 02), dt);
15 }
16
17 [Fact]
0 references
18 public void Test29February2021_ThrowsRangeException()
19 {
20     DateService ds = new DateService();
21     Assert.Throws<ArgumentOutOfRangeException>(() => ds.StringToDateTime("2021-02-29"));
22 }
23
24 [Fact]
0 references
25 public void TestInvalidString_ThrowsRangeException()
26 {
27     DateService ds = new DateService();
28     Assert.Throws<ArgumentOutOfRangeException>(() => ds.StringToDateTime("1993-13-02"));
29 }
30 [Fact]
0 references
31 public void TestEmptyString_ThrowsFormatException()
32 {
33     DateService ds = new DateService();
34     Assert.Throws<FormatException>(() => ds.StringToDateTime(""));
35 }
```

Слика 7.3: Јединично тестирање стрингова

**Интеграциони xUnit тест** који се односи на контролер који формира почетну страну дат је на слици 7.4. Овде се читава страна са дате адресе прочита као стринг и проверава се да ли се у њему налазе одговарајући наслов и тражених пет валута.

```

1  using Xunit;
2  using System.Net.Http;
3
4  namespace eBanking.IntegrationTests
5  {
6      1 reference
7      public class HomeControllerIntegrationTests : IClassFixture<TestingWebAppFactory<Startup>>
8      {
9          private readonly HttpClient _client;
10         0 references
11         public HomeControllerIntegrationTests(TestingWebAppFactory<Startup> factory)
12         {
13             _client = factory.CreateClient();
14         }
15
16         [Fact]
17         0 references
18         public void HomeTest()
19         {
20             var response = _client.GetAsync("http://liss.matf.bg.ac.rs:5000/").Result;
21             response.EnsureSuccessStatusCode();
22             var responseString = response.Content.ReadAsStringAsync().Result;
23
24             Assert.Contains("Currency Explorer", responseString);
25             Assert.Contains("RSD", responseString);
26             Assert.Contains("HKD", responseString);
27             Assert.Contains("BRL", responseString);
28             Assert.Contains("JPY", responseString);
29             Assert.Contains("RUB", responseString);
30         }
31     }

```

Слика 7.4: Интеграциони тест Номе контролера

**Интеграциони Selenium тестови** су примењени за аутоматизацију провере рада графичког окружења. На слици 7.5 приказана су два теста, реализована за прегледач Google Chrome и одговарајући избор његових опција.

Први тест проверава наслов и поднаслов који одговарају страни где је приказана функционалност ExchangeRate.

Други тест уписује одређени датум на ту исту страну, учитава све вредности валута за тај дан и проверава да ли изабрана валута има одговарајућу вредност.

```

11  public AutomatedUITests()
12  {
13      ChromeOptions options = new ChromeOptions();
14      options.AddArgument("--no-sandbox");
15      options.AddArgument("--disable-dev-shm-usage");
16      options.AddArgument("--headless");
17      _driver = new ChromeDriver(options);
18  }
19  [Fact]
20  public void ExchangeRateIndexTest()
21  {
22      _driver.Navigate()
23          .GoToUrl("http://liss.matf.bg.ac.rs:5000/ExchangeRate");
24      Assert.Equal("Index - Currency Explorer", _driver.Title);
25      Assert.Contains("Insert for which date you want the exchange rate list " +
26          "in format YYYY-MM-DD:", _driver.PageSource); _driver.Navigate();
27  }
28
29  [Fact]
30  public void ExchangeRateGenerateListTest()
31  {
32      _driver.Navigate()
33          .GoToUrl("http://liss.matf.bg.ac.rs:5000/ExchangeRate");
34      _driver.FindElement(By.Id("Date")).SendKeys("2021-04-01");
35      _driver.FindElement(By.Id("Load"))
36          .Click();
37      Assert.Contains("HTG", _driver.PageSource);
38      Assert.Contains("94.236046", _driver.PageSource);
39  }

```

Слика 7.5: Selenium аутоматизовани тест графичког окружења

# Библиографија

- [1] Kent Beck. *Test-Driven Development By Example*. Addison-Wesley Professional, 2002.
- [2] Miroslav Mišljenović. Currency explorer, 2021. <http://github.com/miroslav-misljenovic/eBanking>.
- [3] xUnit.net, 2021. <https://xunit.net/>.
- [4] Selenium, 2021. <https://www.selenium.dev/>.
- [5] Roy Osherove. *The Art of Unit Testing, Second Edition*. Manning Publications, Shelter Island, NY 11964, 2014.
- [6] Exchange Rates API, 2021. <https://exchangeratesapi.io/>.
- [7] FusionCharts, 2021. <https://www.fusioncharts.com/>.
- [8] Continuous Deployment or Continuous Delivery? | When To Release, 2021. <https://www.youtube.be/mBzDPRgue6s>.
- [9] Martin Fowler. Continuous Integration. 2006.
- [10] Jenkins, 2021. <https://www.jenkins.io/>.
- [11] Javaone conference: Duke's choice awards winners for 2008. <https://www.oracle.com/java/technologies/>.
- [12] GitHub, 2021. <https://github.com/>.
- [13] Gary Grossgarten. garygrossgarten/github-action-scp, 2021. <https://github.com/garygrossgarten/github-action-scp/blob/master/.github/workflows/scp-example-workflow.yml>.
- [14] GitLab, 2021. <https://gitlab.com/>.



- [15] GitLab. The remote work report by gitlab: The future of work is remote, March 2020. <https://page.gitlab.com/rs/194-VVC-221/images/the-remote-work-report-by-gitlab.pdf>.
  
- [16] Momčil Kojčev. Gitlab vs github: Repositories, ci, deployment, devops, pricing, and documentation, 2020. <https://hackernoon.com/gitlab-vs-github-repositories-ci-deployment-devops-pricing-documentation-and-more-analysis-12qc32n9>.

# Биографија аутора

**Мирослав Ђуро Мишљеновић** рођен је 2. јуна 1993. године, у Београду. Уписао је основну школу „Милица Павловић” 2000. године и завршио са одличним успехом и дипломом Вука Караџића.

Потом завршава природно-математички смер XIV београдске гимназије са одличним успехом.

Математички факултет Универзитета у Београду, смер Информатика, завршава 2018. године са просечном оценом 7,10. Након тога уписује мастер студије на истом смеру и факултету.

Од ваннаставних активности истакао се у различитим областима.

Показао је велико ангажовање као волонтер Црвеног крста почевши од 2010. године.

Поседује мајсторски појас борилачке вештине теквондо.

У периоду од 2002 - 2008. године завршио је музичку школу „Јосип Славенски”, одсек гитара; освојио је друго место на Републичком такмичењу 2004. године.