



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У  
НОВОМ САДУ

---



Никола Трајковић


**Дизајн и реализација дистрибуираног  
*CI/CD* алата са подршком за *2FA*,  
нотификације и корисничку аналитику**

ЗАВРШНИ РАД

Основне академске студије

Нови Сад, 2025.



	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	<b>ЗАДАТАК ЗА ЗАВРШНИ РАД</b>	Датум:

(Податке уноси предметни наставник - ментор)

Студијски програм:	Софтверско инжењерство и информационе технологије		
Студент:	Никола Трајковић	Број индекса:	SV45/2021
Степен и врста студија:	Основне академске студије		
Област:	Софтверско инжењерство и информационе технологије		
Ментор:	Бранко Милосављевић		
<p>НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:</p> <ul style="list-style-type: none"> <li>- проблем – тема рада;</li> <li>- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;</li> </ul>			

### НАСЛОВ ЗАВРШНОГ РАДА:

<p>Дизајн и реализација дистрибуираног CI/CD алата са подршком за 2FA, нотификације и корисничку аналитику</p>
--

### ТЕКСТ ЗАДАТКА:

<p>Анализирати концепте, архитектуру и начин рада постојећих CI/CD алата. Анализирати технологије за имплементацију двофакторске аутентификације. Анализирати механизме за интеграцију CI/CD система са системима за контролу верзија. Дефинисати архитектуру и имплементирати CI/CD систем који омогућава извршавање задатака на више чворова помоћу дистрибуираних агената, поседује двофакторску аутентификацију, нотификације вођене догађајима и кориснички интерфејс за надзор рада система. Документовати решење и дискутовати добијене резултате.</p>
---

Руководилац студијског програма:	Ментор рада:

<p>Примерак за: <input type="checkbox"/> - Студента; <input type="checkbox"/> - Ментора</p>
---



---

# Садржај

---

1	CI/CD алати .....	1
1.1	<i>Jenkins</i> .....	1
1.2	<i>GitHub Actions</i> .....	2
1.3	<i>GitLab CI/CD</i> .....	2
2	Функције система .....	3
2.1	Функционални захтеви .....	3
2.2	Нефункционални захтеви .....	4
3	Архитектура система .....	7
3.1	Мастер компонента .....	7
3.2	Агент компонента .....	9
3.3	База података .....	9
4	Имплементација система .....	11
4.1	Безбедносни механизми .....	11
4.2	Управљање корисницима и агентима .....	14
4.3	Управљање пословима ( <i>Job</i> -овима) .....	15
4.4	Систем нотификација .....	21
4.5	Интеграције са спољним системима .....	23
4.6	Корисничка аналитика .....	24
5	Закључак .....	25
	Биографија .....	31
	Литература .....	33

---

У индустријама као што су авијација, аутомобилска индустрија и телекомуникације, квалитет и поузданост система имају велики значај. Грешке у овим областима могу бити веома скупе или критичне. Због тога стандарди захтевају јасан преглед функционалних и нефункционалних захтева и њихову потпуну покривеност тестовима. Таква пракса омогућава већу безбедност и поузданост софтвера.

Континуирана интеграција и испорука (CI/CD) су темељ савременог софтверског инжењерства. Оне омогућавају бржу и поузданију испоруку апликација. Ипак, често не постоји јасна веза између захтева система и тестних случајева. То доводи до смањене покривености и повећава ризик од пропуштања критичних сценарија. Потребни су алати који повезују управљање захтевима и тестирање. На тај начин сви подаци се налазе на једном месту и избегава се дуплирање. Комуникација између тимова је боља, а захтеви и тестови су лако повезани и прегледни.

Овај рад описује развој CI/CD алата заснованог на мастер-агент архитектури. Алат је намењен сложеним и безбедносно осетљивим окружењима. Омогућава централизовано управљање и координацију аутоматизованих радних токова у дистрибуираном систему. Тако се постижу већа скалабилност и флексибилност. Подржани су кораци као што су клонирање складишта, пренос датотека, извршавање наредби и генерисање извештаја. Безбедност је обезбеђена двофакторском аутентикацијом (2FA) и шифрованом комуникацијом између чворова. Алат садржи аналитике за праћење перформанси и нотификације преко *Microsoft Teams*-а, *Slack*-а и саме апликације.

Циљ рада је да прикаже архитектуру, безбедносне механизме и аналитичке могућности развијеног алата. Посебна пажња посвећена је практичној примени у организацијама које захтевају високу поузданост и контролу процеса. Рад показује да савремен и интегрисан приступ аутоматизацији може унапредити развој и испоруку софтвера, чинећи их ефикаснијим, прегледнијим и лакшим за одржавање.

Развој софтвера данас захтева брзу и поуздану испоруку нових верзија. Због тога су настали бројни CI/CD алати који аутоматизују процес изградње, тестирања и постављања апликације. Њихов циљ је да открију грешке у раној фази и омогуће континуирани развој.

### 1.1 Jenkins

*Jenkins* је један од најраспрострањенијих CI/CD алата отвореног кода. Његова архитектура се заснива на мастер-агент моделу. У овом моделу мастер управља извршавањем послова и распоређује их на агенте. Агенти могу бити покренути локално или на удаљеним серверима.

Агенти извршавају задатке као што су компилација, тестирање и распоред (*deployment*) апликације. Оваква архитектура омогућава расподелу оптерећења и паралелно извршавање послова. Тиме се побољшавају перформансе и скалабилност система.

Комуникација између мастера и агента у *Jenkins*-у се одвија преко *Java Network Launch Protocol* (JNLP) или SSH протокола. JNLP омогућава покретање агента као *Java Web Start* апликације. SSH протокол се користи за удаљене агенте и омогућава сигурнију везу [1].

Предности *Jenkins*-а су велика флексибилност и подршка за више од хиљаду *plugin*-а. Мане су сложено одржавање, ручна конфигурација и потреба за добрим познавањем система. Безбедност зависи од спољних додатака и често захтева додатну конфигурацију.

---

## 1.2 GitHub Actions

*GitHub Actions* је део *GitHub* платформе и нуди интегрисано CI/CD решење. У овом систему не постоји класичан мастер-агент однос. Уместо тога, користи се концепт *Runner*-а. Они представљају извршне јединице сличне агентима. *Runner*-и самостално преузимају послове са *GitHub* сервера и извршавају их. Архитектура је једноставнија, али са мањом контролом над процесима.

*Runner*-и могу бити *GitHub Hosted* или *Self-Hosted* [2]. *GitHub Hosted* је *cloud* инстанца коју обезбеђује *GitHub*. Мастер логика је имплицитно интегрисана у *GitHub* платформу. Она управља оркестрацијом и надзором извршавања радних токова.

Радни токови се дефинишу у *YAML* датотекама унутар репозиторијума. Извршавање радних токова покреће се аутоматски на основу различитих догађаја. Најчешћи догађаји су *push*, *pull request* или креирање новог издања. Сваки радни ток који се извршава на *GitHub Hosted* окружењу покреће се у изолованом виртуелном окружењу. Оваквим приступом се повећава безбедност и стабилност извршавања.

Комуникација се одвија преко *HTTPS REST API*-ја. *Runner*-и периодично шаљу захтеве ка *GitHub* серверу и преузимају послове за извршавање. Овај приступ омогућава сигурну комуникацију, али захтева сталну конекцију.

Предност овог приступа је једноставна конфигурација и чврста интеграција са репозиторијумом кода. Мана је ограничена контрола над инфраструктуром и мања могућност прилагођавања сложеним системима.

## 1.3 GitLab CI/CD

*GitLab CI/CD* интегрише читав *DevOps* процес у једну платформу. Његова архитектура користи мастер-агент модел. *GitLab Server* има улогу мастера, а *GitLab Runner* делује као агент. Мастер управља дефинисаним радним током CI/CD процеса и шаље послове *Runner*-има.

*Runner*-и могу бити локални, удаљени или у *Docker* и *Kubernetes* окружењу. Они преузимају посао од мастера и извршавају задате кораке, као што су *build*, *test* и *deploy*. Након завршетка рада, резултате враћају мастеру. Оваква архитектура омогућава истовремено извршавање више послова. Такође омогућава бољу контролу приступа и једноставније скалирање система.

Комуникација између *GitLab Server*-а и *Runner*-а одвија се преко *HTTP(S)* протокола. *Runner*-и активно контактирају *GitLab Server* преко *API*-ја и преузимају послове. Сва комуникација је шифрована путем *TLS*-а, што обезбеђује сигурност података [3].

*GitLab CI/CD* је стабилан систем, али за велике пројекте захтева снажну инфраструктуру и пажљиво подешавање.



У овом поглављу описане су функционалности и технички захтеви система *Test Hub Mini* (назив развијеног система). Систем је осмишљен као дистрибуирани CI/CD алат који омогућава управљање, извршавање и надгледање радних токова у реалном времену.

Архитектура система заснива се на мастер-агент моделу. Мастер је представљен као централизован систем и координише извршавањем послова. Агенти су извршне јединице задужене за обраду задатака. Систем подржава више типова корака у радном току, као што су клонирање складишта, преузимање и отпремање датотека, извршавање скрипти и генерисање извештаја.

Циљ система је да обезбеди јединствено, сигурно и прошириво решење за континуирану интеграцију и испоруку у дистрибуираним окружењима.

### 2.1 Функционални захтеви

Функционални захтеви дефинишу могућности и понашање система у оквиру дистрибуиране CI/CD архитектуре. Главни циљ је да се омогући стабилан ток рада који подржава:

- управљање корисницима,
- агентима,
- пословима,
- извршавањем,
- аналитиком и
- интеграцијом са спољним сервисима.

#### 2.1.1 Управљање корисницима

Управљање корисницима представља један сегмент система. При иницијалном подизању платформе, систем креира супер администратора. Подаци о овом налогу чувају се у конфигурационом фајлу. Супер администратор може креирати нове администраторе и обичне кориснике. Приликом креирања новог налога систем, генерише иницијалну лозинку која се може променити. Корисницима је омогућено ажурирање личних података, укључујући и промену профилне слике. Безбедност приступа систему обезбеђена је двофакторском аутентикацијом (2FA), преко *Google Authenticator*-а или *Microsoft Authenticator*-а.

#### 2.1.2 Управљање агентима

Управљање агентима представља једну од кључних функционалности система. Систем подржава креирање, измену, брисање и преузимање агента. Сваки агент је задужен за преузимање послова које дефинише мастер и извршавање њихових корака. Систем омогућава лак надзор и управљање статусима агента.

#### 2.1.3 Управљање пословима (*Job*-овима)

Управљање пословима чини једну од основних функционалности система. Сваки посао (*job*) представља радни ток CI/CD процеса и може се састојати од више корака (*step*-ова). Корисници могу креирати нове послове, мењати или брисати постојеће. Приликом покретања посла могу се подесити различити параметри који утичу на извршавање. Систем омогућава праћење извршавања у реалном времену, као и увид у историју свих претходних извршавања. Свака извршена инстанца садржи детаљан приказ статуса појединачних корака и логове који се могу преузети појединачно. У случају да је током извршавања посла дошло до отпремања датотека, систем омогућава преглед и преузимање отпремљених артефаката.

### 2.1.4 Нотификације и праћење статуса

Систем поседује развијен механизам за обавештавање о статусима послова. За сваку промену статуса, систем шаље нотификацију са информацијом о тренутном стању извршавања. Подржане су три врсте нотификација: *Microsoft Teams*, *Slack* и *In-App* (унутар саме апликације). Корисници могу изабрати за које послове желе да примају обавештења и које типове статуса желе да прате. Подешавање обавештења односи се на *In-App* нотификације, које се могу укључивати или искључивати појединачно за сваки посао и сваки статус. За *Slack* и *Microsoft Teams* нотификације подешавање се врши приликом креирања посла. Корисник сам бира да ли ће посао слати обавештења на ове платформе.

### 2.1.5 API и интеграције

Ради лакше интеграције са спољним системима, алат подржава генерисање и брисање API кључева. API кључеви омогућавају сигурну комуникацију са другим сервисима. Поред тога, систем подржава *webhook* интеграције за *GitHub* и *GitLab*, што омогућава аутоматско покретање одређених послова на основу активности у репозиторијуму.

### 2.1.6 Аналитика и извештавање

Систем садржи интегрисани аналитички модул који омогућава праћење активности и понашања корисника у реалном времену. Прикупљају се подаци о географском пореклу корисника, типовима уређаја које користе и најчешћим интеракцијама у систему.

## 2.2 Нефункционални захтеви

### 2.2.1 Безбедност

Систем треба да обезбеди висок ниво заштите података и комуникације између компоненти. Сви пренети подаци морају бити шифровани, а приступ систему ограничен само овлашћеним корисницима. Комуникација између мастера и агента мора бити заснована на сигурним протоколима који гарантују енкрипцију, проверу идентитета и интегритет порука. Поред тога, потребно је осигурати контролу приступа и заштиту спољних интерфејса од неовлашћених захтева.

### 2.2.2 Поузданост

Систем мора бити отпоран на грешке и обезбедити непрекидан рад чак и у случају отказа појединих компоненти. Уколико дође до прекида комуникације или пада агента, остале компоненте морају наставити са радом без утицаја на целокупан процес. Подаци о извршавању послова и статусима морају се чувати на начин који спречава њихов губитак или оштећење.

### 2.2.3 Скалабилност

Архитектура система треба да подржи једноставно проширивање без значајних измена у постојећој структури. Мора бити омогућено додавање нових агената и обрада већег броја послова без смањења перформанси. Систем треба да функционише једнако поуздано у мањим и већим окружењима, уз могућност динамичког прилагођавања оптерећењу.

### 2.2.4 Перформансе

Систем треба да омогући ефикасно извршавање послова и оптимално коришћење ресурса. Обрада података и комуникација између компоненти морају се одвијати без кашњења које би утицало на рад корисника. Распоређивање послова мора бити организовано тако да се избегне преоптерећење појединих чворова и обезбеди равномерна искоришћеност ресурса.

### 2.2.5 Употребљивост

Кориснички интерфејс треба да буде једноставан, прегледан и интуитиван. Све кључне функционалности морају бити лако доступне, а приказ података јасан и разумљив. Систем треба да омогући корисницима

лако праћење статуса послова, нотификација и аналитике у реалном времену, као и прилагођавање приказа сопственим потребама.

#### **2.2.6 Проширивост**

Систем је конципиран тако да се лако може проширити новим функционалностима без већих измена у постојећем коду. Могуће је додати нове типове корака у радним токовима, интеграције са другим сервисима или нове механизме аутентикације. Ова особина омогућава дугорочно одржавање и прилагођавање специфичним потребама организације.

#### **2.2.7 Одрживост и проширивост**

Систем мора бити дизајниран тако да омогући лако одржавање и надоградњу. Код и архитектура треба да буду организовани тако да је додавање нових функционалности могуће без значајних измена постојећег решења. Документација мора бити свеобухватна и ажурна, како би се олакшала будућа развојна и интеграциона унапређења.



На Слици 1 приказана је архитектура система која се заснива на мастер-агент моделу. Мастер представља централну компоненту која управља свим процесима у систему. Он се повезује са пратећим сервисима, као што су база података (*PostgreSQL*), систем за кеширање (*Redis*), складиште артефаката (*MinIO*) и интерфејс за корисничку комуникацију. *Nginx* делује као посредни слој који обезбеђује сигуран приступ и усмеравање захтева ка мастер серверу. Агенти се повезују са системом преко сигурне *WebSocket* везе и извршавају задатке које мастер дефинише.

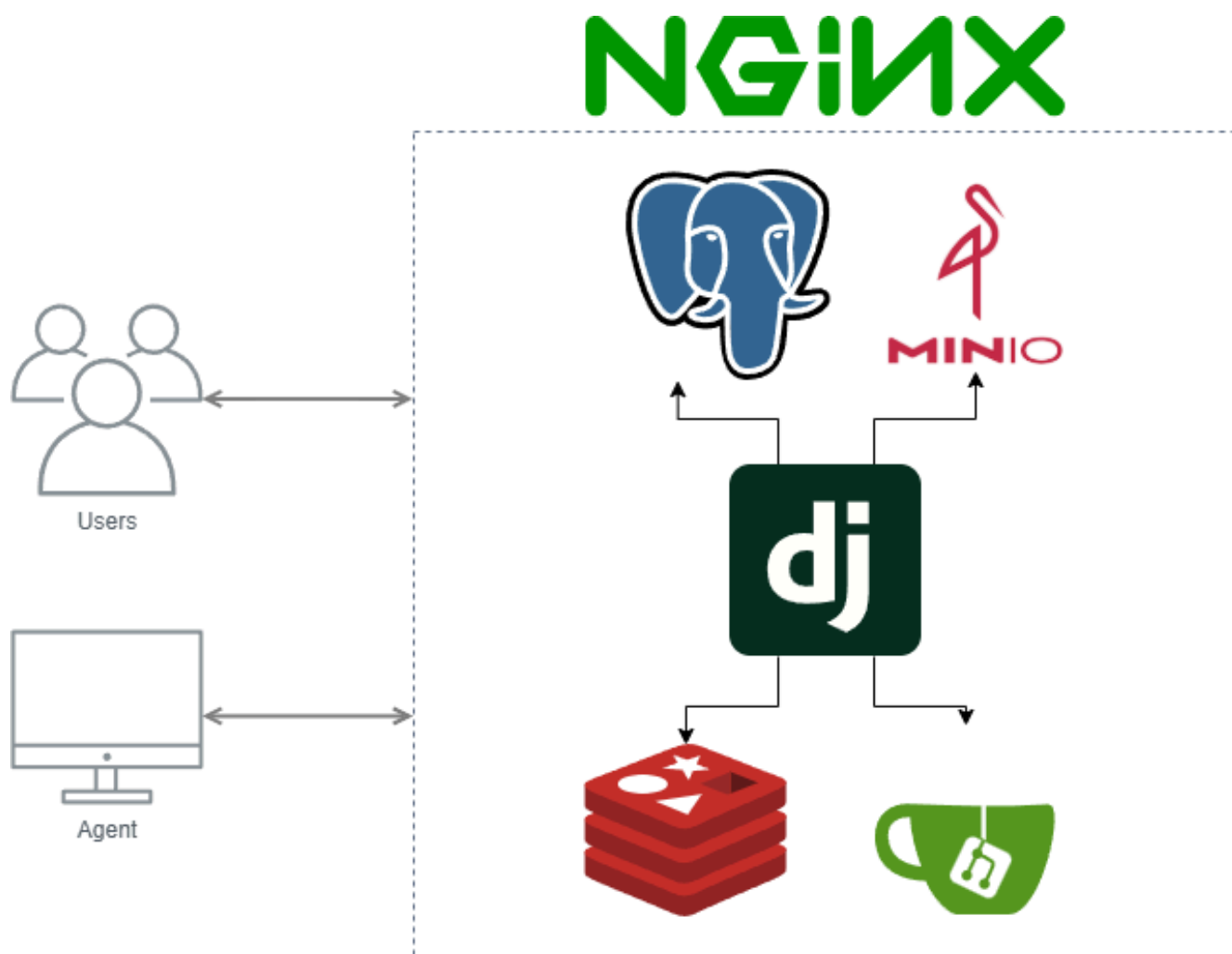
На слици 2 приказан је механизам комуникације између мастер-а и агената. Комуникација се заснива на *WebSocket* протоколу који омогућава двосмерну размену података у реалном времену. За потребе рада система користе се два одвојена *WebSocket* канала. Први канал служи за слање команди и периодичних сигнала (*heartbeat*). Други канал користи за пренос логова и извештаја о извршавању послова. Комуникација је подељена у два канала како би се избегло загушење и осигурао стабилан проток података. Оваква организација обезбеђује поуздану синхронизацију и непрекидан рад система.

### 3.1 Мастер компонента

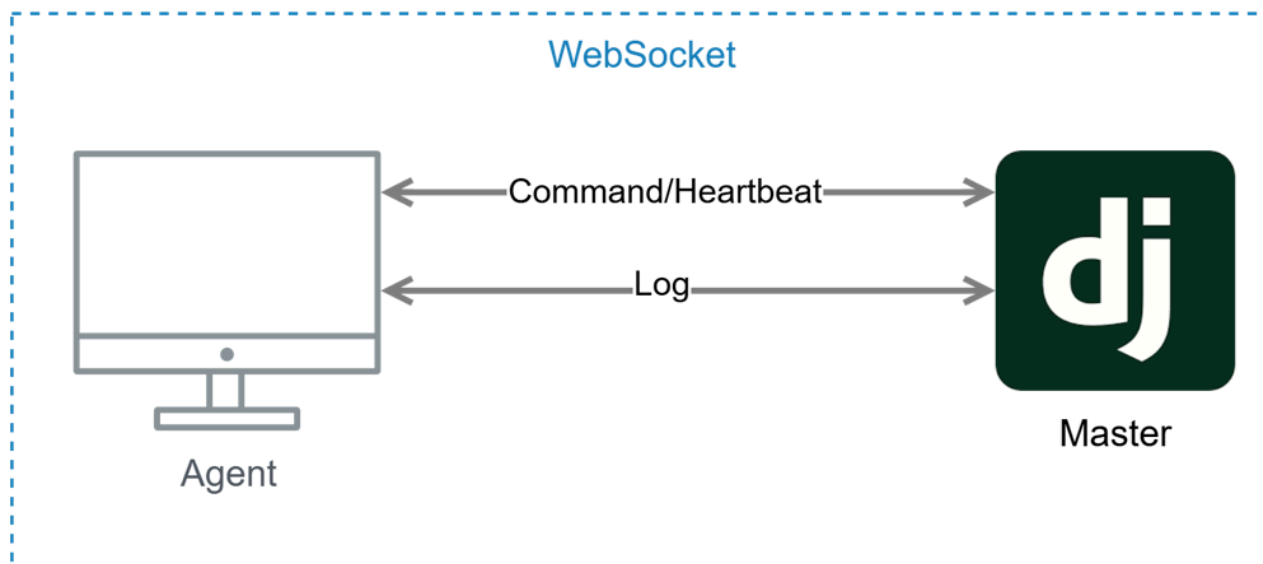
Мастер представља централни део система и имплементирана као *Django* апликација. Задужен је за управљање процесима и оркестрацију послова. Он координише рад свих агената и контролише извршавање задатака које дефинишу корисници.

Кориснички интерфејс развијен је као апликација која комуницира са мастером преко *RESTful API*-ја. Сви захтеви и одговори размењују се у *JSON* формату. Интерфејс омогућава корисницима да управљају свим ресурсима система и прате извршавање послова у реалном времену.

Приликом креирања посла, корисник бира агента који ће га извршити. Мастер прослеђује дефинисани посао изабраном агенту. Агент преузима тај посао и извршава га. Уколико је изабрани агент тренутно заузет, мастер ставља посао у ред чекања. Послови који се већ налазе у реду имају предност у односу на нове захтеве, чиме се обезбеђује фер редослед извршавања.



Слика 1: Архитектура система



Слика 2: Комуникација између мастер-а и агента

Мастер одржава централизовану евиденцију свих дефинисаних послова и њихових статуса, као и информације о агентима и њиховој доступности. Не учествује у самом извршавању, већ управља процесом и надгледа комуникацију између корисника и агената.

Поред управљања пословима, мастер обрађује све захтеве који стижу из корисничког интерфејса. Корисници преко интерфејса могу да креирају, мењају или бришу послове, као и да управљају агентима. Мастер верификује аутентичност захтева, примењује безбедносне политике и прослеђује инструкције одговарајућим сервисима.

### 3.2 Агент компонента

Агент представља извршну компоненту система задужену за обраду послова које шаље мастер. Реализован је као мала *Python* апликација која се изграђује и пакује у *.exe* формат. На тај начин агент се може лако преузети и покренути на било ком рачунару, без потребе за додатним подешавањима или инсталацијом зависности.

Сваки агент ради независно и може бити инсталиран на различитим машинама или унутар изолованих окружења. Оваква организација омогућава равномерну расподелу оптерећења и паралелно извршавање више послова.

Агент периодично шаље *heartbeat* сигнале мастеру како би потврдио своју доступност. На основу тих сигнала, мастер прати активност свих агената у систему. Уколико се *heartbeat* не прими у предвиђеном времену, агент се означава као неактиван. Његови послови се аутоматски паузирају и настављају када се веза поново успостави.

Када агент прими посао, преузима податке које је дефинисао мастер и покреће извршавање корака. Током процеса агент шаље логове и статус сваког корака, што омогућава корисницима да у реалном времену прате напредак извршавања.

Оваква архитектура рада агената обезбеђује поуздано извршавање послова, лако скалирање и једноставно одржавање целокупног система.

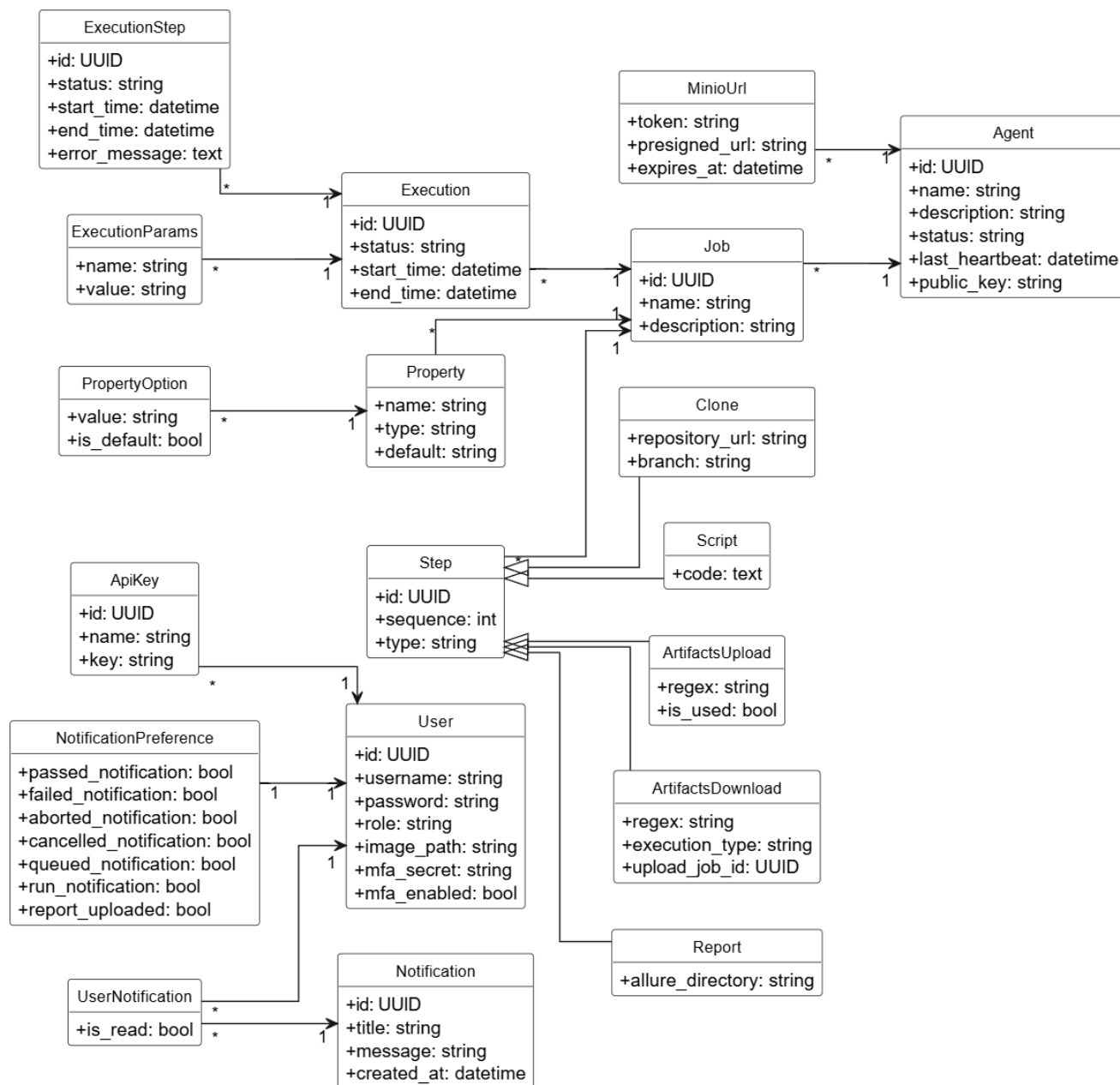
### 3.3 База података

База података представља централно место за чување свих података у систему. Користи се *PostgreSQL* као релациона база података. Изабрана је због стабилности, поузданости и добре подршке за рад у више-корисничким и дистрибуираним окружењима. Сви подаци који се односе на кориснике, агенте, послове, извршења и нотификације чувају се у овој бази, док су артефакти и већи бинарни фајлови смештени у засебном складишту (*MinIO*).

Главни ентитети у систему су:

- *User* – представља корисника система и садржи податке о налогу, улози, слици и стању двофакторске аутентикације.
- *Agent* – представља извршну јединицу задужену за обраду послова. Чува податке о статусу, кључевима и времену последњег *heartbeat*-а.
- *Job* – дефинише посао који се извршава преко изабраног агента. Садржи основне информације и повезане кораке.
- *Step* – описује појединачни корак у извршавању посла.
- *Execution* и *ExecutionStep* – чувају податке о историји извршавања послова и статусима сваког корака.
- *Notification* и *UserNotification* – користе се за слање и евидентирање нотификација унутар система.
- *Property* и *PropertyOption* – омогућавају конфигурацију при покретању послова, са дефинисаним типовима и подразумеваним вредностима.
- *ApiKey*, *GitHubInstallation* и *GitHubAppInstallState* – служе за интеграцију са спољним системима и безбедан приступ преко API интерфејса.

На Сlici 3 приказан је поједностављени дијаграм класа који илуструје структуру и односе између главних ентитета система.



Слика 3: Дијаграм класа система *Test Hub Mini*



У овом поглављу представљена је имплементација дистрибуираног CI/CD система заснованог на мастер-агент архитектури. Поглавље описује начин на који су реализовани функционални и нефункционални захтеви дефинисани у претходним поглављима. Посебан акценат стављен је на механизме комуникације, безбедносне компоненте, као и функционалности које омогућавају управљање пословима, агентима и корисницима.

Систем је дизајниран тако да омогући поуздану, безбедну и флексибилну аутоматизацију процеса интеграције и испоруке софтвера. Имплементација обухвата више међусобно повезаних модула, од којих сваки има јасно дефинисану улогу у целокупном процесу. У наредним одељцима биће детаљно описане главне компоненте система, њихове међусобне везе и кључне технологије које су коришћене при развоју.

### 4.1 Безбедносни механизми

Безбедност система представља један од најважнијих аспеката у дизајну и реализацији дистрибуираног CI/CD алата. Циљ је да се обезбеди поуздана аутентикација, заштита комуникације и спречи неовлашћен приступ подацима и ресурсима. Пошто систем функционише у дистрибуираном окружењу у ком више агената и корисника истовремено комуницира са централним сервером, неопходно је обезбедити висок ниво заштите на свим нивоима архитектуре.

У овом поглављу описани су кључни безбедносни механизми који су примењени у систему. Обухваћени су процес двофакторске аутентикације, контрола приступа и управљање корисничким улогама, шифровање података и сигурна комуникација, као и улога Nginx сервера као заштитног и посредничког слоја. Ови механизми заједно обезбеђују интегритет, поверљивост и поузданост рада целокупног система.

#### 4.1.1 Двофакторска аутентикација (2FA)

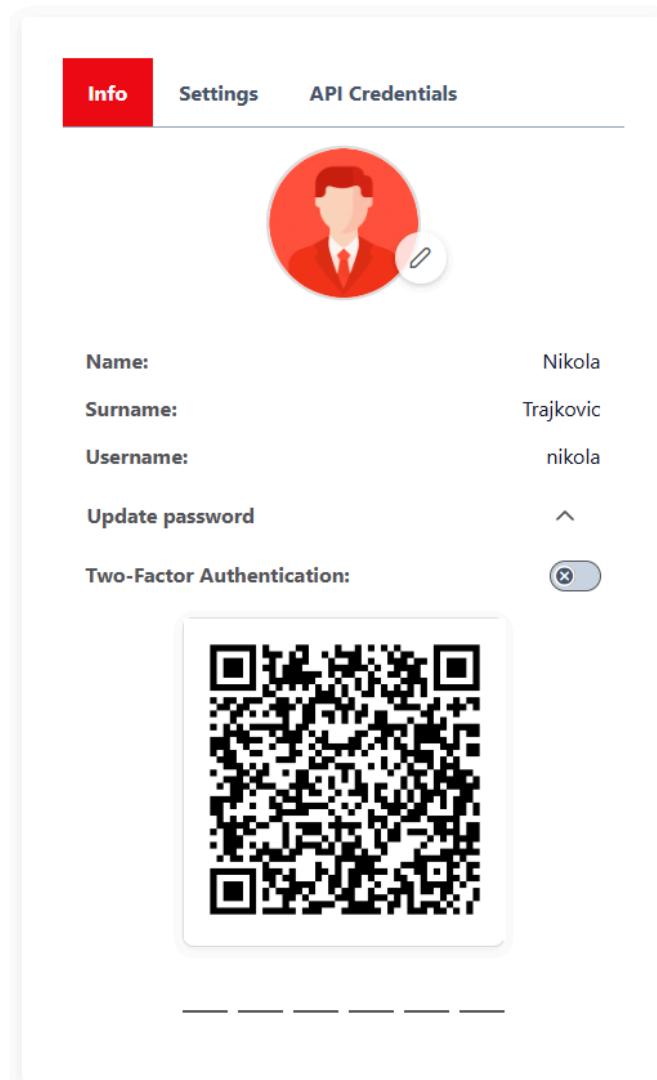
Двофакторска аутентикација представља додатни слој заштите корисничког налога. На овај начин, чак и у случају компромитовања лозинке, неовлашћени приступ систему је онемогућен без физичког приступа корисниковом уређају.

У оквиру система, двофакторска аутентикација је реализована коришћењем временски заснованих једнократних лозинки (TOTP) преко апликација као што су *Google Authenticator* и *Microsoft Authenticator*.

Приликом првог пријављивања, систем генерише јединствени тајни кључ (*mfa\_secret*) за сваког корисника и на основу њега креира QR код. Корисник тај код скенира помоћу изабране аутентикационе апликације, која затим локално генерише нови безбедносни код сваких 30 секунди.

Током процеса пријављивања, након успешне валидације корисничких података, систем захтева унос текућег OTP кода ради завршне провере идентитета. Уколико је унети код исправан, приступ се одобрава и кориснику се издају JWT токени за даљу комуникацију.

На Слици 4 приказан је изглед корисничког интерфејса при активирању двофакторске аутентикације, где се приказује генерисани QR код и поље за унос једнократног кода.



Слика 4: Активирање двофакторске аутентикације (2FA)

#### 4.1.2 Контрола приступа и корисничке улоге

Систем примењује контролу приступа засновану на улогама корисника. Постоје две дефинисане улоге:

- Администратор (*Admin*) и
- Регуларни корисник (*Regular*).

Администратор има овлашћења за управљање корисницима. Може да креира нове налоге и мења постојеће. Регуларни корисник нема ова овлашћења. Ипак, може користити све остале функционалности система, као што су рад са агентима и пословима.

Контрола приступа имплементирана је у оквиру *Django REST Framework* механизма дозвола. Сваки захтев који долази до API-ја пролази кроз проверу аутентификације и улоге. За то се користе класе *IsAdmin* и *IsRegular*, које одређују да ли корисник има право приступа одређеном ресурсу.

```
class IsAdmin(BasePermission):
    def has_permission(self, request, view):
        return request.user.role == Role.ADMIN

class IsRegular(BasePermission):
    def has_permission(self, request, view):
        return request.user.role == Role.REGULAR
```

Листинг 1: Имплементација класа за контролу приступа.

Аутентификација се спроводи помоћу JWT токена, што обезбеђује сигуран и *stateless* приступ систему.

#### 4.1.3 Шифровање и сигурна комуникација

Комуникација између мастера и агената осигурана је употребом више нивоа заштите. Сви подаци се преносе преко *WebSocket* везе која је заштићена SSL/TLS протоколом, чиме се спречава пресретање и измена података током преноса.

При иницијалном повезивању, сваки агент генерише пар криптографских кључева, јавни и приватни. Јавни кључ се шаље мастеру, који га чува у бази у шифрованом облику. Приватни остаје локално на страни агента.

Све поруке које мастер шаље агенту користе хибридни приступ шифровању.

Прво се генерише симетрични кључ који се користи за шифровање садржаја поруке. Тај симетрични кључ се затим шифрује јавним кључем агента помоћу RSA алгоритма и шаље заједно са шифрованим подацима. Агент након пријема поруке користи свој приватни кључ за дешифровање симетричног кључа, а затим њиме дешифрује саму поруку.

На овај начин се обезбеђује интегритет, аутентичност и поверљивост сваке поруке између мастера и агента.

#### 4.1.4 *Nginx* као *reverse proxy* и сигурносни слој

*Nginx* у систему има улогу *reverse proxy* сервера [4] који усмерава све спољне захтеве ка одговарајућим сервисима унутар *Docker* окружења. Поред усмеравања, он обезбеђује и више безбедносних механизма:

- контролу приступа,
- заштиту комуникације и
- изолацију сервиса.

*Nginx* прослеђује HTTP и *WebSocket* захтеве ка *Django* апликацији која ради на порту 8000, док се статички ресурси сервирају директно изнутра *Nginx* окружења.

Захтеви ка складишту података (*MinIO*) пролазе кроз посебан механизам аутентификације који користи *auth\_request* директиву [5]. Сваки захтев се најпре прослеђује *Django* серверу ради валидације токена. Уколико је приступ дозвољен, пропушта ка *MinIO* сервису. На овај начин се спречава неовлашћени приступ фајловима.

Поред HTTP и *WebSocket* комуникације, *Nginx* такође обрађује SSH саобраћај ка систему за верзионисање (*Gitea*) преко *stream* модула, где делује као TCP проху и преусмерава захтеве са порта 2222 на унутрашњи порт 22 [6].

Пример конфигурације *Nginx* сервера приказан је на Слици 5 и илуструје начин прослеђивања захтева ка *Django* апликацији, *MinIO* складишту и *Gitea* систему.

```
server {
    listen 80;
    server_name localhost;
    client_max_body_size 50M;

    location /api/ {
        proxy_pass http://host.docker.internal:8000/api/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /minio/ {
        auth_request /auth/validate-minio-access;
        proxy_pass http://minio:9000;
        proxy_set_header X-Auth-Token $http_x_auth_token;
    }

    location /ws/ {
        proxy_pass http://host.docker.internal:8000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}

stream {
    server {
        listen 2222;
        proxy_pass gitea:22;
    }
}
```

Слика 5: Конфигурација *Nginx reverse proxy* сервера

## 4.2 Управљање корисницима и агентима

Један од кључних аспеката дизајна система представља ефикасно управљање корисницима и агентима. Систем омогућава централизовано креирање и измену, као и дефинисање различитих нивоа приступа у складу са улогом коју корисник има у оквиру система. Поред управљања корисницима, посебна пажња посвећена је и управљању агентима, који представљају извршне јединице задужене за обраду задатака и комуникацију са главним сервером.

#### 4.2.1 Креирање и администрирање налога

Процес управљања корисничким налозима у систему осмишљен је тако да обезбеди једноставно креирање и висок ниво безбедности. Креирање новог налога обавља искључиво администратор, који уноси основне податке као што су корисничко име, име, презиме и почетна улога корисника. Приликом креирања налога, систем аутоматски додељује подразумевану профилну слику и иницијалну лозинку.

Након првог пријављивања, корисник може самостално изменити своју профилну слику и променити иницијалну лозинку. На овај начин се обезбеђује персонализација налога и истовремено повећава безбедност, јер сваки корисник дефинише сопствену шифру. Лозинке се чувају у хешираном облику, што спречава неовлашћен приступ поверљивим подацима.

Оваква архитектура омогућава јасну поделу одговорности између администратора и корисника: администратор креира налог и додељује улогу, док корисник управља сопственим подацима и безбедносним поставкама.

#### 4.2.2 Управљање агентима

Корисници система имају могућност да креирају, мењају и бришу агенте у складу са потребама инфраструктуре.

Приликом креирања новог агента, систем аутоматски генерише *Agent Identification* и *Connection Token*, који служе за јединствено препознавање агента и верификацију његовог идентитета током иницијалне комуникације са системом. Ови параметри користе се само током првог повезивања, како би се спречио неовлашћен приступ и обезбедило да у систем не може бити додат агент који није легитимно регистрован. Након успостављања почетне везе, даља комуникација се обавља преко безбедног канала.

Сваки агент има атрибут *status*, који одражава његово тренутно стање у систему. Статуси могу бити:

- *offline* – агент је искључен или тренутно није повезан са системом.
- *available* – агент је активан и доступан за извршавање нових задатака.
- *running* – агент тренутно извршава посао који му је додељен.

Корисници могу у сваком тренутку пратити статус сваког агента, вршити измене у конфигурацији или уклањати агенте који више нису потребни. Оваквим приступом обезбеђена је флексибилност и скалабилност система, јер се број и конфигурација агената могу динамички прилагођавати у складу са оптерећењем и захтевима рада.

### 4.3 Управљање пословима (*Job*-овима)

У овом поглављу описан је механизам управљања пословима у оквиру система. Биће објашњено како се послови креирају, конфигуришу и покрећу, као и начин на који се чува њихова историја извршавања и генерисани артефакти. Поред тога, детаљно је представљен процес извршавања појединачних корака (*Step*-ова) унутар посла.

#### 4.3.1 Креирање и конфигурисање послова

Креирање новог посла у систему врши се путем форме која омогућава унос основних информација и дефинисање корака (*steps*) који ће се извршавати.

Сваком послу се додељује јединствени идентификатор, назив и опис, као и агент на ком ће се извршавати. Поред тога, у оквиру исте форме могуће је дефинисати *webhook* адресе за *Slack* и *Microsoft Teams*, чиме се омогућава слање нотификација о статусу извршавања директно на изабрану платформу. На Слици 6 је приказан део форме за креирање посла са уносом основних информација и *webhook* адреса

The screenshot shows a form for creating a new job. It has four main sections, each with a label and a text input field:

- Name:** The input field contains the text "Job1".
- Description:** The input field contains the text "first job".
- MS Teams Web Hook URL (optional):** The input field contains the URL "https://prod-232.westeurope.logic.azure.com:443/workflows/5b027ff1c34d-".
- Slack Web Hook URL (optional):** The input field is empty.

Слика 6: Део форме за креирање посла са уносом основних информација и *webhook* адреса.

Кораци представљају логичке јединице које дефинишу ток извршавања посла. Сваки корак описује конкретну акцију коју агент треба да спроведе, попут клонирања репозиторијума, покретања скрипте или преноса резултата извршавања. На Слици 7 је приказан пример конфигурације корака (*steps*) у оквиру једног посла.

The screenshot shows a job configuration form with five steps, each in its own box with a "Remove" button in the top right corner:

- Step 1 - Clone:** Contains fields for "Repository URL", "Branch", "Authentication Type" (a dropdown menu with "SSH Key" selected), "Public Key", and "Private Key".
- Step 2 - Script:** Contains a "Script Code" text area.
- Step 3 - Upload:** Contains a "Regex" text field.
- Step 4 - Download:** Contains fields for "Execution Type" (a dropdown menu with "Last Execution" selected), "Regex", and "Upload Job" (a dropdown menu with "Select a Job" and a search icon).
- Step 5 - Report:** Contains an "Allure Directory" text field.

Слика 7: Пример форме за додавање корака у послу.

Сваки посао може имати сопствене параметре(*properties*) који дефинишу начин извршавања. Параметри могу бити различитих типова у зависности од природе вредности:

- *Text* – унос једноставне текстуалне вредности.
- *Toggle (Boolean)* – омогућава укључивање или искључивање одређене опције.
- *Select* – избор једне вредности из понуђеног скупа.

- *Multiple Select* – избор више вредности из понуђеног скупа.

Параметри се могу дефинисати на више начина унутар конфигурације посла, зависно од синтаксе или окружења које се користи. Најчешћи облици записивања променљивих су:

- `$var`,
- `${var}`,
- `{{var}}` и
- `%var%`.

Сваки од ових начина омогућава динамичко убацивање вредности у време извршавања посла.

На Слици 8 је приказан пример форме за дефинисање различитих типова параметара (*properties*) који се могу користити у послу.

The image displays a configuration interface with four distinct property types, each in a separate panel. Each panel includes a 'Name' input field, a 'Value' input field, and a 'Remove' button.

- Property - Single Line Text:** The 'Name' field contains 'branch' and the 'Value' field contains 'main'.
- Property - Multiple Choice:** The 'Name' field contains 'upload'. The 'Value' field lists three options: '\*.txt' (selected with a red checkmark), '\*.py', and '\*.exe' (each with an unselected checkbox and a trash icon). A '+ Add option' button is present.
- Property - Toggle Switch:** The 'Name' field contains 'isActive' and a toggle switch is shown in the 'off' position.
- Property - Dropdown Menu:** The 'Name' field contains 'download'. The 'Value' field lists two options: '\*.txt' (selected with a red radio button) and '\*.py' (with an unselected radio button and a trash icon). A '+ Add option' button is present.

Слика 8: Пример форме за дефинисање параметара (*properties*) различитих типова у послу.

### 4.3.2 Типови корака и параметри извршавања

Сваки посао (*Job*) у систему се састоји од једног или више корака (*Step*-ова), при чему сваки корак представља одређену акцију која ће бити извршена на агенту. У зависности од врсте корака, систем примењује различите механизме извршавања и приказа конфигурационих параметара.

Подржани типови корака су:

- *Clone* – клонирање Git репозиторијума.
- *Script* – извршавање произвољне скрипте у окружењу агента.
- *Artifacts Upload* – отпремање изабраних датотека као артефаката извршавања.
- *Artifacts Download* – преузимање артефаката.
- *Report* – генерисање извештаја.

#### 4.3.2.1 Clone

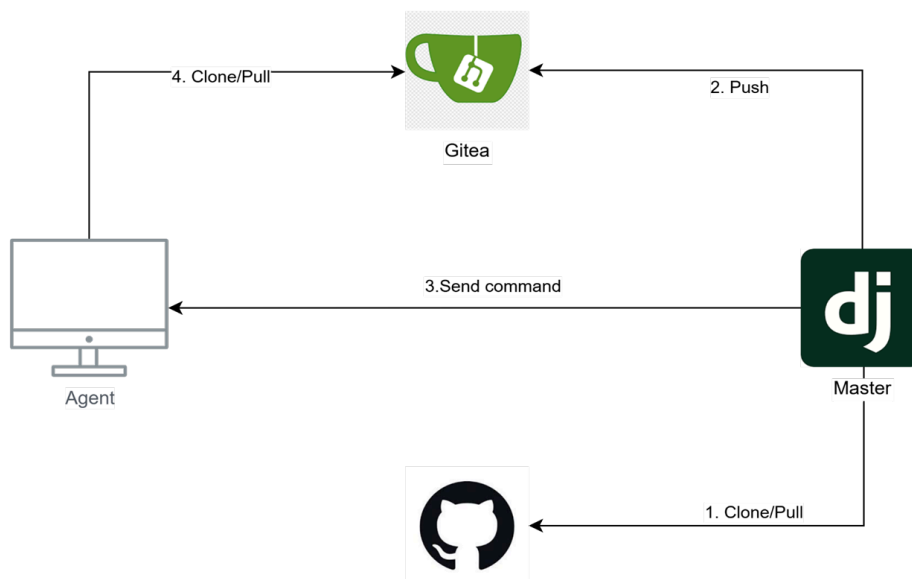
*Clone* корак представља једну од фаза извршавања посла и има задатак да обезбеди изворни код потребан за извршавање наредних корака. Приликом конфигурације посла, корисник бира метод којим ће се репозиторијум клонирати. Систем подржава више начина аутентикације (SSH кључеви, *access token* или *GitHub App*), али како би се очувала безбедност, само мастер има приступ поверљивим подацима корисника.

Мастер врши клонирање из оригиналног Git репозиторијума (нпр. *GitHub* или *GitLab*), након чега преузети код отпрема (*push*) на *Gitea*-у. *Gitea* је локални сервер, самостално хостован *Git* систем. На овај начин се оригинални репозиторијум никада не излаже агентима, а потенцијално осетљиви подаци као што су приступни токени или SSH кључеви остају ограничени искључиво на мастеру.

Приликом иницијалне регистрације или повезивања агента са системом, агент шаље свој јавни SSH кључ мастеру. Мастер затим додаје тај кључ у *Gitea* систем, чиме агенту омогућава приступ одговарајућим репозиторијумима. На овај начин свака комуникација између агента и *Gitea*-е одвија се путем безбедне SSH конекције, уз јасно дефинисане аутентикационе механизме и без потребе да агент поседује било какве корисничке креденцијале.

Агенти затим клонирају и повлаче (*pull*) код искључиво са *Gitea* сервера, чиме се обезбеђује додатни ниво сигурности и централизована контрола над кодом који се извршава у систему.

На Слици 9 приказан је ток извршавања *Clone* корака — од иницијалног преузимања кода од стране Master-а до његовог преноса на *Gitea* и преузимања од стране агената.



Слика 9: Ток извршавања *Clone* корака и комуникација између мастера, *Gitea* сервера и агената.



#### 4.3.2.2 Script

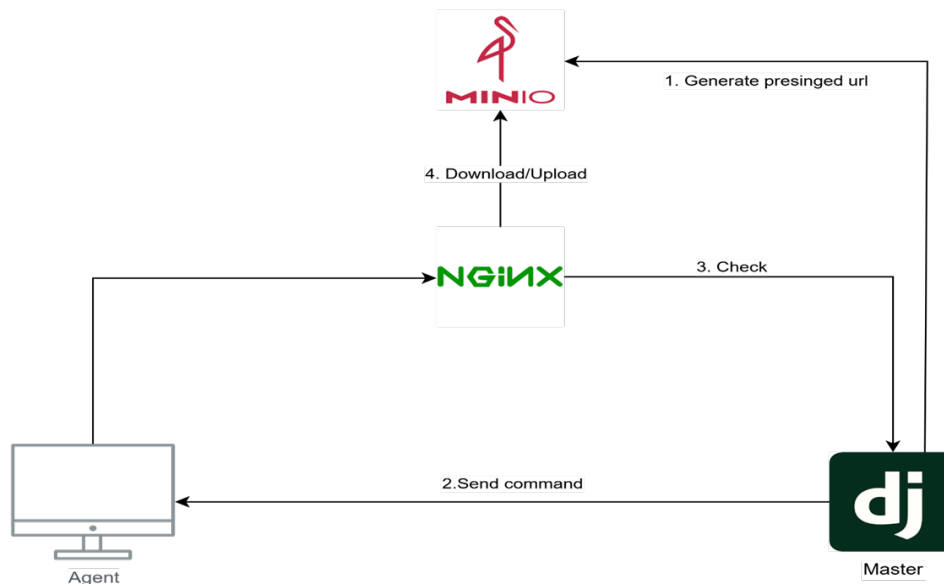
Извршавање корака у оквиру послова реализовано је помоћу посебног механизма агента који омогућава покретање скрипти у контролисаном окружењу. За сваки корак се динамички генерише привремена скрипта која се извршава у засебном процесу, при чему се резултати извршавања прате у реалном времену и прослеђују систему.

Након завршетка извршавања, процес се безбедно прекида, а привремени ресурси се аутоматски уклањају. На овај начин се обезбеђује стабилност агента и поуздано извршавање дефинисаних корака без ризика од нежељених споредних ефеката.

#### 4.3.2.3 Artifacts Upload/Download

Једна од кључних фаза односи се на преузимање и отпремање артефаката који настају током извршавања послова. Артефакти представљају резултате извршавања појединачних корака, као што су изграђене бинарне датотеке, извештаји о тестирању или конфигурациони фајлови, који се користе у наредним фазама процеса.

У оквиру фаза *Artifacts Download* и *Artifacts Upload*, систем обезбеђује сигуран и контролисан пренос артефаката између агената и централног складишта података. На Слици 10 је приказан поступак преузимања и отпремања артефаката у систему.



Слика 10: Ток извршавања *Clone* корака и комуникација између мастера, *Gitea* сервера и агената.

Опис процеса:

1. Генерисање привремене URL адресе:

Мастер сервис иницира процес генерисања *presigned URL* адресе у складишту података (*MinIO*).

1. Прослеђивање команде агенту:

Након успешног генерисања *presigned URL* адресе, мастер сервис путем *Nginx* сервера прослеђује команду агенту, која садржи неопходне параметре за извршење задатка — било да је реч о преузимању или отпремању артефаката.

1. Провера валидности захтева:

Приликом обраде команде, агент остварује комуникацију са *Nginx* прокси сервером, који врши проверу ваљаности и аутентичности захтева, након чега се комуникација безбедно прослеђује ка складишту података.

#### 1. Извршење операције преузимања или отпремања:

Агенти користе добијену *presigned URL* адресу за директно преузимање или отпремање артефаката у складиште (*MinIO*). Оваквим приступом елиминише се потреба за директним приступом бази података или серверу апликације, чиме се постиже виши степен безбедности, као и ефикаснији пренос података унутар система.

Оваква архитектура омогућава сигуран и скалабилан пренос података унутар система, уз минималан ризик од неовлашћеног приступа. Употреба *presigned URL* механизма елиминише потребу да агенти поседују приступне креденцијале за складиште, чиме се повећава ниво безбедности целокупног CI/CD процеса.

#### 4.3.2.4 Report

У оквиру процеса тестирања, систем аутоматски генерише извештај о резултатима у облику *Allure report*-а, који садржи податке о извршеним тестовима, њиховом статусу и релевантним метаподацима. Ради лакшег прегледа, анализе и архивирања резултата, реализован је механизам који омогућава аутоматизовано отпремање ових извештаја на сервер.

Процес отпремања извештаја подразумева идентификацију пројекта ком извештај припада, припрему одговарајуће архиве са подацима и њено слање ка централном систему. На тај начин се обезбеђује да резултати сваког извршавања тестова буду доступни унутар јединственог складишта, чиме се омогућава праћење квалитета током развоја и анализа промена кроз време.

Аутоматизацијом овог поступка елиминише се потреба за ручним интервенцијама, што смањује могућност грешке и убрзава процес тестирања. Овакво решење омогућава поуздану и конзистентну евиденцију резултата тестирања, као и интеграцију са осталим компонентама система за континуирану интеграцију и испоруку.

#### 4.3.3 Извршавање послова

Процес извршавања послова представља централни механизам система и служи као спона између дефинисаних конфигурација, агената и мастера. Циљ овог процеса јесте да обезбеди сигурно, контролисано и предвидиво извршавање различитих корака који чине један посао, као и да омогући праћење и анализу резултата сваког извршавања.

Када се посао активира, мастер приступа анализи конфигурације и припреми параметара потребних за сваку фазу извршавања. Посао се састоји од више корака (*steps*), при чему је за сваки корак дефинисан скуп параметара који одређују његово понашање. Сви ови подаци се организују у структурираном JSON формату, који се затим шаље агенту задуженом за извршавање.

Агент, након пријема инструкција, интерпретира достављени JSON документ и приступа извршавању дефинисаних корака редом. Тиме се обезбеђује потпуна изолација извршења, јер сваки агент делује у сопственом окружењу и користи само параметре који су му додељени.

Систем омогућава континуирано праћење извршавања путем повратних информација које агент шаље мастеру. Ови подаци садрже детаље о току извршавања, времену трајања и евентуалним грешкама, што омогућава да се напредак посла надгледа у реалном времену. По завршетку извршавања, резултати сваког корака се евидентирају и чувају у бази података, чиме се омогућава накнадна анализа и генерисање извештаја.

Поред појединачних резултата корака, систем чува и све записе (логове) који настају током извршавања послова. Ови логови садрже детаљне информације о току рада агента, времену почетка и завршетка сваког корака, као и о евентуалним грешкама или неуспелим извршењима.

#### 4.3.4 Историја извршавања и артефакти

Свако извршавање посла у систему се евидентира и чува у бази података, заједно са свим релевантним параметрима који су коришћени током процеса. Овакав приступ омогућава потпуну реконструкцију поступка у било ком тренутку, што представља основу за анализу, верификацију и унапређење процеса изградње и испоруке.

Поред основних података о покретању и завршетку посла, сваки запис садржи и детаљну листу корака који су извршени у оквиру процеса. За сваки корак бележе се статус, време извршавања и евентуални резултати или грешке. На овај начин систем омогућава прецизно праћење тока извршавања и једноставно откривање потенцијалних неправилности.

Посебан значај имају логови извршавања, који се чувају за сваки појединачни корак. Они омогућавају увид у конкретне активности спроведене током рада система, као и у могуће узроке неуспеха. Поред прегледа логова у оквиру система, кориснику је омогућено и преузимање комплетног лог фајла који садржи детаљан запис читавог извршавања посла.

### 4.4 Систем нотификација

Систем нотификација омогућава корисницима да буду благовремено обавештени о свим догађајима током извршавања послова у систему. На овај начин, корисник може пратити напредак, резултат и статус извршења без потребе за ручним проверама.

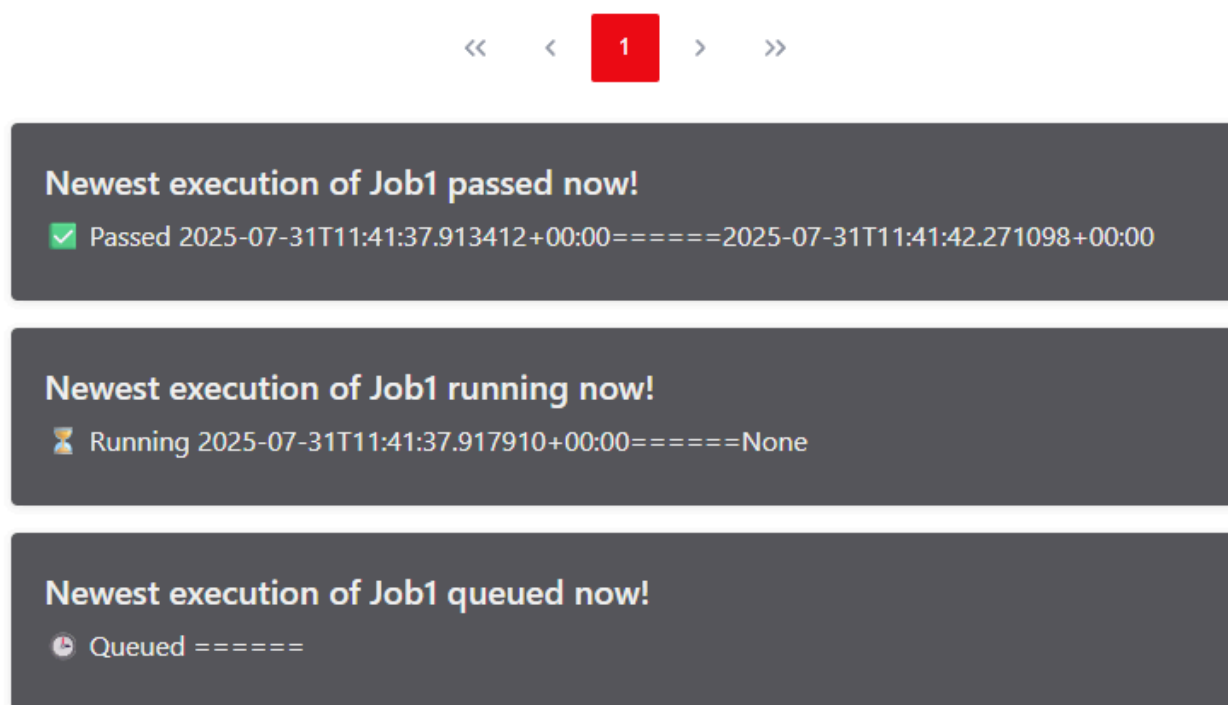
#### 4.4.1 In-App нотификације

In-App нотификације се приказују директно у оквиру корисничког интерфејса и представљају основни вид комуникације између система и корисника. Свака нотификација садржи наслов, опис догађаја и време када је генерисана.

Корисници могу подесити које типове догађаја желе да прате, међу којима су:

- *Passed Execution Notification* – успешно завршено извршење
- *Failed Execution Notification* – неуспело извршење
- *Cancelled Execution Notification* – отказано извршење
- *Aborted Execution Notification* – прекинуто извршење због грешке
- *Queued Execution Notification* – посао додат у ред за извршење
- *Running Execution Notification* – посао у току
- *Report Upload Notification* – извештај је успешно отпремљен.

Поред избора типова догађаја, корисник мора да дефинише за које послове (*jobs*) жели да прима нотификације. Уколико корисник не изабере ниједан посао, неће добијати никакве нотификације. На овај начин се избегава непотребно оптерећење системом нотификација и омогућава се потпуна контрола над релевантним обавештењима. На Слици 11 приказан је пример изгледа in-апп нотификација у корисничком интерфејсу.

Слика 11: Пример приказа *in-app* нотификација у систему.

#### 4.4.2 Slack и Microsoft Teams интеграције

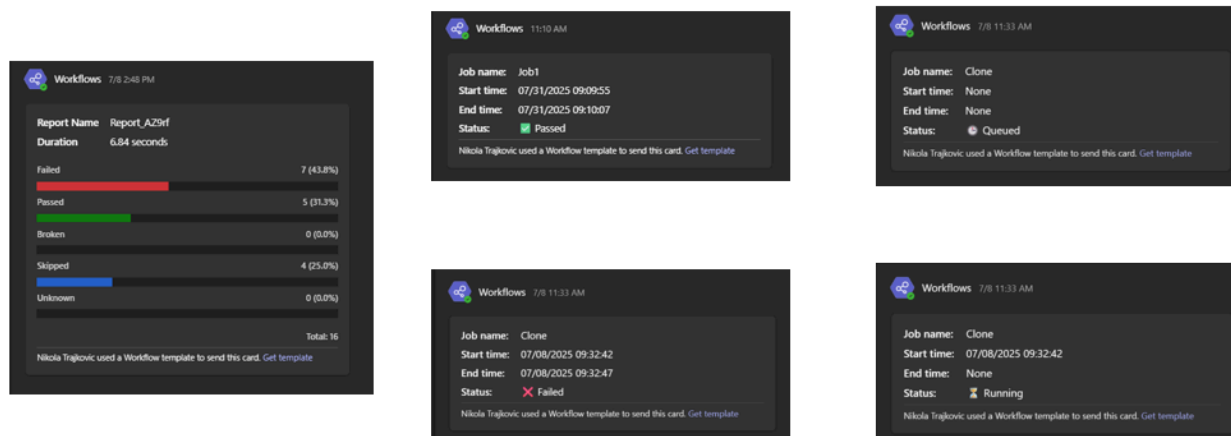
Систем подржава интеграцију са алатима за комуникацију као што су *Microsoft Teams* и *Slack*, омогућавајући слање обавештења о статусу послова директно у изабрани канал. Ова функционалност омогућава да обавештења о извршавању послова (*jobs*) буду достављена у реалном времену, чиме се повећава транспарентност и омогућава бржа реакција на промене у систему.

Део форме за креирање посла омогућава унос URL адреса *webhook*-ва за *Slack* и *Microsoft Teams*. Корисник може унети један или оба линка, у зависности од тога на којој платформи жели да прима нотификације.

Након дефинисања *webhook*-ова, систем аутоматски шаље обавештења о свим релевантним догађајима као што су:

- успешно завршено извршење (*passed*),
- неуспешно извршење (*failed*),
- посао у току (*running*),
- посао у реду за извршење (*queued*) и
- отказано или прекинуто извршење (*cancelled / aborted*).

Нотификације се шаљу у облику JSON порука путем *webhook*-ва, при чему се користе унапред дефинисани шаблони (*templates*) који садрже информације о називу посла, времену почетка и завршетка, као и статусу извршења. На Слици 12 приказан је изглед обавештења послатих путем интеграције са *Microsoft Teams* платформом.



Слика 12: Пример приказа *Microsoft Teams* нотификација.

Креирање *webhook*-ова за ове платформе описано је у њиховој званичној документацији, где су дати кораци за додавање новог *Incoming Webhook*-а у оквиру *Microsoft Teams* [7] и *Slack* [8].

## 4.5 Интеграције са спољним системима

### 4.5.1 Генерисање и управљање API кључевима

У циљу обезбеђивања сигурне комуникације између система и спољних сервиса, имплементиран је механизам за управљање API кључевима. Сваки кључ представља јединствени идентификатор који служи за аутентификацију захтева упућених ка систему и заштиту од неовлашћеног приступа.

Сваки корисник може имати више кључева, при чему је сваки од њих повезан са конкретним именом и наменом. Оваква организација омогућава прецизну контролу приступа и раздвајање права употребе по различитим сервисима или пројектима. Систем спречава дуплирање назива кључева за истог корисника, чиме се обезбеђује јасна структура и једнозначна идентификација сваког кључа.

Генерисање кључа се врши аутоматски приликом његовог креирања, уз употребу криптографски сигурне генерације случајних вредности. На тај начин се минимизује ризик од предвиђања или злоупотребе кључева.

Осим креирања, систем подржава и управљање животним циклусом кључева, као што су брисање, ревокација и верификација током сваког захтева. Кључ је могуће копирати само у тренутку његовог генерисања, након чега више није доступан за преглед из безбедносних разлога. Уколико се изгуби, потребно је обрисати постојећи и генерисати нови кључ. Сви кључеви се чувају у бази података повезаној са корисничким налогом, што омогућава контролу приступа и праћење активности.

Овако дизајниран механизам представља основу за сигурну интеграцију са спољним системима, као што су *GitHub* и *GitLab*, који користе API кључеве за потписивање и верификацију *webhook* захтева.

### 4.5.2 Подршка за *GitHub* и *GitLab* *webhook*-ове

Систем омогућава директну интеграцију са платформама *GitHub* и *GitLab* путем *webhook* механизма, што омогућава аутоматско покретање послова након промена у репозиторијуму. *Webhook* представља HTTP захтев који *Git* сервер аутоматски упућује *Typhoon* систему сваки пут када се догоди неки унапред дефинисани догађај, као што су нови *commit*, отварање *pull request*-а или спајање гране (*merge*).

У конфигурацији *webhook*-а дефинише се:

- *Payload URL*, који представља адресу *Typhoon API*-ја на коју се шаље догађај и садржи параметре као што су идентификатор посла (*job\_id*) и друге вредности потребне за извршавање

- тајни кључ (*secret*), који у оквиру система представља API кључ
- као и тип догађаја који треба да покрећу *webhook*.

По пријему *webhook*-а, сервер врши проверу потписа или токена у зависности од платформе:

- *GitHub* користи HMAC-SHA256 потпис који се шаље у заглављу X-Hub-Signature-256 [9].
- *GitLab* користи токен у заглављу X-Gitlab-Token [10].

Потпис или токен се упоређује са листом важећих API кључева у бази. Уколико је провера успешна, систем покреће дефинисани посао који је прослеђен у параметрима захтева. Поред покретања посла, могуће је проследити и додатне параметре који се аутоматски повезују са подацима из JSON тела догађаја, попут назива гране или репозиторијума.

На овај начин, сваки *commit* или *merge* у изворном коду може аутоматски активирати процес изградње, тестирања и доставе, без потребе за ручном интервенцијом.

## 4.6 Корисничка аналитика

Корисничка аналитика представља важан део система јер омогућава праћење понашања корисника, начина употребе функционалности и укупне ефикасности система. Циљ је да се кроз прикупљање и анализу података обезбеде увиди који помажу у унапређењу корисничког искуства, оптимизацији интерфејса и идентификацији потенцијалних проблема у раду система.

У систему су имплементиране две независне, али комплементарне компоненте за аналитичко праћење — *Umami* и *PostHog*. *Umami* се користи за основну статистику о посећености и употреби интерфејса, док *PostHog* омогућава дубљу анализу интеракција и кретања корисника унутар апликације.

### 4.6.1 Интеграција са *Umami* системом

*Umami* је једноставан и *self-host* систем за праћење корисничке аналитике [11]. Интегрисан је у оквиру веб интерфејса система додавањем кратког *JavaScript* скрипта у HTML документ.

Овим приступом *Umami* аутоматски прикупља податке о посетама, приказима страница и интеракцијама корисника, без потребе за додатним конфигурацијама. Сви подаци се обрађују локално, унутар *Docker* окружења, што обезбеђује потпуну контролу над приватношћу и безбедношћу.

У административном интерфејсу *Umami*-ја могуће је прегледати следеће статистике:

- број јединствених посетилаца,
- учесталост приступа по страницама,
- временска дистрибуција активности и
- преглед коришћених уређаја и прегледача.

### 4.6.2 Интеграција са *PostHog* системом

*PostHog* је платформа за напредну корисничку аналитику и праћење понашања корисника у реалном времену. У оквиру система је интегрисана *cloud* верзија *PostHog*-а, јер *self-host* инстанца захтева *Linux* окружење и додатне ресурсе који нису били доступни током развоја.

Интеграција је реализована путем *React* провајдера [12], чиме се праћење догађаја омогућава на нивоу целе апликације. Овај провајдер омогућава слање података о корисничким активностима, као што су:

- покретање и заустављање послова,
- пријава и одјава корисника и
- интеракције са интерфејсом.

У оквиру овог рада реализован је дистрибуирани CI/CD систем заснован на мастер–агент архитектури. Систем омогућава креирање, управљање и извршавање послова на удаљеним агентима, као и праћење њиховог статуса у реалном времену. Посебна пажња посвећена је безбедности, поузданости и скалабилности решења.

Главни проблем који је решен овим приступом јесте поуздана и безбедна комуникација између дистрибуираних компоненти. То је постигнуто применом *WebSocket* протокола, хибридног шифровања и контроле приступа по улогама. Поред тога, систем обезбеђује једноставан начин додавања нових агената и аутоматско оркестрирање извршавања послова.

Предности овог решења су у његовој модуларности, лакој проширивању и могућности примене у различитим окружењима. Коришћење *Docker*-а поједностављује инсталацију и изолацију компоненти, док интеграција са алатима као што су *Umami* и *PostHog* омогућава увид у коришћење система и понашање корисника.

С друге стране, као главни недостаци могу се издвојити већа сложеност инфраструктуре и потреба за пажљивим управљањем безбедносним сертификатима и кључевима. Такође, у тренутној верзији систем није оптимизован за рад у окружењима са великим бројем агената и извршења истовремено.

Правци даљег развоја обухватају унапређење механизма за динамичко скалирање агената, напредније управљање ресурсима и интеграцију са додатним *DevOps* алатима. Потенцијално проширење обухвата и развој визуелног едитора за дефинисање послова, као и унапређење система извештавања и праћења перформанси.

---



---

## Списак слика

---

Слика 1	Архитектура система .....	8
Слика 2	Комуникација између мастер-а и агента .....	8
Слика 3	Дијаграм класа система <i>Test Hub Mini</i> .....	10
Слика 4	Активирање двофакторске аутентикације (2FA) .....	12
Слика 5	Конфигурација <i>Nginx reverse proxy</i> сервера .....	14
Слика 6	Део форме за креирање посла са уносом основних информација и <i>webhook</i> адреса. ....	16
Слика 7	Пример форме за додавање корака у послу. ....	16
Слика 8	Пример форме за дефинисање параметара ( <i>properties</i> ) различитих типова у послу. ....	17
Слика 9	Ток извршавања <i>Clone</i> корака и комуникација између мастера, <i>Gitea</i> сервера и агената. ....	18
Слика 10	Ток извршавања <i>Clone</i> корака и комуникација између мастера, <i>Gitea</i> сервера и агената. ....	19
Слика 11	Пример приказа <i>in-app</i> нотификација у систему. ....	22
Слика 12	Пример приказа <i>Microsoft Teams</i> нотификација. ....	23

---

---

## Списак листинга

---

Листинг 1	Имплементација класа за контролу приступа. ....	12
-----------	---	----

---

---

## Биографија

---

Никола Трајковић је рођен 19. децембра 2002. године у Врању. Основну школу „Доситеј Обрадовић“ у Врању завршио је 2017. године као носилац Вукове дипломе. Гимназију „Бора Станковић“ у Врању завршио је 2021. године као носилац Вукове дипломе. Исте године уписује Факултет техничких наука у Новом Саду, одсек Софтверско инжењерство и информационе технологије. Положио је све испите предвиђене планом и програмом.

---

---


## Литература

---

- [1] „Architecting for Scale — Jenkins Distributed Builds Architecture“. [На Интернету]. Available at: <https://www.jenkins.io/doc/book/scaling/architecting-for-scale/>
- [2] „About GitHub-hosted runners“. [На Интернету]. Available at: <https://docs.github.com/en/actions/concepts/runners/about-github-hosted-runners>
- [3] „GitLab Architecture Overview“. [На Интернету]. Available at: <https://docs.gitlab.com/development/architecture/>
- [4] „NGINX Reverse Proxy Guide“. [На Интернету]. Available at: <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/>
- [5] „Module ngx\_http\_auth\_request\_module“. [На Интернету]. Available at: [https://nginx.org/en/docs/http/ngx\\_http\\_auth\\_request\\_module.html](https://nginx.org/en/docs/http/ngx_http_auth_request_module.html)
- [6] „Module ngx\_stream\_core\_module“. [На Интернету]. Available at: [https://nginx.org/en/docs/stream/ngx\\_stream\\_core\\_module.html](https://nginx.org/en/docs/stream/ngx_stream_core_module.html)
- [7] „Create Incoming Webhook in Microsoft Teams“. [На Интернету]. Available at: <https://learn.microsoft.com/en-us/microsoftteams/platform/webhooks-and-connectors/how-to/add-incoming-webhook>
- [8] „Sending messages using Incoming Webhooks in Slack“. [На Интернету]. Available at: <https://api.slack.com/messaging/webhooks>
- [9] „Securing your webhooks — GitHub Docs“. [На Интернету]. Available at: <https://docs.github.com/en/webhooks/using-webhooks/securing-your-webhooks>
- [10] „Webhook secrets — GitLab Documentation“. [На Интернету]. Available at: <https://docs.gitlab.com/ee/user/project/integrations/webhooks.html#secret-token>
- [11] „Umami Analytics Documentation“. [На Интернету]. Available at: <https://umami.is/docs>
- [12] „PostHog Documentation“. [На Интернету]. Available at: <https://posthog.com/docs/getting-started/install>

---



	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6
	<b>КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА</b>

Редни број, **РБР**:

Идентификациони број, **ИБР**:

Тип документације, **ТД**: Монографска документација

Тип записа, **ТЗ**: Текстуални штампани материјал

Врста рада, **ВР**: Дипломски - бечелор рад

Аутор, **АУ**: Никола Трајковић

Ментор, **МН**: Др Бранко Милосављевић, редовни професор

Наслов рада, **НР**: Дизајн и реализација дистрибуираног *CI/CD* алата са подршком за *2FA*, нотификације и корисничку аналитику

Језик публикације, **ЈП**: српски/ћирилица

Језик извода, **ЈИ**: српски/енглески

Земља публикација, **ЗП**: Република Србија

Уже географско подручје, **УГП**: Војводина

Година, **ГО**: 2025

Издавач, **ИЗ**: Ауторски репринт

Место и адреса, **МА**: Нови сад, трг Доситеја Обрадовића 6

Физички опис рада, **ФО**: 7/37/0/0/13/0/0  
(поглавља/страница/ цитата/табела/слика/графика/прилога)

Научна област, **НО**: Софтверско инжењерство и информационе технологије

Научна дисциплина, **НД**: Примењене рачунарске науке и информатика

Предметна одредница/Кључне речи, **ПО**: *CI/CD* системи, мастер-агент архитектура, дистрибуирано извршавање задатака

**УДК**

Чува се, **ЧУ**: У библиотеци Факултета техничких наука, Нови Сад

Важна напомена, **ВН**:

Извод, **ИЗ**: Представљена је имплементација система који омогућава дистрибуирано извршавање задатака у оквиру мастер-агент софтверске архитектуре. Систем се састоји од постојећих компоненти отвореног кода и сопствених решења

Датум прихватања теме, **ДП**:

Датум одбране, **ДО**: 31.10.2025.

Чланови комисије, **КО**:

Председник:	Др Горан Сладић, редовни професор
Члан:	Др Мирослав Зарић, редовни професор
Члан, ментор:	Др Бранко Милосављевић, редовни професор

Потпис ментора



UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES  
21000 NOVI SAD, Trg Dositeja Obradovića 6

## KEY WORDS DOCUMENTATION

Accession number, **ANO**:

Identification number, **INO**:

Document type, **DT**: Monographic publication

Type of record, **TR**: Textual printed material

Contents code, **CC**:

Author, **AU**: Nikola Trajković

Mentor, **MN**: Branko Milosavljević, Phd., full professor

Title, **TI**: Design and Implementation of a Distributed CI/CD Tool with Support for 2FA, Notifications, and User Analytics

Language of text, **LT**: Serbian

Language of abstract, **LA**: Serbian

Country of publication, **CP**: Republic of Serbia

Locality of publication, **LP**: Vojvodina

Publication year, **PY**: 2025

Publisher, **PB**: Author's reprint

Publication place, **PP**: Novi Sad, Dositeja Obradovica sq. 6

Physical description, **PD**: 7/37/0/0/13/0/0  
(chapters/pages/ref./tables/pictures/graphs/appendixes)

Scientific field, **SF**: Software Engineering and Information Technologies

Scientific discipline, **SD**: Applied computer science and informatics

Subject/Key words, **S/KW**: CI/CD systems, master-agent architecture, distributed task execution

### UC

Holding data, **HD**: The Library of Faculty of Technical Sciences, Novi Sad, Serbia

Note, **N**:

Abstract, **AB**: The implementation of a system that enables distributed task execution within a master-agent software architecture is presented. The system consists of existing open-source components and custom-built solutions.

Accepted by the Scientific Board on, **ASB**:

Defended on, **DE**: 31.10.2025.

Defended Board, **DB**: President: Goran Sladić, Phd., full professor

Member: Miroslav Zarić, Phd., full professor

Member, Mentor: Branko Milosavljević, Phd., full professor

Mentor's sign

Obrazac Q2.HA.04-05 - Izdanje 1



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА  
21000 НОВИ САД, Трг Доситеја Обрадовића 6

## ИЗЈАВА О НЕПОСТОЈАЊУ СУКОБА ИНТЕРЕСА

Изјављујем да нисам у сукобу интереса у односу ментор – кандидат и да нисам члан породице (супружник или ванбрачни партнер, родитељ или усвојитељ, дете или усвојеник), повезано лице (крвни сродник ментора/кандидата у правој линији, односно у побочној линији закључно са другим степеном сродства, као ни физичко лице које се према другим основама и околностима може оправдано сматрати интересно повезаним са ментором или кандидатом), односно да нисам зависан/на од ментора/кандидата, да не постоје околности које би могле да утичу на моју непристрасност, нити да стичем било какве користи или погодности за себе или друго лице било позитивним или негативним исходом, као и да немам приватни интерес који утиче, може да утиче или изгледа као да утиче на однос ментор-кандидат.

У Новом Саду, дана \_\_\_\_\_

Ментор

\_\_\_\_\_

Кандидат

\_\_\_\_\_