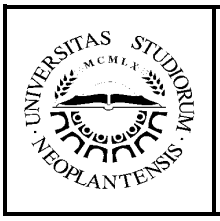


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Срђан Вујадиновић**
Број индекса: **РА 187/2022**

Тема рада: **Систем за тестирање ТВ пријемника са генерисањем тест случаја потпомогнуто претрагом**

Ментор рада: **проф. др Мирослав Поповић**

Нови Сад, јул, 2026

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

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Завршни (Bachelor) рад	
Аутор, **АУ**:		Срђан Вујадиновић	
Ментор, **МН**:		проф. др Мирослав Поповић	
Наслов рада, **НР**:		Систем за тестирање ТВ пријемника са генерисањем тест случаја потпомогнуто претрагом	
Језик публикације, **ЈП**:		Српски / ћирилица	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2026	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/41/11/9/16/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника и рачунарске комуникације	
Предметна одредница/Кључне речи, **ПО**:		генерисање потпомогнуто претрагом, дигитални ТВ пријемник, обезбјеђење квалитета, векторска база података, Appium	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У оквиру овог рада развијен је прототип система који коришћењем генерисања потпомогнутог претрагом предлаже тест случајеве и почетни скелет Appium теста за дигиталне ТВ пријемнике на основу постојеће документације са захтјевима. Систем ту документацију парсира и индексира у векторску базу података, како би за задати упит семантичком претрагом пронашао релевантан садржај, на основу којег велики језички модел генерише приједлог тест случаја, уз пресликавање на одговарајуће захтјеве и критеријуме прихватљивости. Развијено рјешење је евалуирано поређењем са постојећом, ручно писаном документацијом кроз анализу релевантности, покривености критеријума прихватљивости и употребљивости предложеног Appium скелета.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	проф. др Илија Башичевић	
	Члан:	проф. др Иван Каштелан	Потпис ментора
	Члан, ментор:	проф. др Мирослав Поповић	

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**:		Bachelor Thesis	
Author, **AU**:		Srđan Vujadinović	
Mentor, **MN**:		Miroslav Popović, PhD	
Title, **TI**:		System for testing TV receivers with retrieval augmented generation of test cases	
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**:		2026	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/41/11/9/16/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering and Computer Communications	
Subject/Key words, **S/KW**:		retrieval augmented generation, digital tv receiver, quality assurance, vector database, Appium	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This project presents the development of a system prototype that, using retrieval augmented generation, proposes test cases and an initial Appium test skeleton for digital TV receivers, based on requirement documentation. The system indexes this documentation into a vector database, so that for a given query, semantic search retrieves relevant content, based on which a large language model generates a proposed test case, together with a mapping to the corresponding requirements and acceptance criteria. The developed solution was evaluated by comparison with existing, manually written QA documentation, through an analysis of relevance, acceptance criteria coverage, and usability of the proposed Appium skeleton.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Ilija Bašičević, PhD	
	Member:	Ivan Kaštelan, PhD	Mentor's sign
	Member, Mentor:	Miroslav Popović, PhD	

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ИЗРАДУ ЗАВРШНОГ (BACHELOR) РАДА	Датум:

(Податке уноси предметни наставник - ментор)

Врста студија:	а) Основне академске студије б) Основне струковне студије
Студијски програм:	Рачунарство и аутоматика
Руководилац студијског програма:	проф. др Иван Каштелан

Студент:	Срђан Вујадиновић	Број индекса:	РА 187/2022
Област:	Електротехника и рачунарство		
Ментор:	проф. др Мирослав Поповић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ ЗАВРШНОГ (BACHELOR) РАДА:

Систем за тестирање ТВ пријемника са генерисањем тест случаја потпомогнуто претрагом

ТЕКСТ ЗАДАТКА:

1. Упознати се са релевантном литературом и написати теоријске основе рада.
2. Направити пројекат система са деловима за: (1) учитавање, парсирање и структурирање садржаја REQ докумената, (2) имплементацију векторске базе података за индексирање и семантичку претрагу релевантних делова документације, (3) генерисање предлога тест сценарија, уз пресликавање тестова на одговарајуће захтеве и критеријуме прихватљивости, ради подршке праћењу трага информације, и (4) генерисања иницијалног предлога Appium тест скелета.
3. Имплементирати пројектовани систем.
4. Евалуирати пројектовани систем на репрезентативном скупу REQ докумената, кроз поређење генерисаних резултата са постојећом ручно писаном QA документацијом. Евалуација треба да укључи анализу релевантности генерисаних тестова, покривености критеријума прихватљивости, квалитета пресликавања ради праћења трагова и практичне употребљивости предложеног Appium тест скелета.
5. Уочити предности и мане имплементираног решења.
6. Написати текст дипломског рада.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Ментора

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

_____

Захвалност

Велику захвалност дугујем свом ментору, проф. др Мирославу Поповићу, на издвојеном времену, помоћи, корисним савјетима и подршци приликом израде овог рада.

Такође, огромна захвалност мом техничком ментору, Јовану Зубовићу, који ми је био на располагању током практичне израде дипломског рада.

Посебно се захваљујем својој породици и пријатељима на константној подршци током цјелокупног школовања.

САДРЖАЈ

Списак слика

Списак табела

Скраћенице

RAG – (енг. Retrieval Augmented Generation) – генерисање потпомогнуто претрагом

LLM – (енг. Large Language Model) – велики језички модел

STB – (енг. Set-Top Box) – дигитални ТВ пријемник

QA – (енг. Quality Assurance) – обезбјеђење квалитета

API – (енг. Application Programming Interface) – скуп правила и функција за комуникацију између два софтверска система

REQ – (енг. Requirement) – захтјев

JSON – (енг. JavaScript Object Notation) – формат за размјену података

JSONL – (енг. JSON Lines) – формат у којем је сваки ред засебан JSON запис

MCP – (енг. Model Context Protocol) – протокол који омогућава великим језичким моделима да позивају спољашње алате и изворе података

ADB – (енг. Android Debug Bridge) – алат командне линије за комуникацију и управљање андроид уређајима

AST – (енг. Abstract Syntax Tree) – апстрактно синтаксно стабло, хијерархијски приказ структуре изворног кода

1. Увод

У процесу обезбјеђења квалитета (енгл. Quality Assurance, QA) дигиталних ТВ пријемника користи се документација са захтјевима записана у структурираном текстуалном облику. Ова документација садржи велики број појединачних захтјева, који су груписани по функционалним цјелинама. Уз већину захтјева наведени су и одговарајући тест случајеви, повезани са захтјевима преко јединственог идентификатора захтјева. Са растом обима такве документације, ручно проналажење релевантних захтјева, припрема тест случаја и провјера њихове повезаности са захтјевима постају временски захтјевни задаци, подложни пропустима и непотпуној покривености.

Велики језички модели (енгл. Large Language Models, LLM) у посљедњих неколико година показали су значајан напредак у разумијевању и генерисању текста и програмског кода. Ипак, ови модели су ограничени на знање присутно у подацима на којима су обучени. Због тога они повремено генеришу и тзв. халуцинације - одговоре који дјелују увјерљиво, али су чињенично нетачни или у потпуности измишљени [1]. Из тог разлога, њихова класична примјена у процесу тестирања ТВ пријемника би представљала озбиљан ризик. Нетачан и измишљен тест случај, генерисан без ослонца на стварну документацију, може створити погрешну представу о покривености тестирања умјесто да допринесе процесу тестирања.

Генерисање потпомогнуто претрагом (енгл. Retrieval Augmented Generation, RAG) представља један од приступа за ублажавање наведеног проблема. Ова техника се заснива на томе да се прије генерисања одговора претражи спољашња база знања и да се пронађени садржај достави великом језичком моделу као контекст на коме треба да заснује свој одговор. Тако генерисан одговор почива на стварном и провјерљивом садржају, а не искључиво на ономе што је модел запамтио током обуке [2]. На тај начин се може смањити ризик од измишљања непостојећих чињеница.

У циљу рјешавања наведеног проблема, у овом раду развијен је прототип система који комбинује претрагу документације са захтјевима и генерисање потпомогнуто претрагом. Систем учитава, парсира и структурира сву потребну документацију. Након тога је индексира у векторској бази података ради семантичке претраге. На основу корисничког упита, из базе се проналази релевантан контекст, који се доставља великом језичком моделу. На основу тог контекста, велики језички модел генерише приједлог тест случаја, уз пресликавање на одговарајуће захтјеве и критеријуме прихватљивости, или генерише почетни скелет Appium теста. Систем такође омогућава да се, за постојећи захтјев, провјери да ли су сви критеријуми прихватљивости већ покривени постојећим тест случајевима.

Циљ рада је да се испита у којој мјери генерисање потпомогнуто претрагом може допринијети унапређењу процеса анализе захтјева, припреми тест случајева и почетној фази аутоматизације тестирања ТВ пријемника, те да се уоче предности и мане овако имплементираног рјешења.

Рад у наставку прво излаже теоријске основе неопходне за разумијевање рјешења (поглавље 2), затим концепт рјешења (поглавље 3), програмско рјешење (поглавље 4), резултате евалуације (поглавље 5), закључак (поглавље 6) и коришћену литературу (поглавље 7).

2. Теоријске основе

У овом поглављу представљени су појмови потребни за разумијевање система описаног у наставку рада. Најприје су објашњени вјештачка интелигенција и велики језички модели. Након тога су описани генерисање потпомогнуто претрагом, векторске репрезентације текста, семантичка претрага и векторске базе података. Посљедњи дио поглавља односи се на документацију у формату reStructuredText и основни начин рада Appium окружења.

2.1 Вјештачка интелигенција

Вјештачка интелигенција (енгл. Artificial Intelligence, AI) обухвата методе и системе који извршавају задатке за које су потребни закључивање, препознавање образаца, обрада језика или доношење одлука. Један од најзначајнијих савремених приступа у оквиру вјештачке интелигенције јесте машинско учење. Код машинског учења понашање модела не описује се искључиво ручно написаним правилима, већ се обрасци издвајају из података [3].

Дубоко учење представља посебну област машинског учења засновану на неуронским мрежама са већим бројем слојева. Његов развој омогућио је значајан напредак у обради природног језика, препознавању говора, анализи слика и генерисању садржаја. На методама дубоког учења засновани су и велики језички модели. За овај рад посебно су важни, јер могу да обрађују текстуалне инструкције, генеришу описе тест случајева и предложе почетну структуру програмског кода.

2.1.1 Велики језички модели

Велики језички модели (енгл. Large Language Models, LLM) су генеративни модели намијењени обради и генерисању природног језика и програмског кода. Већина савремених модела заснива се на трансформер архитектури, чија је основна карактеристика механизам пажње. Механизам сопствене пажње омогућава да се при обради сваког елемента улазне секвенце узму у обзир његове везе са осталим елементима, без ослањања искључиво на секвенцијалну обраду карактеристичну за старије рекурентне архитектуре [4].

Текст се прије обраде дијели на **токене**, односно мање јединице које могу представљати ријеч, дио ријечи, број или знак интерпункције. Модел затим, на основу улазних токена и образаца научених током обуке, процјењује вјероватноћу наредних токена. На тај начин може да генерише повезан текст, одговор на питање или дио програмског кода.

Инструкција која се доставља моделу назива се **упит** (енгл**.** prompt). Поред самог питања или задатка, упит може да садржи доменски контекст, ограничења излаза и примјере очекиваног облика одговора. Достављање примјера у самом упиту омогућава моделу да прилагоди структуру излаза без додатне обуке, што се назива **учењем у контексту** (енгл. in-context learning) [5].

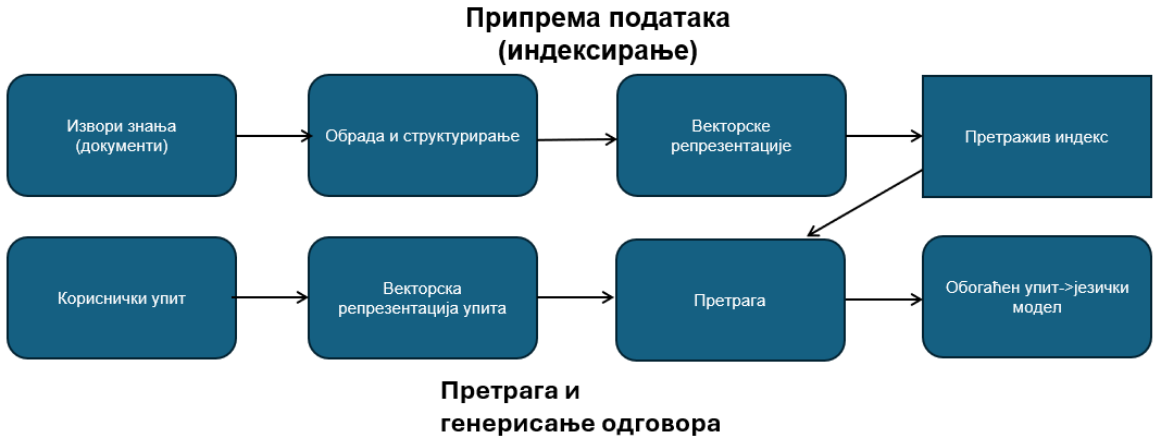
Одговори великог језичког модела нису гарантовано тачни. Модел може да произведе садржај који је граматички исправан и увјерљив, али није поткријепљен поузданим подацима. Та појава се назива **халуцинација** [1]. Поред тога, модел без додатно достављеног контекста нема непосредан приступ интерној документацији организације, па не може поуздано знати називе конкретних захтјева, постојећих тестова или других пројектних артефаката. Један од начина да се моделу обезбиједи доменски контекст јесте **генерисање потпомогнуто претрагом**, описано у наредном одјељку.

2.2 Генерисање потпомогнуто претрагом

Генерисање потпомогнуто претрагом је приступ у коме се велики језички модел комбинује са спољашњим извором знања. Релевантни дијелови документације проналазе се прије генерисања и додају у упит модела. У оригиналном RAG приступу генеративни модел представља параметарску меморију, док претраживи индекс докумената има улогу непараметарске меморије [2].

Ток обраде обично се дијели на два корака. У првом, изворни документи се учитавају, структурирају, претварају у векторске репрезентације и уписују у претраживи индекс. Прије свега тога се обично организују у цјелине које су погодне за претрагу, зависно од структуре и намјене изворне документације. У другом, кориснички упит се такође представља вектором и пореди се са сачуваним документима, а најсличнији резултати чине контекст који се прослиједјује моделу. На тај начин модел не добија цјелокупну документацију, што би било непрактично због ограничене дужине контекста, већ само оне дијелове документације који су за конкретан упит најрелевантнији.

Знање о документацији, у оваквом приступу, не улази у параметре великог језичког модела, већ остаје одвојено у спољашњем и претраживом индексу. Захваљујући томе, ажурирање документације своди се на измјену тог индекса док сам модел остаје непромијењен. Није потребно поновно обучавање модела за сваку измјену изворних података. Ова особина је истакнута и у изворном раду о генерисању потпомогнутом претрагом [2]. Поред тога, пошто је сваки дио контекста повезан са конкретним извором, одговор модела је могуће накнадно повезати са документом на коме почива.



Слика 2.1 - Ток припреме, претраге и генерисања у RAG систему

Ипак, сама претрага не гарантује добијање тачног одговора од модела. Ако претрага врати документ који није стварно повезан са упитом или ако модел погрешно протумачи исправан контекст, резултат и даље може бити непотпун или нетачан. Због тога квалитет оваквог система зависи подједнако од квалитета претраге и квалитета самог генерисања. Управо ово је и разлог зашто је поред самог одговора важно сачувати идентификаторе и метаподатке изворних докумената који су коришћени у генерисању. На тај начин одговор остаје могуће повезати са конкретним изворним садржајем на коме почива.

2.2.1 Векторске репрезентације текста и семантичка претрага

Да би се текст могао претраживати према значењу, потребно га је представити у нумеричком облику. Векторска репрезентација текста (енгл. embedding) јесте низ реалних бројева добијен примјеном модела за кодирање текста. Циљ таквог модела је да семантички сродни текстови добију међусобно блиске векторе, а да се текстови различитог значења налазе на већој удаљености у векторском простору.

Ова особина омогућава семантичку претрагу. За разлику од претраге која зависи од дословног поклапања ријечи, семантичка претрага може повезати упит и документ који користе различите изразе за сродан појам. На примјер, упит о приказу електронског програмског водича (енгл. Electronic Program Guide, EPG) може бити близак захтјеву који користи скраћеницу EPG, иако не постоји поклапање датих ријечи.

За ефикасно поређење реченица и краћих текстова развијени су модели који производе векторске репрезентације погодне за израчунавање сличности. Један од познатијих приступа је Sentence-BERT, који за појединачне реченице формира векторе погодне за семантичко поређење [6]. Конкретан модел и библиотека коришћени у реализованом систему биће описани у поглављу о програмском рјешењу.

За поређење вектора користе се различите мјере сличности и удаљености. Косинусна сличност посматра угао између два вектора, Еуклидска удаљеност њихово праволинијско

растојање у простору, док скаларни производ зависи и од правца и од величине вектора [7]. Основне особине ових мјера приказане су у табели 2.1.

МЈЕРА	ФОРМУЛА	ОСНОВНА ОСОБИНА
Косинусна сличност	$\cos(\theta) = \frac{a \cdot b}{\|a\|\|b\|}$	Поређује угао између вектора и наглашава њихов правац.
Еуклидска удаљеност	$\mathrm{d(a,b)} = \sqrt{(\Sigma_i\,(\mathrm{a_i} - \mathrm{b_i})^2)}$	Мјери праволинијско растојање између двије тачке.
Скаларни производ	$a \cdot b = \sum a_i b_i$	Зависи и од правца и од величине вектора.

Табела 2.1 - Уобичајене мјере сличности и удаљености вектора

2.2.2 Векторске базе података

Векторска база података намијењена је чувању и претраживању векторских репрезентација. Запис у таквој бази обично садржи вектор, јединствени идентификатор и придружене метаподатке, као што су наслов документа, тип документа или идентификатор захтјева. Метаподаци омогућавају да се након претраге пронађени вектор повеже са изворним текстом и другим подацима потребним за даљу обраду.

При извршавању упита, вектор упита пореди се са векторима у бази, након чега се враћа унапријед означен број најсличнијих записа, означен као top-k. Резултати могу бити ограничени прагом сличности, како би се одбацили записи чија је повезаност са упитом недовољна. За веће скупове података могуће је користити и посебне векторске индексе који убрзавају проналажење најближих вектора.

У реализованом систему за ову намјену употријебљена је база Qdrant. Она омогућава чување вектора и припадајућих метаподатака, као и претрагу према изабраној мјери сличности [8]. Конкретна организација података и начин формирања колекција биће представљени у поглављу о програмском рјешењу.

Табела 2.2 - Основни појмови векторске базе података

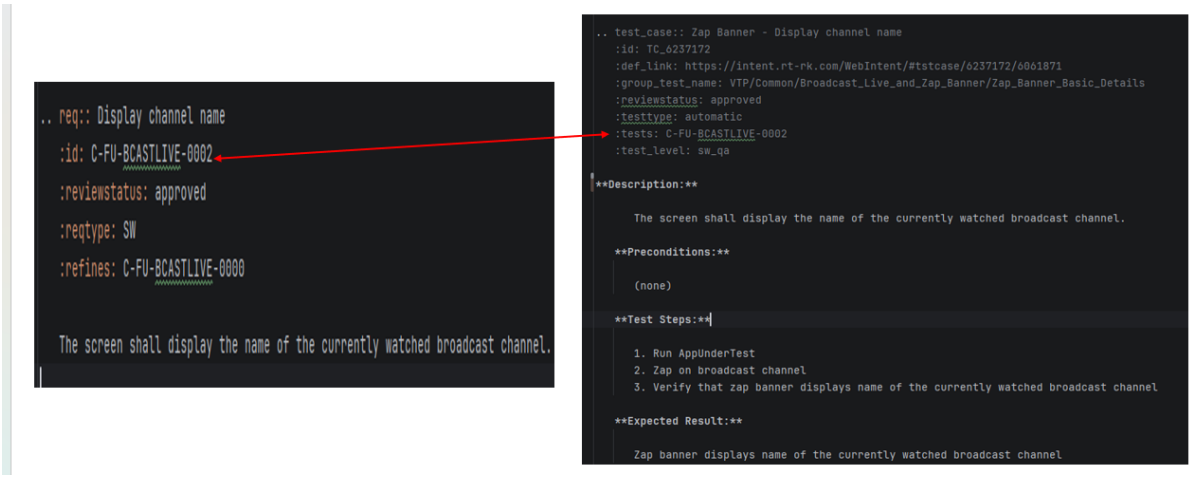
ПОЈАМ	ЗНАЧЕЊЕ
Вектор	Нумеричка репрезентација текстуалног садржаја.
Идентификатор	Јединствена ознака записа у бази.
Праг сличности	Минимална прихватљива оцјена испод које се резултат одбацује.
Top-k	Број најсличнијих резултата који ће се вратити за упит.

2.3 Документација са захтјевима у формату reStructuredText

reStructuredText (RST) је текстуални формат за структурисано документовање, развијен у оквиру пројекта Docutils. Формат користи читљиве текстуалне ознаке за наслове, листе, табеле, референце и друге елементе документа. Директиве представљају проширив механизам којим се у документ уводе посебно означени блокови са аргументима, опцијама и садржајем [9].

Документација анализирана у овом раду користи пројектно дефинисане директиве req и test_case. Оне нису стандардне Docutils директиве, већ структурирани блокови прилагођени QA документацији. Блок req описује захтјев, док блок test_case садржи ручно написан тест случај са описом, предусловима, корацима и очекиваним резултатом. У наставку рада за ову документацију са захтјевима користи се ознака REQ документација.

Сваки захтјев и тест случај имају јединствени идентификатор у пољу :id:. Тест случај се повезује са захтјевом навођењем идентификатора захтјева у пољу :tests:. На тај начин могуће је утврдити који тестови су придружени појединачном захтјеву. Примјер повезивања приказан је на слици 2.2.



Слика 2.2 - Примјер постојећег захтјева и тест случаја

Поред идентификатора, директиве садрже и друга поља као што су статус прегледа, тип захтјева, тип теста, ниво тестирања и веза ка изворном систему за управљање тестовима. Значење најважнијих поља за овај рад наведено је у табели 2.3.

Критеријум прихватљивости представља провјерљив услов који треба да буде испуњен како би се захтјев сматрао реализованим. Сљедљивост означава могућност праћења веза између захтјева, тестова и резултата тестирања. Добро успостављена сљедљивост помаже при процјени покривености захтјева и при анализи утицаја њихових измјена [11]. У анализираним документима критеријуми прихватљивости нису записани као посебно нумерисана поља, па

начин њиховог извођења представља дио пројектованог рјешења и биће описан у наредном поглављу.

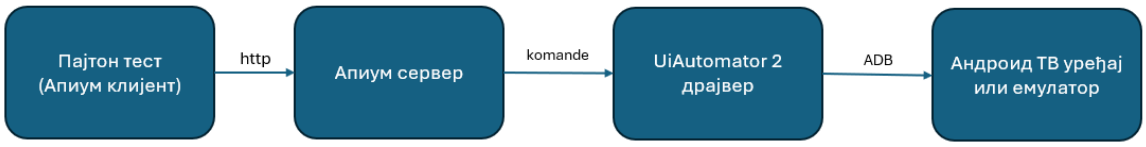
ПОЉЕ	ТИП БЛОКА	ЗНАЧЕЊЕ
:id:	захтјев, тестни случај	Јединствени идентификатор захтјева или тест случаја.
:reviewstatus:	захтјев, тестни случај	Статус прегледа и одобравања документа.
:reqtype:	Захтјев	Тип захтјева, на примјер софтверски захтјев
:refines:	Захтјев	Веза ка општем захтјеву који посматрани захтјев прецизира.
:testtype:	тестни случај	Тип теста, на примјер ручни или аутоматски
:tests:	тестни случај	Идентификатор захтјева који тест покрива.
:test_level:	тестни случај	Ниво тестирања у оквиру QA процеса.
:def_link:	тестни случај	Везе ка изворном запису тест случаја у спољашњем систему.

Табела 2.3 - Опис директива коришћених у QA документацији

2.4 Appium и аутоматизација тестирања

У оквиру овог рада посматрају се ТВ пријемници засновани на Андроид платформи. Поред класичних елемената корисничког интерфејса, такви уређаји подржавају функције карактеристичне за телевизијско окружење, као што су репродукција емитованог садржаја, промјена канала, електронски програмски водич и снимање. Интеракција се у великој мјери извршава помоћу тастера даљинског управљача.

Appium је отворени екосистем намијењен аутоматизацији корисничких интерфејса на различитим платформама, укључујући Андроид. Заснован је на WebDriver архитектури. Тест написан у клијентској библиотеци шаље команде Appium серверу, а сервер их прослеђује одговарајућем драјверу. За Андроид платформу често се користи UiAutomator2 драјвер, који WebDriver команде преводи у операције над Андроид уређајем или емулатором [10].



Слика 2.3 - Поједностављена архитектура Appium аутоматизације на Андроид ТВ платформи

Тестна скрипта може да покрене апликацију, пошаље команду тастера, пронађе елементе корисничког интерфејса и провјери њихово стање. Елементи се проналазе помоћу локатора, односно селектора који описују начин њихове идентификације. Резултат извршавања команде враћа се клијенту, који на основу њега наставља тест или биљежи неуспјех.

Почетни тест скелет представља основну структуру аутоматизованог теста која се касније допуњује конкретним предусловима, корацима, локаторима и провјерама. Начин на који је у овом раду организовано генерисање Appium скелета и његово прилагођавање постојећем програмском оквиру биће описан у поглављима о концепту и програмском рјешењу.

3. Концепт рјешења

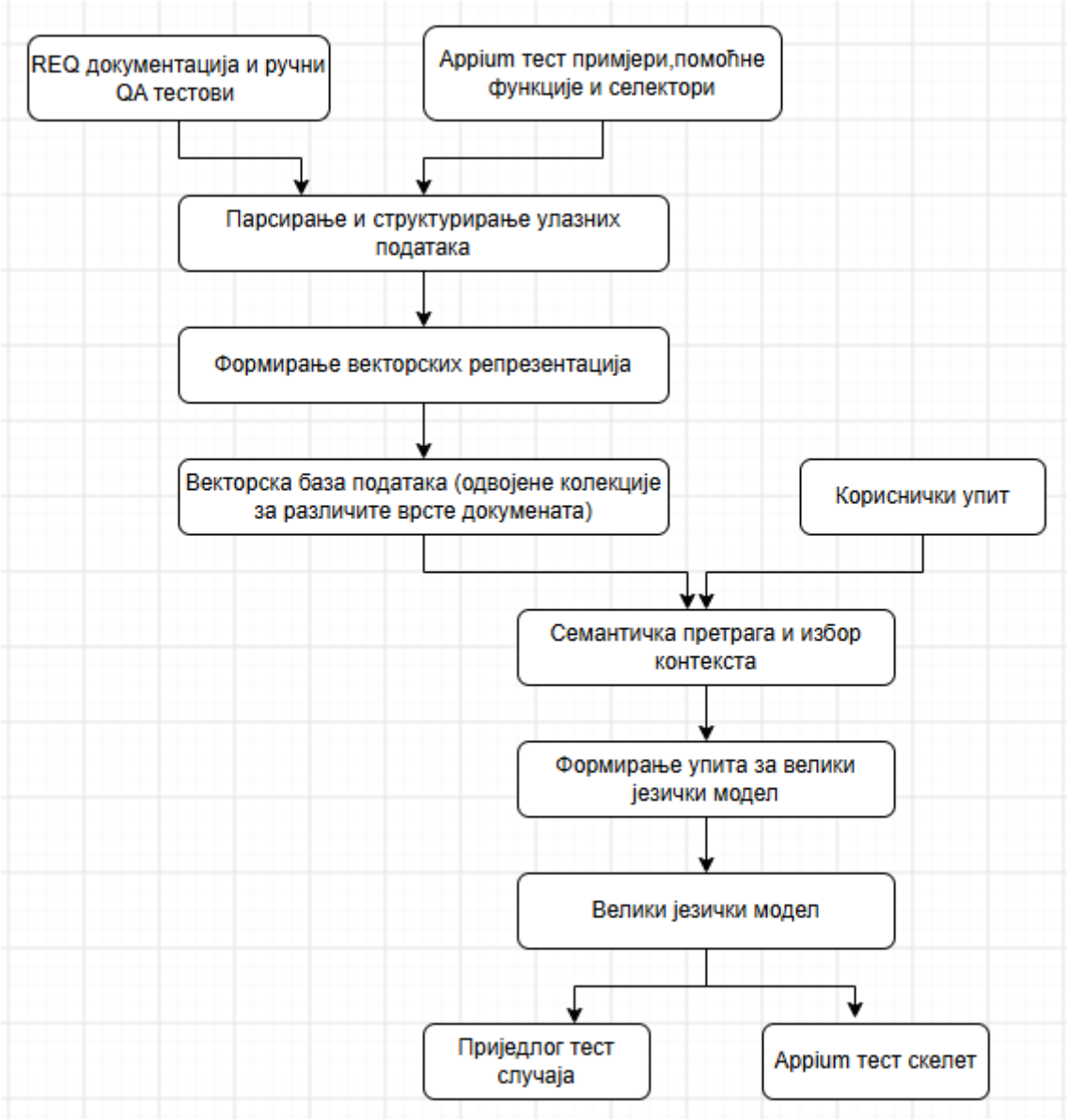
У овом поглављу описана је основна замисао система и начин на који су његове цјелине повезане. Прво је приказана архитектура система, а затим су описани поступци припреме извора знања, организације векторске базе података, семантичке претраге и формирања два излаза: приједлога тест случаја и почетног Appium тест скелета. Конкретни програмски модули, функције, библиотеке и параметри биће представљени у поглављу о програмском рјешењу.

3.1 Архитектура система

Систем је подијељен на двије основне цјелине. Прва цјелина обухвата припрему и индексирање података. Покреће се када се документација први пут дода у систем или када је потребно освјежити садржај векторске базе. Друга цјелина обрађује кориснички упит, проналази одговарајући контекст и позива велики језички модел.

Улазни материјал није ограничен на једну врсту документа. За генерисање тест случаја најважнији су захтјеви и ручно написани тестови. За Appium скелет, поред тога, потребни су још и примјери аутоматизованих тестова, помоћне функције и селектори корисничког интерфејса. Сваки од ових извора прво пролази кроз одговарајући поступак обраде, након чега се добијени подаци записују у структурираном облику прилагођеном врсти извора.

Након структурирања, за сваки запис се формира текст намијењен семантичкој претрази. Тај текст се претвара у векторску репрезентацију и уписује у одговарајућу колекцију векторске базе. Кориснички упит пролази кроз исти поступак претварања у вектор, а претрага враћа документе који су му најближи по значењу. Пронађени документи се затим организују у контекст који се, заједно са корисничким упитом, доставља великом језичком моделу. Архитектура система приказана је на слици 3.1.



Слика 3.1 - Архитектура система за генерисање тест случаја и Appium тест скелета

Табела 3.1 - Главне цјелине система

Цјелина	**Улога у систему**
Парсирање и структурирање	Издваја садржај и метаподатке из различитих улазних датотека.
Формирање векторских репрезентација	Припрема текстове и претвара их у векторе погодне за семантичку претрагу.
Векторска база података	Чува векторе, изворни садржај и метаподатке у одвојеним колекцијама.
Претрага и формирање контекста	Проналази релевантне документе и организује их у контекст за језички модел.
Генерисање	Формира приједлог тест случаја или почетни Appium скелет.
Подршка за провјеру	Издваја и пореди идентификаторе захтјева и критеријума прихватљивости у структурираном излазу.

Припрема података и обрада корисничког упита намјерно су раздвојене. На тај начин индексирање не мора да се понавља при сваком корисничком упиту. Једном припремљени вектори остају сачувани у бази, док се током рада система израчунава само вектор новог упита и претражује постојећа база.

3.2 Припрема извора знања

Квалитет претраге у великој мјери зависи од тога који је садржај издвојен из изворних датотека. Због тога свака врста улаза има засебан поступак обраде. Заједнички циљ је да се при формирању текста намијењеног претрази задрже информације које описују функционалност, начин њене провјере и поријекло записа.

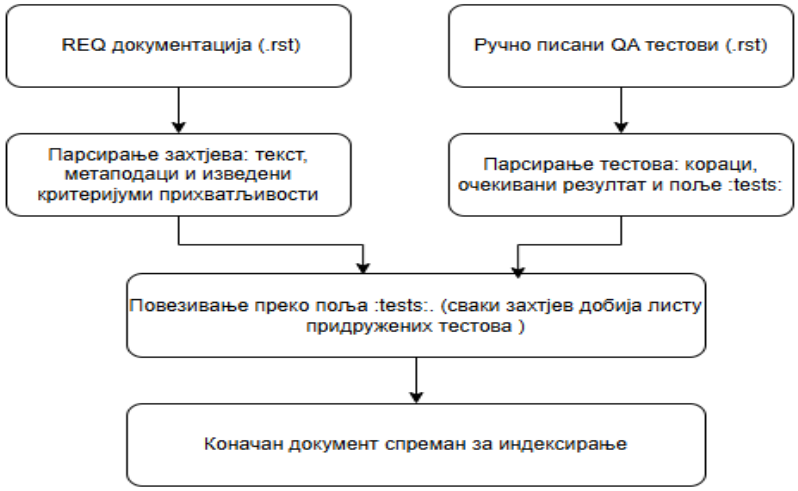
3.2.1 Захтјеви и ручно писани тест случајеви

Документација са захтјевима представља основни извор знања за формирање приједлога тест случаја. Сваки захтјев обрађује се као засебна цјелина и уз његов текст чувају се подаци који га јединствено одређују. Ручно написани тест случајеви обрађују се на сличан начин, при чему су за даљи рад најважнији предуслови, кораци, очекивани резултат и веза са захтјевом који тест провјерава.

Постојећа веза између теста и захтјева задржава се током обраде. На основу ње се за сваки захтјев могу пронаћи њему придружени тестови, па документ припремљен за претрагу, поред описа очекиваног понашања, садржи и доступне примјере начина на који је то понашање раније провјеравано. Овај дио контекста је нарочито користан када корисник тражи нови тест за функционалност која већ има сличне ручне тестове.

Критеријуми прихватљивости у улазним датотекама нису издвојени као посебно нумерисана поља. Зато се из текста захтјева издвајају реченице које описују понашање система и од њих се формирају изведени критеријуми прихватљивости. На примјер, у постојећој документацији налази се захтјев C-FU-APP-0016, у чијем тексту су наведене реченице „Application will be listed in TV input setup menu in Android TV settings“ и „Application should launch as TV input when last used input is set“. Од прве реченице формира се критеријум C-FU-APP-0016-AC01, којим се захтијева да апликација буде приказана у менију за подешавање ТВ улаза. Од друге реченице формира се критеријум C-FU-APP-0016-AC02, којим се захтијева да се апликација покрене као ТВ улаз када је постављена као посљедњи коришћени улаз. На овај начин сваки изведени критеријум остаје непосредно повезан са захтјевом из којег је настао. Први дио ознаке изведеног критеријума прихватљивости представља идентификатор изворног захтјева, ознака AC показује да је ријеч о критеријуму прихватљивости, док завршни број представља његов редни број у оквиру тог захтјева. Тако се и касније, при формирању тест случаја и провјери пресликавања, може јасно утврдити из којег захтјева одређени критеријум потиче.

Основни ток припреме ове документације дат је на слици 3.2.



Слика 3.2 - Ток припреме захтјева и ручно писаних тест случаја

3.2.2 Артефакти Appium оквира

За формирање Appium тест скелета сам текст захтјева није довољан. Он описује шта треба провјерити, али не садржи начин навигације кроз апликацију, називе доступних функција или селекторе елемената корисничког интерфејса. Због тога се као посебни извори знања користе постојећи Appium тестови, датотеке са помоћним и навигационим функцијама, као и селектори корисничког интерфејса.

Постојећи Appium тестови служе као примјери организације и редослиједа корака. Они показују структуру теста, редослијед позива и начин биљежења резултата. За најрелевантнији примјер међу пронађеним тестовима у контекст се може укључити и краћи исјечак изворног кода, јер сама листа коришћених функција понекад није довољна да покаже тачан редослијед и начин њихове употребе. Помоћне и навигационе функције показују које су операције већ доступне у програмском оквиру, док селектори одређују елементе корисничког интерфејса над којима се те операције извршавају. На примјер, постојећи тест Search_Live показује редослијед корака потребних за претрагу и покретање ТВ канала. У њему се функција navigate_to_search() користи за долазак до опције за претрагу, функција check_video_by_successive_frames() провјерава да ли је започета репродукција видео садржаја, док get_last_channel_from_log() из системског записа чита назив покренутог канала. Са друге стране, функција check_one_key() представља примјер употребе селектора. Она на основу назива групе и кључа, на примјер HOME_SCREEN и DISCOVER, преузима одговарајући селектор из рјечника STB_SELECTORS и провјерава да ли је тражени елемент видљив на екрану. На тај начин постојећи тестови показују редослијед корака, функције обезбјеђују већ припремљене операције, а селектори омогућавају проналажење конкретних елемената корисничког интерфејса.

Ове групе артефаката чувају се одвојено, како би се за сваки упит могао изабрати мањи број релевантних примјера, функција и селектора.

3.3 Организација векторске базе података и семантичка претрага

Након структурирања, за сваку врсту документа формира се текст намијењен семантичкој претрази. Тај текст и кориснички упит се претварају у векторе помоћу истог модела, тако да се могу поредити у заједничком векторском простору. Уз вектор у бази остају сачувани изворни садржај и метаподаци који су потребни након претраге.

Векторска база података организована је у четири логички одвојене колекције. Прва садржи захтјеве, изведене критеријуме прихватљивости и повезане ручне тестове. У преостале три чувају се редом описи помоћних функција, сажете представе постојећих Appium тестова и селектори корисничког интерфејса. Оваква организација омогућава да се различити дијелови контекста претражују независно и да се њихови резултати комбинују само онда када су потребни. Да су сви подаци смјештени у једну заједничку колекцију, претрага би морала да разматра и садржај који није релевантан за конкретан тип упита.

При обради упита база враћа ограничен број најсличнијих докумената. Записи који не достигну задати праг сличности не укључују се у контекст, а дупликати се уклањају како би се избјегло непотребно понављање истог садржаја.

За формирање приједлога тест случаја претражује се колекција захтјева. При формирању Appium скелета комбинују се резултати из све четири колекције, јер су поред описа функционалности потребни и примјери постојећих тестова, доступне функције и селектори корисничког интерфејса. Конкретан начин претраге и повезивања ових резултата описан је у поглављу о програмском рјешењу.

3.4 Формирање приједлога тест случаја

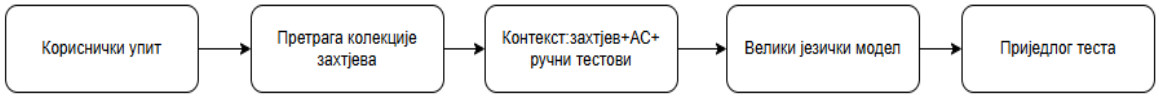
Ток почиње корисничким упитом који описује радњу за коју корисник жели да добије приједлог теста. На основу упита проналазе се релевантни захтјеви. За сваки од њих у контекст се укључују текст захтјева, изведени критеријуми прихватљивости и повезани ручно написани тестови, ако постоје.

Велики језички модел на основу овог контекста формира приједлог тест случаја. Резултат је организован кроз наслов, предуслове, тестне кораке и очекиване резултате. Уз тест се наводе захтјеви и изведени критеријуми прихватљивости на којима је приједлог заснован, чиме се генерисани тест може повезати са конкретним дијелом документације из којег је произашао. Ако претрага не врати тематски повезан контекст, систем треба да укаже на недостатак података, умјесто да формира тест који није поткријепљен документацијом.

На примјер, ако корисник затражи приједлог теста за провјеру информација које се приказују за изабрани догађај у EPG-у, семантичка претрага може пронаћи захтјев C-FU-BCASTEPG-0009. Уз овај захтјев у бази су сачувана три постојећа теста која одвојено провјеравају приказ назива догађаја, његовог описа и времена почетка и завршетка. Модел тако,

поред текста захтјева и изведених критеријума прихватљивости, добија и примјере начина на који је ова функционалност већ провјеравана. На основу тог контекста може да формира приједлог теста у којем се отвара EPG, изабере један догађај и провјере наведене информације, а уз резултат се наводе захтјев и критеријуми прихватљивости на којима је приједлог заснован.

Одговор намијењен кориснику приказује се у текстуалном облику. За потребе аутоматизоване провјере пресликавања користи се посебна варијанта истог тока у којој се резултат записује у структурираном облику. Одвајањем ова два режима задржава се читљив одговор за корисника, док програм може да издвоји идентификаторе потребне за евалуацију. Ток је приказан на слици 3.3.



Слика 3.3 - Ток формирања приједлога тест случаја

3.5 Подршка сљедљивости и провјери пресликавања

Сљедљивост се посматра на два нивоа. Први ниво је постојећа веза између ручно написаног теста и захтјева, преузета из улазне документације. Други ниво је веза генерисаног приједлога са захтјевима и изведеним критеријумима прихватљивости који су коришћени при његовом формирању.

За провјеру пресликавања користи се припремљени скуп случајева, при чему сваки садржи кориснички упит и очекиване идентификаторе. Систем формира структурирани резултат, издваја идентификаторе наведене у добијеном резултату и пореди их са очекиваним вриједностима. На тај начин може се одвојено посматрати пресликавање на нивоу захтјева и на нивоу изведених критеријума прихватљивости.

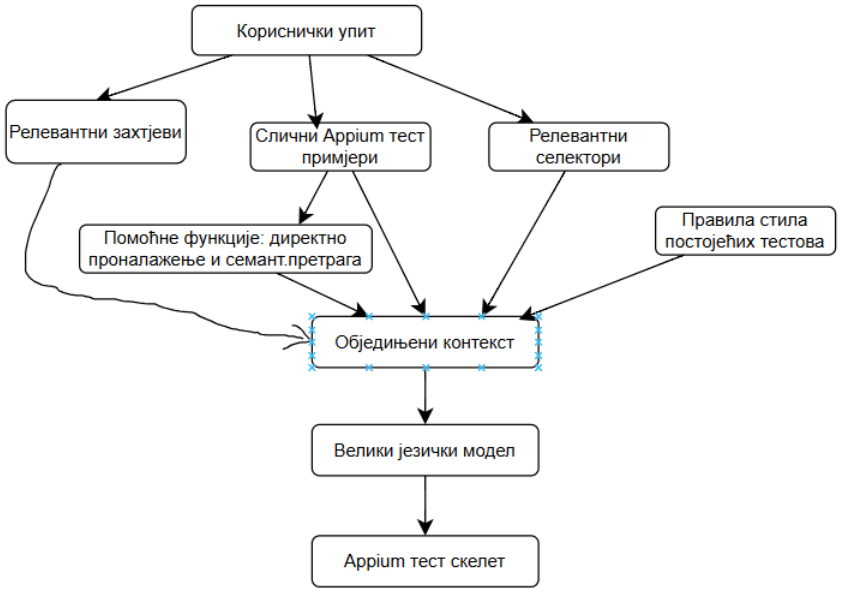
Ова провјера не обухвата оцјену квалитета написаних корака, очекиваних резултата и практичне употребљивости излаза. Ти аспекти биће посебно разматрани у поглављу са резултатима.

3.6 Формирање Appium тест скелета

Формирање Appium скелета почиње корисничким упитом, али се контекст припрема из више извора. Најприје се проналазе релевантни захтјеви и постојећи Appium тестови сличне намјене. На основу пронађених примјера издвајају се помоћне и навигационе функције које су потребне за реализацију траженог теста. Паралелно се проналазе релевантни селектори корисничког интерфејса.

У коначни контекст улазе одабрани захтјеви, сажети примјери тестова, подаци о доступним функцијама, селектори и правила стила којих се модел мора придржавати. На основу тога велики језички модел формира Пајтон скелет који треба да прати уобичајену структуру постојећих

тестова. Од модела се захтијева да користи само функције и селекторе наведене у достављеном контексту. Када потребан податак није доступан, од модела се тражи да то јасно назначи, умјесто да уведе произвољан назив.



Слика 3.4 - Формирање контекста за генерисање Appium скелета

3.7 Повезивање компоненти прототипа

Компоненте су повезане у јединствен прототип који подржава два начина употребе. У директном режиму корисник покреће одговарајуће скрипте преко интерфејса командне линије, при чему прототип обавља претрагу, формира упит са контекстом и позива велики језички модел. Други начин употребе заснива се на MCP протоколу (енгл. Model Context Protocol). Овај протокол омогућава да GitHub Copilot агент користи алате које му MCP сервер ставља на располагање. MCP сервер је дио прототипа који GitHub Copilot агенту омогућава приступ алатима за претрагу базе и припрему контекста. У оквиру овог рада реализован је локални MCP сервер који користи исту претрагу и припрему контекста као и остатак прототипа. Када корисник зада захтјев GitHub Copilot агенту, агент може да позове одговарајући алат на MCP серверу. Сервер затим проналази релевантне податке у локалној бази и враћа припремљени контекст. Коначан резултат не формира MCP сервер, већ GitHub Copilot на основу добијеног контекста.

Припрема улазних података и индексирање се извршавају одвојено од обраде корисничких упита. Након што је документација припремљена, могу се генерисати приједлози тест случајева и Appium скелети без поновног индексирања и парсирања, све док се улазни материјал не промијени. Оваква организација омогућава да се поједине компоненте касније мијењају независно, без промјене основног тока података кроз систем.

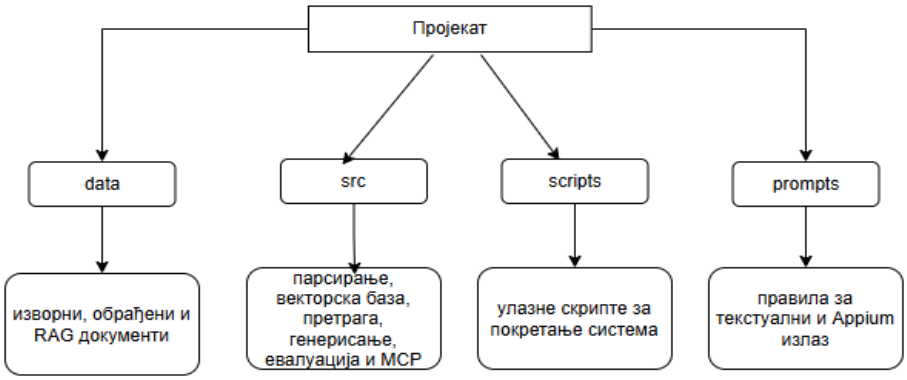
4. Програмско рјешење

Концепти описани у претходном поглављу реализовани су у програмском језику Пајтон. Имплементација је подијељена на двије фазе. У првој се изворна документација и артефакти постојећег Appium оквира парсирају, претварају у структуриране записе и индексирају у локалној Qdrant бази. Друга фаза започиње корисничким упитом. Након тога, систем проналази мањи скуп релевантних записа, од њих формира контекст и прослеђује га великом језичком моделу. У наставку овога поглавља описана је организација изворног кода, начин обраде сваке врсте улаза и токови за добијање приједлога тест случаја и Appium тест скелета.

4.1 Организација пројекта и конфигурација

4.1.1 Организација изворног кода

Пројекат је организован према улози појединих компоненти. Улазне скрипте не садрже главну логику обраде, већ само учитавају конфигурацију и позивају одговарајући модул из пакета src. На тај начин се исти поступци могу користити из интерфејса командне линије и преко MCP сервера, избјегавајући дуплирање кода. Директоријум data је организован у три поддиректоријума, у складу са њиховом улогом у систему: data/raw садржи изворне RST и Пајтон датотеке, data/processed појединачне парсиране записе, а data/rag документе припремљене за индексирање. Локална Qdrant база и генерисани излази чувају се у директоријуму outputs. Путање до података, називи модела, колекција и остали параметри издвојени су у заједнички конфигурациони модул. Правила и шаблони који се користе при формирању упита за велики језички модел чувају се одвојено, у директоријуму prompts, што олакшава њихову накнадну измјену и поновну употребу. Основна организација приказана је на слици 4.1, док су улоге главних програмских цјелина наведене у табели 4.1.



Слика 4.1 - Организација реализованог прототипа

Цјелина	Улога
src/config	Учитавање путања, назива колекција и параметара претраге.
src/parsing	Парсирање RST документатације, ручних тестова, Appium примјера, селектора и помоћних функција.
src/vectorstore	Формирање векторских репрезентација и упис докумената у Qdrant.
src/retrieval	Семантичка претрага и директно проналажење помоћних функција по називу.
src/generation	Формирање упита, припрема контекста и позив великог језичког модела.
src/evaluation	Чување структурираног излаза и израчунавање метрика пресликавања.
src/mcp	Излагање припремљеног RAG контекста спољашњем MCP клијенту.
Scripts	Улазне скрипте за парсирање, индексирање, генерисање, евалуацију и покретање MCP сервера.

Табела 4.1 - Улоге главних програмских цјелина

4.1.2 Конфигурација и приступ великом језичком моделу

Конфигурација је обједињена у класи Settings, дефинисаној у датотеци src/config/settings.py. Она садржи путање до улазних и излазних датотека, називе четири Qdrant колекције, назив embedding модела, праг сличности и параметре за приступ великом језичком моделу. Вриједности се учитавају из датотеке .env, док се за непостављене параметре користе подразумијеване вриједности из кода.

У конфигурацији постоје поља GITHUB_TOKEN и OPENAI_API_KEY, која се користе за избор начина приступа великом језичком моделу. Приступ великом језичком моделу одрађен је тако да функција create_llm_client() бира између два начина повезивања. Ако је постављен GitHub токен, користи се GitHub Models сервис и модел наведен у параметру GITHUB_MODEL. У супротном, велики језички модел се позива преко OpenAI приступа, са моделом наведеним у OPENAI_MODEL. За комуникацију у оба случаја користи се иста OpenAI клијентска библиотека за Пајтон, јер GitHub Models API прати исти интерфејс.

Трећи начин приступа великом језичком моделу не подразумијева његово непосредно позивање из Пајтон кода. У том режиму припремљени контекст се преко MCP сервера ставља на располагање GitHub Copilot агенту, што је детаљније описано у одјељку 4.6.

4.2 Обрада и структурирање улазних података

Скрипта scripts/parse.py покреће парсирање улазних података. Обрада је подијељена на четири одвојена тока, а појединачан ток се покреће навођењем тока уз аргумент --source, на примјер:

python scripts/parse.py --source rst

Аргумент --source може имати једну од сљедеће четири вриједности:

- rst – REQ документација и ручно написани тест случајеви;
- helper – Appium помоћне функције;
- test-example – постојећи Appium тест примјери;
- selectors – селектори корисничког интерфејса, односно подаци који омогућавају да се одређени елемент на екрану пронађе и користи током теста. Селектори су организовани по групама, при чему прва ознака одређује групу елемената, а друга конкретан елемент у тој групи. На примјер, STB_SELECTORS[“HOME_SCREEN”][“DISCOVER”] односи се на елемент „Discover“ на почетном екрану.

Уколико се скрипта покрене без аргумента --source, сва четири тока се извршавају редом, један за другим. Резултат сваког тока је JSON (енгл. JavaScript Object Notation) или JSONL (енгл. JSON Lines) датотека са структурираним записима добијеним из одговарајућег извора. Начин на који сваки од ова четири тока обрађује своју врсту улазних података детаљније је описан у поглављима 4.2.1 и 4.2.2.

4.2.1 Парсирање захтјева, тестова и сљедљивости

Поступак за RST документацију реализован је у модулима rst_utils.py, rst_parser.py, traceability.py и rag_documents.py. Скрипта пролази кроз све датотеке са захтјевима у директоријуму data/raw/req и из њих издваја блокове означене директивом req. За сваки блок се засебно читају наслов и поља :id:, :reviewstatus:, :reqtype: и :refines:, док се преостали текст чува као опис очекиваног понашања. Резултат је низ независних JSONL елемената у датотеци requirements.jsonl, при чему један елемент одговара једном захтјеву и садржи његов текст, метаподатке, изворну датотеку и критеријуме прихватљивости.

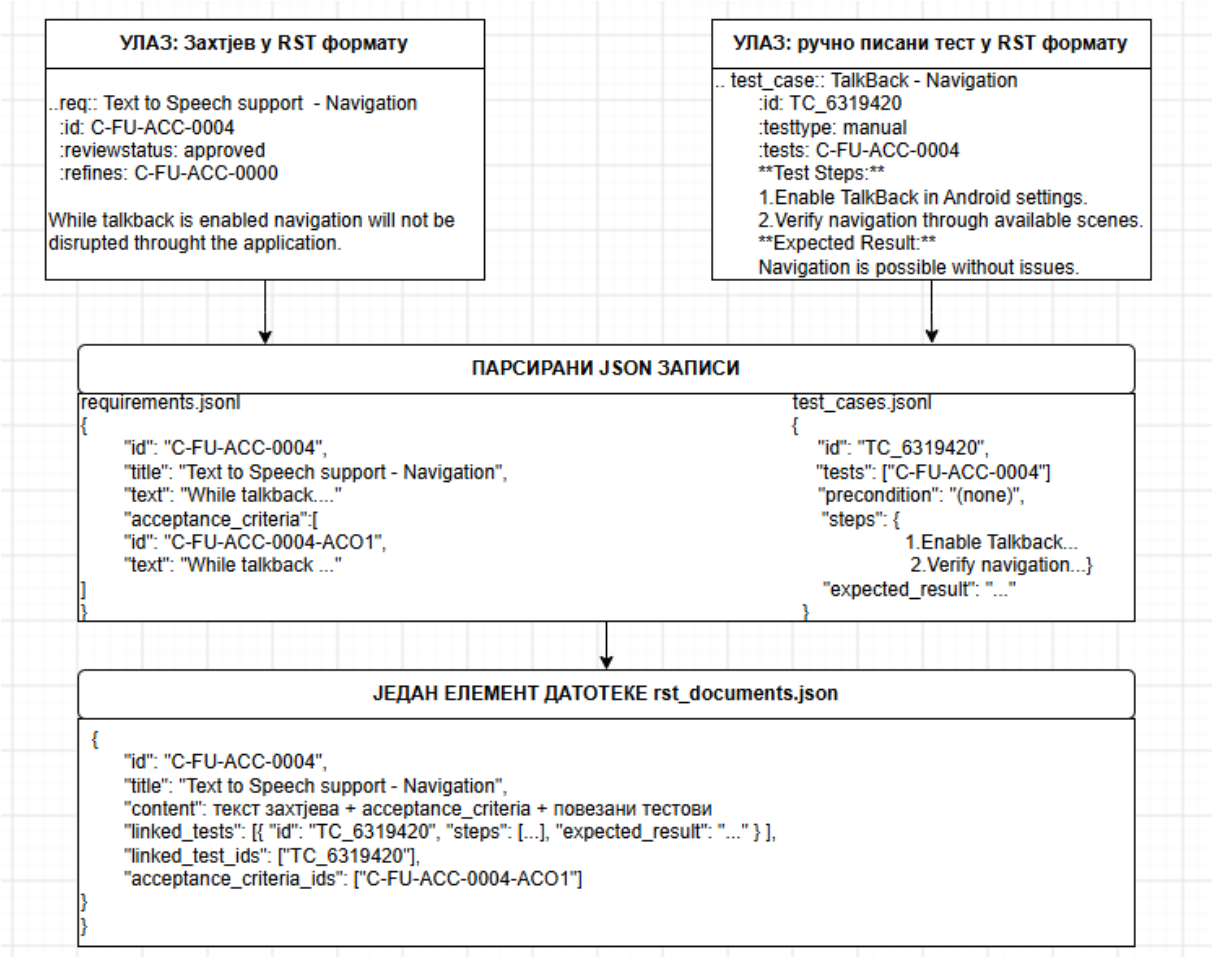
Ручно писани тестови обрађују се на исти начин, али се из сваког test_case блока издвајају опис, циљ, предуслов, нумерисани кораци и очекивани резултат. Поље :tests: садржи идентификатор захтјева који тест провјерава. У коришћеној документацији сваки тест повезан је са једним захтјевом, док исти захтјев може имати више различитих тестова. Сваки тест се као посебан елемент уписује у test_cases.jsonl.

Функција build_traceability() пролази кроз парсиране тестове и за сваки од њих формира запис који садржи идентификатор и наслов теста, идентификатор захтјева, тип теста и путању изворне датотеке. Тако се веза гради у смјеру тест – захтјев, на основу податка који стварно

постоји у RST блоку. Ако је са истим захтјевом повезано више тестова, сваки од њих формира посебну везу у запису сљедљивости.

Критеријуми прихватљивости нису посебно означени у изворним датотекама. Функција extract_acceptance_criteria() зато дијели текст захтјева на реченице и задржава оне које садрже изразе shall, should, must или will. Сваком издвојеном услову додаје се идентификатор који га повезује са изворним захтјевом. На примјер, у ознаци C-FU-ACC-0004-AC01, дио C-FU-ACC-0004 представља идентификатор изворног захтјева, AC означава критеријум прихватљивости, а 01 његов редни број у оквиру тог захтјева. Овим поступком не мијења се изворни захтјев, већ се формира додатна структура која се користи за пресликавање и евалуацију.

Након парсирања, функција build_rag_documents() групише тестове по идентификатору захтјева. За сваки захтјев формира се један елемент датотеке rst_documents.json. Његово поље content садржи текст захтјева, изведене критеријуме и све ручне тестове који су повезани са тим захтјевом. Исти тестови остају доступни и као структурирана листа linked_tests, док metadata садржи идентификаторе критеријума и повезаних тестова. Поље content служи за формирање векторске репрезентације, док се структурирана поља користе за прецизно састављање контекста и приказ идентификатора. Примјер цијелог тока од RST блокова до елемента за индексирање приказан је на слици 4.2.



Слика 4.2 - Примјер претварања RST блокова у JSON записе и RAG документ

4.2.2 Обрада артефаката Appium оквира

За генерисање Appium скелета обрађују се три додатна извора: датотеке са помоћним и навигационим функцијама, постојећи аутоматизовани тестови и рјечник STB_SELECTORS. На примјер, navigate_to_search() и navigate_to_settings() представљају навигационе функције, док су check_one_key(), check_video_by_successive_frames() и get_last_channel_from_log() примјери помоћних функција које се користе за провјеру стања апликације и добијање података потребних током теста. И овдје једна помоћна или навигациона функција, један Appium тест или један селектор постају засебан JSON запис који се касније може независно пронаћи семантичком претрагом.

Модул helper_parser.py користи апстрактно синтаксно стабло (енгл. Abstract Syntax Tree, AST) Пајтон кода и задужен је за обраду помоћних и навигационих функција. За сваку функцију издвајају се назив, потпис са подразумијеваним аргументима, путања увоза, docstring, листа позваних функција и изворни код. Навигационе функције чине посебну групу и служе за кретање кроз кориснички интерфејс апликације и долазак до одређеног екрана или ставке. Међу помоћним функцијама, према њиховој намјени, разликују се функције за провјеру корисничког интерфејса, ADB (енгл. Android Debug Bridge) и logcat функције, као и остале помоћне функције. Функције за провјеру утврђују да ли је очекивани елемент приказан или није приказан на екрану. На примјер, navigate_to_search() служи за долазак до претраге, док check_one_key() провјерава да ли је тражени елемент корисничког интерфејса видљив. ADB функције користе се за непосредно извршавање команди на Андроид уређају, као што то ради run_adb_command(). Logcat функције читају и обрађују системски запис уређаја, па тако get_last_channel_from_log() из њега издваја назив покренутог канала. Остале помоћне функције обављају једноставније заједничке послове који се користе у више тестова, као што је упис поруке у датотеку функцијом log_to_file().

Парсер додатно анализира два обрасца који су се показали важним при генерисању. Ако функција враћа један од улазних аргумената након измјене, у запис се додаје ознака да повратну вриједност треба поново додијелити промјенљивој. Ако се аргументи користе за приступ рјечнику STB_SELECTORS, биљежи се да функција очекује текстуални назив групе и кључа, а не саму вриједност која је под тим називом сачувана у рјечнику. На примјер, функција check_one_key() може примити “HOME_SCREEN” и “DISCOVER”, а затим на основу њих из рјечника STB_SELECTORS пронаћи одговарајући селектор.

Постојећи Appium тестови обрађују се модулом test_example_parser.py. Главна тестна функција не одређује се само према редослиједу дефиниција, већ се тражи позив из функције омотача којом се тест предаје функцији run_test(). Из главне тестне функције издвајају се помоћне и навигационе функције, селектори, ADB уноси и редослијед притисака тастера. На основу тога се формира сажети ток теста који описује како се апликација покреће, којом навигацијом се долази до екрана и које се провјере извршавају.

Поред сажетог тока, из главне тестне функције се издваја и исјечак кода који садржи специфичну логику теста. Уобичајени дио за покретање апликације и чекање почетног екрана се

изостављa, како би примјер више простора посветио оним корацима који разликују конкретан тест. Ако исјечак користи константу дефинисану на нивоу модула, додаје се и њена дефиниција.

Селектори се парсирају из рјечника STB_SELECTORS. AST парсер проналази групе и њихове кључеве, а за сваки кључ чува стратегију лоцирања, вриједност локатора и исправан облик употребе, на примјер STB_SELECTORS[“HOME_SCREEN”][“DISCOVER”].

Резултати три описана поступка уписују се у code_documents.json, test_example_documents.json и selector_documents.json.

4.3 Формирање векторских репрезентација и Qdrant база

4.3.1 Embedding модел и индексирање

Процес индексирања покреће се скриптом scripts/index.py. Скрипта може да индексира једну изабрану колекцију или све колекције редом. За сваку колекцију се из одговарајуће JSON датотеке учитава листа структурираних елемената, а затим се сваки елемент појединачно претвара у текст за формирање векторске репрезентације. Цијела JSON датотека и цијела RST документација никада се не шаљу моделу као један улаз. Код захтјева, сваки захтјев обрађује се засебно, заједно са његовим критеријумима прихватљивости и повезаним тестовима. Код осталих извора засебно се обрађују једна функција, један сажет примјер Appium теста или један селектор.

У оквиру овог рада реализован је модул src/vectorstore/embeddings.py за формирање векторских репрезентација. Модул користи постојећи модел BAAI/bge-m3 и библиотеку sentence-transformers. Модел и његов токенизер учитавају се тек при првој употреби и затим остају у меморији за наредне елементе. Функција create_embedding() формира и нормализује векторску репрезентацију сваког елемента. За чување и претрагу добијених вектора одабрана је векторска база података Qdrant. У њеним колекцијама користи се косинусна мјера, којом се одређује семантичка сличност између корисничког упита и сачуваних записа. Поред самог вектора, у бази се чувају и структурирани подаци потребни за касније формирање контекста.

Садржај који се претварао у вектор прилагођен је врсти елемента. За захтјев се спајају идентификатор, наслов и поље content. Код функције се користе назив, потпис, путања увоза, категорија, docstring и називи позваних функција. Appium примјер се описује доменом, коришћеним артефактима и сажетим током, док текст селектора садржи групу, кључ, локатор и примјер употребе. Овај избор је важан јер вектор треба да представља значење елемента, док се цјелокупни метаподаци чувају одвојено у садржају Qdrant записа.

Прије формирања вектора токенизер израчунава дужину појединачног текста. У конфигурацији је MAX_TOKENS постављен на 8192, у складу са изабраним embedding моделом. Ако елемент пређе ту вриједност, програм исписује упозорење са његовим идентификатором и бројем токена. Провјера се обавља засебно за сваки JSON елемент.

При обнови колекције најприје се на основу првог елемента одређује димензија вектора. Ако колекција већ постоји, она се брише и поново формира са косинусном мјером. За сваки елемент затим се припрема Qdrant запис типа PointStruct, који садржи нумерички идентификатор, вектор и садржај записа са изворним структурираним подацима. Подјела на четири колекције приказана је у табели 4.2.

Колекција	Садржај
rag_requirements	Један запис по захтјеву, са изведеним критеријумима и листом свих повезаних ручних тестова.
rag_code	Један запис по помоћној или навигационој функцији, са потписом и метаподацима за правилну употребу.
rag_test_examples	Један сажети запис по постојећем Appium тесту, са током, коришћеним функцијама и исјечком кода.
rag_selectors	Један запис по селектору корисничког интерфејса, са групом, кључем и локатором.

Табела 4.2 - Qdrant колекције и њихов садржај

4.3.2 Семантичка претрага

Семантичка претрага је реализована у модулу src/retrieval/search.py. Кориснички упит се прво претвара у вектор истим embedding моделом који је коришћен при индексирању. Qdrant затим проналази записе из изабране колекције чије су векторске репрезентације најсличније упиту. Резултати чија је оцјена нижа од конфигурисаног прага сличности се одбацују, а евентуално поновљени записи уклањају се из коначног скупа резултата.

Сврха претраге није да се великом језичком моделу пошаље што већа количина података, већ да се издвоји неколико примјера који најбоље одговарају корисничком упиту. Поред смањења дужине упита, тиме се ограничава утицај тематски удаљених захтјева и тестова. Број резултата и праг сличности могу се прилагођавати различитим врстама упита без промјене кода претраге.

При претрази функција за генерисање Appium скелета семантичка претрага допуњена је директним проналажењем по називу. Из најсличнијих тест примјера најприје се издвајају коришћене помоћне и навигационе функције. Њихови записи се непосредно проналазе у code_documents.json, а затим се спајају са функцијама добијеним семантичком претрагом. Директно пронађене функције имају предност, док се дупликати уклањају према називу. Ова комбинација је уведена зато што назив функције из стварног теста поузданије показује да је она потребна него сам њен положај међу резултатима семантичке претраге.

4.4 Формирање приједлога тест случаја и провјера пресликавања

4.4.1 Генерисање приједлога тест случаја

Директно генерисање приједлога тест случаја покреће се скриптом scripts/llm_test_case.py. Након уноса корисничког питања, систем претражује колекцију захтјева и издваја мањи број записа који су по значењу најближи упиту. Функција build_test_case_prompt() на основу корисничког питања и пронађених записа формира цио упит који се прослеђује великом језичком моделу. У њему се најприје наводи задатак који модел треба да изврши, а затим контекст преузет из документације. За сваки пронађени захтјев у контекст се укључују његов идентификатор, наслов, текст и изведени критеријуми прихватљивости. Ако захтјев има повезане ручно написане тестове, додају се и њихови предуслови, кораци и очекивани резултати. Ови подаци показују на који начин је посматрана функционалност већ провјеравана и моделу служе као ослонац при формирању новог приједлога.

Иста скрипта подржава више врста питања. Ако корисник тражи постојеће тестове за одређени захтјев, модел добија инструкцију да прикаже податке пронађене у документацији, умјесто да ствара нови тест. Код питања о покривености модел пореди текст захтјева и изведене критеријуме прихватљивости са повезаним тестовима и указује на дијелове који нису обухваћени. Тек када упит тражи нови тест сценарио, модел генерише приједлог новог тест случаја.

Након припреме цјелокупног упита, функција ask_llm() прослеђује га изабраном великом језичком моделу и враћа текст добијеног одговора. Модел добија исто ограничење описано у поглављу о концепту рјешења, да користи искључиво захтјеве и идентификаторе присутне у достављеном контексту.

Када се формира нови тест случај, одговор садржи наслов, предуслове, тестне кораке и очекиване резултате. На крају се наводи захтјев на којем је тест заснован, као и веза између појединих критеријума прихватљивости и корака којима су они обухваћени. На тај начин корисник може да провјери на којем дијелу документације је генерисани приједлог заснован.

Добијени одговор приказује се у терминалу, након чега скрипта наставља да прихвата нова питања све док корисник не унесе команду exit или quit. У овом начину рада одговор је намијењен читању и ручном прегледу, па се не претвара у унапријед дефинисану JSON структуру. Структурирани излаз користи се само у посебном режиму намијењеном евалуацији, описаном у наредном одјељку.

4.4.2 Структурирани режим за евалуацију

За провјеру пресликавања користи се посебна варијанта истог упита. Она задржава пронађени контекст, али захтијева да цијели одговор од модела буде један JSON објекат. Одговор садржи статус, податке о недостајућим информацијама, врсту сценарија, листе идентификатора

захтјева и критеријума прихватљивости, предуслове, кораке и очекиване резултате. Ако доступни контекст није довољан за формирање сценарија, модел и даље враћа исту структуру, али са статусом insufficient_context.

Скрипта scripts/evaluate.py учитава унапријед припремљене евалуационе случајеве из датотеке traceability_cases.json. Сваки случај садржи кориснички упит, као и листе идентификатора захтјева и критеријума прихватљивости за које се очекује да буду пронађени. За сваки од ових упита покреће се исти поступак претраге и генерисања, након чега се добијени одговор чува заједно са идентификатором случаја и улазним упитом.

Функција extract_json_object() издваја JSON садржај из одговора модела, док validate_scenario_fields() провјерава да ли су присутна сва обавезна поља, да ли листе садрже текстуалне вриједности и да ли статус и врста сценарија припадају дозвољеном скупу вриједности. У излазној JSONL датотеци чувају се и евентуалне грешке настале приликом парсирања или валидације, како би се неисправни одговори ипак прегледали накнадно.

За сваки евалуациони случај познате су очекиване листе идентификатора. На примјер, за упит „Give me a test case for verifying parental restriction in the live banner“ очекује се да модел у свом одговору наведе идентификатор захтјева C-FU-BCASTLIVE-0013 и идентификаторе критеријума прихватљивости C-FU-BCASTLIVE-0013-AC01, C-FU-BCASTLIVE-0013-AC02 и C-FU-BCASTLIVE-0013-AC03, на којима је генерисани тест заснован. Модул traceability_metrics.py затим пореди идентификаторе које је модел навео у одговору са очекиваним идентификаторима за тај евалуациони случај. Прецизност показује колики дио добијених идентификатора припада очекиваном скупу, одзив показује колики дио очекиваних идентификатора је модел навео, док F1 мјера обједињује ове двије вриједности. Микро резултат израчунава се над свим идентификаторима из свих случајева заједно, док макро резултат представља просјек метрика добијених за сваки појединачан случај. Захтјеви и изведени критеријуми прихватљивости оцјењују се одвојено.

Овим поступком провјерава се исправност пресликавања генерисаног резултата на изворну документацију. Квалитет формулације тестних корака и практична употребљивост добијеног сценарија не оцјењују се овим метрикама, већ се посебно анализирају у поглављу са резултатима.

4.5 Формирање Appium тест скелета

4.5.1 Припрема проширеног контекста

Appium ток се покреће скриптом scripts/llm_appium_code.py. Све креће од корисничког описа траженог теста на основу кога се претражују захтјеви, постојећи Appium тест примјери и селектори. Пронађени тест примјери се затим користе за прецизније проналажење помоћних функција, јер се из њих већ зна који су позиви функција били коришћени у сличном сценарију.

За сваки релевантан Appium примјер у контекст се уносе наслов и домен теста, називи главне функције и функције омотача, коришћене помоћне и навигационе функције, селектори и

сажети ток извршавања. Ако је у тесту препознат редослијед притисака даљинског управљача, он се преноси као посебна инструкција. Само за најрелевантнији примјер додаје се краћи исјечак стварног кода. На тај начин модел добија конкретан примјер структуре и редослиједа корака, а упит се не оптерећује цјелокупним садржајем свих пронађених тестних датотека.

Упит за претрагу кода проширује се називима функција, селектора и тастера издвојених из тест примјера. Потребни записи функција преузимају се поступком описаним у одјељку 4.3.2. За сваку изабрану функцију модел добија њен тачан потпис, путању увоза, категорију и docstring. Када је парсер утврдио да повратну вриједност функције треба сачувати или да њен одређени аргумент мора бити текстуални кључ селектора, та напомена се додаје одмах уз потпис функције. Контекст селектора садржи назив групе, кључ, стратегију лоцирања и исправан начин приступа селектору у рјечнику.

4.5.2 Формирање упита и директно генерисање кода

Функција build_appium_code_prompt() обједињује четири групе података: захтјеве, постојеће примјере Appium тестова, доступне функције и селекторе. На почетку упита моделу се задаје инструкција да користи само достављене пројектне артефакте и да јасно означи недостајући податак умјесто да измишља функцију или селектор. Посебно се наглашава да тестни корак мора довести систем у стање које треба провјерити. На примјер, модел мора употријебити доступну навигациону функцију или одговарајуће команде даљинског управљача којима ће се доћи до траженог екрана, умјесто претпоставке да је већ приказан тај екран.

На крају упита додаје се садржај датотеке prompts/appium_test_style.txt. Она садржи правила која одређују облик генерисаног теста. Дефинисани су обавезни импорт изрази и константе, структура главне тестне функције, начин покретања апликације и употреба блока try/finally. Правила одређују и услове под којима се резултат теста може поставити на “passed“, као и обавезан упис QA резултата функцијом write_qa_qc_results().

Иста датотека дефинише структуру функције омотача и улазне функције run_main_test(), чиме генерисани скелет прати организацију постојећих тестних датотека. Помоћне функције третирају се као пројектни API. Ако помоћна функција већ извршава тражену радњу, модел треба да је позове, умјесто да понавља њене унутрашње притиске тастера, WebDriverWait позиве или ADB команде.

Функција run_appium_code_query() прије позива модела исписује број токена коначног упита, а затим користи исти LLM клијент као у току за генерисање текстуалног тест случаја. Добијени Пајтон код приказује се у терминалу. Резултат представља почетни Appium тест скелет који треба да се прегледа и, уколико је потребно, ручно доради.

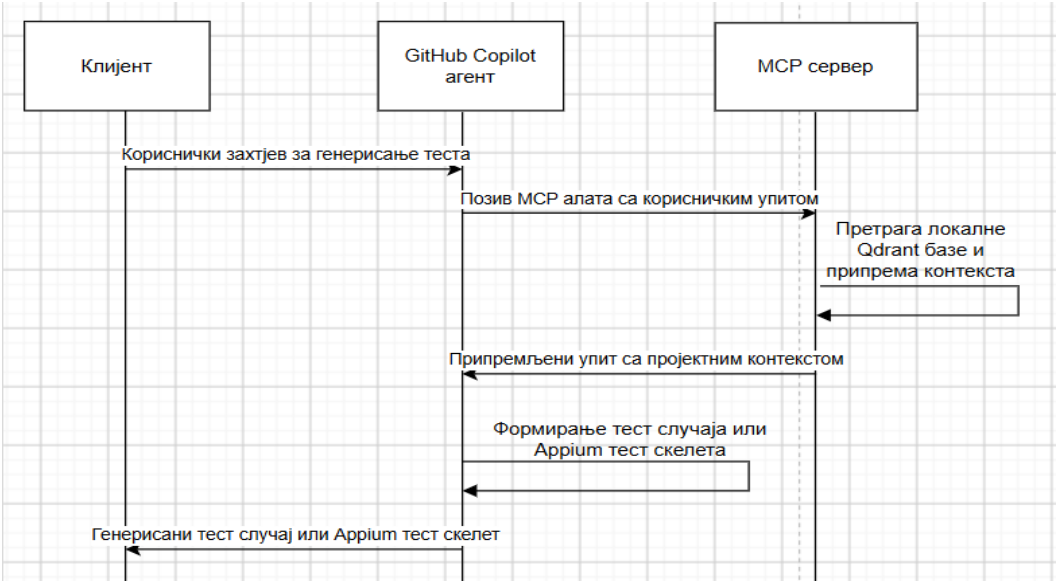
4.6 MCP сервер и режим GitHub Copilot агента

У оквиру овог рада реализован је локални MCP сервер у модулу src/mcp/server.py, помоћу библиотеке FastMCP, а покреће се скриптом scripts/run_mcp_server.py. Да би Visual Studio Code знао како да покрене овај сервер, у датотеци .vscode/mcp.json наведени су Пајтон команда која се извршава, директоријум из којег се сервер покреће и .env датотека из које се учитавају потребна подешавања. Након овакве конфигурације, GitHub Copilot агент може да препозна MCP сервер као локални извор пројектних података и да позове његове алате.

Сервер излаже један алат за припрему контекста текстуалног тест случаја и један алат за припрему контекста потребног при генерисању Appium теста. Када корисник Copilot агенту опише тест који жели да добије, агент може да позове одговарајући MCP алат. Сервер тада претражује локалну Qdrant базу, издваја релевантне захтјеве, тест примјере, помоћне функције и селекторе и од њих формира цио упит са пројектним контекстом и правилима која резултат треба да поштује.

Оба MCP алата користе исту локалну Qdrant базу и исте функције за формирање упита као и директне Пајтон скрипте. Разлика је у томе што MCP сервер сам не позива велики језички модел и не формира коначан одговор. Он GitHub Copilot агенту враћа припремљен упит који садржи кориснички захтјев, релевантне податке пронађене у пројекту и правила за генерисање резултата. На основу тог садржаја Copilot агент затим може да предложи тест случај или да генерише цијелу Пајтон датотеку са Appium тест скелетом.

MCP сервер самостално не чита нити мијења произвољне датотеке у радном простору. Ако корисник затражи прављење новог тестног фајла, ту радњу извршава GitHub Copilot агент у оквиру дозвола које има у Visual Studio Code окружењу. Улога MCP сервера остаје ограничена на проналажење релевантних пројектних података и припрему контекста који је потребан за генерисање. Комуникација између клијента, GitHub Copilot агента и MCP сервера приказана је дијаграмом секвенце порука на слици 4.3.



Слика 4.3 – Дијаграм секвенце порука између клијента, GitHub Copilot агента и MCP сервера

4.7 Покретање и повезивање прототипа

Свака улазна скрипта позива једну или више функција из пакета src. Скрипте саме не садрже логику парсирања, претраге или генерисања, већ дате цјелине повезују у одређеном редослиједу. Уобичајени редослијед рада је да се након измјене улазног материјала прво покрене parse.py, затим index.py, а послије тога један од токова за генерисање или евалуацију. Парсирање и индексирање не понављају се за сваки кориснички упит, јер Qdrant колекције остају сачуване у директоријуму outputs/qdrant_local. Најважније скрипте и њихове улоге приказане су у табели 4.3.

Скрипта	Намјена
scripts/parse.py	Парсира изабрани извор или сва четири извора и уписује JSON/JSONL датотеке.
scripts/index.py	Обнавља изабрану Qdrant колекцију или све четири колекције.
scripts/llm_test_case.py	Омогућава кориснику да великом језичком моделу поставља питања у вези са QA документацијом и покреће ток за генерисање приједлога текстуалног тест случаја.
scripts/llm_appium_code.py	Омогућава кориснику да опише жељени тест и покреће комуникацију са великим језичким моделом ради генерисања Appium тест скелета.
scripts/evaluate.py	Покреће припремљене случајеве за провјеру пресликавања и чува структурирани излаз.
scripts/run_mcp_server.py	Покреће локални MCP сервер за припрему контекста у спољашњем агент клијенту.

Табела 4.3 - Најважније улазне скрипте

Овакво повезивање задржава јасну границу између припреме базе знања и њене употребе. Након парсирања и индексирања, исти припремљени подаци могу се користити у свим токовима прототипа, без поновне обраде улазне документације за сваки кориснички упит.

5. Резултати

У овом поглављу приказани су резултати испитивања реализованог прототипа. Испитивање је спроведено задавањем различитих упита систему и анализом добијених одговора. Посебно су посматрани рад са постојећом QA документацијом, пресликавање резултата на захтјеве и критеријуме прихватљивости, генерисани тест случајеви и добијени Appium тест скелети.

За испитивање су коришћене двије врсте упита. Прву врсту чине припремљени евалуациони случајеви, код којих су унапријед познати идентификатори захтјева и критеријума прихватљивости који се очекују у одговору. Другу врсту чине кориснички упити за практичну провјеру рада система. Код њих се не пореде унапријед задати идентификатори, већ се прегледа сам одговор који је систем дао. Међу овим упитима били су захтјеви за проналажење постојећих тестова, провјеру њихове повезаности са захтјевима, провјеру покривености критеријума прихватљивости, генерисање новог тест случаја и генерисање Appium тест скелета. Провјеравани су и упити за које у документацији не постоји одговарајући садржај.

5.1 Поставка испитивања

Први корак при испитивању био је припрема улазних података. Најприје је покренута скрипта scripts/parse.py, којом су обрађени захтјеви, ручно написани тестови, постојећи Appium тест примјери, помоћне функције и селектори. Након тога је покренута скрипта scripts/index.py, којом су припремљени записи индексирани у одговарајуће Qdrant колекције. Тек након парсирања и индексирања покретани су упити за генерисање и евалуацију.

За формирање векторских репрезентација при индексирању, као и за претварање корисничког упита у вектор при претрази, коришћен је модел BAAI/bge-m3. Праг сличности при претрази био је постављен на 0,4.

Током развоја и провјере прототипа коришћена су сва три реализована начина приступа великом језичком моделу. Директне Пајтон скрипте коришћене су преко GitHub Models и OpenAI

приступа, док је трећи начин рада испробан помоћу GitHub Copilot агента и реализованог MCP сервера. Циљ рада није био поређење великих језичких модела, па се због тога у раду не оцјењује који од коришћених модела даје боље резултате.

Улазни подаци обухватили су 53 RST датотеке са захтјевима, 135 RST датотека са ручно написаним тест случајевима, 206 Пајтон датотека са постојећим Appium тест примјерима, 10 Пајтон датотека са помоћним и навигационим функцијама и једну датотеку са селекторима корисничког интерфејса. Обим података добијен парсирањем ових датотека приказан је у табели 5.1.

Врста података	Број записа
Захтјеви	1236
Ручно написани тест случајеви	1674
Везе између захтјева и тестова	1754
Помоћне и навигационе функције	143
Постојећи Appium тест примјери	206
Селектори корисничког интерфејса	668

Табела 5.1 - Обим података коришћених при испитивању

При индексирању су у одговарајуће Qdrant колекције уписана 1236 записа са захтјевима, 143 записа са помоћним и навигационим функцијама, 206 записа са Appium тест примјерима и 668 записа са селекторима. Ручно написани тестови и њихове везе са захтјевима користе се као дио података придружених колекцији са захтјевима.

За бројчану провјеру пресликавања припремљен је скуп од 100 евалуационих случајева. Сваки случај садржи кориснички упит и унапријед познате идентификаторе захтјева и критеријума прихватљивости који се очекују у одговору од LLM-а. На основу њих се рачунају метрике пресликавања описане у претходном поглављу. Примјер једног од постојећих евалуационих случајева дат је на слици 5.1.

```
{
  "case_id": "TRC-022",
  "query": "How would I test adding a broadcast channel to a favourite list from Broadcast EPG?",
  "expected_requirement_ids": [
    "C-FU-BFAV-0010"
  ],
  "expected_acceptance_criteria_ids": [
    "C-FU-BFAV-0010-AC01",
    "C-FU-BFAV-0010-AC02"
  ]
},
```

Слика 5.1 - Примјер унапријед припремљеног евалуационог случаја

Овим случајевима провјерава се да ли је систем навео одговарајуће захтјеве и критеријуме прихватљивости. Они не показују колико је добро написан генерисани тест случај.

Због тога су, поред њих, коришћени и кориснички упити за практичну провјеру рада система. Код тих упита посебно су прегледани добијени одговори, коришћење података из пројектне документације и практична употребљивост резултата.

5.2 Примјери рада са тестном документацијом

У овом одјељку приказано је неколико конкретних примјера рада са постојећом тестном документацијом. Примјери обухватају провјеру покривености критеријума прихватљивости, генерисање новог тест случаја и утицај постојећих тестова на добијени резултат.

5.2.1 Провјера покривености критеријума прихватљивости

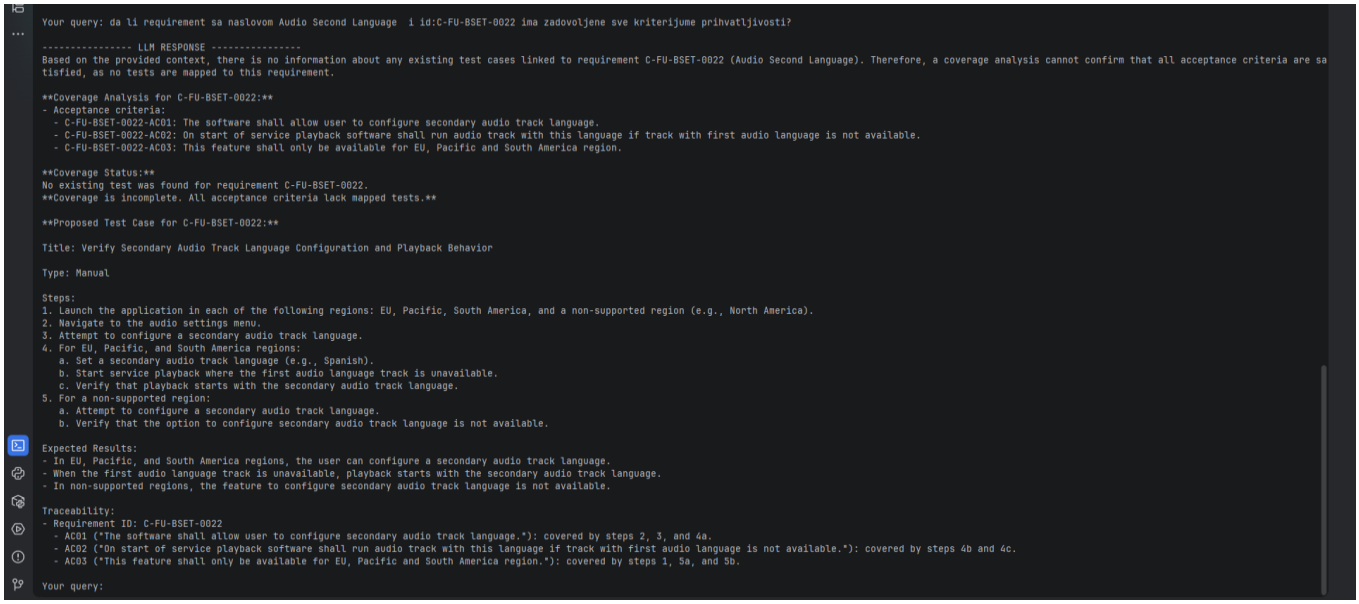
Као први примјер коришћен је захтјев C-FU-TIMESHIFT-1002 – Settings control. Од система је тражено да пронађе постојеће тестове који су повезани са овим захтјевом и провјери да ли они покривају све критеријуме прихватљивости датог захтјева. Систем је успјешно пронашао постојећи тест TC_6312039 - TIMESHIFT – Settings control. Тест садржи четири корака. У првом и трећем кораку Timeshift се укључује и искључује, другим кораком се провјеравају доступне Timeshift контроле након укључивања, док се четвртим провјерава да су оне искључене и сакривене након искључивања Timeshift функције. Поређењем ових корака са критеријумима прихватљивости датог захтјева систем је правилно повезао први критеријум прихватљивости са корацима 1 и 3, други критеријум прихватљивости са кораком 2 и трећи критеријум прихватљивости са кораком 4. На основу тога је закључено да постојећи тест у потпуности покрива сва три критеријума прихватљивости датог захтјева. Добијени одговор приказан је на слици 5.2.

```
Your query: Провјери да ли за захтјев C-FU-TIMESHIFT-1002 - Settings control постоје повезани тестни случајеви и да ли задовољавају све његове критеријуме прихватљивости
---------- LLM RESPONSE ----------
Да — за захтјев **C-FU-TIMESHIFT-1002 - Settings control** постоји повезан тестни случај.

**Постојећи повезани тест:**

- **Test ID:** TC_6312039
  **Test Title:** Timeshift - Settings control
  **Test Type:** manual
  **Source File:** `VTP/Common/Timeshift/tests.rst`
  **Short Description:** User shall be able to set default Timeshift on/off state from Broadcast Settings screen (C-FU-BSET-0043). When default Timeshift is enabled, all timeshift
related UI controls and functionality shall be enabled and application shall present them to user. When default Timeshift is disabled, all timeshift related UI controls and functio
nality shall be disabled and hidden from user.

**Процјена покривености критеријума прихватљивости:**

- **C-FU-TIMESHIFT-1002-AC01:** User shall be able to set default Timeshift on/off state from Broadcast Settings screen (C-FU-BSET-0043).
  **Покривено:** да, корацима 1 и 3, који укључују омогућавање и онемогућавање default Timeshift поставке из Preferences/Broadcast Settings контекста.

- **C-FU-TIMESHIFT-1002-AC02:** When default Timeshift is enabled, all timeshift related UI controls and functionality shall be enabled and application shall present them to user.

  **Покривено:** да, кораком 2.

- **C-FU-TIMESHIFT-1002-AC03:** When default Timeshift is disabled, all timeshift related UI controls and functionality shall be disabled and hidden from user.
  **Покривено:** да, кораком 4.

**Закључак:**
На основу датог контекста, покривеност изгледа **комплетна** за све наведене acceptance criteria захтјева **C-FU-TIMESHIFT-1002**.
```

Слика 5.2 - Примјер провјере покривености критеријума прихватљивости

Провјерен је и супротан случај, када захтјев нема повезан тест. За захтјев C-FU-BSET-0022 - Audio Second Language систем је утврдио да у доступној документацији нема повезаног теста, па критеријуми прихватљивости нису покривени постојећим тестовима. Након тога је предложио нови тест случај који покрива сва три критеријума прихватљивости датог захтјева.

Овај примјер показује да се систем може користити и за уочавање захтјева за које је потребно додати тест.



Слика 5.3 – Примјер захтјева без повезаног теста и приједлог новог тест случаја

5.2.2 Генерисање новог тест случаја и утицај доступне документације

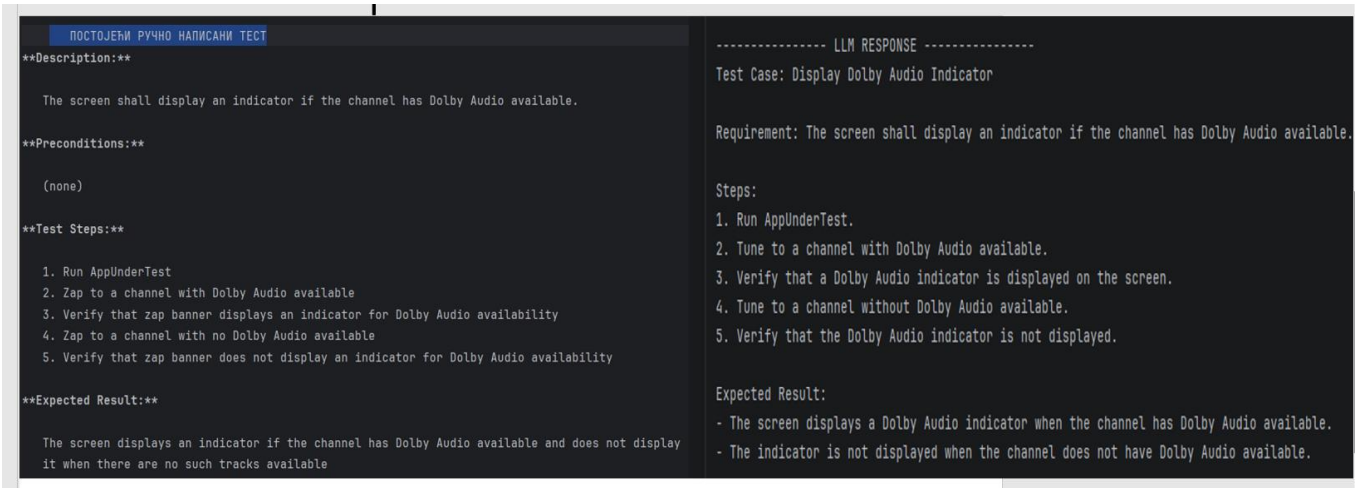
Главни дио испитивања односио се на генерисање нових тест случајева. Посматрано је колико генерисани тест одговара захтјеву и колико је сличан већ постојећим тестовима. За дио провјера коришћен је сљедећи поступак. Из документације је изабран захтјев за који већ постоји ручно написан тест. Тај тест је затим привремено уклоњен из података доступних систему. Након поновног парсирања и индексирања, од система је затражено да за исти захтјев генерише нови тест случај. На крају је генерисани тест упоређен са тестом који је претходно уклоњен.

Један од таквих примјера био је захтјев C-FU-BCASTLIVE-0016 – Display Dolby Audio indicator. У тестној документацији за овај захтјев већ постоји тест TC_6067178 – Zap Banner – Display Dolby Audio indicator. Да модел не би могао само да се ослони на тај тест, он је прије постављања упита привремено уклоњен из података доступних систему. Захтјев је остао доступан, а од система је затражено да сам предложи тест случај. Постојећи ручно написани тест прво покреће апликацију и прелази на канал који има Dolby Audio. Затим се провјерава да је Dolby Audio индикатор приказан. Након тога се прелази на канал без Dolby Audio подршке и провјерава да индикатор није приказан.

Исти основни ток добијен је и у генерисаном тесту. Модел је предложио покретање апликације, избор канала са Dolby Audio подршком и провјеру приказа индикатора, а затим избор канала без Dolby Audio подршке и провјеру да индикатор није присутан. На тај начин су обухваћене и позитивна и негативна провјера које постоје у ручно написаном тесту.

У овом примјеру генерисани тест је веома близак ручно написаном тесту. Један од разлога је што су у документацији остали други тестови за сличне функционалности. На примјер, тест за Hard of Hearing аудио индикатор такође прво провјерава канал на којем је одговарајући аудио

садржај доступан, а затим канал на којем није доступан. Систем је зато могао да пронађе примјер сличног начина тестирања иако је директно повезани тест био уклоњен. На слици 5.4. дато је поређење ручно написаног теста и генерисаног тест случаја за дати примјер.



Слика 5.4 – Поређење ручно написаног и генерисаног тест случаја

Исти поступак са привременим уклањањем постојећег теста поновљен је и на другим функционалностима, као што су родитељска контрола, ажурирање EPG података и промјена канала тастерима даљинског управљача. У примјерима у којима су у документацији постојали сродни тестови, генерисани тестови били су ближи начину на који су ручно написани тестови већ организовани. Сродни тестови моделу дају примјер редослиједа корака и начина на који се функционалност провјерава у постојећем окружењу.

Када таквих примјера нема, резултат може бити општији. На примјер, то се види на захтјеву C-FU-BCASTLIVE-0429 – Teletext support – recording. У тексту овог захтјева се наводи да телетекст није доступан док је снимање у току и да при покушају његовог покретања треба приказати одговарајућу поруку. За овај захтјев у постојећој документацији није постојао повезани тест. На кориснички упит за генерисање новог тест случаја за постојећи захтјев, систем је предложио да се покрене снимање, затим покуша отварање телетекста и провјери да ли је приказана порука да телетекст није доступан током снимања. Међутим, у тест је додао и провјеру шта се дешава након заустављања снимања. Иако тај корак дјелује логично, он није наведен у самом захтјеву.

Овај примјер показује да генерисани тест није увијек спреман за директну употребу. Када постоје слични тестови у документацији, резултат је обично ближи начину на који QA инжењери већ пишу тестове. Када таквих примјера нема, систем и даље може да предложи основни тест, али је потребно пажљивије провјерити да ли је модел додао кораке који нису наведени у захтјеву.

На крају је провјерено и понашање система када упит није повезан са пројектном документацијом. На питање „Који је главни град Србије?“ систем није дао одговор из општег знања модела, већ је навео да у доступном контексту нема довољно података. Овај примјер показује да модел одговор заснива искључиво на контексту који му је систем обезбиједио.

5.3 Резултати евалуације пресликавања

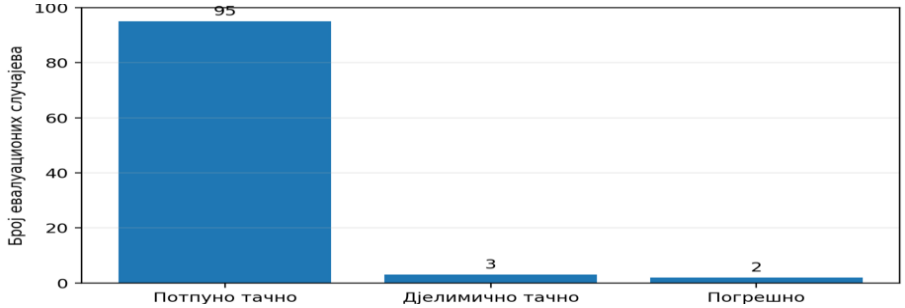
За бројчану провјеру пресликавања покренуто је свих 100 евалуационих случајева, претходно споменутих у одјељку 5.1. За сваки случај упоређени су идентификатори захтјева и критеријума прихватљивости које је модел навео са идентификаторима који су били унапријед задати као очекивани. Добијени резултати приказани су у табели 5.2. Значење коришћених метрика описано је у одјељку 4.4.2. Прецизност показује колики је дио идентификатора које је модел навео исправан, одзив показује колики је дио очекиваних идентификатора пронађен, док F1 обједињује ове двије мјере. Микро резултат се рачуна над свим идентификаторима заједно, док макро резултат представља просјек резултата добијених за појединачне евалуационе случајеве.

Ниво провјере	Прецизност	Одзив	F1
Захтјеви – микро	0,916	0,980	0,947
Критеријуми прихватљивости – микро	0,953	0,981	0,967
Захтјеви – макро	0,966	0,980	0,969
Критеријуми прихватљивости – макро	0,967	0,975	0,968

Табела 5.2 - Резултати евалуације пресликавања

На нивоу захтјева у скупу је било 100 очекиваних идентификатора. Систем је навео 98 од њих, пропустио два и додао још девет идентификатора који нису били очекивани. Због тога је микро одзив 0,980, док је микро прецизност 0,916. На нивоу критеријума прихватљивости било је укупно 208 очекиваних идентификатора. Систем је исправно навео 204, пропустио четири и додао десет додатних. За овај ниво добијени су микро прецизност 0,953, микро одзив 0,981 и F1 вриједност 0,967. Макро резултати су такође високи. Пошто макро резултат посматра сваки евалуациони случај посебно, ове вриједности показују да је систем имао добре резултате у већини појединачних случајева, а не само када се сви идентификатори посматрају заједно.

Поред збирних мјера прегледан је и резултат сваког евалуационог случаја. У 95 од 100 случајева систем је тачно навео и захтјеве и све очекиване критеријуме прихватљивости. У три случаја дио пресликавања је био тачан, али је неки идентификатор недостајао или је додат вишак. У преостала два случаја није пронађен очекивани захтјев. Ова расподјела приказана је на слици 5.5.



Слика 5.5 - Расподјела евалуационих случајева према резултату пресликавања

Резултати показују да је пресликавање било потпуно тачно у већини испитаних случајева. Грешке су се појавиле када је систем навео више сродних захтјева него што је потребно или када је изабран погрешан захтјев из исте функционалне области. Због тога је важно да уз генерисани тест остану приказани идентификатори захтјева и критеријума прихватљивости, како би их QA инжењер могао провјерити.

5.4 Практична употребљивост Appium тест скелета

Практична употрбљивост Appium тест скелета провјеравана је поређењем генерисаног скелета са постојећим Appium тестовима из пројекта. За различите функционалности систему је описано какав тест треба правити, а добијени код је затим упоређен са начином на који је иста или слична функционалност већ тестирана у постојећим Appium тестовима. Посматрано је да ли је структура теста исправна, да ли су изабране одговарајуће функције и селектори и да ли су кораци извршени исправним редослиједом.

Основна структура генерисаног скелета у већини проба била је исправна. Генерисани код је углавном садржао листу тестних корака, почетну вриједност резултата "failed", блок try/finally, упис резултата функцијом write_qa_qc_results(), функцију омотача и улазну функцију run_main_test(). Ова правила су већ задата у датотеци appium_test_style.txt, описаној у одјељку 4.5.2, па је модел имао јасан шаблон по којем треба да формира тест.

Добри резултати добијени су код краћих провјера. Једна од њих односила се на приказ родитељског рејтинга у листи канала. Од система је тражено да направи Appium тест који отвара листу канала и провјерава да ли је за канал приказана информација о родитељском рејтингу. За ову функционалност у пројекту већ постоји Appium тест Channel List Info – Parental Rating, па је генерисани скелет упоређен са њим. Генерисани скелет је задржао исти основни ток провјере и користио постојеће функције open_channel_list() и check_one_key(). Слично је било и код теста за приказ детаља тренутног догађаја у EPG-у, гдје су коришћене функције navigate_to_broadcast_epg(), longpress_center() и check_custom_keys() . Код оваквих тестова, када је број корака мали и потребне функције већ постоје у пројекту, добијени скелет је био веома близак постојећем коду.

На резултат је утицао и начин на који је кориснички упит написан. Када је упит био превише општи, модел није имао довољно података да тачно одреди шта треба провјерити и како до тог дијела апликације треба доћи. У једној таквој проби од система је само тражено да провјери присуство елемента на екрану, без навођења екрана и самог елемента. У одговору су зато остала празна мјеста за селектор или је модел покушао сам да изабере шта треба провјерити. Када се у упиту јасно наведе шта тест треба да уради и шта на крају треба да провјери, добијени код је ближи жељеном тесту.

Највише разлика уочено је код навигације кроз више екрана. Ово се добро види на провјери у којој се најприје покрене репродукција live канала, затим се корисник враћа кроз претходне екране, прелази на Broadcast дио апликације и бира други канал. На крају тест треба да провјери да ли је репродукција настављена и да ли је заиста покренут други канал. За ову функционалност у пројекту постоји Appium тест Playback_After_Zap, па је генерисани скелет упоређен са његовим кодом. У оваквом тесту важан је и редослијед тастера даљинског управљача. Команда back враћа претходни екран, right и down помјерају означену ставку кроз мени, док center потврђује тренутни избор. Ако недостаје један од ових корака, тест може да заврши на погрешноом дијелу апликације иако су коришћене функције исправно изабране.

У првом упиту био је описан читав ток провјере, али није било додатно наглашено да се за прелазак на Broadcast дио користи више тастера даљинског управљача. Систем је исправно изабрао функције navigate_to_live_channel(), check_video_by_successive_frames() и get_last_channel_from_log() , али је за тај прелазак употријебио само команду right. У постојећем тесту након ње слиједе још down и center, како би се означила одговарајућа ставка и потврдио избор. Затим је иста функционалност затражена прецизнијим упитом, у којем је додатно наглашено да се за прелазак на Broadcast користи више тастера, без навођења њиховог редослиједа. У новом генерисаном тесту, добијен је низ right, down и center, исти као у постојећем Appium тесту. Поређење најважнијих дијелова постојећег теста и добијених Appium скелета је приказано на слици 5.6.

Постојећи тест

Playback_After_Zap.py

```
navigate_to_live_channel(driver)
video_present = check_video_by_successive_frames()
channel_before = get_last_channel_from_log(
    device_uuid, log_file_path)

for _ in range(3):
    press_key("back")
    sleep(7)
press_key("right")
sleep(10)
press_key("down")
sleep(3)
press_key("center")
sleep(10)

video_present = check_video_by_successive_frames()
channel_2 = get_last_channel_from_log(
    device_uuid, log_file_path)
```

Први генерисани скелет

општи упит

```
navigate_to_live_channel(driver)
video_present = check_video_by_successive_frames()
channel_before = get_last_channel_from_log(
    device_uuid, log_file_path)

for _ in range(3):
    press_key("back")
    sleep(DELAY)

# претпостављено да је right довољан
press_key("right")
sleep(DELAY)

video_present_broadcast =
    check_video_by_successive_frames()
channel_after = get_last_channel_from_log(
    device_uuid, log_file_path)
```

Након прецизнијег упита

наглашено да се користи више тастера

```
navigate_to_live_channel(driver)
video_present = check_video_by_successive_frames()
channel_before = get_last_channel_from_log(
    device_uuid, log_file_path)

for _ in range(3):
    press_key("back")
    sleep(DELAY)

# прелазак на Broadcast
press_key("right")
sleep(DELAY)
press_key("down")
sleep(DELAY)
press_key("center")
sleep(DELAY)

video_present_broadcast =
    check_video_by_successive_frames()
channel_after = get_last_channel_from_log(
    device_uuid, log_file_path)
```

Слика 5.6 – Поређење постојећег Appium теста и генерисаних скелета

Код дужих тестова потребна је већа ручна провјера. На примјер, у једној проби од система је тражено да направи тест у којем се преко EPG-а закаже више снимања, а затим отвори дио Library и провјери да ли су та снимања приказана међу заказаним снимањима. Систем је пронашао одговарајуће функције за отварање EPG-а и дијела Library, као и потребне провјере, али је поједине кораке навигације између екрана поједноставио. Код оваквих тестова важно је на ком се екрану апликација тренутно налази и која је ставка означена, јер један погрешан притисак

тастера може да промијени наставак теста. Због тога генерисани скелет може да послужи као добра почетна основа, али је потребно ручно провјерити навигацију и редослијед корака.

На основу ових проба може се закључити да је генерисани Appium скелет најупотребљивији код краћих и јасно описаних тестова. Основни шаблон кода је углавном исправно формиран, а постојеће функције и селектори се добро бирају када у документацији постоји сличан примјер. Највише пажње захтијева навигација кроз више екрана, па QA инжењер прије употребе треба да провјери редослијед радњи и дијелове који зависе од тренутног стања апликације.

5.5 Ограничења и могућности унапрјеђења

Испитивање је показало да резултат у великој мјери зависи од података који су доступни систему. Када документација садржи добро описан захтјев и сличне постојеће тестове, систем има довољно информација да формира употребљив приједлог новог теста. Ако таквих примјера нема, генерисани тест је обично општији. Због тога квалитет и количина постојеће QA документације директно утичу на квалитет добијеног резултата.

Друго ограничење односи се на семантичку претрагу. Она проналази документа чији је садржај најсличнији корисничком упиту, али најсличнији документ не мора увијек бити и онај који је за конкретан упит најважнији. Код сродних функционалности може се десити да модел добије контекст који је тематски веома близак траженом, али се односи на другачији случај употребе. Пошто модел одговор формира на основу достављеног контекста, погрешно одабран контекст може да утиче и на коначни резултат.

Тренутна верзија користи Qdrant базу у локалном режиму, при чему Пајтон програм директно приступа бази сачуваној на рачунару. Предност оваквог приступа је једноставније покретање и независтан рад без додатног сервиса који би представљао заједничку тачку приступа. Ако би систем користило више одвојених процеса, једна могућност би била да сваки од њих ради са сопственом локалном базом. На тај начин процеси могу радити независно и паралелно, али се подаци налазе у више копија које је потребно засебно одржавати. Друга могућност је покретање Qdrant-а као посебног сервиса, на примјер у докер окружењу, којем би више процеса приступало као заједничкој бази. Предност овог приступа је што сви користе исто стање података, док је недостатак увођење додатног сервиса кроз који пролазе захтјеви и који при већем оптерећењу може представљати уско грло. Због тога оба приступа имају предности и недостатке, а избор зависи од начина употребе система.

У реализованом рјешењу већ је могуће засебно индексирати захтјеве, Appium примјере, помоћне функције и селекторе. На тај начин није потребно обнављати све колекције када се промијени само једна врста података. Ипак, при поновном индексирању изабране колекције поново се обрађују сви њени записи, што код већег броја докумената може да траје дуже. Даље унапрјеђење могло би да буде индексирање само нових или промијењених записа.

Ограничена је и количина пројектног контекста која се може прослиједити великом језичком моделу у једном упиту. Због тога се моделу не шаље цијела база, већ само контекст који је претрагом изабран за конкретан упит. Ако у том контексту недостаје важан захтјев, тест, функција или селектор, модел ту информацију нема приликом формирања одговора. Ово је посебно важно код генерисања Appium тест скелета, гдје се у исти упит укључује више различитих врста података.

Директни Пајтон начин рада, преко GithubModels или OpenAI кључа, тренутно не чува историју разговора. Свако ново питање обрађује се као засебан упит, па модел не добија претходно питање и одговор, за разлику од приступа преко GitHub Copilot агента. Као једно од проширења могла би се увести историја сесије, тако да корисник може да постави додатно питање које се односи на претходно добијени резултат.

Код генерисања Appium скелета највеће ограничење остају дужи тестови са више корака навигације. Код таквих тестова није довољно пронаћи одговарајуће функције и селекторе, већ је потребно исправно одредити и редослијед радњи кроз више екрана апликације. Као даље унапрјеђење могла би се додати аутоматска провјера генерисаног Пајтон кода и његово пробно извршавање у тестном окружењу, како би се дио грешака открио прије ручне провјере.

6. Закључак

У овом раду развијен је прототип система за подршку тестирању дигиталних ТВ пријемника коришћењем генерисања потпомогнутог претрагом. Циљ је био да се постојећа документација са захтјевима и тестовима искористи као извор знања за велики језички модел, како би генерисани резултати били засновани на стварним пројектним подацима и могли да се повежу са захтјевима и критеријумима прихватљивости. Реализовани систем обухвата обраду REQ документације, семантичку претрагу, генерисање приједлога тест случаја са повезивањем на захтјеве и критеријуме прихватљивости, као и формирање почетног Appium тест скелета.

Реализовано рјешење најприје парсира и структурира захтјеве, ручно написане тестове, постојеће Appium тестове, помоћне функције и селекторе. Припремљени записи се затим индексирају у Qdrant векторској бази. За кориснички упит проналази се релевантан контекст који се прослеђује великом језичком моделу. На основу тог контекста систем може да предложи нови тест случај, провјери повезаност постојећих тестова са критеријумима прихватљивости или формира почетни Appium скелет. Поред директног рада преко Пајтон скрипти, омогућен је и приступ истом припремљеном контексту преко MCP сервера и GitHub Copilot агента.

Испитивање је показало да систем у већини случајева исправно повезује генерисани резултат са изворном документацијом. Од 100 евалуационих случајева, у 95 су исправно наведени очекивани захтјеви и сви очекивани критеријуми прихватљивости. За захтјеве је добијена микро F1 вриједност 0,947, док је за критеријуме прихватљивости износила 0,967. У преосталим случајевима грешке су се односиле на изостављене или додатне идентификаторе и на избор сродног, али неодговарајућег контекста. Ови резултати показују да реализовани приступ може да обезбиједи добру основу за праћење веза између генерисаног теста и документације, али да добијено пресликавање ипак треба да остане доступно QA инжењеру за провјеру.

Практичне провјере показале су и да квалитет генерисаног теста зависи од садржаја који је доступан систему. Када у документацији постоје добро описани захтјеви и сродни тестови,

генерисани приједлог је ближи начину на који се тестови већ пишу у пројекту. Када таквих примјера нема, резултат може бити општији или садржати корак који није наведен у захтјеву. Због тога генерисани тест случај не треба посматрати као коначан резултат који се користи без провјере, већ као приједлог који је потребно прегледати и по потреби дорадити.

Сличан закључак важи и за Appium дио рјешења. Код краћих и јасно описаних тестова генерисани скелет је углавном задржавао структуру постојећег тестног оквира и користио одговарајуће функције и селекторе. Највише разлика појављивало се код дужих тестова са више корака навигације, гдје једна погрешно изабрана команда може да промијени наставак читавог теста. Због тога Appium резултат треба посматрати као почетни скелет који може да олакша припрему аутоматизованог теста, али чију навигацију и завршне провјере треба ручно прегледати прије употребе.

Даљи развој система могао би да буде усмјерен на ефикасније ажурирање и коришћење векторске базе, као и на аутоматску провјеру и пробно извршавање генерисаног Appium кода. На тај начин би се дио грешака могао открити прије ручног прегледа, а систем би био погоднији за употребу над већом количином пројектне документације.

На основу спроведеног испитивања може се закључити да генерисање потпомогнуто претрагом може да буде корисна подршка при анализи постојеће документације, припреми нових тест случајева и почетној фази аутоматизације тестирања. Главна предност реализованог приступа је што се уз генерисани тест наводе захтјеви и критеријуми прихватљивости на којима је тест заснован. На тај начин може се провјерити на основу којих података из документације је тест формиран. Реализован прототип не замјењује QA инжењера, али му може обезбиједити почетни резултат који се затим провјерава и по потреби допуњује.

7. Литература

[1] Z. Ji et al., “Survey of Hallucination in Natural Language Generation,” ACM Computing Surveys, vol. 55, no. 12, art. 248, 2023. doi: 10.1145/3571730.

[2] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020.

[3] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson, 2021.

[4] A. Vaswani et al., “Attention Is All You Need,” Advances in Neural Information Processing Systems, vol. 30, pp. 5998–6008, 2017.

[5] T. B. Brown et al., “Language Models are Few-Shot Learners,” Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.

[6] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks,” in Proceedings of EMNLP-IJCNLP, pp. 3982–3992, 2019. doi: 10.18653/v1/D19-1410.

[7] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge University Press, 2008.

[8] Qdrant, “Qdrant Documentation,” [Online]. Available: https://qdrant.tech/documentation/

[9] Docutils Project, “reStructuredText Markup Specification and Directives,” [Online]. Available: https://docutils.sourceforge.io/rst.html

[10] Appium, “Appium Documentation,” [Online]. Available: https://appium.io/docs/en/latest/

[11] International Software Testing Qualifications Board, Certified Tester Foundation Level Syllabus, Version 4.0.1, 2024. [Online]. Available: https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTFL_Syllabus_v4.0.1.pdf