УНИВЕРЗИТЕТ У БЕОГРАДУ

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

РЕАЛИЗАЦИЈА МОБИЛНЕ АПЛИКАЦИЈЕ ЗА ПРОНАЛАЖЕЊЕ И ОЦЕЊИВАЊЕ МАЈСТОРА НА УРЕЂАЈИМА СА АНДРОИД ОПЕРАТИВНИМ СИСТЕМОМ

Ментор:

проф. др Марија Пунт

Кандидат:

Андреј Јокић 2022/3015

Београд, август 2024.

САДРЖАЈ

1.Увод

Тема ове тезе је пројектовање апликације за проналазак и рецензију мајстора на Андроид мобилној платформи. Апликација омогућава корисницима повезивање са поузданим и квалитетним мајсторима у њиховом подручју како би олакшали процес проналажења и ангажовања професионалних услуга.

Проналазак квалитетног и поузданог мајстора представља изазов у већим градовима где се људи претежно баве канцеларијским пословима. Проценат становништа који чине мајстори у оваквима градовима је релативно мали, и поставља се питање како пронаћи доброг мајстора. Људи се у оваквим ситуацијама окрећу препорукама пријатеља, али број оваквих препорука је лимитиран, као и доступност и квалитет мајстора. Циљ овог рада је реализација мобилне апликације која ће корисницима омогућити повезивање са поузданим и квалитетним мајсторима у њиховом подручју како би олакшали процес проналажења и ангажовања професионалних услуга. Апликација ће корисницима пружити детаљне информације о мајсторима и њиховим услугама, напредне опције за претрагу мајстора, преглед рецензија и сатисфакцију клијената одређеним мајстором, као и интересантне и поучне приче подељене од стране регистрованих корисника.

Систем ће бити реализован коришћењем већег броја технологија. Мобилни део система ће бити имплементиран у Котлин програмском језику. Графички кориснички интерфејс мобилне апликације биће имплементиран помоћу *Jetpack Compose* библиотеке. Комуникација са сервером биће реализована преко HTTP протокола користећи *Retrofit* библиотеку. Серверски део система ће бити имплементиран помоћу *SpringBoot* радног оквира, који ће податке складиштити у релационој бази података *MySQL*.

Теза је организована у шест поглавља. Прво поглавље представља увод који се осврће на саму област тезе, њену структуру и коришћене технологије приликом имплементације. Циљ другог поглавља је упознавање читаоца са проблемом. Садржи детаљну анализу постојећих решења, као и предлог новог. Треће поглавље описује технологије коришћене за реализацију система и разлоге за њихов одабир.

Четврто поглавље представља осврт на саму реализацију система и даје детаљан опис архитектуре система, имплементације компонената и примене коришћених технологија. У петом поглављу, описан је рад система и његов графички кориснички интерфејс у формату корисничког упутства. Последње поглавље садржи скраћени увод, са рекапитулацијом самог рада и конкретних закључака изведених током реализације, као и кратак параграф о могућој надоградњи система.

2. ПРЕГЛЕД ПОСТОЈЕЋИХ И ПРЕДЛОГ НОВОГ РЕШЕЊА

Као део израде овог рада, било је потребно упознати се са доступним решењима и понудити ново. Истраживање постојећих решења је неопходно за разумевање тржишних захтева и корисничких очекивања. Анализа конкурентних апликација омогућава идентификацију њихових предности и недостатака, што помаже у креирању иновативног и функционалног производа који боље задовољава потребе корисника. Додатно, овакво истраживање може указати на могуће технолошке и дизајнерске недостатке, чиме се минимизују ризици током развоја апликације. Истраживање обухвата како мобилне апликације тако и интернет решења, чиме се постиже свеобухватнији увид у различите приступе и технолошке оквире који се користе у решавању сличних проблема. Разумевање предности и ограничења обе врсте апликација омогућава боље доношење одлука у погледу функционалности, корисничког искуства и приступачности.

Пре почетка израде рада анализирани су сајтови „ProfiMajstor”, „PozoviMajstora”, „NadjiMajstora” и мобилна апликација „Postaj”. Наведене апликације представљају тренутне лидере на тржишту проналаска мајстора и опслужују највећи број корисника. Анализа је спроведена на основу неколико критеријума, укључујући интегритет мајстора, реалну оцену квалитета њихових услуга, као и приступачност. Поред тога, испитане су напредне опције претраге мајстора, ниво детаља о понуђеним услугама и постојање занимљивог садржаја који доприносе корисничком искуству и времену проведеном користећи апликацију. Додатно, кључна ставка у анализи је преглед рецензија и претходних радова мајстора. На основу истраживања објављеног на *nicejob* платформи [1], чак 88% корисника верује рецензијама исто колико и препорукама од стране пријатеља. Број рецензија је такође битна ставка јер око 59% потенцијалних клијената жели да види најмање 2 до 3 рецензије пре било какве одлуке.

2.1. Анализа сајта „ProfiMajstor”

Сајт „ProfiMajstor” представља тип решења искључиво намењен за клијенте услуга мајстора.

Мајстори нису директно видљиви корисницима и не могу интераговати са њима на било који начин. Сајт функционише тако што клијенти контактирају саме руководиоце сајта (преко електронске поште или позивом) и објашњавају своје захтеве, као што је приказано на слици 2.1.

Руководиоци сајта представљају посредника између клијената и мајстора, и врше проналазак одговарајућег мајстора. Постојање посредника који има улогу агенције доноси неке добре али и неке лоше особине. Посредник у конекцији може помоћи људима који се не осећају пријатно за овакав тип комуникације и помоћи им да конкретно формулишу своје захтеве. Додатно, неки мајстори такође нису добри или немају стрпљења у комуникацији са клијентима. Постојање централизованог места за проналаза мајстора може пружати веће поверење и сигурност клијентима јер посредник често врши проверу квалификација и репутације мајстора. Није познато након инспекције сајта на који начин се мајстори региструју код руководиоца сајта.

Постојање посредника доноси и све лоше особине које агенције поседују. Сво знање о мајсторима се налази у рукама руководиоца сајта који то може злоупотребити. Цене трошкова услуга су непознате и углавном веће због провизије коју посредник узима, што може одвратити клијенте од позива. Посредник може ограничити иницијалну директну комуникацију између мајстора и клијента, што понекад може довести до неспоразума или недовољно персонализоване услуге. Додатно, сајт не поседује могућност прегледа и рецензија мајстора, једну од кључних корисничких захтева [2].

Слика 2.1. Контакт мајстора на сајту „ProfiMajstor” [2]

2.2. Анализа сајта „PozoviMajstora”

Сајт „PozoviMajstora” представља други тип решења. Функционише тако што корисници лично претражују доступне мајсторе на сајту без постојања било каквог посредника. Овака тип решења кориснику нуди већу контролу и флексибилност, али може бити и захтевнији и компликованији за коришћење, нарочито ако корисник нема довољно знања или времена да се посвети процесу претраге [3].

Систем пружа две опције претраге мајстора. Прва је текстуално поље, приказано на слици 2.2.1, у које корисник може унети било шта. Није конкретно објашњен начин употребе овог поља на сајту. Након дубље анализе, донет је закључак да поље служи као филтер за мајсторе на основу унете стручне делатности. Унета текстуална вредност се пореди са вредностима које су мајстори пријавили током регистрације. Овакав тип претраге је врло проблематичан и захтева познавање стручних термина од стране клијената услуга. Врло је подложан људским грешкама и може довести до непредвиђених ситуација.

Слика 2.2.1. Поље за текстуалну претрагу мајстора на сајту „PozoviMajstora” [3]

Друга опција претраге јесте навигација кроз категорије стручних делатности које сајт пружа помоћу графичког корисничког интерфејса. Категорије су подељене у два нивоа да би обезбедиле лакшу навигацију кориснику. Након што корисник изабере категорију првог, а затим и другог нивоа, приказује се листа имена свих мајстора који пружају изабрану услугу.. Овакав тип претраге омогућава корисницима да лако и брзо пронађу жељену категорију захваљујући визуелним елементима као што су иконе, слике или инфографике. Корисници који нису сигурни у тачне називе или описе услуга могу лакше да се сналазе путем слика или икона, јер оне дају јасну асоцијацију на одређену врсту мајстора или услуге.

Додатно, визуелно атрактиван интерфејс може привлачи пажњу новим корисницима и повећава време проведено на сајту. Неке од негативних страна оваквог типа претраге могу бити прекомерна сложеност и спорија брзина претраге. Графичка навигација може бити спорија у поређењу са текстуалним претраживачем, посебно ако је корисник већ упознат са сајтом или конкретном услугом коју тражи. Дизајн графичких елемената има кључну улогу у практичности претраге. Превелика количина визуелних елемената или лоше дизајниране иконе и графички елементи, могу довести до конфузије и фрустрације корисника. На слици 2.2.2. приказан је одабир категорије првог нивоа.

Слика 2.2.2. Графички интерфејс за претрагу мајстора на сајту „PozoviMajstor” [3]

Поред претраге мајстора по стручним делатностима, сајт омогућава и регистрацију и пријаву корисника који на тај начин откључавају приступ додатним функционалностима система. Пријављени корисници имају приступ приватним подацима мајстора као што су број телефона или слике претходних радова. Додатно, пријављени корисници могу видети рецензије бивших клијената одређеног мајстора.

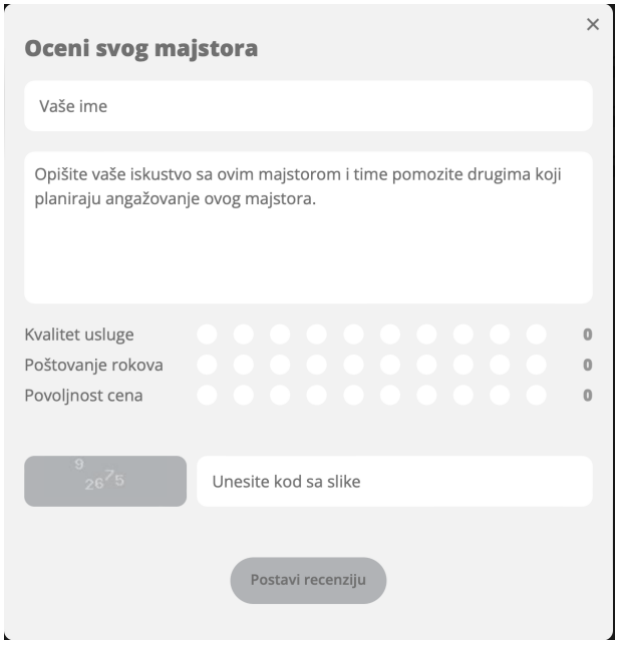
Непријављени корисници могу само да претражују мајсторе преко неког од два описана метода. Међутим, након претраге и проналаска одговарајућег мајстора, корисник нема могућност увида у квалитет и интегритет датог мајстора. Штавише, с обзиром да су приватни подаци мајстора сакривени, корисник нема ни могућност да контактира мајстора. Систем је у поптуности посвећен пријављеним корисницима и потпуно занемарује велики број потенцијалних клијената који немају регистроване налоге у систему. Додатно, сајт не поседује никакав садржај који би кориснике дуже задржао, као на пример мајсторске приче.

2.3. Анализа сајта „NadjiMajstora”

Сајт „NadjiMajstora” спада у категорију другог типа решења, где корисници лично претражују и контактирају мајсторе без постојања посредника. Систем такође пружа могућност претраге мајстора преко текстуалног поља или навигацијом кроз категорије стручних делатности користећи графички кориснички интерфејс. Међутим, овај сајт поседује напреднији систем текстуалне претраге који је мање подложан људским грешкама приказан на слици 2.3.1. Наиме, у текстуално поље могуће је унети стручну делатност, пребивалиште или име/презиме мајстора. Није потребно унети потпуно идентичну текстуалну вредност коју систем очекује. Алгоритам претраге покушава да уклопи унету вредност са кључним речима из система. Првих пар карактера одређене кључне речи довољне су да би их систем укључио у резултат. Ипак, већина недостатака овог приступа су и даље присутни. Грешка у уносу произвешће празну листу резултата [4].

Слика 2.3.1. Поље за текстуалну претрагу на сајту „NadjiMajstora” [4]

Претрага мајстора користећи навигацију система по категоријама стручних делатности функционише на поптуно идентичан начин као и код сајта „PozoviMajstora” и неће бити даље разматран. Такође, сајт корисницима пружа могућност регистрације мајстора у систем, остављања рецензија на мајсторе и увид у детаљне податке одабраног мајстора. Главна разлика јесте што овај сајт не захтева од корисника да буду пријављени на систем. Непријављени корисници имају пристум свим функционалностима система. Штавише, сајт не поседује опцију пријављивања корисника на систем. Сви клијенти мајстора су анонимни корисници сајта. Овакав приступ доприноси брзом и лаком приступу информацијама без потребе за уношењем личних података. Пошто не захтева пријављивање, сајт је доступан свима, што може привући већи број корисника, укључујући оне кориснике који желе брзе и једноставне услуге или оне који само повремено користе услуге и не желе да се региструју. Додатно, систем не мора да управља системима за аутентификацију и складиштење корисничких података, чиме се смањују сложеност и трошкови одржавања система. Омогућава се фокусирање на основне функционалности и побољшано корисничко искуство.



Слика 2.3.2. Графички интерфејс за претрагу мајстора на сајту „NadjiMajstora” [4]

Са друге стране, могућност остављања рецензије од стране анонимних корисника сајта уводи доста недостатака. Пошто корисници нису пријављени, нема гаранције да су заиста користили услуге мајстора о којима пишу рецензије. Ово може довести до лажних или злонамерних рецензија, што смањује кредибилитет и поузданост система оцењивања. Додавање рецензије приказано је на слици 2.3.2.

Отворен систем за рецензије је подложан спам коментарима или злонамерним активностима, као што су масовно објављивање негативних рецензија од стране конкуренције или других заинтересованих страна. Без пријаве, администратори сајта не могу пратити историју рецензија одређених корисника. Ово отежава идентификацију и уклањање корисника који систематски злоупотребљавају платформу. Са психолошке стране, непријављени корисници могу осетити мању одговорност за своје рецензије јер њихов идентитет није повезан са коментаром. Према чланку објављеном на *Wiley* платформи, анонимност може подстаћи неодговорне или непоштене рецензије [5].

Додатну функционалност коју овај сајт пружа јесте визуелни приказ области града које мајстор покрива на интерактивној мапи. Ову информацију мајстор уноси приликом регистрације на систем. Интерактивна мапа приказана је на слици 2.3.3. На овај начин, корисници могу врло брзо видети које области мајстор покрива без потребе да читају или разумеју сложене текстуалне описе. Текстуални приказ захтева од корисника да прочита и протумачи списак насеља, улица или области, што може бити збуњујуће и мање интуитивно. Интерактивна мапа омогућава прецизно одређивање граница покривености, чиме се избегавају нејасноће. Корисници могу директно видети да ли се њихова тачна локација налази унутар покривене зоне. Додатно, визуелно атрактиван и динамичан приказ на мапи чини сајт привлачнијим и ангажујућим за кориснике. То може подстаћи кориснике да дуже остану на сајту и истраже више опција.

Слика 2.3.3. Визелни приказ оперативног подручја мајстора на сајту „PozoviMajstor” [4]

2.4. Анализа мобилне апликације „Postay”

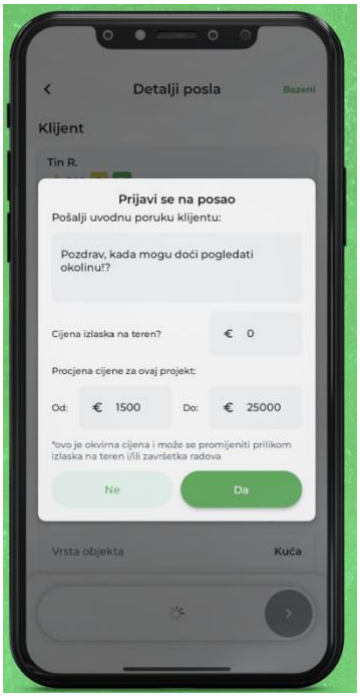
Детаљна анализа постојећих решења подразумева и анализу решења имплементираног за још неку платформу поред интернета. Одабрана је мобилна апликација „Postay”, доступна на *IOS* и Андроид оперативним системима. С обзиром да су апликације поприлично сличне, у овом раду фокусираћемо се искључиво на верзију апликације имплементирану за Андроид оперативни систем [6].

Ова апликација представља трећи тип решења за проблем проналаска мајстора за одговарајући тип услуге. Главна разлика јесте то да је ток контроле у овом систему обрнут. Наиме, уместо да клијенти претражују мајсторе на основу унетих критеријума, мајстори претражују тражене услуге од стране клијената. Додатно, функционалности за клијенте и мајсторе подељене су у две потпуно одвојене мобилне апликације. Функционалности клијента смештене су апликацију под називом „Postay”, док су функционалности мајстора смештене у апликацију „Postay Partner”. Подршка за непријављене клијенте не постоји, односно, апликација захтева од корисника да се региструје и пријави на систем пре употребе.

Апликација функционише тако што клијенти прво креирају тражене услуге у облику огласа на које мајстори могу да се пријаве. Након креирања огласа, клијент чека да се један или више мајстора пријави као заинтересован за тражену услугу. Овакав приступ поједностављује процес и штеди време клијенту које би потрошио на претрагу мајстора јер не морају активно да претражују мајсторе, већ само поставе оглас и чекају да се мајстори сами пријаве. С обзиром да су мајстори који се пријављују на оглас већ заинтересовани за одређену услугу, клијент одмах добија понуде од релевантних и доступних мајстора и не губи време на заузете мајсторе.

Међутим, овакав приступ значајно смањује контролу коју клијент поседује. Клијенти немају активну контролу над претрагом и могу бити ограничени само на понуде које добију, што може бити фрустрирајуће ако је понуда слаба или неадекватна. Додатно, просечно време чекања на анговажовање мајстора из перспективе клијента се значајно повећава. Постављање огласа и чекање на пријаве је знатно спорији процес у поређењу са директним претраживањем и контактирањем мајстора. Ако је услуга хитна, овакав приступ је потпуно неупотребљив.

Након креирања огласа за тражену услугу, она бива видљива од стране свих мајстора. Мајстор може да се пријави за одређени оглас и попуњава неопходне податке за своју услугу, као што су цена и оквирно време трајања радова. На слици 2.4.1. приказана је форма за пријаву мајстора. Опција пријављивања више мајстора на један оглас поспешује мотивисаност мајстора да понуде боље цене, услове или брже време извршења, што може бити корисно за клијента. Са друге стране, може доћи до негативног ефекта изгладњивања појединог мајстора – ако клијенти увек одаберу другог мајстора, дати мајстор никад неће добити посао. С обзиром да се мајстори могу пријавити на било који оглас, постоји могућност да понуде своје услуге клијентима које иначе не би биле видљиве у обичној претрази, због маркетиншких или неких других разлога. На овај начин повећава се униформност расподеле послова мајсторима и смањује просечно чекање на посао мајстора.



Слика 2.4.1. Пријава мајстора за услугу на апликацији „Postay” [6]

Када се мајстор пријави за одређени оглас, поново чека да аутор огласа изабере баш њега. Из перспективе мајстора са пуно понуда, поново се губи време. Када клијент изабере одређеног мајстора, период синхронизације је завршен. Овакав тип синхронизације идентичан је *3-way-handshake* механизму који се користи у TCP протоколу за успостављање везе између клијента и сервера [7].

Након завршеног посла, клијент и мајстор имају могућност оцењивања извршене услуге мајстора, односно интегритета тражене и реалне услуге клијента. С обзиром да је ово једини начин оцењивања мајстора и клијената, интегритет, веродостојност и поштеност рецензија је загарантован.

2.5. Предлог новог решења

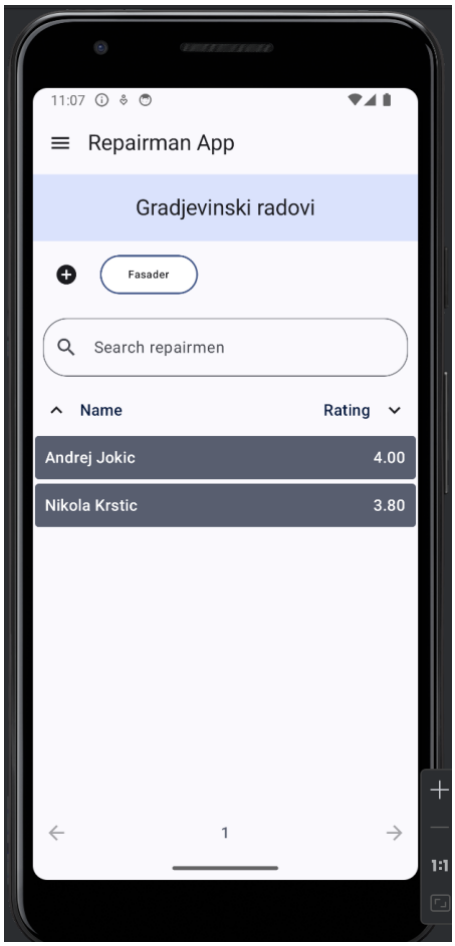
На основу детаљне анализе постојећих решења, одлучено је да ће решење бити имплементирано као мобилна апликација за оперативни систем Андроид. Кључан критеријум јесте тај што је број постојећих мобилних апликација знатно мањи у односу на број веб апликација и нуде врло лимитиране функционалности. Решење је дизајнирано са циљем да испуни кључне критеријуме разматране у претходном потпоглављу и надомести функционалне недостатке које анализирана решења поседују.

Интегритет корисника система и њихових акција намеће се као главни нефункционални захтев који систем мора да испуњава. Без подршке система за интегритет акција корисника већина функционалних захтева постају неупотребљиви. На пример, систем за рецензију мајстора више није веродостојан и потпуно бескорисан. Да би се обезбедио интегритет корисника, систем мора да омогући регистрацију и пријаву корисника на систем. Међутим, већина корисника не жели или нема времена да се региструје на систем. Да би апликација била приступачна што већем броју корисника, систем мора да обезбеди непријављеним корисницима све функционалности које не захтевају проверу интегритета.

За реализацију потражње решења услуге изабран је други тип решења – клијенти ће имати опцију да претражују мајсторе помоћу текстуалног поља или навигацијом кроз категорије стручних делатности које апликација пружа помоћу графичког корисничког интерфејса. Додатно у односу на постојећа решења, систем пружа опцију претраге мајстора на основу више критеријума. На пример, систем омогућава претрагу мајстора који пружа услуге и паркетара и молера. На овај начин корисници могу пронаћи одговарајућег мајстора за више потенцијалних услуга. На слици 2.5. приказан је екран за напредну претрагу.

Такође, апликација пружа могућност сортирања мајстора на основу просечне оцене или имена лексикографкси. Пријављени корисници имаће опцију да оцене квалитет услуге мајстора и додају кратак опис којим би помогли у избору будућим корисницима система. Пријављени мајстори имаће опцију додавања интересантих прича које други пријављени или непријављени корисници могу да прочитају.

На овај начин продужава се просечно време проведено на сајту корисника. Додатно, пријављени мајстори имаће опцију додавања слика својих претходних радова за одређену услугу. На овај начин повећава се сигурност и поверење клијената у изабраног мајстора.



Слика 2.5. Претрага мајстора у предложеном решењу

У табели 2.5., приказана су сва решења анализирана у овом поглављу, као и кључни критеријуми узети у обзир током анализе.

Табела 2.5. Преглед функционалности анализираних и предложеног решења

	ProfiMajstor	PozoviMajstora	PronadjiMajstora	Postay	Предлог
Употребљивост сајта код непријављених корисника	ДА	НЕ	ДА	НЕ	ДА
Претрага мајстора	НЕ	ДА	ДА	НЕ	ДА
Напредна претрага мајстора по категоријама	НЕ	НЕ	ДА	НЕ	ДА
Интегритет акција корисника	НЕ	ДА	НЕ	ДА	ДА
Рецензије на мајсторе	НЕ	ДА	ДА	ДА	ДА
Приказ претходних радова мајстора	НЕ	НЕ	НЕ	НЕ	ДА
Занимљив садржај који би задржао кориснике	НЕ	НЕ	НЕ	?	ДА
Практичан преглед својих акција	НЕ	НЕ	НЕ	ДА	ДА
Напредна претрага мајстора по више категорија	НЕ	НЕ	НЕ	НЕ	ДА
Манипулација резултата претраге мајстора	НЕ	НЕ	ДА	ДА	ДА
Приказ оперативног подручја мајстора на мапи	НЕ	НЕ	ДА	НЕ	НЕ

3. КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ

Предложено решење биће имплементирано као мобилна апликација за оперативни систем Андроид користећи програмски језик Котлин. За имплементацију графичког корисничког интерфејса изабрана је библиотека *Jetpack Compose*. Серверски део система биће имплементиран у радном оквиру *SpringBoot*, који нуди робустну и скалабилну основу за развој HTTP сервера. За складиштење и управљање подацима изабрана је *MySQL* база података, захваљујући њеној поузданости, перформансама и широкој примени у индустрији. Избор ових технологија заснован је на релевантности у савременој пракси мобилног програмирања, значајном броју активних корисника и програмера, као и богатој подршци заједнице која омогућава континуирано побољшање и одржавање софтверских решења. Овај приступ гарантује да ће систем бити не само технички адекватан, већ и компатибилан са тренутним тржишним стандардима и очекивањима корисника.. Додатно, обилна и лако доступна документација омогућава бржи развој и одржавање система.

3.1. *MySQL* база података

Базе података представљају један од основних стубова савремене информационе инфраструктуре, омогућавајући ефикасно складиштење, управљање и приступ великим количинама података. Постоје два основна типа база података - релационе и нерелационе. Избор зависи од специфичних захтева сваког пројекта. Релационе базе погодне су за апликације са јасно дефинисаним и конзистентним подацима, које захтевају напредне упите са комплексним релацијама између више табела. Нерелационе базе су бољи избор за скалабилне системе који управљају великим количинама неструктурираних или полу-структурираних података. За израду овог пројекта изабрана је релациона база, јер је пројекат релативно једноставан и захтева јасно дефинисану структуру података која омогућава лако управљање и манипулацију. У табели 3.1., приказане су предности и мане два типа база [8].

MySQL је један од најпопуларнијих релационих система за управљање базама података (RDBMS) са широком применом, укључујући веб апликације, управљање подацима у облаку и аналитичке системе.

Основни концепт релационих база података заснива се на коришћењу табела за представљање података. У овим табелама, колоне дефинишу атрибуте података, док редови представљају записе. Ова структура омогућава лако креирање односа између различитих скупа података путем заједничких атрибута, познатих као кључеви. У бази *MySQL*, примарни кључеви се користе за јединствену идентификацију редова у табели, док страни кључеви омогућавају успостављање односа између различитих табела.

MySQL подржава Структурисани језик упита (SQL) као стандардни језик за управљање подацима. SQL омогућава дефинисање, манипулацију и контролу приступа подацима кроз наредбе као што су *SELECT*, *INSERT*, *UPDATE* и *DELETE*. Ово обезбеђује да корисници могу лако извршавати комплексне упите и манипулације подацима са минималним напором. Једна од значајних карактеристика *MySQL* базе јесте његова подршка за трансакције, које омогућавају груписање вишеструких операција у једну логичку јединицу рада. Ово се постиже употребом ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*) својстава, које обезбеђују интегритет података у случају грешака или неочекиваних прекида рада система. *MySQL* подржава разне механизме за управљање трансакцијама, као што су *InnoDB* и *NDB* (*MySQL Cluster*) формат складиштења. Додатно, обезбеђује подршку за репликацију података, омогућавајући копирање и синхронизацију података између различитих сервера. Ова функционалност је кључна за обезбеђивање високог степена доступности и отпорности система, омогућавајући лако постављање кластерских конфигурација и балансирање оптерећења [9].

Табела 3.1. Поређење релационих и нерелационих база података

Својство	Нерелационе базе	Релационе базе
Доступност	Добра	Добра
Конзистенција	Лоша	Добра
Складиштење података	Огромне количине	Средње – Велике колине
Перформансе	Високе	Ниске
Поузданост	Лоша	Добра
Скалабилност	Висока	Висока (али скупља)

Са аспекта перформанси, *MySQL* пружа бројне оптимизације и алате као што су индекси, кеширање упита и партиционисање табела. Индекси значајно побољшавају брзину извршавања упита, омогућавајући бржи приступ релевантним подацима. Кеширање упита смањује учесталост приступа диску кеширањем резултата често коришћених упита у меморији, док партиционисање табели омогућава ефикасно управљање великим скуповима података кроз растављање табела на мање сегменте. У погледу скалабилности и интероперабилности, *MySQL* је компатибилан са бројним платформама и оперативним системима, укључујући *Linux, Windows* и *macOS*. Као отворени и широко прихваћени стандард, *MySQL* подржан је од стране великог броја програмских језика и оквира, као што су Јава, *Kotlin*, и *Python*, омогућавајући лаку интеграцију у различите апликационе и технолошке околине.

Закључно, *MySQL* представља робусан и флексибилан систем за управљање релационим базама података, који обезбеђује висок ниво перформанси, безбедности и поузданости. Његова способност да се прилагоди различитим корисничким потребама и применама чини га једним од главних избора за управљање подацима у модерном дигиталном окружењу.

3.2. Програмски језик Јава

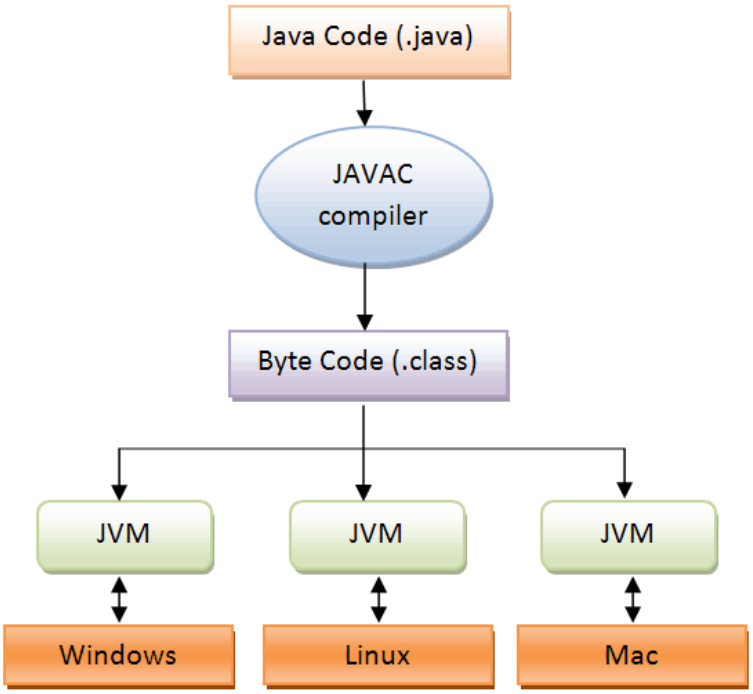
Јава је објектно-оријентисани програмски креиран од стране компаније *Sun Microsystems* 1995. године. Еволуирао је од скромних почетака до покретања великог дела данашњег дигиталног света, пружајући поуздану платофрму на којој су изграђене многе услуге и апликације. Нови, иновативни производи и дигиталне услуге и даље се ослањају на овај језик

Једна од најзначајнијих карактеристика Јаве јесте њен концепт „*Write Once, Run Anywhere*", што у преводу значи да програм написан у Јави може да се покреће на било којој платформи која има инсталиран *Java Virtual Machine* (JVM). Код написан у Јави преводи се у бајткод, који JVM интерпретира и извршава. Захваљујући овоме, Јава апликације извршаваће се идентично на било којој платформи, без обзира на хардвер или оперативни систем. Друга важна карактеристика Јаве јесте потпуно објектно-оријентисани приступ, односно све програмске компоненте у Јави дефинисане су као објекти. Овакав приступ омогућава бољу модуларност програма и поновну употребу кода, због чега је развој и одржавање софтвера ефикаснији и јефтинији [10].

Главни концепти објектно-оријентисаног програмирања имплементирани у Јави укључују енкапсулацију, наслеђивање, полиморфизам и апстракцију.

Треће препознатљиво својство Јаве јесте механизам за аутоматско управљање меморијом, односно аутоматско сакупљање смећа (*garbage collection*), који минимизира ризик од меморијских цурења (*memory leak*) и других безбедносних пропуста програмера. Јава такође подржава вишенитно програмирање, што омогућава развој апликација које истовремено извршавају више таскова, побољшавајући перформансе и ефикасност коришћења системских ресурса. Вишенитно програмирање у Јави омогућава асинхроно извршавање и бољу искоришћеност језгара код мултипроцесорских система.

Иако интерпретирана природа Јаве може донекле умањити перформансе у поређењу са нативним програмским језицима као што су *C++* или *Rust*, напредне технике компилације као што је *Just-In-Time* (JIT) компилација су значајно побољшале брзину и ефикасност Јава апликација. JIT компилација преводи бајткод у машински код током извршавања, чиме се повећавају перформансе апликација. На слици 3.2. приказан је поступак превођења Јава кода у машински код.



Слика 3.2. Процес компилације пројекта на програмском језику Java [11]

3.3. Радни оквир *Spring Boot*

Java Spring представља радни оквир отвореног програмског кода имплементиран у Јави намењен за креирање микросервиса. Развијен је од стране компаније *PivotalTeam* и користи за имплементацију независних *Spring* апликација високог квалитета. Микросервисна архитектура омогућава рашчлањивање велике монолитне апликације у мање, „лакше" и независне делове од којих сваки има различите одговорности. На овај начин олакшава се развој апликације и повећава скалабилност.

Spring Boot представља алат за креирање *Spring* интернет апликација и микросервиса на бржи и једноставнији начин. Главна предност оквира *Spring Boot* јесте његова способност да минимизује конфигурацију потребну за покретање сложеног интернет апликативног система. Кроз употребу механизма ауто-конфигурације, *Spring Boot* аутоматски подешава потребне зависности и параметре на основу библиотека које су присутне у пројекту. Ова функционалност значајно убрзава развојни циклус, омогућавајући развојном тиму да се фокусира на пословну логику и специфичне захтеве пројекта уместо на ручну конфигурацију.

Подршка за уграђене сервере као што је *Tomcat* омогућава покретање апликације као самостални JAR фајл без потребе за екстерним апликационим сервером. Додатно, *Spring Boot* садржи предефинисан скуп зависних библиотека које поједностављују додавање функционалности апликацији. На пример, *spring-boot-starter-web* укључује потребне библиотеке за развој веб апликација, док *spring-boot-starter-data-jpa* обухвата библиотеке за интеграцију са базом података. Ово својство знатно олакшава развој апликације у локалу. *Spring Boot* такође интегрише *Spring Security*, који омогућава лаку имплементацију аутентикације и ауторизације корисника. Сигурносне конфигурације могу бити дефинисане путем једноставних анотација и конфигурационих фајлова [12]. На слици 3.3. приказан је лого радног оквира *Spring Boot.*

Слика 3.3. Лого радног оквира Spring Boot [12]

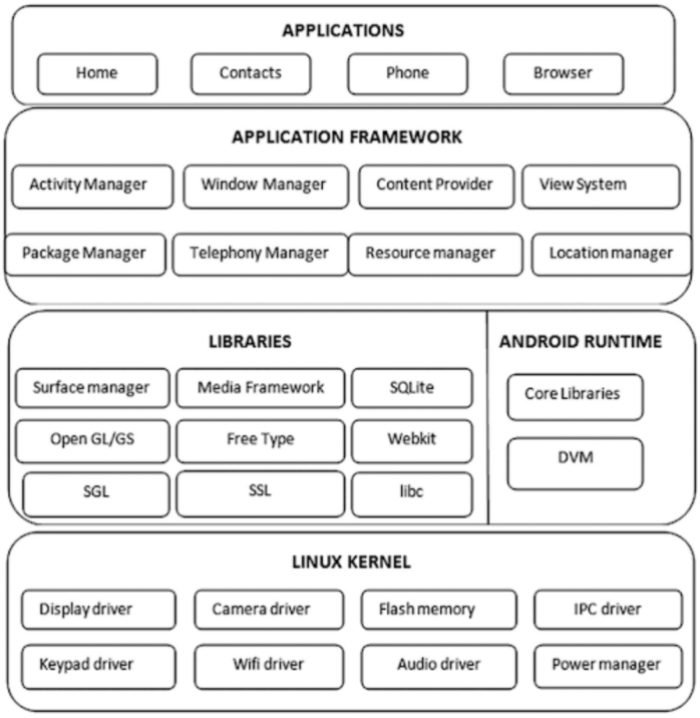
3.4. Оперативни систем Андроид

Андроид је најпопуларнији оперативни систем за мобилне уређаје, развијен од стране компаније *Google* и бесплатан за употребу на широком спектру хардвера. Почевши од свог лансирања у 2008. години, Андроид се брзо учврстио као предводник на тржишту захваљујући својој отворености, подршци за различите уређаје и бројним могућностима за персонализацију. Са преко 2 милијарде активних уређаја, Андроид екосистем је постао највећа платформа за дистрибуцију мобилних апликација. У овој секцији биће површно објашњени историја, архитектура, кључне функционалности и утицај Андроида на мобилну технологију.

Изворни код Андроида објављен је од стране компаније *Google* и доступан је под лиценцом отвореног кода, иако се већина Андроид уређаја испоручује у комбинацији софтвера отвореног кода и власничких лиценци, између осталог и власничког софтвера неопходног за приступ *Google* сервисима. Андроид је популаран међу технолошким компанијама које захтевају готове, јефтине, прилагодљиве и лаке оперативне системе за своје високотехнолошке уређаје.

Архитектура Андроида дизајнирана је тако да омогући високе перформансе и флексибилност. Састоји се из више слојева. У срцу Андроида налази се *Linux* кернел, који пружа основне системске услуге као што су безбедност, управљање меморијом, процесима, мрежом и остале драјвере за хардвер. Следећи слој јесте извршно окружење - укључује *Android Runtime* (ART) који управља извршењем апликација. ART доноси значајна побољшања у односу на свог претходника, *Dalvik Virtual Machine*, посебно у погледу перформанси и управљања меморијом. Андроид користи бројне *C/C++* библиотеке као што су *WebKit* за претрагу интернета, *OpenGL ES* за графику, и *SQLite* за управљање базама података. Следећи слој, под називом *Application Frameowork*, пружа висок ниво API, омогућавајући програмерима да користе основне функције уређаја као што су телефонски позиви, подаци из сензора, и управљање апликацијама. Последњи слој на врху архитектуре представљају апликације које свакодневно користимо, укључујући системске апликације као што су позивање, размена порука, и управљање контактима [13]. Архитектура Андроида приказана је на слици 3.4.

Андроид оперативни систем познат је по низу иновативних функција и функционалности које га чине привлачним за кориснике и програмере. Отворени системи као што је Андроид омогућавају произвођачима уређаја и програмерима значајну флексибилност у прилагођавању и оптимизацији софтвера за своје потребе. Андроид подржава различите типове уређаја, укључујући телефоне, таблет рачунаре, паметне сатове, телевизоре и аутомобилске системе, чиме подстиче једноставну интеграцију и подршку за широк спектар уређаја. Интеграција Гугл услуга као што су *Google Play Store, Gmail, Google Maps* и *Google Assistant* обезбеђује корисницима приступ великом броју апликација и сервиса који се користе у сваком реалном систему.



Слика 3.4. Архитектура оперативног система андроид [13]

3.5. Програмски језик Котлин

Котлин је модеран програмски језик развијен од стране *JetBrains* компаније и најчешће се користи за развој мобилних Андроид апликација. Креиран је са намером да буде потпуно компатибилан са програмским језиком Јава, наследи њене предности и превазиђе мане примећене током њене вишедеценијске употребе. Штавише, нека развојна окружења намењена за програмирање у Котлину (као што је *IntelliJ*) садрже алате за аутоматско превођење Јаве у Котлин код.

Интероперабилност са Јавом омогућава рад са постојећим Јава библиотекама и пројектима без икаквих проблема, чиме се олакшава и подстиче прелазак на Котлин. Концепт „*Write Once, Run Anywhere*" који омогућава Јава показао се као велики успех, и подржан је од стране Котлина који такође користи JVM за интерпретацију и извршавање кода. Котлин смањује потребу за шаблонским кодом који је чест у Јави. На пример, неке типичне Јава конструкције као што су инспектори и мутатори атрибута, *equals(), hashCode(), и toString()* у Котлину могу врло једноставно бити генерисане аутоматски. Додатно, Котлин је дизајниран са уграђеном безбедношћу против изузетка нултне вредности (*NullPointerException*), који се показао као врло проблематичан. У Котлину, програмери морају експлицитно да означе промењљиве које могу да буду *Null*. На овај начин, смањује се број грешака које настају услед нултних вредности.

Иако је дизајниран као објектно оријентисани програмски језик, Котлин подржава функционалне програмске парадигме које омогућавају чистији и концизнији код, као што су ламбда изрази, функције вишег реда, непромењиве колекције и много више. Котлин поседује типску инференцију која омогућава да компајлер сам одреди типове израза на основу контекста, чиме се смањује потреба за експлицитним навођењем типова у коду и додатно повећава конзицност и читљивост кода. Додатно, Котлин уводи уграђену подршку за корутине, које олакшавају и поједностављују асинхроне (неблокирајуће) операције као што су HTTP захтеви преко интернета или сложени упити над базом података [14]. На слици 3.5. приказан је лого програмског језика Котлин.

Слика 3.5. Лого програмског језика Kotlin [14]

3.6. Радни оквир *Jetpack Compose*

Jetpack Compose је модеран, декларативни радни оквир за развој графичког корисничког интерфејса (GUI) на Андроиду, развијен од стране компаније *Google*. Намера ове платформе је да поједностави и унапреди процес креирања визуелно привлачних и функционалних апликација. *Compose* значајно смањује количину кода потребног за креирање UI у поређењу са XML базираним приступом коришћеним у традиционалном Андроид апликацијама. Компоненте графичког корисничког интерфејса дефинишу се и конфигуришу користећи компактни и читљиви програмски језик Котлин, што поједностављује развој, тестирање и одржавање апликација.

Jetpack Compose користи декларативни приступ уместо традиционалног императивног приступа. Уместо детаљног описивања редоследа операција које изграђују UI, програмери описују како треба да изгледа кориснички интерфејс у одређеном стању апликације. Додатно, *Compose* подржава реактивно ажурирање UI. Промене у стању апликације аутоматски покрећу поновно израчунавање и рендеровање компоненти корисничког интерфејса, чиме се избегавају замке повезане са ручним ажурирањем UI и побољшава одзивност апликације. Додатно, само компоненте које захтевају промену бивају пононо рендероване, чиме се повећавају перформансе апликације. Још једна карактеристика овог радног оквира јесте акценат на креирање модуларних, поново употребљивих компоненти које могу бити лако комбиноване и конфигурисане. Овај приступ не само да смањује дуплирање кода већ и омогућава бољу организацију и одржавање кода. *Compose* је интегрисан са осталим *Jetpack* библиотекама, као што су *LiveData, Navigation* и *ViewModel*, што омогућава једноставно управљање животним циклусом и навигацијом између различитих екрана [15]. На слици 3.6. приказан је лого библиотеке *Jetpack Compose.*

Слика 3.6. Лого библиотеке Jetpack Compose [15]

4. РЕАЛИЗАЦИЈА СИСТЕМА

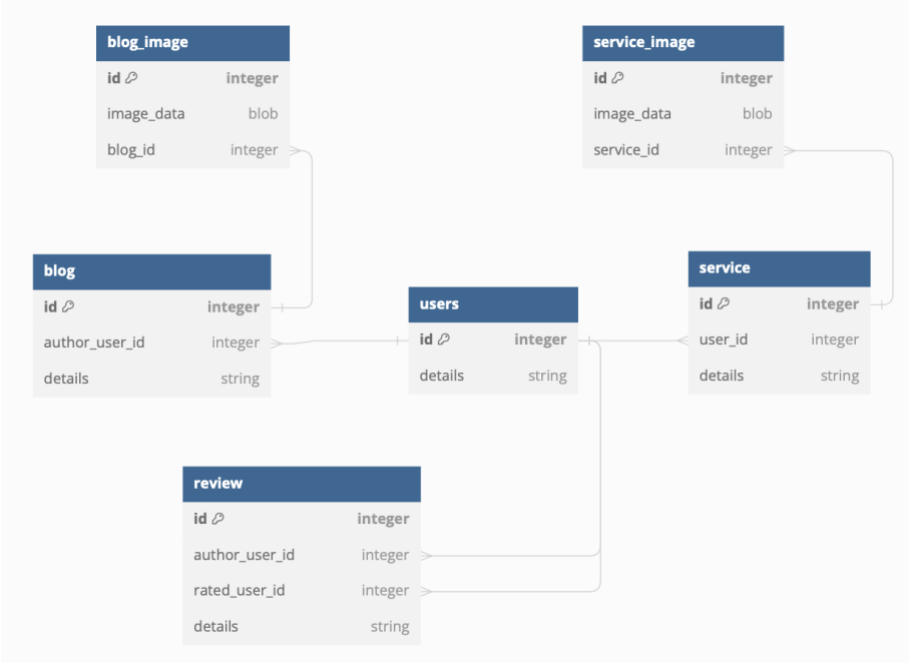
У овом поглављу објашњена је реализација предложеног решења за описани проблем користећи технологије описане у поглављу три. Систем је дизајниран и имплементиран тако да испуни све функционалне и нефункционалне захтеве забележене у табели 1. Апликација је реализована као део већег система који решава проблем проналаска одговарајућег мајстора на веб сајту и Андроид мобилној апликацији. Израда овог рада обухвата имплементацију HTTP бекенд сервера и мобилне апликације. Главне ставке система које је било потребно реализовати су складиштење и миграција података, архитектура бекенд дела система, аутентикација и ауторизација корисника, имплементација бекенд дела система, архитектура мобилне апликације и имплементација мобилне апликације.

Најпре, било је потребно дизајнирати структуру базе података и креирати неопходне табеле и релације за складиштење података апликације. Након тога, одрађен је дизајн архитектуре бекенд система коју ће пратити имплементација. Пре почетка имплементације, било је потребно решити проблем ауторизације и аутентикације корисника. Затим, дизајниран је API интерфејс преко којег комуницирају бекенд и мобилна апликација. Коначно, имплементиран је бекенд API сервер који опслужује захтеве корисника мобилне апликације. Пре почетка имплементације мобилне апликације, било је потребно дизајнирати архитектуру система. Упоредо са имплементацијом бекенд сервера, а након дизајна, имплементирана је и мобилна апликација. Апликација је имплементирана итеративним приступом – једна по једна функционалност имплементирана је упоредо на серверској и клијентској (мобилној) страни.

4.1. Складиштење и миграција података

За складиштење података апликације коришћена је *MySQL* релациона база података. Најпре, одрађен је дизајн структуре базе према функционалним захтевима описаним у поглављу два. Креиран је корисник са специјалним привилегијама за администрацију базе и направљене су табеле које представљају ентите апликације.

Табела *user* представља регистроване кориснике система. Сваки корисник има следеће атрибуте: име и презиме, јединствену адресу електронске поште и шифру. Ради сигурности корисничких података, било је неопходно енкриптовати шифру пре чувања у базу. Табела *blog* представља корисничке приче које регистровани корисници могу да објаве. Садржи страни кључ аутора из табеле *user*, као и остале описне детаље које корисник уноси при креацији блога. Табела *service* представља услуге које одређени мајстор пружа. Садржи страни кључ из табеле *user* који идентификује мајстора, остале детаље везане за услугу, као и категорију услуге. Један мајстор не може креирати више услуга исте категорије. Релација између корисника и блога односно услуге је „*one-to-many*”, односно један корисник може креирати више приче/услуга. Табела *review* представља рецензије корисника и садржи стране кључеве корисника који је дао и који је примио оцену. Табеле *blog_image* и *service_image* представљају слике које одговарају специфичном блогу/услузи. Слике су претходно енкодоване помоћу *Base64*, и чувају се у бинарном облику као тип „*blob*”. На слици 4.1. приказана је архитектура табела базе података.



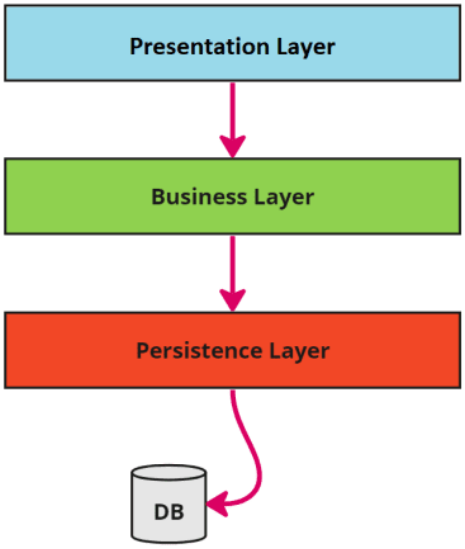
Слика 4.1. Архитектура табела базе података

За верзионисање и управљање миграцијама базе података коришћен је *Flyway* алат, који омогућава да се промене у бази примене аутоматски и на контролисан начин.

Миграције се дефинисане као верзионисане SQL скрипте које се аутоматски покрећу у исправном редоследу приликом покретања сервера, обезбеђујући да база остане синхронизована са апликацијом. Додатно, *Flyway* прати историју свих миграција, чиме олакшава преглед промена и отклањање проблема. Овакав приступ омогућава тимовима да лакше управљају променама базе у развојном процесу и у продукционом окружењу [16].

4.2. Архитектура бекенд дела система

Дизајн система је врло битна ставка у дефинисању производа и његове архитектуре. Неопходно је да дизајн, интерфејси, класе, подаци и модули задовоље системске захтеве. Због тога, добар дизајн система представља кључ за оптималан развој софтвера. Како се бекенд део система ослања на Spring Boot радни оквир, одлучено је да имплементација прати препоручену тролојну архитектуру. Тролојна архитектура структурира апликацију на три слоја: за презентацију, за сервис и за приступ подацима. Сваки слој представља другачији ниво и тип апстракције софтвера. Њихова улога јесте рашчлањивање апликације на мање делове које је лакше развијати и одржавати. Када корисник пошаље захтев серверу, ток података ће проћи кроз сва три слоја [17]. Кључно правило у тролојној архитектури јесте да сваки слој има приступ само једном слоју испод себе, приказано на слици 4.2.1.



Слика 4.2.1. Приказ тролојне архитектура [17]

Презентациони слој представља највиши слој и садржи интерфејс за комуникацију са клијентима. Садржи искључиво једноставну логику потребну за конструкцију одговора. У њему су дефинисани контролери који представљају улазне тачке (*endpoint*) система. Додатно, контролери врше валидацију и припрему улазних података за сервисни слој.

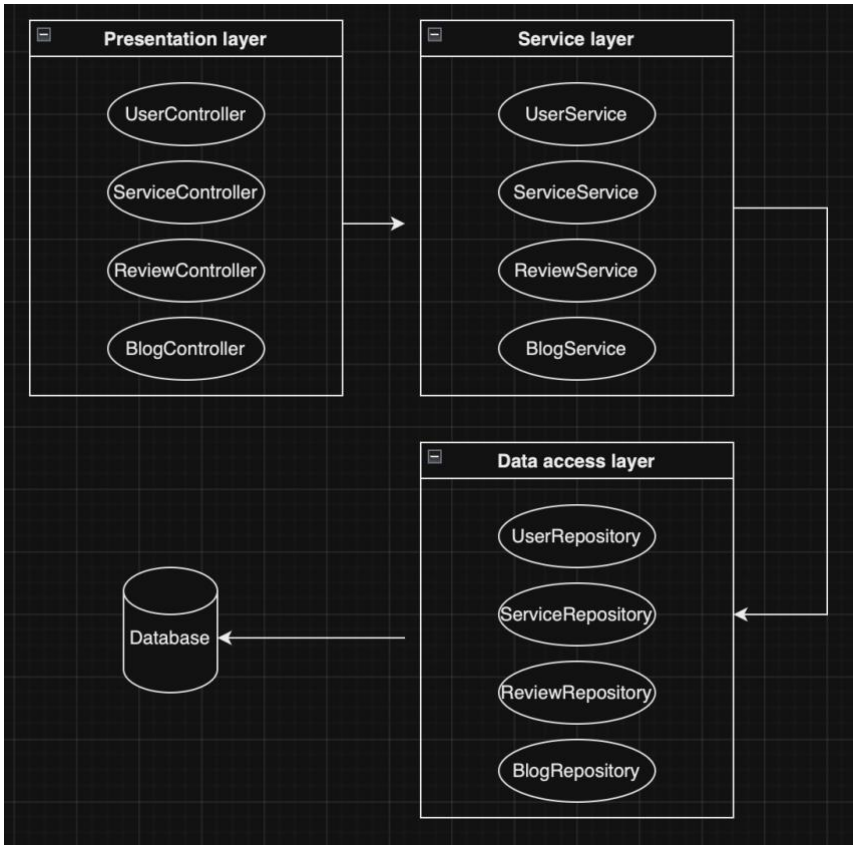
Сервисни слој налази се између презентационог слоја и слоја за приступ подацима. Садржи сва пословна правила и алгоритме потребне за функционисање апликације - примењује пословна правила пре него што се подаци проследе презентационом слоју или слоју за приступ подацима. Одвајање пословне логике у сервисни слој омогућава лакшу измену или рефакторинг кода, јер се промене у пословној логици не одражавају директно на презентациони или слој за приступ подацима. Ово побољшава тестабилност, одрживост и читљивост кода.

Слој за приступ подацима управља интеракцијом са базом података и другим изворима података. Одговоран за све операције читања, уписа, ажурирања и брисања података у бази података. Садржи методе које комуницирају директно са базом, користећи ORM (*Object-Relational Mapping*) алат *Hibernate*. Репозиторијумски слој апстрахује сложеност базе података од осталих делова апликације. На овај начин виши слојеви, попут сервисног и презентационог, не морају да знају специфичности базе података (попут SQL синтаксе или детаља о повезивању), што побољшава читљивост и одрживост кода. Централизовање приступа подацима у овом слоју омогућава поновну употребу истих метода и логике у различитим деловима апликације. То смањује дуплирање кода и олакшава одржавање, јер се промене у логики приступа подацима морају направити само на једном месту. Најважније, централизовани приступ обезбеђује универзалну истину.

Трослојна архитектура побољшава одрживост, флексибилност и скалабилност апликација. Одвајање одговорности олакшава развој, тестирање и одржавање јер је свака компонента фокусирана на специфичан аспект апликације. Слојевита структура омогућава лакше јединично тестирање (*unit testing*) и интеграционо тестирање (*integration testing*). Сваки слој може бити тестиран независно од других, што поједностављује процес тестирања и побољшава укупни квалитет софтвера.

Додатно, олакшава се скалирање апликације. На пример, презентациони слој се може лако заменити или ажурирати да подржава различите уређаје или интерфејсе, док се сервисни и репозиторијумски слојеви могу оптимизовати за перформансе или променити како би подржали нове технологије база података или API стандарде.

У овом систему сваки слој подељен је на четири компоненте према принципу поделе одговорности (*separation of concerns*). На овај начин повећава се модуларност система, чиме се олакшава имплементација и одржавање. Прва компонента обрађује захтеве везане за корисничке налоге, као што су регистрација и пријава на систем, или дохватање података о одређеном кориснику. Друга компонента опслужује корисничке захтеве везане за услуге мајстора. Садржи методе за манипулацију претраге мајстора, као што су дохватање мајстора на основу изабране категорије или додавање нове услуге. Трећа компонента имплементира функционалне захтеве за реценцију услуга мајстора. Садржи методе за додавање рецензије и дохватање свих рецензија за одређеног мајстора. Последња компонента обрађује захеве везане за корисничке приче (блогове). Садржи методе за додавање, брисање и дохватање блогова. На слици 4.2.2. приказана је архитектура бекенд дела система.



Слика 4.2.2. Архитектура бекенд дела система

4.3. Аутентикација и ауторизација корисника

Главни нефункционални захтев система јесте обезбеђивање интегритета мајстора и рецензије корисника. Интегритет се може осигурати провером права и привилегија корисничких акција. Наиме, само корисници пријављени на систем имају могућност остављања рецензије. Додатно, потребно је обезбедити осетљиве податке корисника од илегалних и злонамерних акција других корисника. Аутентикација и ауторизација су кључни аспекти безбедности у веб апликацијама, који осигуравају да само овлашћени корисници имају приступ осетљивим подацима и функционалностима.

Аутентикација је процес провере идентитета корисника. Она осигурава да корисник који покушава да приступи систему заиста јесте онај за кога се представља. Најчешће укључује унос корисничког имена и лозинке, али може обухватати и друге методе као што су биометријски подаци, токени засновани на времену (TOTP), и двофакторска аутентикација (2FA). Сврха аутентикације је да спречи неовлашћене појединце од приступа систему, минимизирајући ризик од крађе идентитета и неовлашћеног приступа поверљивим информацијама.

Ауторизација долази након успешне аутентикације и одређује којим ресурсима и акцијама корисник има приступ. Док аутентикација потврђује идентитет корисника, ауторизација дефинише дозвољене акције корисника. Ауторизација се често реализује путем корисничких улога и привилегија, где се корисницима додељују одређени нивои приступа на основу њихове улоге у систему (нпр. администратори, уредници, корисници). Овај механизам спречава злоупотребу система и осигурава да се корисничке акције конструктивно ограниче ради заштите података и функционалности.

Комбинација аутентикације и ауторизације омогућава свеобухватан приступ контроли приступа у веб и мобилним апликацијама. Ови процеси су критични за одржавање интегритета, поверљивости и безбедности информационих система, чиме се штите поверљиви подаци и одржава поверење корисника у апликацију. Аутентикација и ауторизација у овом систему реализоване су користећи *JWT Authentication Filter* компоненте доступне у *Spring Boot* радном оквиру. Функционише као филтер, односно као посредник који пресреће HTTP захтеве пре него што они допру до улазних тачака апликације (контролера), и проверава валидност JWT (*JSON Web Token*) присутног у заглављу захтева.

Аутентикација је имплементирана у три корака. Најпре *JwtAuthFilter* пресреће долазни HTTP захтев и екстрахује JWT из заглавља захтева, конкретно из „*Authorization*" заглавља. Типично, заглавље изгледа овако: *Authorization: Bearer <JWT>*, где је *<JWT>* сам токен. Потом, проверава се валидност извученог токена. Процес укључује проверу потписа токена, проверу времена истека (*expiration time*) и других обезбеђених атрибута. Ако неки од наведених корака не буде успешан, клијенту се враћа HTTP одговор са статусним кодом 403 који кориснику сигнализира да је сервер примио захтев али је одбио да га обради [18].

Ако је токен валидан, филтер екстрахује корисничке информације које служе за аутентификацију корисника, конкретно *email*. На основу ових информација, филтер може поставити идентификационе податке корисника у специфичан објекат за аутентификацију (*UserAuthProvider*). Након успешне аутентификације врши се ауторизација, ондосно филтер проверава да ли корисник има потребне привилегије или улоге за извршење захтева. Коначно, ако сви кораци успешни, захтев се прослеђује улазној тачки сервера (контролеру) ради даље обраде. На слици 4.3., приказан је део кода задужен за валидацију токена.

```
public Authentication validateToken(String token) {  1 usage   krsnik
    Algorithm algorithm = Algorithm.HMAC256(secretKey);
    JWTVerifier verifier = JWT.require((algorithm)).build();
    DecodedJWT decodeJWT = verifier.verify(token);
    User user = User.builder()
            .email(decodeJWT.getIssuer())
            .id(decodeJWT.getClaim( s: "id").asLong())
            .firstName(decodeJWT.getClaim( s: "firstName").asString())
            .lastName(decodeJWT.getClaim( s: "lastName").asString())
            .status(UserStatus.valueOf(decodeJWT.getClaim( s: "status").asString()))
            .build();
    return new UsernamePasswordAuthenticationToken(user,  credentials: null, Collections.emptyList());
}
```

Слика 4.3. Део кода за валидацију JWT токена

JWT токен креира се и прослеђује кориснику приликом успешне регистрације или пријаве на систем. Садржи све корисничке податке неопходне за аутентикацију и ауторизацију, као што су јединствени идентификатор корисника у бази, електронска пошта, име, презиме и тип корисника (регуларни корисник или администратор). Сваки токен садржи поље „*expiresAt*”, који представља време истека валидности токена. Коначно, токен се пре слања клијенту енкриптује *HMAC256* алгоритмом, како би се осигурала безбедност осетљивих корисничких података које садржи.

4.4. Имплементација бекенд дела система

За реализацију бекенд дела система, односно HTTP сервера, било је потребно имплементирати три слоја описана у поглављу 4.2. Слој за приступ подацима имплементиран је користећи *JpaRepository* интерфејс доступан у оквиру *SpringDataJPA*, који обезбеђује напредне функционалности за рад са релативним базама података користећи *Java Persistence API* (JPA). *JpaRepository* обезбеђује основне *CRUD* методе и уклања потребу за ручним имплементирањем основних операција за рад са базом података, као што су *save(), findById(), findAll(), count(), deleteById(), и deleteAll()*. Додатно, подржава и креирање прилагођених упита користећи методе које следе договорене конвенције о именовању (*query methods*). На пример, креирање методе *findByEmail(String email)* аутоматски генерише SQL упит заснован на имену методе. *JpaRepository* такође подржава трансакције, коришћењем аннотација као што су *@Transactional* за означавање метода које би требало извршити у склопу трансакције. То осигурава конзистентност и интегритет података у случају сложених операција или непредвиђених грешака. На слици 4.4.1, приказана је имплементација репозиторијума.

```
public interface BlogRepository extends JpaRepository<BlogEntity, Long> { }  3 usages  new *
public interface ReviewRepository extends JpaRepository<ReviewEntity, Long> {  2 usages  krsnik *  1 related problem
    List<ReviewEntity> findByRatedUser(UserEntity rated_user_id);  1 usage  new *
    List<ReviewEntity> findByCreatorUser(UserEntity creator_user_id);  1 usage  new *
    Boolean existsByCreatorUserIdAndRatedUserId(Long creator_user_id, Long rated_user_id);  1 usage  new *
}
public interface ServiceRepository extends JpaRepository<ServiceEntity, Long> {  2 usages  new *  1 related problem
    List<ServiceEntity> findByL1Category(String l1Category);  1 usage  new *
    List<ServiceEntity> findByL2Category(String l2Category);  1 usage  new *
    List<ServiceEntity> findByUserId(Long userId);  1 usage  new *
    boolean existsByUserIdAndL2Category(Long userId, String l2Category);  1 usage  new *
}
public interface UserRepository extends JpaRepository<UserEntity, Long> {  2 usages  new *  1 related problem
    Optional<UserEntity> findByEmail(String email);  3 usages  new *  5 related problems
}
```

Слика 4.4.1. Део кода за имплементацију репозиторијума

Ентитетски објекти имплементирани су као JPA класе користећи анотацију *@Entity*, која омогућава објектно-релационо мапирање (ORM). На овај начин *Spring Data JPA* аутоматски креира SQL упите потребне за рад са датом табелом. Додатно, ентитетске класе користе библиотеку *Lombok* која аутоматски генерише инспекторе и мутаторе атрибута класе, чиме се смањује количина шаблонског кода.

Слој за пословну логику имплементиран је као скуп обичних Јава класа. Садржи пословну логику и методе за трансформацију ентита базе података у доменске објекте апликације (DTO – *Data Transfer Objects*) дефинисане API интерфејсом и обрнуто. У ову сврху, користи се *MapStruct* библиотека за мапирање у Јави приказана на слици 4.4.2. *MapStruct* аутоматски генерише имплементације метода за трансформацију између ентитетских и доменских објеката.

```
@Mapper(componentModel = "spring")  10 usages  1 implementation  krsnik *
public interface UserMapper {
  User toDomain(UserEntity entity);  1 implementation  krsnik
  UserEntity toEntity(User user);  1 implementation  krsnik
  UserEntity signUpRequestToUser(SignUpRequest signUp);  1 usage  1 implementation  krsnik
}
```

Слика 4.4.2. Део кода за мапирање ентитетских у доменске класе

Презентациони слој имплементиран је као скуп контролера који дефинишу *RESTful* API за комуникацију са клијентом. Користе анотације *Spring* библиотеке за руковање HTTP захтевима и дефинисање рута. Сваки контролер одговоран је за одређени део система, на пример, *UserController* управља корисничким операцијама као што су пријава, регистрација и добијање корисничких података. Улазне тачке система (*endpoint*) дефинисане су као методе класе контролера и описане користећи *Spring* анотације као на слици 4.4.3.

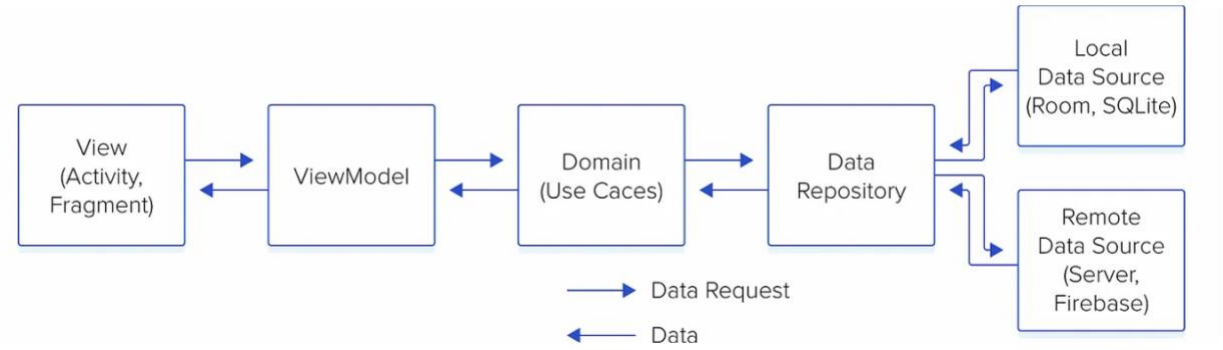
```
@RestController  krsnik *
@RequiredArgsConstructor
@RequestMapping(value = "/api/v1")
public class UserController {

  private final UserService userService;  1 usage
  private final UserAuthProvider userAuthProvider;  1 usage

  @PostMapping("/login")  no usages  krsnik
  public ResponseEntity<User> login(@RequestBody CredentialsRequest credentials) {
    var user = userService.login(credentials);
    user.setToken(userAuthProvider.createToken(user));
    return ResponseEntity.ok(user);
  }

  ...
}
```

Слика 4.4.3. Део кода за имплементацију контролера

4.5. Архитектура мобилне апликације

За архитектуру мобилне Android апликације изабран је MVVM образац, препоручен од стране доменских експерата. MVVM (*Model-View-ViewModel*) је архитектонски образац који се примарно користи у развоју графичког корисничког интерфејса, нарочито Андроид апликација. MVVM настоји да раздваји кориснички интерфејс од пословне логике и података апликације. *ViewModel* служи као посредник између приказа и модела, излагајући податке и команде за које се приказ везује. Типичан ток података у MVVM апликацијама приказан је на слици 4.5. Циљ је раздвајање одговорности, чиме се повећава тестабилност, одрживост и флексибилност апликације. Избегавање снажних зависности између корисничког интерфејса, радног оквира и извора података омогућава измену зависности без промене комплетне структуре пројекта. MVVM апликација поседује следећа својства:

- Реактивна и слојевита архитектура
- Једносмерни ток података у свим слојевима (UDF)
- Сепарација одговорности између слојева
- Приказ осликава податке апликације
- Асинхрони позиви
- *Dependency Injection* образац



Слика 4.5. Ток података у мобилној апликацији [19]

Модел представља извор података и пословну логику апликације. Може укључивати локалну базу података, репозиторијуме, или веб сервисе. У контексту Андроид апликације, модел не треба директно поседовати ниједну информацију о приказу (*View*) или *ViewModel*. Главна функција јесте рад са подацима, било да је у питању преузимање података са сервера, локалне базе података или обављање интерних прорачуна. Представља универзалну истину о подацима апликације.

Приказ (*View*) је одговоран за приказивање података кориснику и мапирање корисничког уноса у интеракције са *ViewModel* објектом. У Андроиду, приказ обухвата активности (*Activity*), фрагменте (*Fragment*), и остале UI компоненте као што су виџети (*Widgets*). Приказ је одговоран само за приказивање података и прикупљање корисничког уноса. Не треба да садржи пословну логику или било коју врсту обраде података. Преко *StateFlow* објеката, приказ посматра промене у *ViewModel* објекту и ажурира UI компоненте када до њих дође.

ViewModel служи као мост између Приказа и Модела. Садржи логику презентације и припрема податке који ће бити приказани у Приказу. Одговоран је за обраду корисничких акција из приказа и за позивање метода из Модела за обраду података. Не би требало да зна ништа о специфичном Приказу који га користи, што олакшава тестирање. *ViewModel* класа у Андроиду специјално је дизајнирана да преживи промене конфигурације, као што је ротација екрана.

MVVM у Андроид апликацији функционише на следећи начин. Корисник интерагује са приказом (нпр. кликне на дугме или унесе текст у поље). Приказ прослеђује кориснички унос *ViewModel* објекту путем позива метода *ViewModel* класе. *ViewModel* обрађује овај унос и ажурира податке у Моделу. Ово укључује HTTP позив ка серверском делу систем преко репозиторијума. Када се подаци ажурирају у Моделу, *ViewModel* добија ажуриране податке и припрема их за приказ. *ViewModel* користи *StateFlow* објекат да би обавестио приказ о променама података континуирано. Приказ „посматра“ промене у *StateFlow* објектима из *ViewModel* класе и ажурира кориснички интерфејс у складу са новим подацима.

Иако је MVVM образаз препоручљив за имплементацију Андроид апликација, поседује и неке негативне стране. У пројектима малих размера, где апликација има само неколико екрана и једноставну логику, MVVM може изгледати претерано сложено и

захтевати више кода него што је потребно. Додатно, увођење MVVM-а може резултирати писањем доста шаблонског кода (*boilerplate code*) [19].

4.6. Имплементација мобилне апликације

Мобилна апликација реализована је у програмском језику Котлин. Било је потребно имплементирати два независна слоја описана у претходном поглављу – слој за приступ апликативним подацима и слој за приказ графичких компоненти.

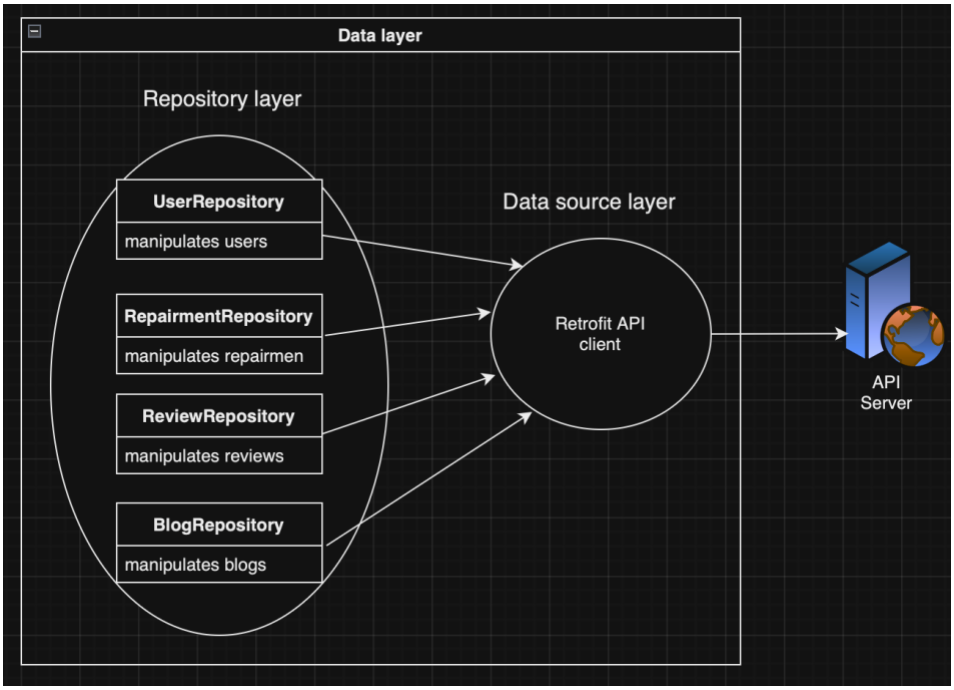
Слој за приступ апликативним подацима подељен је у две компоненте. Прва компонента представља извор апликативних података и извози интерфејс за њихов приступ. Током израде овог рада, имплементиран је само један извор података - серверски део система. За мрежну комуникацију са серверским делом система коришћен је *Retrofit*, библиотека развијена од стране компаније *Square*. Омогућава једноставну и ефикасну конструкцију HTTP захтева и обраду HTTP одговора. *Retrofit* претвара HTTP API у Котлин интерфејс и омогућава лако позивање *RESTful* веб сервиса. Користи анотације над интерфејсима за дефинисање HTTP метода и њихових путања, чиме омогућава једноставну и читљиву синтаксу. Додатно, подржава аутоматску конверзију JSON одговора у Котлин објекте коришћењем библиотека као што су *Gson, Moshi* или *Kotlin Serialization*. На овај начин избегава се ручно парсирање JSON фајлова. *Retrofit* се лако интегрише са Котлин корутинама, омогућавајући реактивно и асинхроно руковање захтевима и одговорима неопходно за MVVM пројекат. Подржава интерцепторе за пресретање и модификацију захтева који су коришћени за додавање JWT токена у заглавље. На слици 4.6.1. приказана је имплементација API клијента.

Слика 4.6.1. Део кода за имплементацију API клијента

Друга компонента слоја за приступ апликативним подацима пружа скуп интерфејса доступних слоју приказа (конкретније ViewModel објектима) за манипулацију подацима. Ова компонента имплементирана је према *Repository* пројектном узорку. *Repository* је дизајн образац који делује као посредник између апликације и једног или више извора података (као што су базе података, веб сервиси, или кеш меморија). Састоји се из скупа класа одговорних за пружање података остатку апликације. Представљају централизовано место за промену података и одговорне су за разрешење конфликата између више извора података. Апстрахују изворе података од остатка апликације, чиме се омогућава промена извора података без потребе за поновним превођењем остатка система.

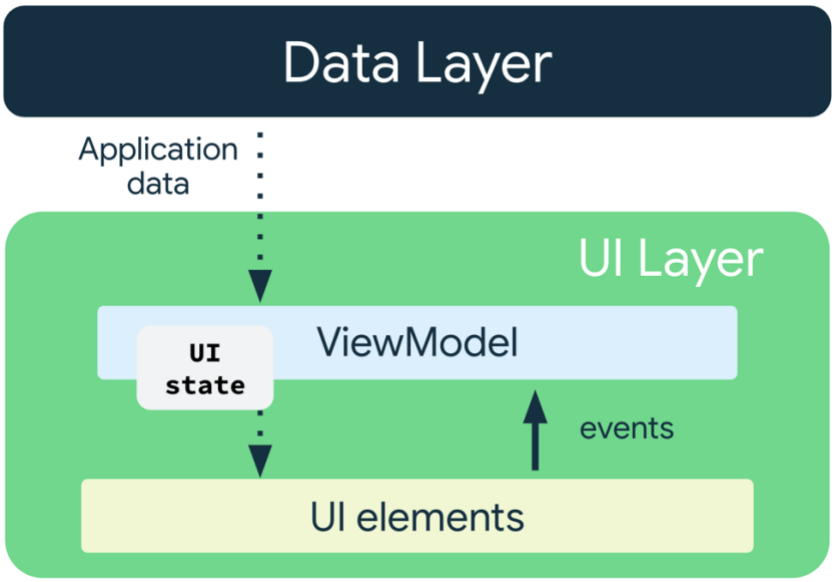
У овом систему, имплементирана су четири „репозиторијума”, која пружају методе за манипулацију одвојеним деловима система. То су *UserRepository* која обрађује захтеве везане за кориснике система, *RepairmentRepository* за манипулацију претраге мајстора - дохватање мајстора на основу изабране категорије или додавање нове услуге. Трећи репозиторијум *ReviewRepository* управља реценцијама услуга мајстора. Садржи методе за додавање рецензије и дохватање свих рецензија за одређеног мајстора. Последња компонента обрађује захеве везане за корисничке приче (блогове). Садржи методе за додавање, брисање и дохватање блогова. Последњи репозиторијум *BlogRepository* обрађује захеве везане за корисничке приче (блогове). Садржи методе за додавање, брисање и дохватање блогова.

На слици 4.6.2. приказана је архитектура слоја за приступ подацима мобилне апликације.



Слика 4.6.2. Архитектура мобилне апликације

Улога слоја за приказ јесте да прикаже апликативне податке на екрану и буде примарна тачка интеракције са корисником. Приликом промене података, било корисничком акцијом (нпр. притисак дугмета) или екстерним сигналом (нпр. одговор на захтев са сервера), кориснички интерфејс се ажурира тако да осликава те промене. Концептуално, кориснички интерфејс представља визуелну презентацију стања апликације примљену од слоја за приступ подацима. Ипак, ти апликациони подаци су углавном у другом формату од оног потребног за конструкцију корисничког интерфејса. На пример, можда је само део тих података неопходан за приказ или је потребно искомбиновати више извора података за конструкцију података за приказ. Слој за приказ представља цевовод за конверзију промена апликативних података у форму неопходну за њихов приказ. Састоји се из графичких елемената и њихових стања. На слици 4.6.3 приказана је архитектура слоја приказа.



Слика 4.6.3. Архитектура слоја приказа [20]

Графички елементи имплементирани су користећи *Jetpack Compose* библиотеку. Подељени су у пет пакета који одговарају њиховим улогама. Четири пакета одговарају описаним пакетима у слоју за приступ апликативним подацима. Пети пакет садржи заједничке компоненте које се користе од стране више других пакета, као што су *Base64Image* за приказ слике из *Base64* формата, или *ClippedIconButton* који представља елипсоидно дугме. Кориснички интерфејс састоји се од градивних елемената (функција) које се комбинују приликом конструкције одабраног екрана. Градивни елементи дефинисани су аргументима функције.

Навигација између екрана имплементирана је помоћу *Jetpack Navigation* библиотеке. Попут *Compose* библиотеке, *Navigation* је библиотека за Андроид која омогућава једноставно и декларативно управљање навигацијом унутар апликације, укључујући прелазе између екрана, навигационе акције и дубоко умрежавање(*deep linking*). Навигација се имплементира у два корака: дефинисање навигационих рута (*NavGraph*) и дефинисање навигације између навигационих рута приликом корисничких акција (*NavHost*).

Навигационе руте дефинишу се за сваки екран навигације, односно за сваку графичку компоненту највишег нивоа. На слици 4.6.4. приказана је дефиниција рута неких од екрана.

```
object NavigationRoute {
    const val PROFILE = "Profile"
    const val BLOG = "Blog"
    ...
}
```

Слика 4.6.4. Део кода за дефиницију навгиационих рута

Затим потребно их је повезати преко навигационог контролера (*NavHostController* класе) и креирати навигациони граф. Ова класа омогућава програмерима да дефинишу и контролишу прелазе између различитих графичких елемената на декларативан начин, чувајући стање и хијерархију навигације током живота апликације. На слици 4.6.5. приказан је део кода за навигацију између екрана.

```
class AppNavigationActions(val navController: NavHostController) {
    new *
    fun navigateToBlogScreen(blogId: Int) {
        navController.navigate("${NavigationRoute.BLOG}/$blogId")
    }
    ...
}
```

Слика 4.6.5. Део кода за навигацију помоћу контролера

Коначно, потребно је имплементирати *NavHost* класу која садржи навигационе руте и навигациони контролер. Представља контејнер који дефинише на који начин се креирају све руте апликације. *NavHost* заједно са *NavController* класом омогућава динамичко додавање, уклањање и модификацију рута, при чему се дестинације аутоматски ажурирају у складу са дефинисаном навигацијом. Омогућава повезивање *ViewModel* класа са навигационим графом ради дељења података везаних за кориснички интерфејс између дестинација унутар тог графа. Додатно, аутоматски подржава акције унапред и уназад присутне на сваком телефону. На слици 4.6.6. приказана је имплементација *NavHost* класе.

```
@Composable
fun AppNavHost(
    navigationActions: AppNavigationActions,
    modifier: Modifier = Modifier
) {
    NavHost(
        navController = navigationActions.navController,
        startDestination = NavigationRoute.PROFILE
    ) {
        composable(
            route = "${NavigationRoute.BLOG}/{id}",
            arguments = listOf(
                navArgument(name = "id") {
                    type = NavType.IntType
                }
            )
        ) {
            BlogScreen(
                navigateOnBlogDeleted = { navigationActions.navController.popBackStack() },
                modifier = modifier
            )
        }
    }
    ...
}
```

Слика 4.6.6. Део кода за имплементацију NavHost-а

Стање корисничког интерфејса (UI *state*) је атрибут који описује кориснички интерфејс. Имплементиран је у облику простих Котлин класа за графичке елементе екрана, и сложених *ViewModel* класа за комплетан екран. Класе графичких елемената имплементиране су користећи *Kotlin Data Class* механизам, који аутоматски генерише инспекторе и мутаторе неопходних атрибута. *ViewModel* класе наслеђују *ViewModel* апстрактну класу имплементирану у *androidx.lifecycle* пакету. Ова класа омогућава управљање и чување података везаних за кориснички интерфејс на начин који је отпоран на промене конфигурације, као што су ротација екрана или промена језика. На слици 4.6.7. приказана је имплементација класе за чување стања екрана.

```
data class BlogUiState(
    val blog: Blog? = null,
    val canDeleteBlog: Boolean = false
)
Andrej Jokic *
@HiltViewModel
class BlogViewModel @Inject constructor(
    private val blogsRepository: BlogsRepository,
    private val userRepository: UserRepository,
    private val savedStateHandle: SavedStateHandle,
): ViewModel() {
    // Get blogId set by compose navigation
    private val blogId: Int = savedStateHandle.get<Int>("id")!!

    private val _uiState = MutableStateFlow(BlogUiState())
    val uiState: StateFlow<BlogUiState> = _uiState
```

Слика 4.6.7. Имплементација класа за чување стања корисничког интерефејса

За разрешавање зависности између компонената коришћен *је Dependency Injection* (DI) механизам. DI омогућава класама да декларишу своје зависности без њихове конструкције. Током извршавања програма, друга класа је одговорна за обезбеђивање и прослеђивање ових зависности. На овај начин, омогућено је скалирање кода без потребе за дупликацијом или додатном комплексности. Штавише, пружају могућност једноставне промене између тест и продукционих имплементација [20].

Конкретно, коришћена је *Hilt* библиотека према препоруци компаније *Google*. *Hilt* обилази стабло зависности, аутоматски креира објекте и прослеђује зависности декларисане за време компилације. Најпре, неопходно је *Hilt* библиотеци пријавити класу коју нека зависност очекује и пружити начин за њену конструкцију. На слици испод видимо пријаву репозиторијума који *ViewModel* конзумира. На слици 4.6.8. приказан је део кода за пријаву репотизоријума *BlogRepository Hilt* библиотеци.

```
@Module
@InstallIn(SingletonComponent::class)
object RepositoryModule {
    Andrej Jokic
    @Provides
    @Singleton
    fun provideBlogsRepository(apiService: ApiService): BlogsRepository =
        BlogsRepositoryImpl(apiService)
    ...
}
```

Слика 4.6.8. Део кода за пријаву репозиторијума *Hilt* библиотеци

Сваки ViewModel такође је потребно пријавити *Hilt* библиотеци. Ово се постиже специјалном анотацијом *@HiltViewModel*. Након тога, могуће је декларисати зависности које ће *Hilt* аутоматски креирати и убацити у објекат током извршавања програма, као на слици 4.6.9.

```
Andrej Jokic *
@HiltViewModel
class BlogsViewModel @Inject constructor(
    private val blogsRepository: BlogsRepository,
    private val userRepository: UserRepository
): ViewModel() {
    ...
}
```

Слика 4.6.9. Имплементација ViewModel-а преко Hilt библиотеке

5. Опис рада система

У овом поглављу дефинисано је корисничко упуство које покрива главне функционалности система. Кроз поглавље, уз помоћ слика графичког корисничког интерфејса, биће описан рад система. Апликација је подељена у три компоненте између којих је могућа навигација коришћењем бочног менија. Први екран омогућава пријаву и регистрацију нових корисника на систем, односно инспекцију свог профила и својих акција за пријављене кориснике. Други екран обухвата главну функционалност система – претрагу и рецензије мајстора. Садржи напредне могућности претраге и омогућава свим корисницима инспекције базе мајстора. Трећи екран садржи корисничке приче објављене од стране мајстора. Додатно, пријављени корисници имају могућност додавања нове приче.

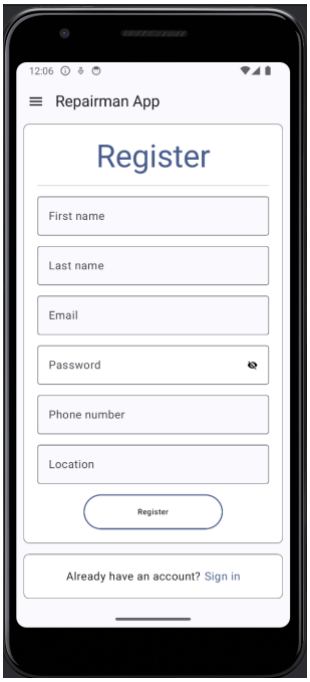
5.1. Навигација

Апликација садржи бочни мени који омогућава корисницима навигацију између главних екрана и приказан је на слици 5.1. Изабрана опција обојена је светло плавом бојом, чиме се корисницима указује на њихову тренутну позицију.

Слика 5.1. Навигација кроз главне екране апликације

5.2. Пријава и регистрација на систем

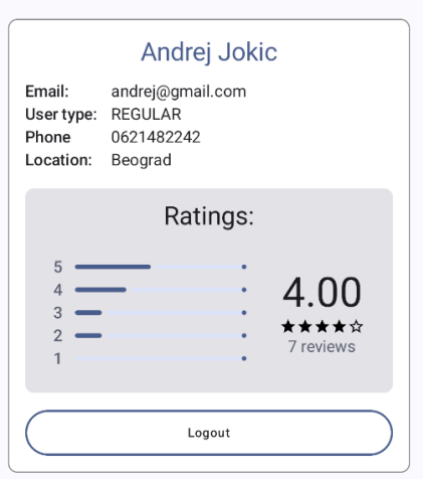
На екрану профилне странице, непријављени корисници имају могућност пријаве, односно регистрације на систем. Приликом пријаве, уносе се email и шифра подешени током регистрације. За успешну регистрацију неопходно је унети име, презиме, *email*, шифру, број мобилног телефона и пребивалиште као на слици 5.2. *Email* мора да испуњава познати образац за електронске адресе и буде јединствен у систему. Систем пријављује грешку ако је унети мејл заузет. Додатно, шифра мора да испуњава одређене критеријуме, као што су минимална дужина, присуство слова, бројева и специјалних карактера.



Слика 5.2. Регистрација корисника на систем

5.3. Преглед профила пријављеног корисника

На екрану профилне странице, пријављени корисници имају увид у своје личне податке и извршене акције. Страница је подељена у четири секције. Прва секција приказује личне податке корисника и оцене забележене од рецензија других корисника. Оцене су представљене на два начина: излистане све оцене и просечна оцена. Додатно, екран садржи опцију за одјављивање са система. Прва секција приказана је на слици 5.3.1.

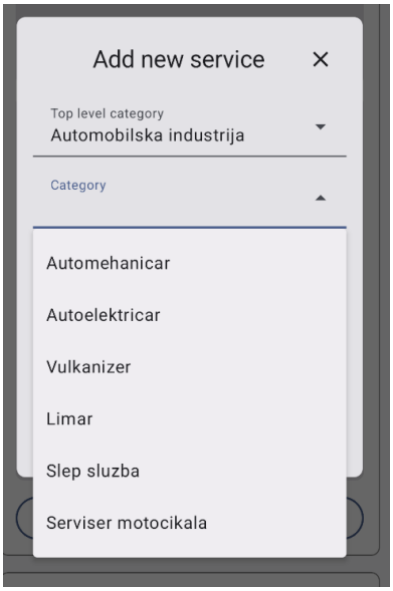


Слика 5.3.1. Секција о подацима пријављеног корисника

Друга секција приказује све регистроване услуге корисника и приказана је на слици 5.3.2. Свака услуга садржи категорију, опис и опционио, фотографије претходних радова. Додатно, сваку услугу могуће је обрисати кликом на иконицу канте у доњем десном углу.

Слика 5.3.2. Секција за приказ услуга мајстора

На дну секције налази се дугме за додавање нове услуге. Притиском на дугме отвара се дијалог приказан на слици 5.3.3., за попуњавање информација о услузи. Корисник прво бира категорију услуге највишег нивоа преко падајућег менија. Затим, излиставају се услуге у тој категорији од којих корисник бира једну помоћу падајућег менија. Исту услугу није могуће више пута регистровати и оне нису поновно излистане. Испод се налази поље за унос описа о услузи који треба да садржи све информације неопходне корисницима при одабиру мајстора, као што је цена или оперативно подручје. Опционо, могуће је додати слике претходних радова.



Слика 5.3.3. Додавање услуге мајстора

Трећа секција приказује исечак објављених прича корисника. Сваки исечак приказује наслов приче, кратки опис, датум објављивања и једну слику. Кликом на исечак приказује се страница за детаљни приказ корисничке приче.

Четврта секција приказује три најновије рецензије корисника на пријављеног мајстора и приказана је на слици 5.3.4. Свака рецензија садржи име аутора, датум објаве, кратак опис и оцену приказану у облику попуњених и празних звездица.

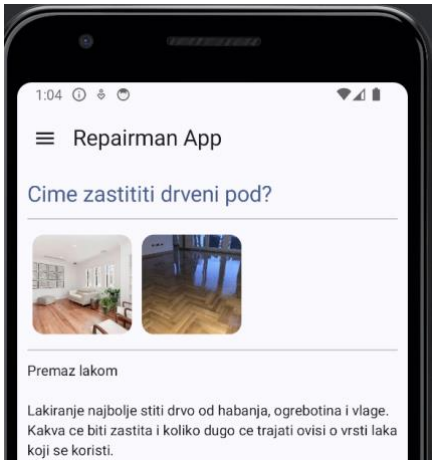
Слика 5.3.4. Секција за приказ најновијих рецензија

5.4. Преглед корисничких прича

Други главни екран апликације приказује исечке за све објављене корисничке приче. Сваки исечак садржи наслов приче, име аутора и датум објављивања, кратак опис и једну од приложених слика. Пријављеним корисници приказује се дугме за додавање нове приче у доњем десном углу екрана. Екран са корисничким причама приказан је на слици 5.4.1.

Слика 5.4.1. Екран за приказ корисничких прича

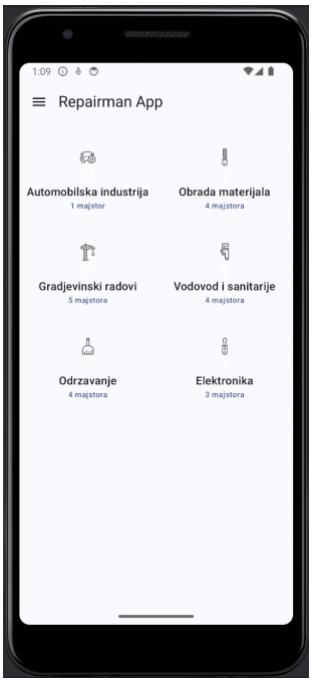
Притиском на неки од исечака приказује се екран са детаљним приказом изабране корисничке приче, приказан на слици 5.4.2. На дну екрана, могуће је обрисати дату причу. Причу могу обрисати само аутор и администратор система.



Слика 5.4.2. Приказ детаља о корисничкој причи

5.5. Претрага мајстора

Трећи екран омогућава главну функционалност апликације – претрагу и контакт мајстора. Најпре, потребно је одабрати категорију највишег нивоа за тражену услугу. На екрану су излистане све категорије и број мајстора у свакој категорији, као на слици 5.5.1.

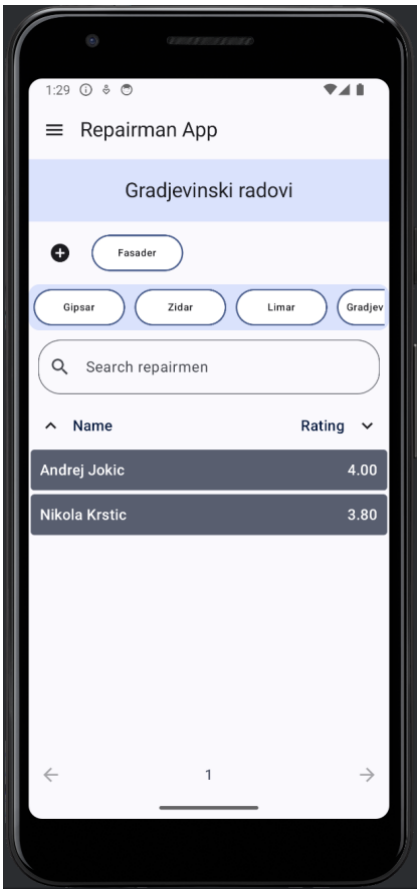


Слика 5.5.1. Екран за одабир категорије услуга мајстора

Притиском на жељену категорију приказује се екран са свим мајсторима који пружају бар једну услугу изабране категорије. Ради лакшег прегледа, мајстори су подељени на странице кроз које је могуће навигирати користећи мени на дну екрана. Мени приказује тренутно изабрану страницу и садржи дугмиће за прелаз на претходну, односу следећу страницу. Ради лакшег прегледа, за сваког мајстора, приказани су само име, презиме и његова просечна оцена.

Постоје три команде за манипулацију прегледа мајстора. Мајсторе је могуће сортирати на основу просечне оцене или имена, користећи дугмиће са сликом стрелице изнад листе мајстора. Друга команда представља текстуално поље у које је могуће унети име и/или презиме мајстора. Само мајстори чије име/презиме одговара унетом тексту бивају излистани. Трећа команда омогућава корисницима додавање и брисање услуга. Само мајстори који пружају све одабране услуге бивају приказани.

Уколико ниједна услуга није изабрана, приказани су сви мајстори изабране категорије. Изабране услуге излистане су испод имена одабране категорије. Услуга се додаје притиском на иконицу + изнад поља за претрагу мајстора. Притиском на ово дугме, појављује се мени са свим услугама изабране категорије. Притиском на облачић неке услуге, она се додаје у листу одабраних услуга. Услугу је могуће уклонити притиском на облачић услуге унутар листе изабраних услуга. Екран за претрагу мајстора приказан је на слици 5.5.2.



Слика 5.5.2. Екран за напредну претрагу мајстора

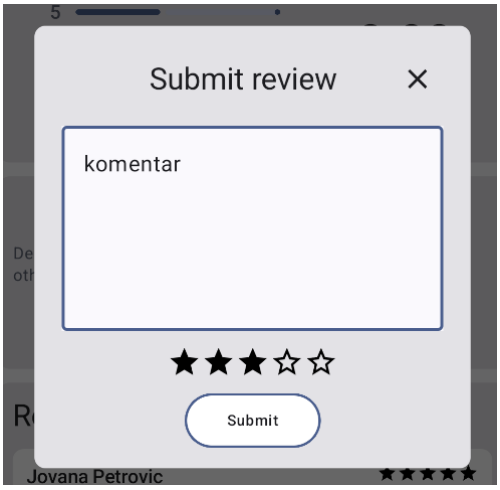
5.6. Инспекција мајстора

Притиском на одређеног мајстора излистаног на страници за преглед мајстора, приказује се страница са детаљним информацијама о мајстору. На врху странице, поред имена мајстора, приказана су два дугмета за контакт мајстора. Притиском на дугме у облику телефонске слушалице, отвара се апликација за телефонирање инсталирана на сваком Андроид уређају. Притиском на дугме у облику електронске поште, отвара се апликација за слање текстуалних порука. Телефонски број мајстора аутоматски је уписан у поље примаоца позива/поруке.

Страница за инспекцију мајстора подељена је у три секције. За навигацију између секција користи се мени на дну странице. Иконица изабрана секције обојена је светло плавом бојом. Прва секција приказује све услуге које мајстор пружа. Ова секција изгледа идентично као секција регистрованих услуга на корисничком профилу описана у трећем потпоглављу и неће бити поново описана. Додатно, ако је улоговани корисник администратор система, видљиво је дугме за брисања услуге. На слици 5.6.1. приказан је екран за инспекцију мајстора.

Слика 5.6.1. Екран за инспекцију мајстора

Друга секција садржи детаљни приказ рецензија корисника на датог мајстора. Приказана је просечна оцена и све рецензије. Поново, секција је идентична као на профилној страници и неће бити поново описана. Додатно, улоговани корисници имају опцију остављања рецензије на мајстора. Један корисник може оставити само једну рецензију на истог мајстора. Приликом притиска на дугме „*review repairman*”, отвара се дијалог за уношење рецензије, приказан на слици 5.6.2. Приликом уношења рецензије, потребно је унети коментар и изабрати оцену у виду попуњених стрелица.



Слика 5.6.2. Остављање рецензије на мајстора

Трећа секција приказује контакт информације мајстора – број мобилног телефона и пребивалиште.

6. Закључак

У овом раду описана је реализација система за проналазак и рецензију мајстора на Андроид мобилној платформи. Апликација корисницима омогућава једноставно повезивање са поузданим и квалитетним мајсторима у њиховом подручју, с циљем олакшања напорног процеса проналажења и ангажовања професионалних услуга. Дат је увод у поменути проблем и детаљно су анализирана постојећа решења из перспективе менаџера пројекта, али и перспективе корисника. Пројектован је систем који на оптималан начин решава препознате функционалне и нефункционалне захтеве. Описане су технологије коришћење за имплементацију система и њихова примена у апликацији. Објашњена је архитектура система, као и имплементација појединачних компонената. На крају, приказан је рад система кроз корисничко упуство и низ слика графичког корисничког интерфејса саме апликације.

Мобилна апликација представља део већег система који укључује и веб сајт. Конкретно, серверски део система развијан је паралелно са колегом. Било је неопходно увести систем за верзионисање с циљем избегавања конфликата. Промене у бази података решене су коришћењем *Flyway* миграција на оптималан начин. Јављали су се проблеми приликом промене API уговора који бекенд сервер пружа. Проблем је било могуће избећи коришћењем приступа *API-Contract-First,* односно дефинисањем комплетног API уговора пре почетка имплементације мобилне и веб апликације.

Систем је могуће проширити новом функционалношћу која улогованом кориснику аутоматски предлаже мајсторе на основу његове локације. Додатно, могуће је увести персонализацију и унапредити предложене мајсторе на основу претходних искустава корисника. Попут сајта ”NadjiMajstora”, могуће је убацити интерактивну мапу која визуелно приказује оперативну област мајстора. Корисничке приче могуће је категоризовати према категоријама услуга, и омогућити корисницима претрагу мајстора преко њих. Коначно, систем се може унапредити додавањем аутоматизованих тестова имплементацијом јединичних тестова (*unit tests*), тестова за компоненте (*components tests*) и интреграционих тестова (*E2E tests*).

Литература

[1] Customer reviews. Доступно на: https://get.nicejob.com/resources/why-customer-reviews-are-important-and-how-to-get-more (23.08.2024)

[2] Сајт ProfiMajstor. Доступно на: https://www.profimajstor.rs/ (05.08.2024)

[3] Сајт PozoviMajstora. Доступно на: https://pozovimajstora.com/ (05.08.2024)

[4] Сајт NadjiMajstora. Доступно на: https://www.nadjimajstora.rs/ (05.08.2024)

[5] Reviews. Доступно на: https://onlinelibrary.wiley.com/doi/abs/10.1002/mar.21565/ (23.08.2024)

[6] Мобилна апликација Postay. Доступно на: https://postaj.app/ (05.08.2024)

[7] TCP handshake. Доступно на: https://www.coursera.org/articles/three-way-handshake/ (23.08.2024)

[8] Databases. Доступно на: https://towardsdatascience.com/relational-vs-non-relational-databases-f2ac792482e3/ (23.08.2024)

[9] MySQL. Доступно на: https://dev.mysql.com/doc/ (23.08.2024)

[10] Java. Доступно на: https://www.ibm.com/topics/java/

[11] Java portable. Доступно на: https://devrant.com/rants/2096719/which-programming-languages-can-create-portable-programs-that-can-run-on-every-o (23.08.2024)

[12] SpringBoot. Доступно на: https://en.wikipedia.org/wiki/Spring_Boot (23.08.2024)

[13] Android architecture. Доступно на: https://www.elluminatiinc.com/android-architecture/ (23.08.2024)

[14] Kotlin. Доступно на: https://www.twitchtime.com/blogs/kotlin-android-app-development/ (23.08.2024)

[15] Android UI. Доступно на: https://developer.android.com/develop/ui/compose/documentation/ (23.08.2024)

[16] Flyway migrations. Доступно на: https://www.baeldung.com/database-migrations-with-flyway/ (10.08.2024)

[17] Microservices. Доступно на: https://dev.to/jnavez/make-your-microservices-tastier-by-cooking-them-with-a-sweet-onion-34n2/ (23.08.2024)

[18] JWT. Доступно на: https://jwt.io/introduction/ (13.08.2024)

[19] Android architecture. Доступно на: https://www.toptal.com/android/android-apps-mvvm-with-clean-architecture/ (23.08.2024)

[20] Android ui layer. Доступно на: https://developer.android.com/topic/architecture/ui-layer/ (02.08.2024)

[21] Hilt. Доступно на https://developer.android.com/training/dependency-injection/hilt-android/ (15.08.2024)

СПИСАК СЛИКА

СПИСАК ТАБЕЛА