



УНИВЕРЗИТЕТ У НОВОМ  
САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ  
НАУКА У НОВОМ САДУ




Маја Варга

# ВЕБ АПЛИКАЦИЈА ЗА РУКОВАЊЕ ПАМЕТНИМ СВЕТЛИМА

Дипломски рад  
- Основне академске студије -

Нови Сад, 2024.



	УНИВЕРЗИТЕТ У НОВОМ САДУ <b>ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА</b> 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	<b>ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА</b>	Лист: 1/1

(Податке уноси предметни наставник - ментор)

Врста студија:	Основне академске студије
Студијски програм:	Софтверско инжењерство и информационе технологије
Руководилац студијског програма:	проф. др Мирослав Зарић

Студент:	Маја Варга	Број индекса:	SV 54/2020
Област:	Инжењерство софтвера за Internet/Web of Things		
Ментор:	Др Милан Видаковић, редовни професор		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: <ul style="list-style-type: none"> <li>- проблем – тема рада;</li> <li>- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;</li> <li>- литература</li> </ul>			

### НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

Веб апликација за руковање паметним светлима
--

### ТЕКСТ ЗАДАТКА:

Задатак рада представља развој веб апликације за руковање паметним светлима произвођача <i>WiZ Connected</i> . Серверски део апликације треба имплементирати у <i>Python</i> програмском језику коришћењем <i>Flask</i> радног оквира. Клијентски део решења треба имплементирати користећи <i>React</i> радни оквир. Апликација треба да подржи управљање, груписање и организацију светала по некретнинама и просторијама. Спецификацију система треба представити употребом <i>UML</i> дијаграма.
--

Руководилац студијског програма:	Ментор рада:

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





# САДРЖАЈ

1.	УВОД .....	7
2.	ПРЕГЛЕД СЛИЧНИХ СИСТЕМА .....	9
3.	КОРИШЋЕНЕ СОФТВЕРСКЕ ТЕХНОЛОГИЈЕ .....	13
3.1	<i>Flask</i> .....	13
3.2	<i>pywizlight</i> .....	14
3.3	<i>React</i> .....	15
3.4	<i>PostgreSQL</i> .....	16
4.	СПЕЦИФИКАЦИЈА .....	19
4.1	Спецификација захтева .....	19
4.1.1	Функционални захтеви .....	19
4.1.2	Нефункционални захтеви .....	23
4.2	Спецификација система .....	25
4.2.1	Модел података .....	25
4.2.2	Архитектура система .....	27
4.2.3	Интеракција са светлима .....	33
5.	ИМПЛЕМЕНТАЦИЈА .....	37
5.1	Помоћне функције .....	37
5.2	Функционалности .....	41
6.	ДЕМОНСТРАЦИЈА .....	53
7.	ЗАКЉУЧАК .....	69
8.	ЛИТЕРАТУРА .....	71
9.	БИОГРАФИЈА .....	75
	КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА .....	77
	KEY WORDS DOCUMENTATION .....	79

## 1. УВОД

Паметни кућни уређаји представљају све више популарну алтернативу традиционалним уређајима. Њихова највећа предност је могућност управљања путем паметног телефона или рачунара, често и са удаљених локација. Да би се то омогућило, сваки произвођач паметних уређаја нуди апликације за њихово руковање. Међутим, најчешће су то мобилне апликације, што значајно онемогућује коришћење рачунара у исту сврху.

*WiZ Connected* [1] (у даљем тексту *WiZ*) је произвођач паметних кућних уређаја, најпопуларнији по својим паметним светлима. Управљање је могуће преко званичне мобилне апликације. За рачунаре не постоји јединствена апликација, већ само могућност коришћења уређаја преко других платформи, са ограниченим бројем могућности. Из тог разлога, задатак овог рада је развој веб апликације фокусиране на паметна светла поменутог произвођача.

Серверски део решења је имплементиран користећи *Flask* [2] радни оквир и *pywizlight* [3] библиотеку за управљање светлима. Клијентски део решења је имплементиран користећи *React* [4] радни оквир. За складиштење података је одабрана *PostgreSQL* [5] база података.

Решење представља јединствену апликацију за управљање светлима, чиме се издваја од свих осталих платформи које нуде само интеграцију са њима. Нуди већи број функционалности од већине, као и интуитивнији кориснички интерфејс.

Рад је организован у седам поглавља. Након увода, у другом поглављу се разматрају постојећа решења, у трећем поменуто технологије коришћене у имплементацији, након чега се у четвртом даје спецификација захтева решења. У петом поглављу је детаљно описана имплементација решења, након

чега се у шестом представља демонстрација рада битних функционалности апликације. У завршном, седмом поглављу следи рекапитулација решења и разматрање могућих праваца даљег развоја апликације.



## 2. ПРЕГЛЕД СЛИЧНИХ СИСТЕМА

У овом поглављу се описују постојеће платформе за управљање паметним уређајима и разматрају њихове предности и мане. Критеријуми за њихов одабир су:

- мора да подржава интеграцију са паметним светлима произвођача *WiZ*,
- намењена је за употребу на рачунару,
- може се инсталирати и користити на *Windows* оперативном систему, или представља веб апликацију и
- бесплатна је.

На основу наведених критеријума, одабране су четири платформе:

- *Home Assistant*,
- *openHAB*,
- *ioBroker* и
- *Homey*.

*Home Assistant* [6][7] је платформа отвореног кода за интеграцију и управљање паметним уређајима написана у *Python* [8] програмском језику. Представља једно од најпопуларнијих решења са подршком за преко 1000 произвођача и сервиса. Главна предност платформе јесте што је у потпуности изолована на локалну мрежу рачунара, што значи да се управљање уређајима и чување података врши искључиво на рачунару, без потребе *cloud* инфраструктуре.

У оквиру интеграције са произвођачем *WiZ* могуће је:

- аутоматски детектовати и конфигурисати нова светла,
- ручно конфигурисати нова светла,
- укључити и искључити светла,
- мењати боју светла или температуру (за жуту/белу светлост),
- мењати осветљеност светла,

- укључивати предефинисане динамичне ефекте светала,
- мењати брзину динамичних ефеката и
- дефинисати сцене, тј. груписати светла и дефинисати њихово стање (боја, осветљеност, температура, ефекат, брзина ефекта) након активације сцене.

Поседује кориснички интерфејс за извршавање свих наведених акција. Међутим, представља главну ману платформе, зато што нема јасну структуру и захтева учење корака којима се долази до жељеног приказа.

*openHAB* [9] је платформа отвореног кода за интеграцију и управљање паметним уређајима написана у *Java* [10] програмском језику. Слично претходној, управљање и складиштење података се врши локално на рачунару, са могућношћу интеграције са *cloud* сервисима. У оквиру интеграције са произвођачем *WiZ* нуди све акције као и *Home Assistant*. Главна мана ове платформе јесте сложеност и недостатак интуитивности њеног коришћења. Концепт платформе [11] се заснива на ручном дефинисању објеката и акција које се извршавају над њима, што може захтевати писање програмског кода. Не поседује кориснички интерфејс за брзо и једноставно управљање, што чини ову платформу неадекватном за већину редовних корисника сличних платформи.

*ioBroker* [12] је платформа отвореног кода за интеграцију и управљање паметним уређајима написана у *JavaScript* [13] програмском језику. Као и претходне, функционише у потпуности локално на рачунару. Нуди подршку за преко 600 произвођача и сервиса. У оквиру интеграције са произвођачем *WiZ* нуди све претходно наведене акције осим аутоматског детектовања нових светала и дефинисања сцена, што је једна од мана ове платформе. Њена главна мана је начин на који управља светлима. Без употребе додатних библиотека, стање светла се поставља директним уносом вредности у текстуална поља. Она немају назначену структуру и границе вредности, што чини ову платформу најнеприступачнијом за редовне кориснике сличних платформи.

*Homey* [14] је платформа за интеграцију и управљање паметним уређајима са подршком за преко 1000 произвођача. Једина је од наведених платформи која је затвореног кода. У зависности од произвођача и његових уређаја, складиштење података и управљање се може обављати и преко *cloud* инфраструктуре, али и преко локалне мреже рачунара. У оквиру интеграције са *WiZ* уређајима, управљање и складиштење се врши преко *cloud* инфраструктуре. Нуди све претходно наведене акције осим аутоматске и ручне конфигурације светала и дефинисања сцена. Доступна светла се директно преузимају из *WiZ*-ове базе података при интеграцији, чиме се губи потреба за конфигурацијом на платформи. Међутим, главна мана платформе је недостатак могућности дефинисања сцена. Главна предност платформе јесте његов кориснички интерфејс, који је најједноставнији за коришћење од свих претходно поменутих.



## 3. КОРИШЋЕНЕ СОФТВЕРСКЕ ТЕХНОЛОГИЈЕ

У овом поглављу ће се детаљно разматрати технологије коришћене за имплементацију решења поменуте у уводном делу рада. У поглављу 3.1 следи опис *Flask* радног оквира, у поглављу 3.2 *pywizlight* библиотеке, у поглављу 3.3 *React* радног оквира и у поглављу 3.4 *PostgreSQL* базе података.

### 3.1 *Flask*

*Flask* је радни оквир за писање веб апликација написан у *Python* програмском језику. Спада у групу микрофрејмворка (*microframework*) [15], који нуде минималан број функционалности како би омогућили правилно рутирање, прихватање и враћање одговора на пристигле веб захтеве. Овим се смањује величина и комплексност самог радног оквира, али и повећава слобода програмера да самостално бира додатне библиотеке које се баве стандардним функционалностима попут:

- валидације пристиглих порука у оквиру захтева,
- објектно-релационог мапирања,
- руковања отпремања садржаја и
- слично.

*Flask* је базиран на раду *Werkzeug*, *Jinja* и *Click* библиотеке.

*Werkzeug* [16] је библиотека која представља скуп алата за имплементацију других радних оквира за веб, укључујући и *Flask*. Главна улога ове библиотеке јесте руковање веб захтевима, тј. прихватање захтева од сервера, његово рутирање до одговарајуће функције апликације и слање одговора. Служи као спона између *Flask* апликације и сервера на коме је покренута. Поред тога, обезбеђује и једноставан сервер на коме је могуће сервирати *Flask* апликације.

*Jinja* [17] је библиотека за креирање динамичких *HTML* страница на серверској страни. Нуди хибридную синтаксу за писање *HTML* кода, која омогућава убацивање логичких израза и променљивих у странице. Овим се постиже персонализација садржаја који апликација враћа, чиме она може да сервира статички садржај прилагођен кориснику и његовим подацима који се чувају на серверу. Уколико се *Flask* користи за имплементацију и серверског и клијентског дела, *Jinja* има функцију креирања корисничког интерфејса на клијентској страни.

*Click* [18] је библиотека за креирање интерфејса командне линије (*Command-Line interface, CLI*) [19]. Нуди једноставно креирање произвољних команди везаних за апликацију, као и аутоматско креирање навигације (листа доступних команди) и произвољно форматирање исписа. *Flask* је користи за генерисање свих његових команди и форматирање исписа логова у конзоли приликом покретања апликације и њеног рада.

За имплементацију овог рада коришћен је *Flask* верзије 3.0.3.

## 3.2 *pywizlight*

*pywizlight* је библиотека за управљање паметним светлима произвођача *WiZ* имплементирана у *Python* програмском језику. Намењена је да подржи комуникацију са светлима преко *UDP* [20] протокола, слањем команди преко мреже на коју је повезан рачунар. Команде се шаљу у формату према званичној документацији објављеној од стране произвођача [21]. Нуди два начина слања команди: преко интерфејса командне линије, и директним позивањем доступних функција унутар кода.

Главна предност библиотеке јесте што функционише у потпуности преко локалне мреже рачунара. Ово омогућује повећану безбедност управљања светлима, зато што се извршене команде не шаљу на *cloud* сервере произвођача. Међутим,

библиотека не гарантује потпуну изолацију од комуникације са њима, зато што светла шаљу своје тренутно стање на *cloud* ван контроле библиотеке.

Предуслов за коришћење и детекцију светала у оквиру библиотеке јесте да она буду претходно конфигурисана кроз званичну мобилну апликацију произвођача. Према томе, ова библиотека не тежи да замени ту апликацију, већ да омогући комуникацију са светлима и преко рачунара. *Home Assistant* описан у претходном поглављу је једна од платформи која је користи за имплементацију своје интеграције са *WiZ* светлима.

За имплементацију овог рада коришћен је *pywizlight* верзије 0.5.14.

### 3.3 *React*

*React* је радни оквир за креирање корисничких интерфејса базиран на *JavaScript* програмском језику. Основна идеја радног оквира је креирање корисничког интерфејса сачињеног од једне стране, где се у зависности од корисничке интеракције мења приказ на њој. *React* се придржава декларативне парадигме [22], чиме се кориснички интерфејс дизајнира тако да опише шта треба приказати на страници, а радни оквир је задужен да ажурира приказ у зависности од датог описа. Практично, ово значи да програмер наводи које податке је потребно приказати, а *React* генерише приказ са тим подацима и ажурира га када ти подаци буду измењени.

*React* странице је могуће писати у *JavaScript* и *TypeScript* [23] језику. Код који се пише је типично комбинација логичког кода (функције и променљиве) писаног у једном од поменутих језика и хибридног *HTML* кода, који је проширен да подржи приказ тренутног стања променљивих или резултата логичких израза. Ово је још познато и као *JSX* [24] синтакса. Уколико је коришћен *JavaScript*, код се чува у фајлу са екстензијом *.jsx*, а у случају да је коришћен *TypeScript* *.tsx*.

Странице у *React* радном оквиру су сачињене од једне или више компоненти. Компонента представља једну логичку целину апликације, на пример страницу, један њен део (навигација, форма за унос података, заглавље,...) или једно поље (текстуални унос, дугме, падајући мени,...). Могу се секвенцијално слагати и међусобно угњеждавати. Приликом угњеждавања, компоненте могу слати податке само у једном смеру, од родитељске компоненте до потомачке. Уколико је потребно обезбедити двосмерну комуникацију између компоненти, потомачкој компоненти се шаље функција из родитељске компоненте која мења њено стање (*callback* функција).

Да би обезбедио ефикасно генерисање и ажурирање приказа код честих промена података, *React* у радној меморији чува виртуално стабло *HTML* објеката, такозвано виртуално *DOM* (*Document Object Model*) стабло [25]. За разлику од правог *DOM* стабла које представља тренутни приказ који је генерисан на екрану, виртуални *DOM* је везан за променљиве и надгледа њихово тренутно стање. Уколико се вредност променљиве измени, виртуални *DOM* се последично мења и пореди са тренутним приказом. Измене се пропадају само на делове који су захваћени променама, чиме се смањује време освежавања приказа и повећавају перформансе апликације.

За имплементацију овог рада коришћен је *React* верзије 18.3.1 писан у *TypeScript* језику.

### 3.4 PostgreSQL

*PostgreSQL* је систем за управљање релационим базама података заснованим на *SQL* упитном језику [26]. Нуди могућност складиштења података и управљања њима, што обухвата претрагу и ажурирање података, дефинисање права приступа подацима, манипулацију шеме базе података и остало. Поред основних, *PostgreSQL* нуди проширени скуп *SQL* наредби које омогућују креирање нових типова података, креирање



функција и рутина које се извршавају над подацима и проширују могућности претраге података. Акције које се извршавају над базом података су *ACID* [27] трансакције које:

- се успешно извршавају само уколико је успешно извршен сваки њен корак (атомичност, *Atomicity*),
- трансформишу базу података из једног валидног стања у друго валидно стање, односно, одржава конзистентност података у складу са дефинисаним ограничењима над њима (конзистентност, *Consistency*),
- се међусобно не нарушавају уколико се истовремено извршавају, тј. остављају базу података у стању у којем би била да се оне секвенцијално извршавају (изолованост, *Isolation*) и
- гарантују да ће се извршити уколико су успешне, чак и ако дође до прекида рада система (издржљивост, *Durability*).

За имплементацију овог рада коришћен је *PostgreSQL* верзије 15.3.



## **4. СПЕЦИФИКАЦИЈА**

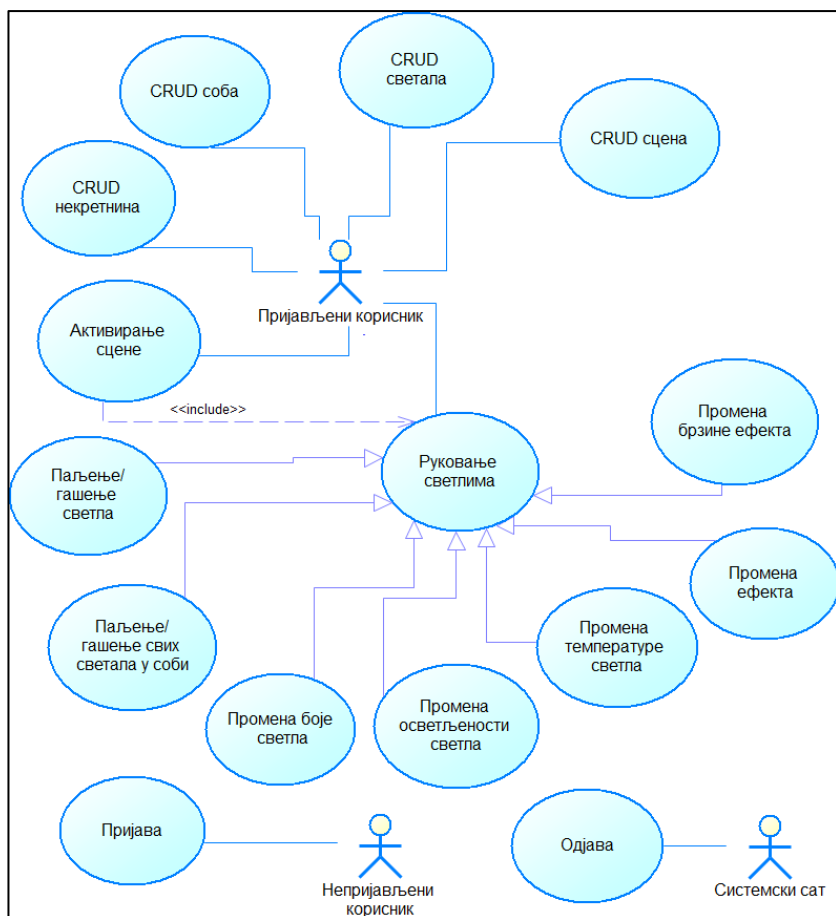
У следећем поглављу ће бити представљена спецификација система. У поглављу 4.1 следи спецификација захтева, а у поглављу 4.2 спецификација система.

### **4.1 Спецификација захтева**

У овом поглављу ће се представити сви захтеви које апликација испуњава. У поглављу 4.1.1 су детаљно описани сви функционални захтеви тј. функционалности система, а у поглављу 4.1.2 сви нефункционални захтеви.

#### **4.1.1 Функционални захтеви**

На слици 1 је приказан дијаграм случајева коришћења апликације.



Слика 1 Дијаграм случајева коришћења

Непријављени корисник не може користити функционалности система пре пријаве. Пријава подразумева унос корисничке лозинке која је предефинисана и чува се на серверској страни апликације. Након успешне пријаве, кориснику су доступне све функционалности система и започиње се његова сесија. Сесија траје 24 часа, након чега ће систем аутоматски одјавити корисника. У том случају, неопходна је поновна пријава.

*CRUD* на дијаграму представља четири основне функционалности које се могу извршити над објектом: креирање тј. додавање новог (*Create*), читање тј. добављање података о објекту (*Read*), ажурирање података о објекту (*Update*) и брисање објекта (*Delete*).

Некретнина је највећа организациона јединица у систему. Омогућује груписање светала у случају да је корисник власник више некретнина. Креирање нове некретнине подразумева унос њеног имена и имена прве просторије у њој, чиме се омогућује кориснику да одмах након креирања може додавати светла. Измена подразумева могућност промене имена некретнине. Брисање ће последично изазвати брисање свих просторија, светала у њима и сцена које су повезане са просторијама.

Просторија је организациона јединица повезана са некретнином. Нуди логичку поделу некретнине на произвољан број јединица у којима се налазе светла. Креирање нове просторије подразумева унос њеног имена и одабир некретнине којој се придружује. Измена подразумева могућност промене имена просторије. Брисање ће последично изазвати и брисање свих светала у просторији, као и сцена које су јој придружене. Уколико је некретнина са којом је просторија повезана остала без просторија, брише се и она.

Светло је основна јединица система којом он управља. Процес креирања новог светла започиње аутоматском претрагом свих доступних светала на мрежи. Предуслов да светла буду пронађена јесте да морају бити регистрована помоћу званичне мобилне апликације. Ово обезбеђује да светло добије своју *IP* адресу на мрежи, што га чини видљивим процесу детекције. Уколико су светла пронађена, приказује се листа свих нађених инстанци којима корисник придружује име, просторију којој припада и тип (сијалица, лампа, светлосна трака). Током регистрације светла је могуће послати команду за његово укључивање, како би се сазнало које светло је у питању. Корисник није дужан да региструје сва пронађена светла одједном. Када је процес завршен, нова светла се више неће

приказивати у резултатима аутоматске претраге. Измена подразумева могућност промене имена и просторије којој је светло придружено. Измена просторије светла не утиче на измену просторије којој припадају сцене које га обухватају. Брисање светла узрокује његово уклањање из сцена у којима је укључено, као и поновно приказивање у резултатима аутоматске претраге, чиме се може поновно регистровати.

Сцена је појам који описује оркестрацију једног или више светала у једној просторији како би се постигао жељени изглед без потребе конфигурације појединачних светала. Креирање нове сцене подразумева унос њеног имена и просторије којој се придружује. Након тога се дефинише која светла у тој просторији учествују у сцени и режим њиховог рада приликом активације сцене. Активацијом сцене се сва светла придружена сцени укључују према задатом режиму, а остала искључују. Измена подразумева могућност промене имена сцене, придруживања или уклањања светала из сцене, промену режима рада светала, као и промену појединачних параметара режима. Брисањем сцене се губи могућност њене активације.

Светлима која су регистрована се може мењати режим рада и његови параметри. Постоје три могућа режима рада:

- режим боје – светлост је описана са три компоненте: црвена ( $R$ ), зелена ( $G$ ) и плава ( $B$ ),
- режим жутог/белог светла – светлост је описана температуром израженом у келвинима и
- режим ефекта – ефекти су предефинисане динамичне промене светла специфичне за WiZ. Постоји 34 подржаних ефеката којима је придружен идентификациони број, којим је овај режим и дефинисан.

Параметри које је могуће променити су:

- боја – изражена преко своје три компоненте, вредности се крећу између 0 и 255,

- температура – број у границама минималне и максималне дозвољене вредности за светло, типично од 2200 до 6700 келвина (зависи од модела светла),
- осветљеност – број између 0 и 255, мења јачину светлости у било ком режиму,
- ефекат – идентификациони број између 1 и 34 којима је придружен и назив и
- брзина ефекта – број између 10 и 200, мења брзину прелаза између две боје у оквиру ефекта (за ефекте који подржавају подешавање брзине).

Физичка светла интерно чувају своје последње активно стање (режим рада и параметре). Приликом укључивања светла, уколико није наведен режим рада, укључиће се у последње запамћено стање. У супротном, светла се укључују према задатом режиму. На нивоу просторије постоји могућност укључивања и искључивања свих светала у једном кораку, под условом да постоје светла придружена тој просторији.

#### **4.1.2 Нефункционални захтеви**

Предуслов за могућност руковања светлима јесте да рачунар буде повезан на исту мрежу као и светла, и да светла буду претходно конфигурисана преко званичне мобилне апликације, чиме добија своју *IP* адресу на мрежи. Међутим, ово не гарантује да светло задржава адресу, већ је могуће (и врло често се дешава) да светло периодично добије нову адресу. Како је предуслов за управљање светлима познавање његове *IP* адресе, систем мора да има механизам како да у разумном временском периоду ажурира адресе које су измењене, без потребе ангажовања корисника.

Корисник је у могућности да прати тренутно стање светала. Систем мора да омогући периодично ажурирање корисничког интерфејса, како би се увек приказивало најактуелније стање.

Приликом регистровања светала, корисник не мора да води рачуна о техничким аспектима светла које региструје (*MAC*

адреса, подршка за боје, минимална и максимална дозвољена температура, могућност промене осветљења,...). Систем мора да подржи аутоматску детекцију релевантних параметара светала, како би обезбедио једноставну регистрацију и неометано руковање. У складу са могућностима светала, кориснику није потребно откривати функционалности које оно не подржава (нпр. светлима која немају подршку за режим боје не треба приказивати поља за њену измену).

Кориснички интерфејс мора да буде имплементиран поштујући принципе прилагодљивог веб дизајна, како би се могао правилно приказивати на свим димензијама екрана. Све компоненте (секције странице, поља за унос,...) морају бити обележене називом и/или симболом који је представља. Фонт слова мора да буде читљив, довољно велик и адекватне величине у складу са наменом (наслов секције, ознака поља, текст уноса, ...).

Валидација података се мора извршити и са стране серверске апликације приликом прихватања захтева, и са стране клијентске апликације приликом прослеђивања података прикупљених у форми. Сва поља која су обавезна морају да буду обележена као таква. Приликом измене неких од података објекта, потребно је у пољима за унос приказати њихове тренутне вредности.

Акције које не резултују физичком променом стања светала морају имати адекватан одзив. Ово подразумева приказ поруке са описом акције и успешности њеног извршења. Акције које резултују брисањем неког од објеката морају бити претходно потврђене у дијалогу, где се представљају све могуће последице акције.

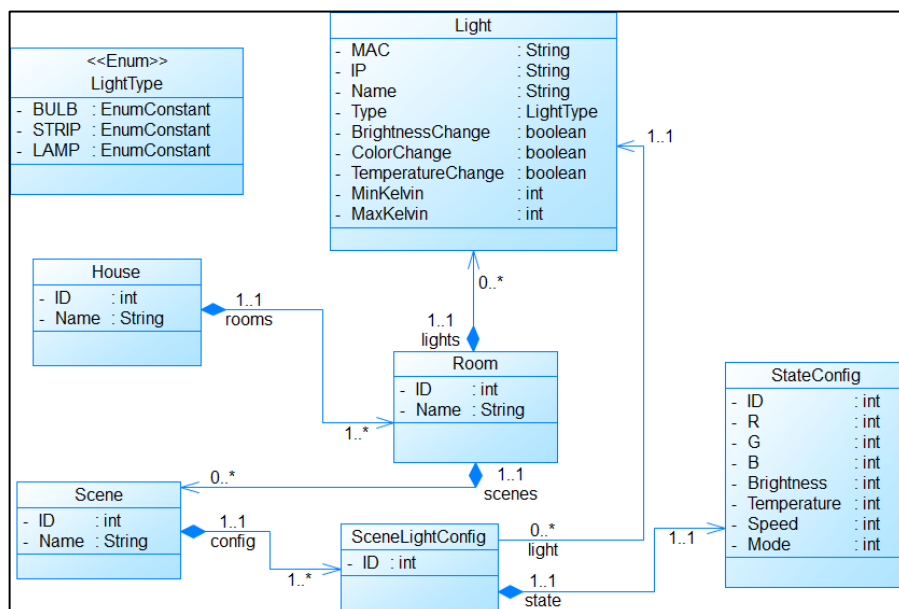


## 4.2 Спецификација система

У овом поглављу ће се разматрати општи изглед система и интеракција између његових компоненти. У поглављу 4.2.1 је представљен модел података система, у поглављу 4.2.2 архитектура система, а у поглављу 4.2.3 се налази детаљнији преглед интеракције система и светала.

### 4.2.1 Модел података

На слици 2 је приказан дијаграм класа система.



Слика 2 Дијаграм класа система

Некретнина (*House*) садржи идентификациони број (*ID*), назив (*Name*) и листу просторија (*rooms*) које су са њом повезане. Некретнина мора да поседује барем једну просторију, док једна просторија може бити везана само за једну некретнину.

Просторија (*Room*) садржи идентификациони број (*ID*), назив (*Name*), листу светала које се налазе у тој просторији

(*lights*) и листу сцена које су везане за светла у просторији (*scenes*). Просторија не мора да садржи светла, чиме не може садржати ни сцене. Светло може да се налази у само једној просторији у једном тренутку, али може променити просторију којој припада. Просторија не мора да има сцене асоциране са светлима које се у њој налазе.

Светло (*Light*) се идентификује уместо идентификационог броја својом *MAC* адресом, која је такође јединствена за свако светло. Поред тога, садржи назив (*Name*) и тип (*Type*) који може бити:

- сијалица (*BULB*),
- светлосна трака (*STRIP*) и
- лампа (*LAMP*).

Светлом се управља преко његове *IP* адресе. Физичке карактеристике светла су:

- могућност промене осветљења (*BrightnessChange*),
- могућност рада у режиму боје (*ColorChange*),
- могућност промене температуре и рада у режиму жутог/белог светла (*TemperatureChange*),
- минимална температура светлости у келвинима (*MinKelvin*) и
- максимална температура светлости у келвинима (*MaxKelvin*).

Сцена (*Scene*) садржи идентификациони број (*ID*), назив (*Name*) и листу конфигурација стања рада светала које су укључене у сцену (*config*). Сцена мора да садржи барем једно исконфигурисано светло.

Конфигурација светла у сцени (*SceneLightConfig*) садржи идентификациони број (*ID*) и повезује светло (*light*) и стање (*state*) у коме ће се наћи након активације сцене. Светло не мора да учествује у сценама, али уколико учествује, мора да има подешено стање.

Стање светла у сцени (*StateConfig*) садржи идентификациони број (*ID*) и конфигурацију режима рада и његових параметара светла. Ти параметри су:

- црвена компонента боје (*R*),
- зелена компонента боје (*G*),
- плава компонента боје (*B*),
- осветљеност (*Brightness*),
- температура светла (*Temperature*),
- ефекат (*Mode*) и
- брзина ефекта (*Speed*).

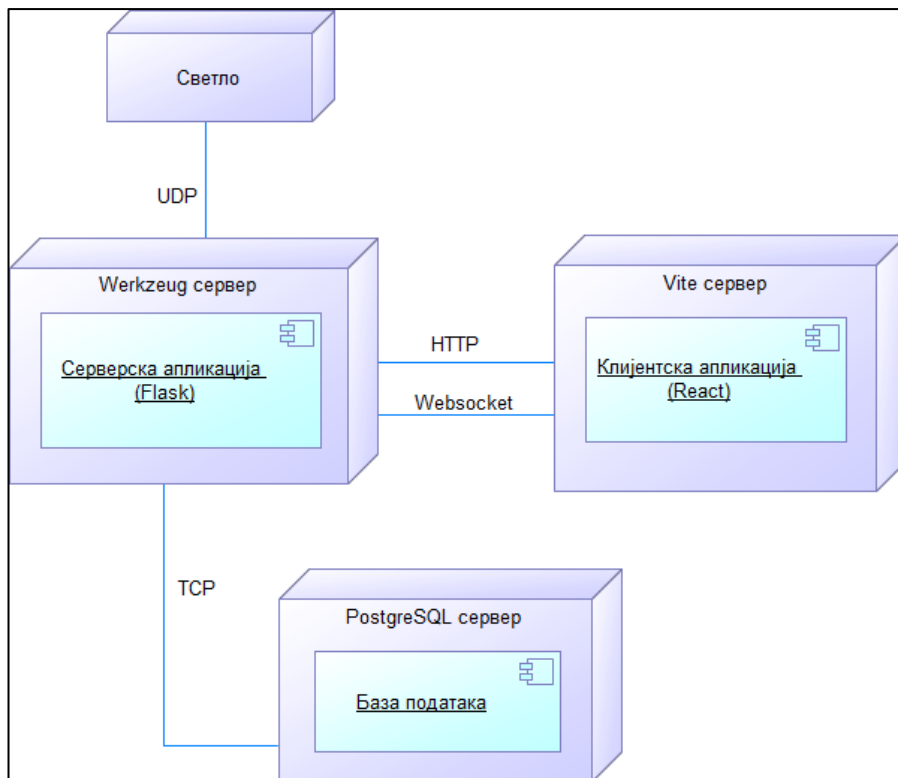
У зависности од жељеног режима рада, неке од вредности ће бити постављене на вредност ван опсега, како би се нагласило да тај параметар не игра улогу у стању светла. Уколико је тражено стање у режиму рада боје, вредности ефекта, брзине ефекта и температуре неће бити постављене. Уколико је тражено стање у режиму рада жутог/белог светла, вредности црвене компоненте, зелене компоненте, плаве компоненте, ефекта и брзине ефекта неће бити постављене. Уколико је тражено стање у режиму рада ефекта, вредности црвене компоненте, зелене компоненте, плаве компоненте и температуре светла неће бити постављене.

#### 4.2.2 Архитектура система

Систем је подељен на три логичке целине (компоненте):

- клијентска апликација – сервира кориснички интерфејс,
- серверска апликација – обезбеђује функционалности система и поседује логику за управљање светлима и
- база података – складишти податке поштујући модел дат у поглављу 4.2.1.

На слици 3 је приказан дијаграм размештаја компоненти система.



Слика 3 Дијаграм размештаја компоненти

Клијентска апликација написана помоћу *React* радног оквира је сервирана помоћу *Vite* [28] алата, који нуди сервирање клијентске апликације на сервер који се налази локално на рачунару. Након подизања сервера, клијентској апликацији се типично приступа са локалне мреже на порту 5173 (<http://localhost:5173>). Клијентска апликација комуницира са серверском апликацијом на два начина, у зависности од жељене операције. За све операције које укључују руковање светлима, активацију сцене и добављање тренутног стања светала користи се *websocket* [29] протокол. Ово обезбеђује брзи пренос порука између две компоненте и несметано добављање стања светала, без потребе за константним слањем захтева за добављање тренутног стања. За све остале операције, користи се *HTTP* протокол [30], који омогућује комуникацију по моделу захтев-

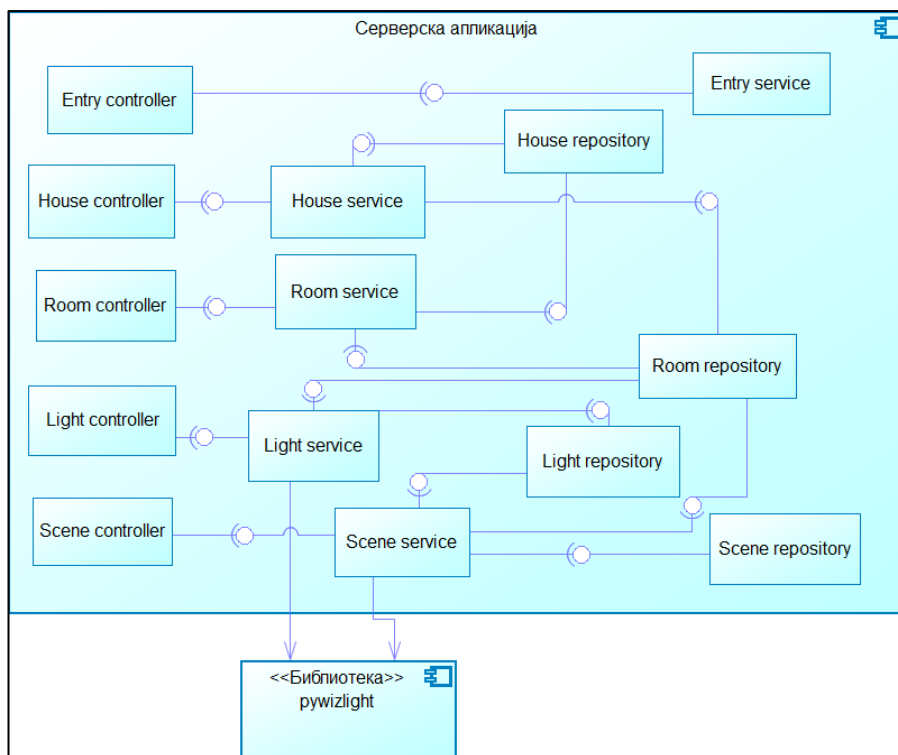
одговор. Клијентска апликација шаље *HTTP* захтеве одређеног типа, док серверска апликација прима те захтеве и обрађује их. Типови *HTTP* захтева које апликација шаље су:

- захтеви за додавање нових објеката и ауторизацију (*POST*),
- захтеви за ажурирање података о објекту (*PUT*),
- захтеви за добављање објеката, његових података и статуса ауторизације (*GET*) и
- захтеви за брисање објеката (*DELETE*).

Серверска апликација написана помоћу *Flask* радног оквира користи *Werkzeug* сервер. *Werkzeug* је претходно описан у поглављу 3.1. Приликом покретања *Flask* апликације покреће се сервер на који се апликација поставља, типично доступна на локалној мрежи на порту 5000 (*http://127.0.0.1:5000*). Како би се омогућила комуникација са светлима, серверска апликација користи *pywizlight* библиотеку, која команде светлима шаље преко *UDP* протокола.

База података се налази у оквиру *PostgreSQL* сервера типично на локалној мрежи на порту 5432. Комуникација серверске апликације и базе података се одвија помоћу *TCP* протокола [32]. Међутим, ова комуникација је апстрахована користећи *Flask-SQLAlchemy* [33] библиотеку за објектно-релационо мапирање, чиме се све операције везане за успостављање *TCP* конекције и размене података пребацују у одговорност библиотеке, а не серверске апликације. Библиотека нуди додатну апстракцију ове комуникације, где се наредбе за добављање и манипулацију података написане у *Python* програмском коду интерно преводе у *SQL* наредбе које се шаљу ка бази. Овим се избегава потреба за директним писањем *SQL* кода које је подложно грешкама, као и познавањем интерне структуре шеме базе података.

На слици 4 је приказан дијаграм компоненти серверске апликације.



Слика 4 Дијаграм компоненти серверске апликације

Серверска апликација прати архитектуру сачињену од контролера, сервиса и репозиторијума. Свака од ових компоненти је задужена за по један објекат или надлежност система, чиме код постаје читљивији и једноставнији за одржавање.

Контролери су софтверске компоненте надлежне за прихватање и слање одговора на захтеве пристигле са мреже. Контролери су сачињени од функција којима се придружује *HTTP* метод (*POST*, *PUT*, *GET*, *DELETE*) (уколико се очекује *HTTP* захтев) и путања. Функција ће се окинути само уколико је метод и путања са које је захтев пристигао идентична. Контролер прима све параметре захтева, укључујући његово заглавље, параметре путање и тело које може садржати прослеђене објекте. Ти објекти могу бити фајлови или *JSON* [34]

објекти који служе за креирање или ажурирање постојећих инстанци модела података система. Ове објекте називамо *DTO* (*Data Transfer Object*) објектима. Контролери не садрже пословну логику система, већ прослеђују захтев и пристигле податке даљем слоју у хијерархији (сервисима). Након што се захтев обради, контролер враћа статус обраде захтева и, уколико је потребно, податке у форми *DTO* објеката назад пошиљаоцу.

Контролери које серверска апликација садржи су контролер ауторизације корисника (*Entry controller*), контролер некретнина (*House controller*), контролер просторија (*Room controller*), контролер светала (*Light controller*) и контролер сцена (*Scene controller*). Контролери светала и сцена, поред прихватања *HTTP* захтева такође служе и за комуникацију преко *websocket* протокола за потребе управљања светлима и активације сцена. У том случају, функција која се окида се дефинише само преко своје путање и не мора се слати одговор пошиљаоцу.

Сервиси су софтверске компоненте које садрже пословну логику система, тј. дефинишу низ корака за извршавање функционалности система. Сервиси нуде своје функције контролерима и осталим сервисима за позивање. Добијају прослеђене податке од контролера, извршавају логику над тим подацима и шаљу повратну информацију назад ка контролеру. Сервиси не садрже логику за управљање и комуникацију са базом података, већ све захтеве који захтевају интеракцију са њом прослеђују последњем слоју (репозиторијумима). Сервиси могу комуницирати са осталим сервисима зарад извршавања потребне логике над сродним објектима. Уколико је логика коју треба извршити над њима једноставна, постоји могућност и директне комуникације са репозиторијумима сродних објеката (нпр. сервис некретнина (*House service*) комуницира са репозиторијумом некретнина (*House repository*), али и са репозиторијумом просторија (*Room repository*), пошто су те две класе повезане).

Сервиси које серверска апликација садржи се мапирају на претходно описане контролере. Сервис сцена и сервис светала директно комуницирају са *pywizlight* библиотеком зарад комуникације са светлима.

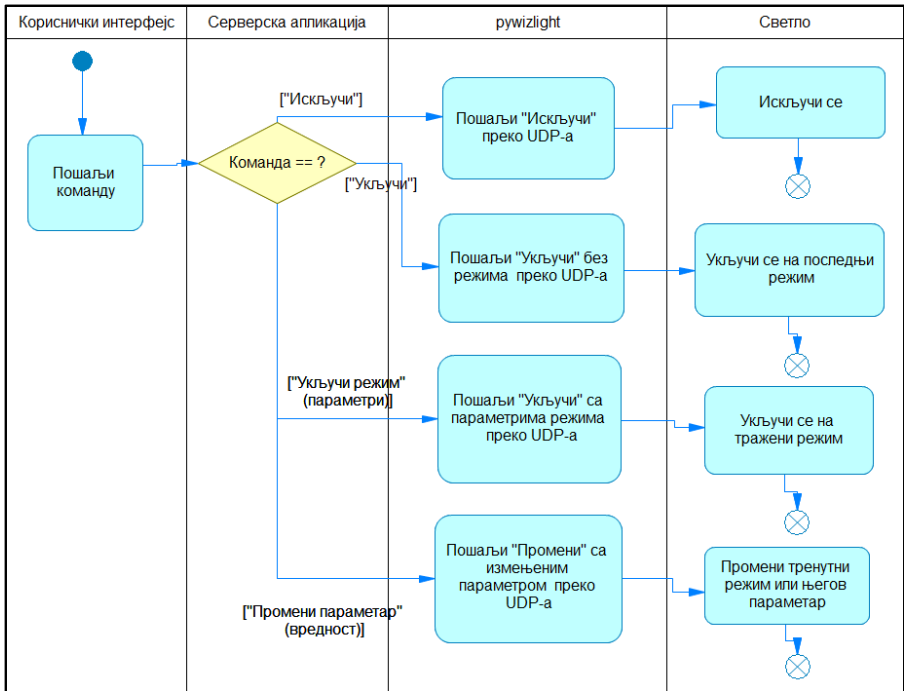
Репозиторијуми су софтверске компоненте надлежне за комуникацију са базом података. Типично се једна инстанца репозиторијума везује за једну класу објеката у систему, али то није правило (нпр. репозиторијум сцена (*Scene repository*) се везује за класу сцене (*Scene*), конфигурације светла у сцени (*SceneLightConfig*) и стања светла у сцени (*StateConfig*), зато што последње две класе немају смисла изван контекста сцене). Поред извршавања операција над објектима базе података, надлежне су за успостављање и прекид комуникационог канала са базом након извршене операције, иако се ово често апстрахује користећи библиотеке за објектно-релационо мапирање.

Репозиторијуми које серверска апликација садржи се мапирају на претходно описане контролере и сервисе. Контролер и сервис ауторизације представљају изузетак, зато што се њима не придружује и репозиторијум ауторизације. Разлог томе јесте што се ауторизација корисника врши без кориснички дефинисаних лозинки. Лозинка за пријаву је јединствена на нивоу апликације и чува се у склопу сервиса за ауторизацију, чиме се уклања потреба за комуникацијом са базом података.



### 4.2.3 Интеракција са светлима

На слици 5 је приказан дијаграм активности управљања светлима.



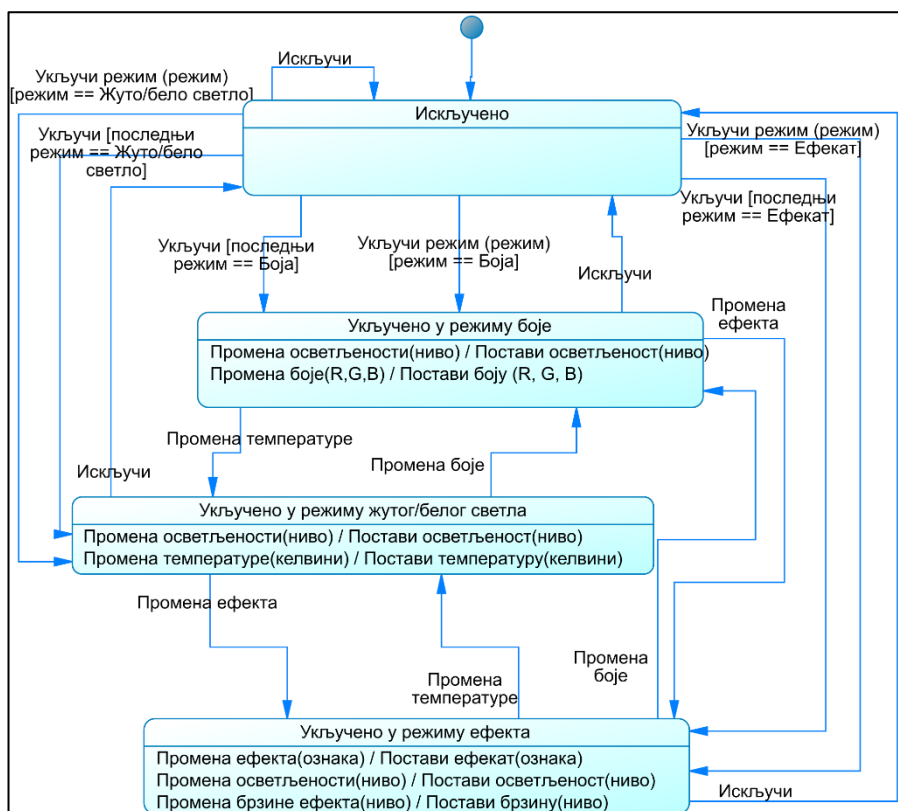
Слика 5 Дијаграм активности управљања светлима

Корисник интерагује са елементима корисничког интерфејса, што резултује слањем управљачких команди ка серверској апликацији. Серверска апликација разликује тип команде према путањи са које је пристигла (*websocket* путања која представља засебан канал комуникације) и изгледу пристиглог *DTO* објекта команде.

Постоји јасна разлика између два типа укључивања: са и без дефинисаног режима. Уколико корисник укључи светло без подешавања параметара режима (преко прекидача на корисничком интерфејсу), светло се укључује на последњи запамћени режим у којем је било пре искључивања. Приликом активације сцене, режим светла и његови параметри су већ

предефинисани и запамћени у бази података, тако да се светлу шаље команда укључивања са прочитаним параметрима режима.

Када корисник подешава параметре режима, команде које се шаљу се не тумаче као промене статуса рада светла (укључено или искључено), већ само као промене режима рада или изгледа тренутног режима. Да ли ће доћи до промене режима или само изгледа тренутног зависи од команде која се шаље. На слици 6 је приказан дијаграм прелаза стања светла у зависности од команде која му је пристигла.



Слика 6 Дијаграм прелаза стања светла

Иницијално стање светла је искључено. Из свакој другог режима је могуће искључити светло, док искључивање светла уколико је оно већ у том стању (могуће током активације сцене)

не резултује променом. У зависности од последњег запамћеног режима, укључивање без параметара ће поставити режим рада на један од приказаних. Укључивање са параметрима директно поставља режим рада на тражени. Уколико се светлу мењају параметри везани за тренутно активан режим, светло остаје у том режиму, мењајући само свој изглед. У супротном, у зависности од параметра који је промењен и невезан за тренутни, мења се режим рада на један од два преостала.

На пример, уколико је тренутно активан режим жутог/белог светла, и светло добије команду за промену боје у црвену, светло прелази у режим боје и светли црвено. Уколико је након тога пристигла команда за промену боје у зелену, светло остаје у том режиму, мењајући само боју светлости на зелену. Уколико корисник пошаље команду за искључивање, последњи запамћени режим постаје режим зелене боје и светло се искључује. При поновном укључивању, светло ће бити у режиму зелене боје и светлеће зелено.

Команда промене осветљености никад не доводи до промене режима рада, већ само изгледа тренутно активног. Промена брзине ефекта је могућа само уколико је активан ефекат коме је могуће подесити брзину, стога ни он не може довести до промене режима рада.



## 5. ИМПЛЕМЕНТАЦИЈА

У овом поглављу ће бити представљена имплементација битних функционалности система, које прате типичан ток коришћења апликације. У поглављу 5.1 се налазе описи помоћних функција и дељених понашања функционалности апликације, а у поглављу 5.2 функционалности система. Клијентски део апликације се неће разматрати, с обзиром да не садржи кључну логику система и служи само за приказ и слање података и команди на серверски део. Интеракција са корисничким интерфејсом ће бити демонстрирана у поглављу 6.

### 5.1 Помоћне функције

На листингу 1 се налази код функције за парсирање тела пристиглог захтева.

```
def request_parser(schema, object_class):
    json_data = request.get_json()
    try:
        data = schema.load(json_data)
    except ValidationError as err:
        raise
    ValidationException(format_errors(err.messages
    ), 400)
    return object_class(**data)
```

Листинг 1 Функција за парсирање тела пристиглог захтева

Функција прима параметар `schema` који представља инстанцу класе која наслеђује класу `Schema`. `Schema` је класа која је доступна из библиотеке *marshmallow* [35], која је коришћена за валидацију пристиглих података у телу захтева. Други параметар `object_class` представља класу у коју ће се пристигли подаци претворити, уколико је валидација успешно прошла. `request` је објекат доступан у оквиру *Flask* радног оквира и садржи све релевантне информације о пристиглом

захтеву, укључујући путању са које је пристигао, параметре путање који су прослеђени, информације у заглављу захтева и тело захтева. Функција прихвата тело захтева у *JSON* формату и на основу прослеђене шеме валидира податке. Уколико валидација није била успешна, шаље се порука о грешци за поља која нису правилно прослеђена. У супротном, креира се и враћа објекат прослеђене класе. Функција *format\_errors* је такође помоћна функција која прима објекат који садржи све валидационе грешке шеме и форматира их зарад бољег приказа и парсирања на клијентском делу апликације.

Сваки пристигли захтев изузевши пријаву и проверу ауторизације мора претходно да се ауторизује. На листингу 2 је приказана логика провере ауторизације корисника након што је захтев пристигао.

```
@app.before_request
def before_request_func():
    if request.method == 'OPTIONS':
        return '', 204
    if 'enter' in request.endpoint or 'check' in
request.endpoint:
        return
    if not entry_service.authorized:
        return "Session expired! Please enter
password again.", 401
```

Листинг 2 Провера ауторизације пристиглог захтева

На сваки пристигли захтев, проверава се да ли је пристигао захтев за проверу *CORS* (*Cross-Origin Resource Sharing*) правила, која дефинишу који захтеви и са које локације су дозвољени. *OPTIONS* је *HTTP* метода којом се ово проверава, и апликација увек дозвољава овај тип методе. Уколико је пристигао захтев за пријавом (*'enter'* се налази у путањи) или провером да ли је корисник ауторизован (*'check'* се налази у путањи), корисник не мора бити ауторизован. У супротном, за све остале захтеве важи да им није дозвољено обрађивање све док корисник није пријављен, што се осликава у стању

вредности статичке променљиве `authorized` у оквиру сервиса ауторизације.

Слично овоме, све поруке пристигле кроз *websocket* комуникациони канал се такође ауторизују. На листингу 3 се налази имплементација декоратора (специјалне функције написане зарад проширивања могућности других функција, пише се изнад заглавља функције коју проширује са знаком `@` пре назива) за ауторизацију *websocket* захтева.

```
def authorize(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not entry_service.authorized:
            disconnect()
            return False
        return f(*args, **kwargs)
    return decorated_function
```

Листинг 3 Ауторизација *websocket* захтева

Комуникациони канал се прекида (`disconnect()`) уколико корисник није ауторизован. У супротном, позива се функција за обраду захтева са прослеђеним параметрима.

Приликом процеса претраге доступних светала на мрежи, потребно је познавати *broadcast* адресу мреже на коју је рачунар повезан. Ово је специјална резервисана адреса мреже помоћу које се рачунар може обратити свим уређајима на њој, укључујући и паметна светла. Да се ова адреса не би ручно рачунала и проналазила, постоји помоћна функција за то, представљена на листингу 4.

```
def get_broadcast_address():
    gateways = netifaces.gateways()
    default_gateway = gateways.get('default')
    if default_gateway is None:
        return None
    default_interface =
    default_gateway[netifaces.AF_INET][1]
    addresses =
```

```

netifaces.ifaddresses(default_interface)
    if netifaces.AF_INET in addresses:
        for addr_info in
addresses[netifaces.AF_INET]:
            if 'broadcast' in addr_info:
                return addr_info['broadcast']
return None

```

Листинг 4 Функција за проналажење *broadcast* адресе мреже

Најпре се проналази подразумевани (*default*) *gateway* мреже (`default_gateway = gateways.get('default')`), који омогућује проналажење тренутног мрежног интерфејса на који је рачунар повезан (`default_gateway[netifaces.AF_INET][1]`). За мрежни интерфејс се везују *IP* адресе за фамилије адреса верзије 4 (*IPv4*) или 6 (*IPv6*), при чему за рад са светлима је коришћена фамилија верзије 4 (`netifaces.AF_INET`). Уколико мрежни интерфејс има *IP* адресу верзије 4, преузима се и враћа *broadcast* адреса мреже. Уколико рачунар није повезан на мрежу, или је повезан на мрежу која нема постављену *broadcast* адресу (*VPN* мреже, *IPv6* мреже,...) функција ће вратити `None`. Функција се ослања на библиотеку *netifaces* [36], која служи за претрагу мрежних интерфејса рачунара и њихових параметара.

*pywizlight* библиотека долази са познатим проблемима, који проузрокују интерне грешке у извршавању кода за управљање светлима. Како би се тај код правилно извршио, направљена је посебна функција која служи за правилан опоравак програма од грешке када се она деси. Прослеђује јој се функција са њеним параметрима, а враћа се њена повратна вредност, уколико је правилно извршена. Код функције је дат на листингу 5.

```

async def __wrapper(function, **kwargs):
    try:
        ret_val = await function(**kwargs)
    try:
        del wizlight.__del__
    except AttributeError:
        pass

```



```

        return ret_val
    except WizLightConnectionError:
        pass

```

Листинг 5 Функција за извршавање управљачких команди

## 5.2 Функционалности

Функционалности које ће се разматрати су:

- пријава корисника,
- креирање нове некретнине,
- регистрација нових светала,
- измена светла,
- укључивање/искључивање светала,
- промена режима и параметара рада светла,
- креирање сцене,
- активација сцене и
- брисање некретнине.

На листингу 6 је приказана функција за прихватање захтева за пријаву у контролеру ауторизације.

```

@entry_blueprint.route('', methods=['POST'])
def enter():
    try:
        data = request_parser(passwordSchema,
                                Password)
        message = entry_service.enter_app(data)
        return message, 200

    except ValidationException as ex:
        return ex.message, ex.status_code

```

Листинг 6 Прихватање захтева за пријаву у контролеру

Декоратор изнад назива функције означава да захтев долази са путање */enter* (претходно је дефинисан префикс за све путање које се везују за контролер ауторизације као */enter*, у овом случају путања нема наставак) и коришћена *HTTP* метода је *POST*. Валидира се пристигли захтев, валидирана лозинка се шаље на обраду у сервис ауторизације и уколико је ауторизација

успешна, шаље се порука о успеху са статусом 200 (*OK*). У супротном, враћа се порука о грешци и статус из фамилије клијентских грешака (бројеви од 400-500, типично 404 - *Not Found* (тражени ресурс не постоји) и 400 – *Bad Request* (грешка у подацима или недозвољена акција)).

У даљем тексту се неће наводити листинг са изгледом кода за прихватање *HTTP* захтева у контролеру, зато што је за сваку функционалност скоро идентичан приказаном. Разликују се само по путањи, методу и шеми која се валидира. За *websocket* захтеве ће из истих разлога такође бити приказан само један пример.

На листингу 7 је приказана логика за пријаву корисника на систем у сервису ауторизације.

```
def __cancel_authorization():
    global authorized
    authorized = False

def enter_app(password: Password):
    global authorized
    if
bcrypt.checkpw(str.encode(password.password),
entry_password):
    authorized = True
    scheduler.add_job(__cancel_authorization,
'date', run_date=datetime.now() +
timedelta(days=1))
    return "Successful authorization!"
    else:
        raise ValidationException("Incorrect
password!", 401)
```

Листинг 7 Пријава корисника

Лозинка се чува у шифрованом облику у променљивој `authorized`. Алгоритам коришћен за шифровање је *bcrypt* [37]. Након успешне пријаве (прослеђена лозинка и лозинка за ауторизацију која се чува (`entry_password`) се поклапају), покреће се позадински процес који ће за 24 часа од момента

пријаве „одјавити“ корисника, тј. поставити вредност променљиве `authorized` на `False`.

На листингу 8 је приказана логика додавања нове некретнине и његове прве просторије у сервису некретнина.

```
def add(new_house: NewHouse):
    house_added =
    house_repository.add(new_house.houseName)
    _ = room_repository.add(new_house.roomName,
    house_added)
    return "House successfully added!"
```

Листинг 8 Додавање нове некретнине

Логика је једноставна и подразумева основну операцију додавања у бази података. На листингу 9 и 10 је редом приказана логика додавања нове некретнине и нове просторије у њој у базу података.

```
def add(house_name: str):
    house = House(name=house_name)
    db.session.add(house)
    db.session.commit()
    return house
```

Листинг 9 Додавање некретнине у базу података

```
def add(room_name: str, house: House):
    room = Room(name=room_name)
    house.rooms.append(room)
    db.session.commit()
    return room
```

Листинг 10 Додавање просторије у базу података

Коришћење објектно-релационог мапера је свело операције над редовима табела у бази на операције над објектима класа. У овом случају то су класе `House` и `Room`, чија се дефиниција атрибута и односа са другим класама пресликава на колоне и стране кључеве табела у бази података. Према томе, приступ

колонама у табели се пресликава на приступ атрибутима класе (*Room(name=...)* и *House(name=...)*), а приступ објектима из повезаних табела се пресликава на приступ листи објеката друге класе (*rooms* листа у *house* објекту). Свако извршење акција над базом података се мора потврдити (*commit()*) на нивоу једне сесије (*db.session*).

Након што постоји некретнина и просторија у некретнини, могуће је регистровати светла. На листингу 11 је приказана логика проналажења доступних светала на мрежи, што је почетни корак у процесу регистрације светла.

```
async def discover():
    with current_app.app_context():
        all_lights = light_repository.get_all()
        lights_initialized = []
        broadcast_address = get_broadcast_address()
        lights = await
discovery.discover_lights(broadcast_space=broadcast_addr
ess)
    for light in lights:
        if any(light.mac in
vars(existing_light).values() for existing_light in
all_lights):
            continue
        light_type = await light.get_bulbtype()
        light_initialized = NewLight(mac=light.mac,
                                     ip=light.ip,
                                     name='',
                                     type='',
brightnessChange=light_type.features.brightness,
colorChange=light_type.features.color,
temperatureChange=light_type.features.color_tmp,
minKelvin=light_type.kelvin_range.min,
maxKelvin=light_type.kelvin_range.max,
                                     roomId=0)

        lights_initialized.append(light_initialized)
    ...
```

```
return lights_initialized
```

#### Листинг 11 Проналажење светала на мрежи

Најпре се добављају сва претходно регистрована светла. Проналажење светала на мрежи се омогућује коришћењем функције `discovery.discover_lights()` доступне из *pywizlight* библиотеке. Уколико се на мрежи пронађе светло са *MAC* адресом која већ постоји међу регистрованим светлима, она се неће укључити у листу пронађених светала. Уколико светло није претходно регистровано, добављају се његове физичке карактеристике (`light.get_bulbtype()`, такође из библиотеке *pywizlight*), прави се објекат са параметрима светла и додаје у листу. Та листа се враћа у одговору захтева, након чега се приказује у клијентској апликацији одакле се прелази на други корак регистрације, приказан на листингу 12.

```
def add(light: NewLight):
    global all_lights
    room =
    room_repository.get_by_id(light.roomId)
    if room is None:
        raise ValidationException("Room with the
specified ID does not exist!", 404)
    else:
        try:
            new_light =
            light_repository.add(light, room)
            all_lights.append(new_light)
            return "Light successfully added!"
        except Exception:
            raise ValidationException("Light
with the specified MAC address already exists!",
404)
```

#### Листинг 12 Додавање светла

Ради повећања отпорности система на грешке, проверава се постојање просторије са прослеђеним идентификационим бројем и светла са прослеђеном *MAC* адресом. Клијентска апликација враћа исти објекат који је добио у претходном кораку са попуњеним атрибутима имена, типа и просторије којој светло

припада. Светло се након тога додаје у базу података, али и у локалну листу свих светала система. Ова листа омогућује да позадински процеси који периодично ажурирају *IP* адресе светала и добављају њихово тренутно стање не морају приступати бази података зарад њиховог добављања. Клијентска апликација заправо враћа листу попуњених објеката које је добио, након чега се за свако светло у листи позива функција са листинга 12.

На листингу 13 је приказана логика измене података о светлу.

```
def modify(modifications: ModifyLight):
    light =
light_repository.get_by_mac(modifications.mac)
    if light is None:
        raise ValidationException("Light with the
specified MAC address does not exist!", 404)
    else:
        if light.room_id != modifications.roomId:
            new_room =
room_repository.get_by_id(modifications.roomId)
            if new_room is None:
                raise ValidationException("Room with the
specified ID does not exist!", 404)
            else:
                room_repository.move_light(new_room,
light)
        light_repository.modify(light,
modifications.name)
        return "Light successfully modified!"
```

Листинг 13 Измена светла

Измена светла подразумева слање *MAC* адресе светла, новог имена и идентификационог броја просторије којој ће припасти, при чему се њихово постојање у бази података проверава (важи за измене било ког објекта – модификацији атрибута објекта увек претходи провера постојања). Уколико је измењена просторија којој светло припада, ажурира се листа светала за прослеђену просторију

(room\_repository.move\_light(new\_room, light)). Назив светла се свакако ажурира.

Након регистрације светала, кориснику су доступне функционалности управљања. Управљање светлима се одвија преко претходно успостављене *websocket* везе. На листингу 14 је приказана функција за прихватање захтева за укључивање светла у контролеру светала.

```
@socket.on('turn_on')
@authorize
def turn_on(light_ip):
    asyncio.run(light_service.turn_on(light_ip))
```

Листинг 14 Прихватање захтева за укључивање светла

Функција се окида уколико је пристигла порука кроз ‘turn\_on’ канал којим је прослеђена *IP* адреса светла које треба укључити. Захтев проверава да ли је корисник претходно ауторизован (позивање декоратора @authorize описаног у поглављу 5.1). Акције које подразумевају комуникацију са светлима су асинхроне, стога се увек позивају из функције asyncio.run() како би се правилно сачекао крај извршења. У даљем тексту се неће наводити изглед функција за прихватање *websocket* захтева, зато што су скоро идентичне приказаној, разликујући се само у називу канала и команде која се извршава.

Листинг 15 приказује логику слања команде укључивања ка светлу, чиме оно мења свој статус рада.

```
async def turn_on(ip: str):
    async def execute(ip: str):
        light = wizlight(ip)
        await light.turn_on(PilotBuilder())
    await __wrapper(execute, ip=ip)
```

Листинг 15 Укључивање светла

wizlight је класа доступна из *pywizlight* библиотеке и прима *IP* адресу светла коме се шаље команда. PilotBuilder је такође класа доступна из библиотеке и служи за постављање режима и његових параметара. Пошто нису прослеђени параметри, светло се укључује на последње запамћено стање.

Слично укључивању, светло се искључује позивањем команде `await light.turn_off()` уместо `await light.turn_on(PilotBuilder())` на листингу 15.

Управљање режимом рада и параметрима подразумева слање објекта команде (Command), чије се вредности тумаче и на основу тих закључака одлучује да ли ће се мењати режим, или његови параметри. Са аспекта *pywizlight* библиотеке, ово подразумева слање исте команде (`light.turn_on()`) са разликом у PilotBuilder објекту који се прослеђује. Логика за слање команде је приказана на листингу 16.

```
async def trigger_command(command: Command):
    async def execute(command: Command):
        light = wizlight(command.ip)
        if command.r > -1 and command.g > -1 and
command.b > -1:
            await
light.turn_on(PilotBuilder(rgb=(command.r,
command.g, command.b)))
        elif command.mode > -1:
            await
light.turn_on(PilotBuilder(scene=command.mode))
        elif command.temperature > -1:
            await
light.turn_on(PilotBuilder(colortemp=command.tem
perature))
        elif command.brightness > -1:
            await
light.turn_on(PilotBuilder(brightness=command.br
ightness))
        elif command.speed > -1:
            await light.send({"method":
"setPilot", "params": {"speed": command.speed}})
        else:
```



```

        pass
    await __wrapper(execute, command=command)

```

Листинг 16 Слање команде за промену стања светла

-1 је ознака да прослеђени параметар није промењен. У зависности од параметара који нису постављени на ту вредност, мења се режим или његов изглед према претходно описаној логици у поглављу 4.2.3. Због познатог проблема са мењањем брзине ефекта у библиотеци, уместо очекиване `light.turn_on(PilotBuilder(speed=command.speed))` команде је искоришћена еквивалентна алтернатива која је приказана на листингу 16.

Када постоје регистрована светла, постоји могућност креирања сцене. На листингу 17 је приказана логика креирања нове сцене.

```

def add(scene: NewScene):
    lights_config = []
    room = room_repository.get_by_id(scene.roomId)
    if room is None:
        raise ValidationException("Room with the
specified ID does not exist!", 404)
    else:
        for light_config in scene.config:
            light_config =
LightColorConfigBasic(**light_config)
            light =
light_repository.get_by_mac(light_config.light_mac)
            if light is None:
                raise ValidationException("Light with
the specified MAC address does not exist!", 404)
            else:
                light_config.config =
ColorOrModeConfigBasic(**light_config.config)
                config = SceneLightConfig(light=light,
state=StateConfig(r=light_config.config.r,
g=light_config.config.g,
b=light_config.config.b,
brightness=light_config.config.brightness,

```

```
speed=light_config.config.speed,
temperature=light_config.config.temperature,
mode=light_config.config.mode))
        lights_config.append(config)
        scene_repository.add(room,
Scene(name=scene.name, lightsConfig=lights_config))
    return "Scene successfully added!"
```

```

        await
light.turn_on(PilotBuilder(colortemp=config.state.temperature,
        brightness=config.state.brightness))
        if config.state.mode != -1:
            if config.state.speed != -1:
                await
light.turn_on(PilotBuilder(scene=config.state.mode,
        brightness=config.state.brightness,
        speed=config.state.speed))
            else:
                await
light.turn_on(PilotBuilder(scene=config.state.mode,
        brightness=config.state.brightness))
        await __wrapper(execute, id=id)

```

Листинг 18 Активација сцене

Проверава се да ли постоји сцена са прослеђеним идентификационим бројем. Логика слања команде ка светлу је слична оној за промену стања светла представљена на листингу 16. Активација сцене проузрокује укључивање светала у предефинисани режим рада (дефинисан приликом креирања сцене), чиме се команди која се шаље светлу придружују групе параметара, уместо појединачних (нпр. температура и осветљеност или ефекат, осветљеност и брзина).

На листингу 19 је приказана логика брисања некретнине са прослеђеним идентификационим бројем.

```

def delete(house_id: int):
    house = house_repository.get_by_id(house_id)
    if house is None:
        raise ValidationException("House with
the specified ID does not exist!", 404)
    else:
        house_repository.delete(house)
        return "House successfully removed!"

```

Листинг 19 Брисање некретнине

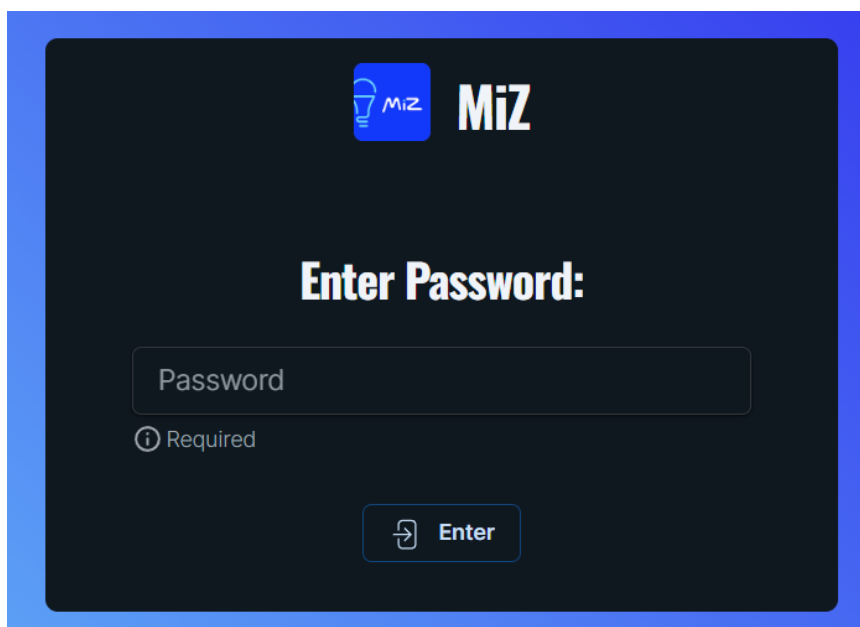
Брисање свих ентитета се врши на основу прослеђеног идентификационог броја. Брисању следи провера постојања

објекта са задатим бројем. Након тога брисање се врши у бази података, где у зависности од објекта који се брише може доћи до последичног (каскадног) брисања објеката повезаних са њим. У примеру некретнине, брисање ће проузроковати и брисање свих просторија, светала у тим просторијама и сцена које су асоциране са њима. Ово се ради аутоматски, на основу модела шеме базе података дефинисаног преко модела класа.

## 6. ДЕМОНСТРАЦИЈА

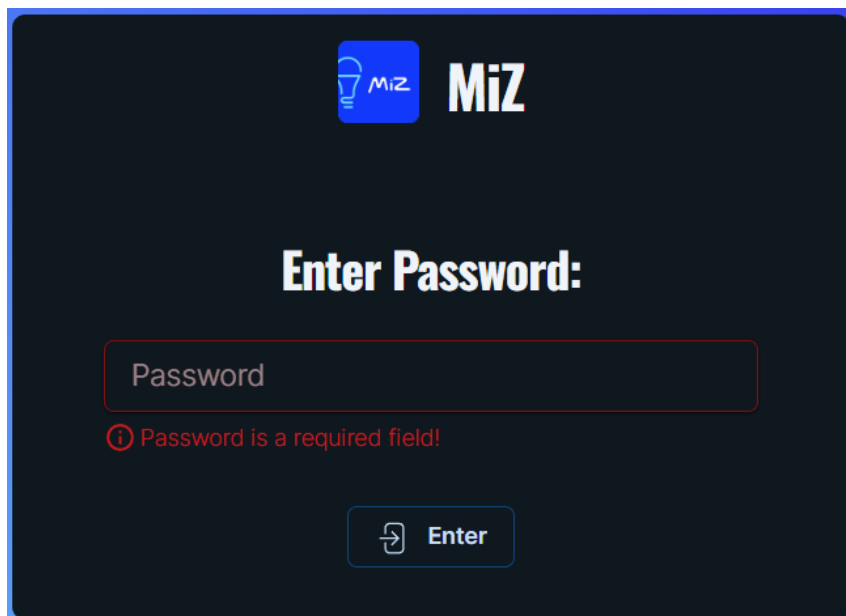
У овом поглављу ће бити представљена демонстрација коришћења апликације и њених функционалности описаних у претходним поглављима.

На слици 7 је приказана форма за пријаву корисника.

The image shows a login interface for 'MiZ'. At the top, there is a logo consisting of a lightbulb icon and the text 'MiZ'. Below the logo, the text 'Enter Password:' is displayed in a large, bold font. Underneath this text is a text input field with the placeholder text 'Password'. Below the input field, there is a small information icon (an 'i' in a circle) followed by the word 'Required'. At the bottom of the form, there is a button with a right-pointing arrow icon and the text 'Enter'.

Слика 7 Форма за пријаву корисника

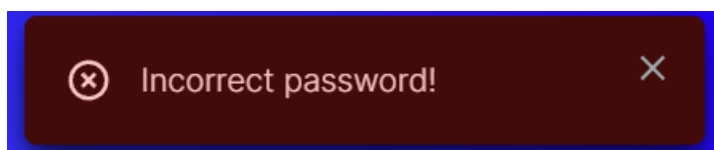
Корисник се преусмерава на ову форму сваки пут када је потребно да се пријави (на почетку коришћења и сваки следећи пут када га систем одјави). Поље које је обавезно се увек означава са *Required* ознаком испод уноса. Уколико корисник притисне дугме за слање лозинке (*Enter*) и валидација поља није успешна (поље је празно), поље се означава црвеном бојом и на месту ознаке *Required* се приказује порука да је поље потребно попунити. На слици 8 је приказан изглед форме у том случају.

The image shows a dark-themed login interface. At the top left is a logo with a lightbulb icon and the text 'MiZ'. To its right is the text 'MiZ' in a large, bold, white font. Below this, the text 'Enter Password:' is centered in a bold, white font. Underneath is a rectangular input field with a red border and the placeholder text 'Password'. Below the input field, a red error message 'Password is a required field!' is displayed, preceded by a red circular icon containing a white 'i'. At the bottom center is a button with a white right-pointing arrow icon and the text 'Enter'.

Слика 8 Изглед форме са неуспешном валидацијом поља

У општем случају, за сва поља која имају валидациону логику која није задовољена, приказиваће се порука са информацијом како треба попунити поље.

Уколико је корисник унео лозинку и послао захтев за ауторизацијом, али лозинка није исправна, приказује се искакајућа (*popup*) порука о насталој грешци у горњем десном углу екрана. Пример изгледа поруке за неисправну лозинку је приказан на слици 9.

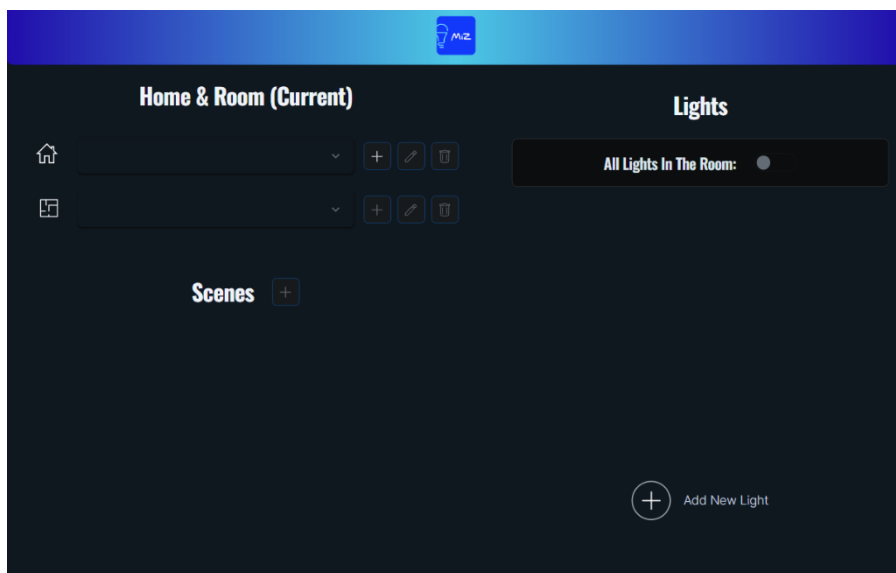


Слика 9 Искакајућа порука грешке

Грешку је могуће одмах одстранити са екрана притиском знака ×, иако ће се то свакако десити аутоматски након 5

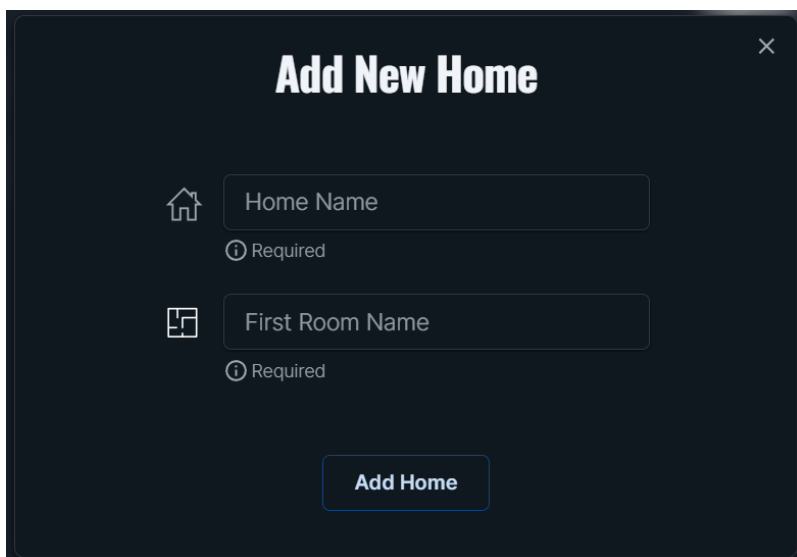
секунди. У општем случају, за све настале грешке које нису везане за валидацију уноса, приказиваће се искакајућа порука са описом грешке.

Након успешне пријаве, корисник се преусмерава на главну страницу, одакле има преглед свих некретнина и њихових просторија (*Home & Room (Current)*), светала у тој просторији (*Lights*) и сцена (*Scenes*) које постоје. На слици 10 је приказан изглед странице на почетку коришћења, када још не постоји регистрована некретнина.



Слика 10 Главна страница

Кликом на дугме означено симболом + у линији са симболом куће се отвара дијалог за креирање нове некретнине, приказан на слици 11.



**Add New Home**

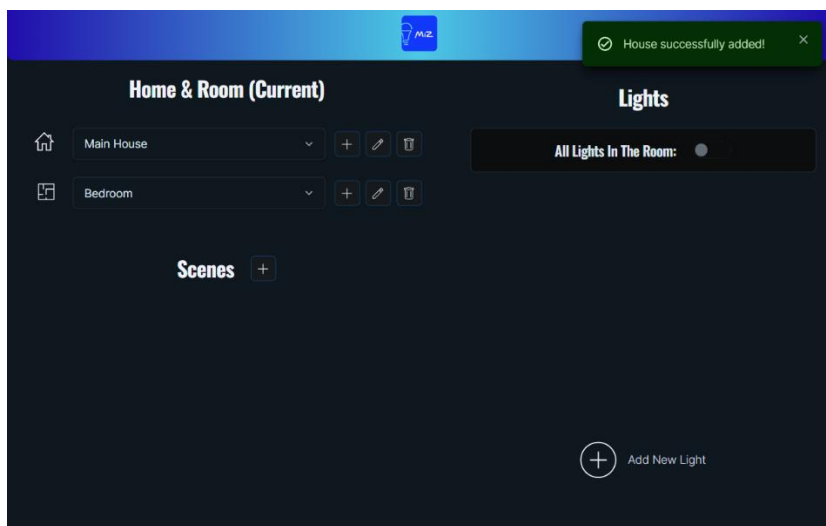
Home Name  
Required

First Room Name  
Required

**Add Home**

Слика 11 Дијалог за креирање нове некретнине

Креираће се некретнина под називом *Main House* и просторијом под називом *Bedroom*. Кликом на дугме *Add Home* ће се освежити страница, која прелази у стање приказано на слици 12.



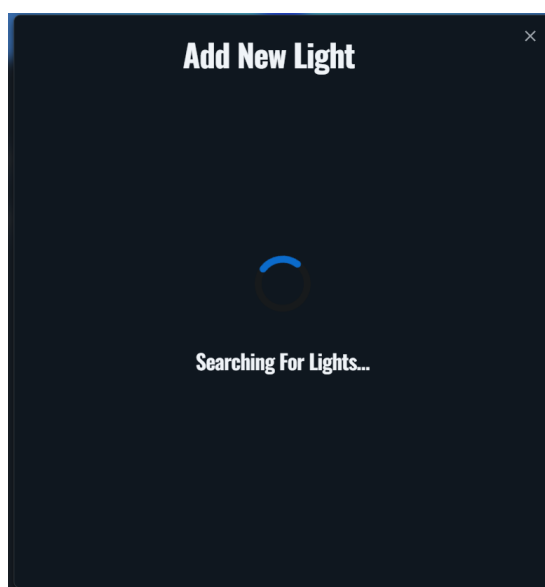
Слика 12 Главна страница након успешног додавања некретнине



У оквиру секције за некретнине и просторије се приказује новонастала некретнина и просторија. Налазе се у оквиру падајућих менија, одакле корисник може да се преусмери на друге некретнине и просторије. Приказ се мења у зависности од одабране некретнине и просторије. Уколико је промењена некретнина, мења се листа опција у падајућем менију просторија на оне које припадају тренутној. Уколико је промењена просторија, мења се приказ листе светала и сцена које јој припадају.

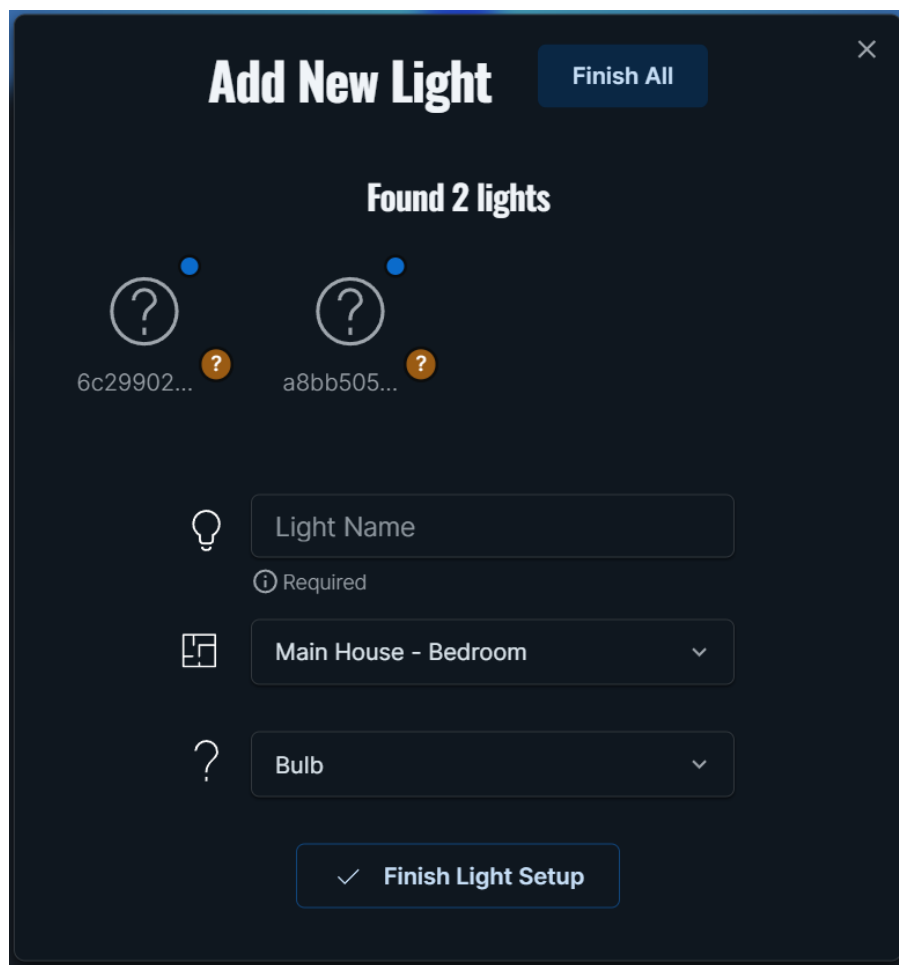
У свим случајевима (уместо акција које рукују светлима) у којима је акција успешно извршена, у горњем десном углу ће се приказивати искакајућа порука са поруком о успешном извршењу акције. За разлику од поруке о грешци која је црвене боје, ова порука је зелена.

Када постоји регистрована просторија, могуће је регистровати светла. Кликом на дугме *Add New Light* отвара се дијалог у којем је започет процес претраге светала, приказан на слици 13.



Слика 13 Почетни изглед прозора за регистрацију светала

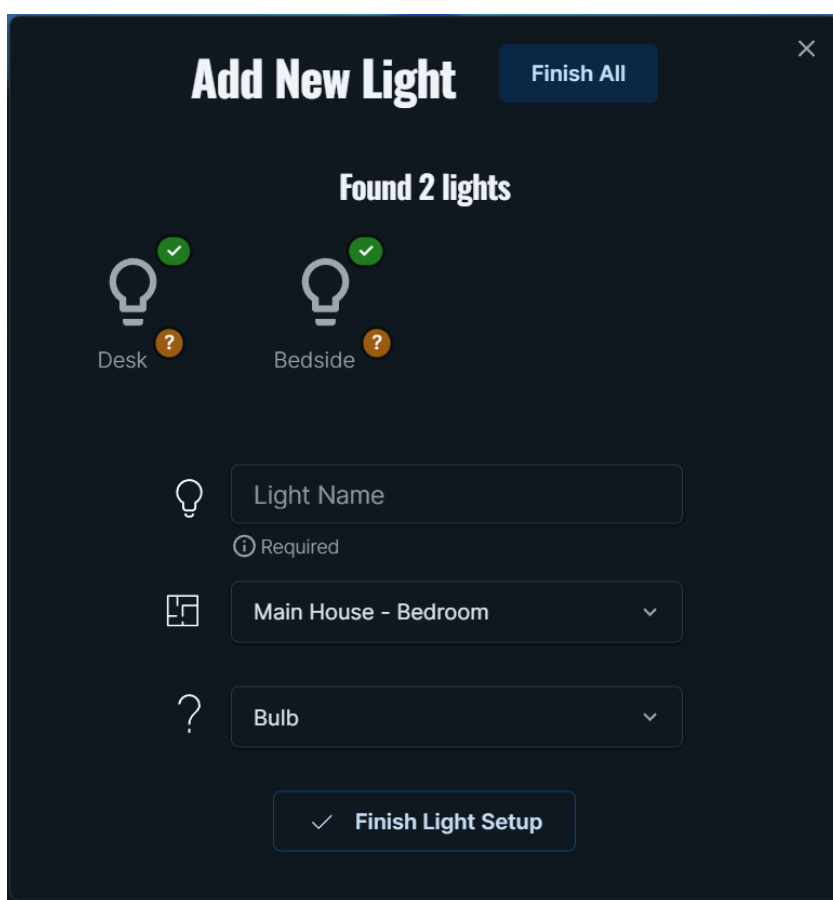
Уколико нису пронађена светла, кориснику се приказује порука и дугме за покретање поновне претраге. У супротном, приказује се листа пронађених светала и дијалог изгледа као на слици 14.



Слика 14 Дијалог за регистрацију пронађених светала

Светлима се приказује њихова *MAC* адреса, поред које се налази дугме са знаком ? које служи да се пошаље команда за укључивање тог светла. Пошто процес регистрације не познаје које физичко светло припада којој *MAC* адреси, ово је начин на

који се оно може идентификовати. Процес регистрације једног светла почиње кликом на слику или *MAC* адресу која се приказује, након чега је потребно унети назив светла (*Light Name*), селектовати просторију којој припада из падајућег менија (приказује се у формату Назив некретнине – Назив просторије) и тип светла. Кликком на дугме *Finish Light Setup* се завршава процес креирања једног светла. Назив и слика светла се мења према подацима унетим у форми. У овом примеру, региструју се два светла под називом *Desk* и *Bedside* која ће бити сијалице и налазити се у просторији *Bedroom*. Након уноса података, дијалог изгледа као на слици 15.



**Add New Light** Finish All

**Found 2 lights**

Desk Bedside

Light Name

Required

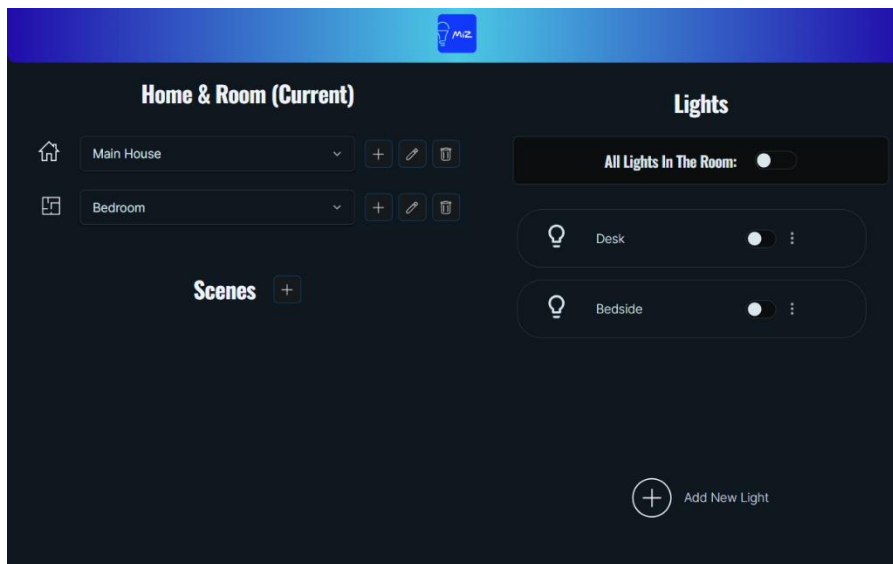
Main House - Bedroom

Bulb

✓ Finish Light Setup

Слика 15 Дијалог са конфигурисаним светлима

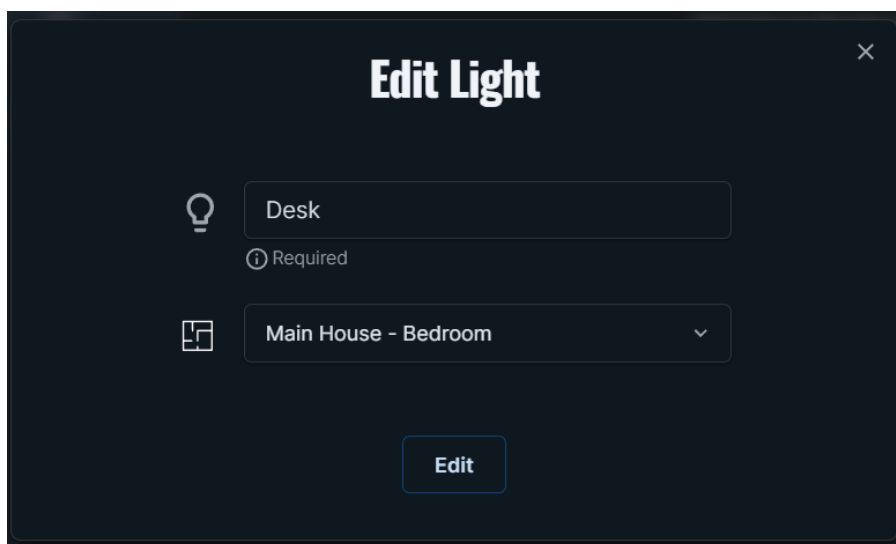
Кликом на дугме *Finish All* се освежава страница, након чега је видљива порука о успешном додавању светала, и светла постају видљива на главној страници. Њен изглед дат је на слици 16.



Слика 16 Главна страница са приказом светала

Корисник је у могућности да укључи сва светла у соби кликом на прекидач поред назива *All Lights In The Room:*. Тај прекидач и остали приказани поред назива светла мењају боју у плаво. Појединачна светла се идентично укључују преко својих прекидача. Поновни клик на прекидач га враћа у искључено стање и тиме се светла искључују. Укључивање преко прекидача подразумева укључивање светла на последњи запамћени режим. Поред слике, назива и прекидача за светло налази се симбол три тачке. Кликом на њега отвара се мени са ставкама *Edit* и *Delete* које преусмеравају корисника на дијалоге за измену и брисање светала. У општем случају, дугме означено оловком или називом *Edit* служи за измену, а дугме означено кантом за отпатке или називом *Delete* за брисање објекта.

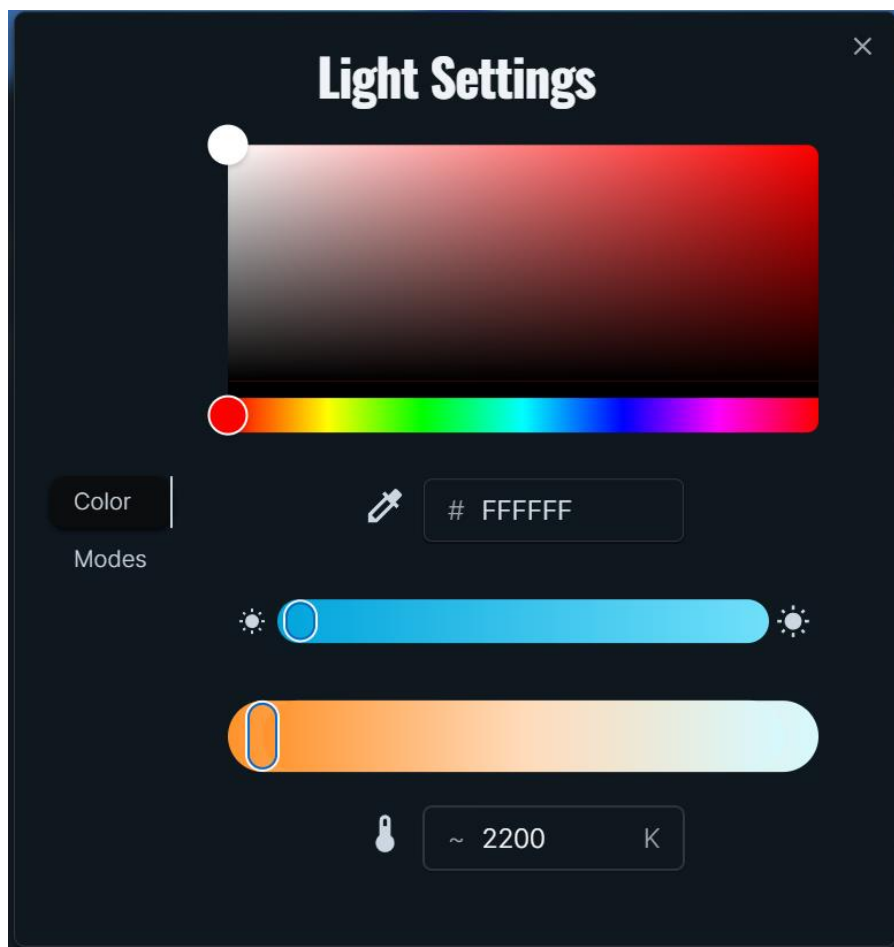
Кликом на ставку *Edit* из менија отвара се дијалог за измену података о светлу приказан на слици 17.



Слика 17 Дијалог за измену података о светлу

За сваки дијалог измене важи да су поља за унос вредности које се ажурирају претходно попуњена тренутним вредностима. У случају светла, могућа је промена назива и/или просторије којој припада. У овом примеру ће се променити само назив (*Desk EDIT*), како би се у следећим корацима могла креирати сцена са више светала. Уколико би дошло до измене просторије, светло се не би више приказивало на главној страници за одабрану просторију. Кликом на дугме *Edit* се приказ освежава, приказује се порука о успешној измени и приказује се назив *Desk EDIT* уместо *Desk* са слике 16.

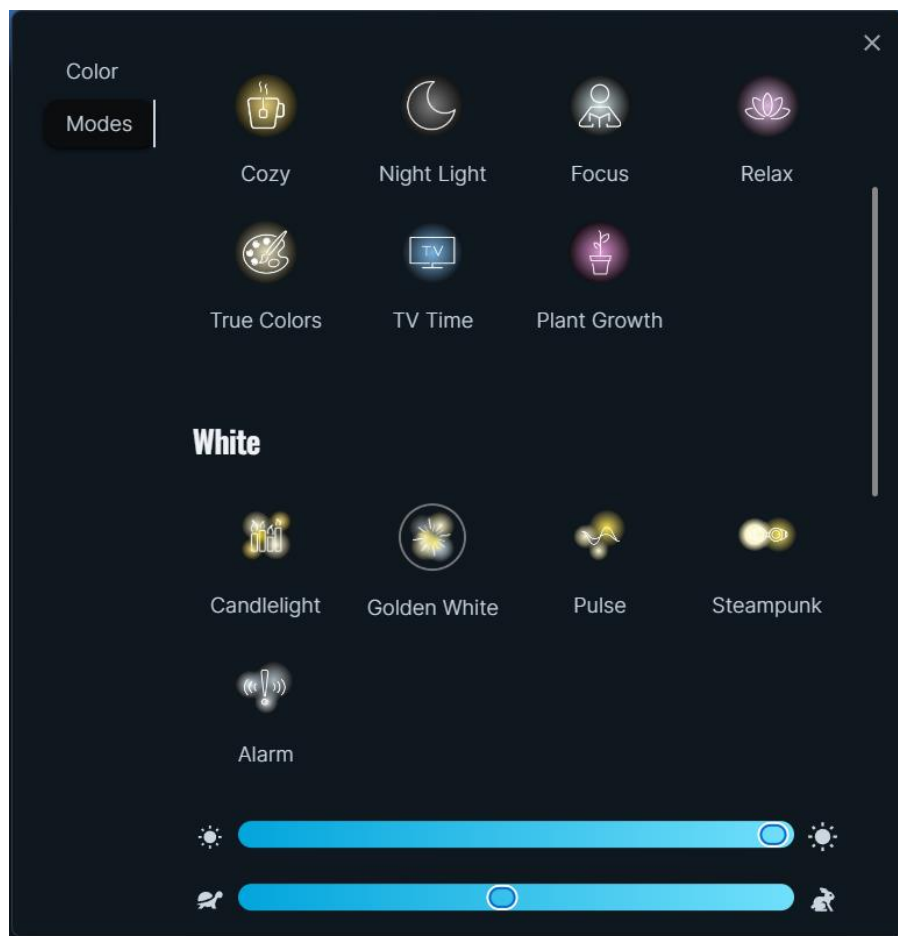
Кликом на назив светла се отвара дијалог за управљање приказан на слици 18.



Слика 18 Почетни изглед дијалога за управљање светлом

Крајња горња компонента служи за одабир боје светла. У највећем правоугаонику је могуће манипулисати нијансом једне боје, а преко клизача испод је могуће променити боју. Боја се може дефинисати и преко текстуалног поља испод, уписом хексадецималне ознаке боје. На клизачу испод је могуће променити осветљење (крајње лево – најмање, крајње десно – највише). На клизачу испод се може подесити температура жутог/белог светла, а може се и унети директно у текстуално поље испод.

Кликом на ставку *Modes* отвара се друга секција дијалога, намењена подешавању ефекта. Кликом на неку од ставки, на дну дијалога се приказују клизачи за промену осветљења и брзине у зависности да ли активан ефекат подржава промене. Дијалог у овом стању изгледа као на слици 19.

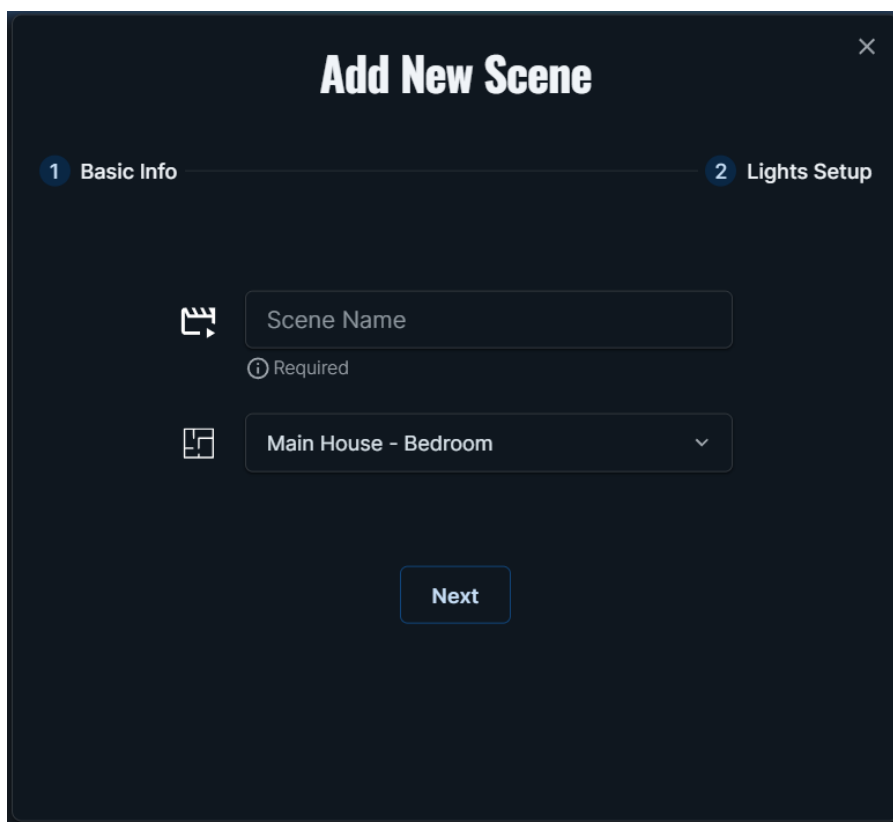


Слика 19 Дијалог за управљање светла на секцији ефеката

Активан ефекат је оивичен кружницом беле боје.

Регистрација светла омогућује креирање сцена. Кликом на дугме означено симболом + поред назива *Scene* из главног

менија отвара се дијалог за креирање нове сцене приказан на слици 20.



**Add New Scene**

1 Basic Info 2 Lights Setup

Scene Name

Required

Main House - Bedroom

Next

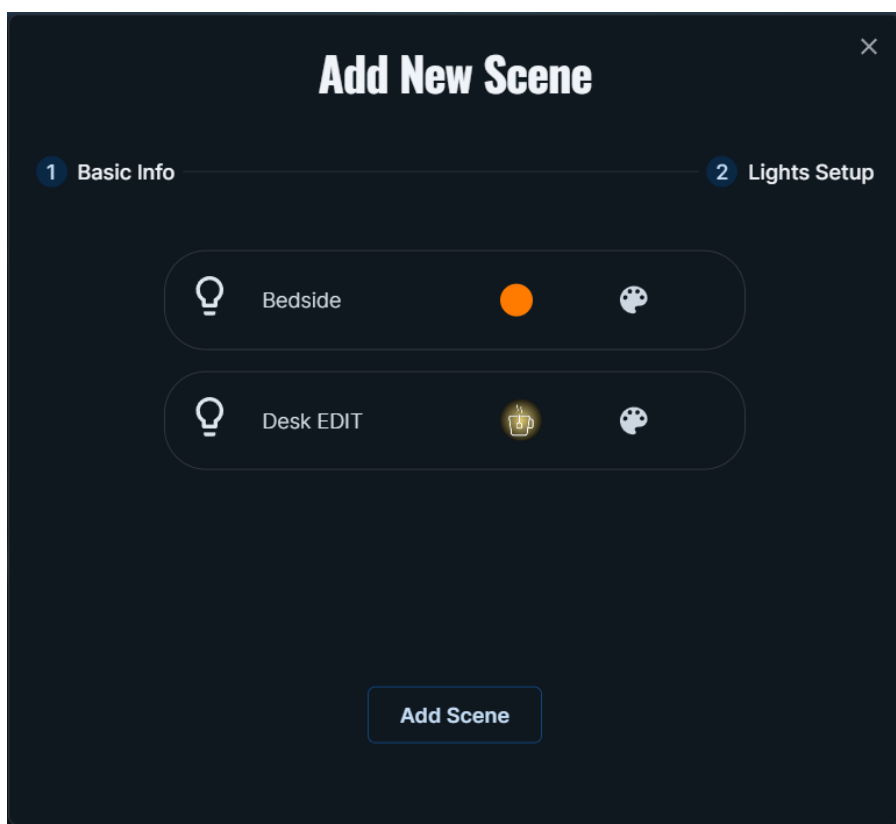
Слика 20 Почетни изглед дијалога за креирање сцене

Креирање сцене је подељено у два корака. Први корак (*Basic Info*) је унос назива сцене (*Scene Name*) и одабир просторије којој се сцена придружује. Одабир просторије утиче на листу светала које ће се приказати у другом кораку (*Lights Setup*). Није могуће прећи на други корак уколико није унето име сцене.

Кликом на дугме *Next* се прелази на други корак креирања сцене, где се приказује листа светала у одабраној просторији. Поред сваког назива светла, налази се иконица која осликава



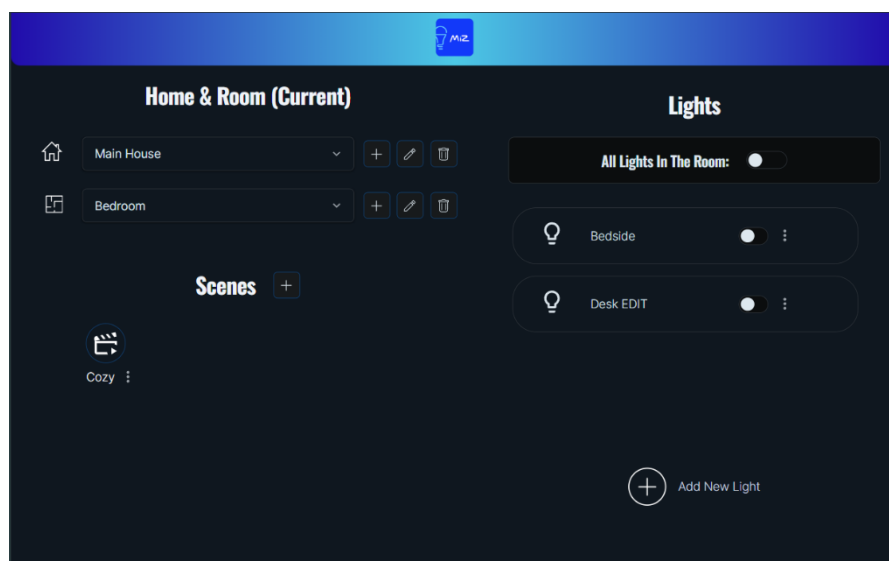
жељено стање тог светла у сцени, које је почетно постављено на *Off* вредност. Кликом на дугме означено иконицом палете се приказује идентичан дијалог приказан на сликама 18 и 19, одакле је могуће поставити параметре стања светла. У зависности од одабира, иконица стања поред назива се мења. У овом примеру, креираће се сцена под називом *Cozy* у којој ће *Desk* (*Desk EDIT* према последњој измени) светло бити постављено на ефекат под називом *Cozy*, а *Bedside* ће бити у режиму наранџасте боје. Дијалог изгледа као на слици 21.



Слика 21 Изглед дијалога за креирање сцене након подешавања светала

Кликом на дугме *Add Scene* се освежава страница, након чега је видљива порука о успешном додавању сцене. Сцена

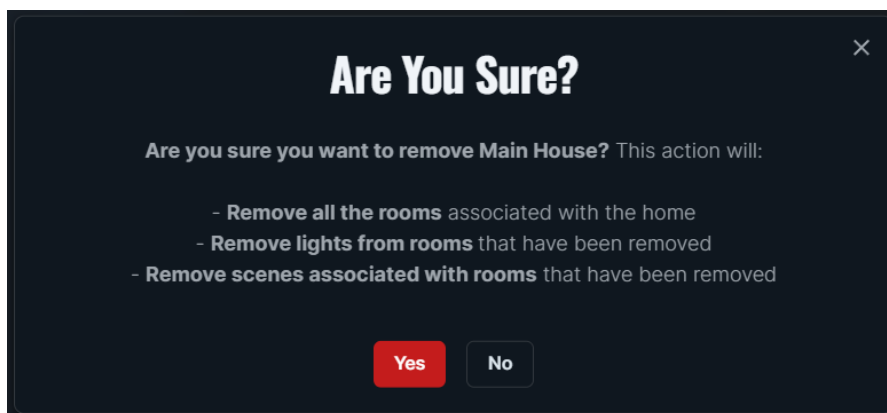
постаје видљива на главној страници. Њен изглед дат је на слици 22.



Слика 22 Главна страница са приказом сцене

Сцена се активира кликом на слику изнад њеног назива. Дугме са симболом три тачке поред назива сцене има исто значење као и код приказа светла.

Кликом на дугме означено иконицом канте за отпатке у линији са симболом куће отвара се дијалог за брисање некретнине приказан на слици 23.



Слика 23 Дијалог за брисање објекта

Сваки дијалог за брисање приказује све последице брисања објекта. У овом случају, то је брисање свих просторија, светала и сцена које су асоциране са тим просторијама. Уколико се притисне дугме *No*, брисање се неће извршити и дијалог ће се затворити. У супротном, кликом на дугме *Yes* се објекат брише, чиме се приказ освежава и главна страница прилагођава према последицама брисања. У случају брисања некретнине, главна страница изгледа идентично слици 10.



## 7. ЗАКЉУЧАК

Због све већег броја паметних кућних уређаја, произвођачи су корисницима понудили апликације за једноставно управљање њиховим производима. Један од њих је и *WiZ*, који је својим корисницима понудио мобилну апликацију за руковање паметним светлима. Решење описано у раду тежи да надомести недостатак веб апликације, како би постојала могућност руковања и помоћу рачунара.

Решење садржи све основне операције за управљање светлима, као и операције за њихово груписање и организацију. Управљање се врши преко локалне мреже, чиме је комуникација са светлима једноставнија и безбеднија. Кориснику је доступан интуитиван кориснички интерфејс у оквиру којег има приступ свим доступним операцијама. Подаци који се прикупљају се трајно складиште у бази података, чиме се обезбеђује несметани наставак рада апликације након искључења серверске апликације или корисничког интерфејса.

Решење се издваја од постојећих алтернатива као једино решење које је фокусирано на *WiZ* паметна светла. Нуди већи број функционалности од појединих представљених платформи. Нуди једноставан кориснички интерфејс прилагођен функционалностима система, дизајниран према званичној мобилној апликацији произвођача. Изглед и једноставност коришћења је главна предност решења у односу на остале платформе, које ради интеграције са већим бројем произвођача нуде генерички кориснички интерфејс тежи за коришћење.

Могуће будуће унапређење апликације је имплементација напреднијих функционалности које нуди званична мобилна апликација. Једна од њих је подршка за заказивање укључивања и искључивања светала, чиме би се могао дефинисати распоред по коме светла раде. Ово би последично омогућило креирање

циклуса рада, где би било могуће дефинисати режим рада светла током једног дана (нпр. циркадијални ритам рада светла).

Друго унапређење би било имплементација паљења светла на основу промена у мрежном сигналу светала (*SpaceSense* [38]). Овим се омогућује укључивање светала након детекције покрета без потребе других физичких сензора. Ова функционалност је још увек у раној фази развоја и за званичну мобилну апликацију, тако да се њена имплементација не може очекивати у скорије време.

Поред поменутих функционалности, други правац даљег развоја апликације укључује увођење подршке за језичку локализацију и прилагодљивост особама са општећеним видом. Ово подразумева могућност превођења апликације са изворно енглеског на друге језике, повећања величине слова и елемената корисничког интерфејса и промене палете боја апликације (за кориснике са далтонизмом).

## 8. ЛИТЕРАТУРА

- [1] <https://www.wizconnected.com>
- [2] Ghimire, D. (2020). Comparative study on Python web frameworks: Flask and Django.
- [3] <https://github.com/sbidy/pywizlight>
- [4] Gackenheimer, C. (2015). Introduction to React. Apress.
- [5] Worsley, J., & Drake, J. D. (2002). Practical PostgreSQL. " O'Reilly Media, Inc."
- [6] <https://www.home-assistant.io>
- [7] <https://community.home-assistant.io/t/homeassistant-for-newcomers-what-it-is-what-is-hassio-hassos-hassbian-101-and-cookies/123004>
- [8] Python, W. (2021). Python. Python releases for windows, 24.
- [9] <https://www.openhab.org/about/who-we-are.html>
- [10] Eckel, B. (2003). Thinking in JAVA. Prentice Hall Professional.
- [11] <https://www.openhab.org/docs/concepts>
- [12] <https://www.iobroker.net>
- [13] Flanagan, D. (2011). JavaScript: The definitive guide: Activate your web pages. " O'Reilly Media, Inc."
- [14] <https://homey.app>
- [15] <https://web.archive.org/web/20171117015927/http://flask.pocoo.org/docs/0.10/foreword>
- [16] <https://werkzeug.palletsprojects.com/en/3.0.x>
- [17] <https://jinja.palletsprojects.com/en/3.1.x>
- [18] <https://click.palletsprojects.com/en/8.1.x>
- [19] [https://en.wikipedia.org/wiki/Command-line\\_interface](https://en.wikipedia.org/wiki/Command-line_interface)

- [20] Postel, J. (1980). User datagram protocol (No. rfc768).
- [21] <https://docs.pro.wizconnected.com>
- [22] Lloyd, J. W. (1994, September). Practical Advtanages of Declarative Programming. In GULP-PRODE (1) (pp. 18-30).
- [23] Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. In ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28 (pp. 257-281). Springer Berlin Heidelberg.
- [24] Sengupta, D., Singhal, M., & Corvalan, D. (2016). Getting Started with React. Packt Publishing Ltd.
- [25] Goodman, D. (2002). Dynamic HTML: The definitive reference: A comprehensive resource for HTML, CSS, DOM & JavaScript. " O'Reilly Media, Inc."
- [26] Groff, J. R., Weinberg, P. N., & Oppel, A. J. (2002). SQL: the complete reference (Vol. 2). McGraw-Hill/Osborne.
- [27] Yu, S. (2009). Acid properties in distributed databases. Advanced eBusiness Transactions for B2B-Collaborations, 17.
- [28] <https://vitejs.dev/guide>
- [29] Fette, I., & Melnikov, A. (2011). The websocket protocol (No. rfc6455).
- [30] Berners-Lee, T., Fielding, R., & Frystyk, H. (1996). Hypertext transfer protocol--HTTP/1.0 (No. rfc1945).
- [31] Gardner, J. (2009). The web server gateway interface (wsgi). The Definitive Guide to Pylons, 369-388.
- [32] Forouzan, B. A. (2002). TCP/IP protocol suite. McGraw-Hill Higher Education.
- [33] <https://flask-sqlalchemy.palletsprojects.com/en/3.1.x>
- [34] Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., & Vrgoč, D. (2016, April). Foundations of JSON schema. In Proceedings



of the 25th international conference on World Wide Web (pp. 263-273).

- [35] <https://marshmallow.readthedocs.io/en/stable>
- [36] <https://github.com/al45tair/netifaces>
- [37] Sriramy, P., & Karthika, R. A. (2015). Providing password security by salted password hashing using bcrypt algorithm. ARPN journal of engineering and applied sciences, 10(13), 5551-5556.
- [38] <https://www.wizconnected.com/en-us/explore-wiz/spacesense>



## 9. БИОГРАФИЈА



Маја Варга је рођена 21.12.2001. у Новом Саду. Основно образовање је стекла у основној школи „Људовит Штур“ у Кисачу. Средње образовање је стекла у средњој електротехничкој школи „Михајло Пупин“ у Новом Саду на смеру Електротехничар информатичких технологија. Школске 2020/21 године се уписује на Факултет техничких наука у Новом Саду на студијски програм

Софтверско инжењерство и информационе технологије. Положила је све испите предвиђене планом и програмом и стекла услов за одбрану завршног рада. Добитница је престижне државне стипендије „Доситеја“ Фонда за младе таленте, као и приватне стипендије компаније „Levi9“ у школској 2023/24 години за успехе постигнуте током студија.



## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, <b>РБР</b> :	
Идентификациони број, <b>ИБР</b> :	
Тип документације, <b>ТД</b> :	монографска публикација
Тип записа, <b>ТЗ</b> :	текстуални штампани документ
Врста рада, <b>ВР</b> :	дипломски рад
Аутор, <b>АУ</b> :	Maја Варга
Ментор, <b>МН</b> :	др Милан Видаковић, редовни професор
Наслов рада, <b>НР</b> :	Веб апликација за руковање паметним светлима
Језик публикације, <b>ЈП</b> :	српски
Језик извода, <b>ЈИ</b> :	српски / енглески
Земља публикавања, <b>ЗП</b> :	Србија
Уже географско подручје, <b>УГП</b> :	Војводина
Година, <b>ГО</b> :	2024
Издавач, <b>ИЗ</b> :	ауторски репринт
Место и адреса, <b>МА</b> :	Нови Сад, Факултет техничких наука, Трг Доситеја Обрадовића 6
Физички опис рада, <b>ФО</b> :	7 / 60 / 0 / 0 / 42 / 0 / 0
Научна област, <b>НО</b> :	Софтверско инжењерство и информационе технологије
Научна дисциплина, <b>НД</b> :	Софтверско инжењерство
Предметна одредница / кључне речи, <b>ПО</b> :	WiZ, паметна светла, React, Flask, pywizlight
<b>УДК</b>	
Чува се, <b>ЧУ</b> :	Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад
Важна напомена, <b>ВН</b> :	
Извод, <b>ИЗ</b> :	Задатак рада представља развој веб апликације за руковање паметним светлима произвођача <i>WiZ Connected</i> . Серверски део апликације је имплементиран у <i>Python</i> програмском језику коришћењем <i>Flask</i> радног оквира. Клијентски део решења је имплементиран користећи <i>React</i> радни оквир. Апликација подржава управљање, груписање и организацију светала по некретнинама и просторијама.
Датум прихватања теме, <b>ДП</b> :	
Датум одбране, <b>ДО</b> :	
Чланови комисије, <b>КО</b> :	
председник	др Милан Сегединац, ванредни професор

члан	др Марко Марковић, ванредни професор
ментор	др Милан Видаковић, редовни професор
Потпис ментора	

## KEY WORDS DOCUMENTATION

Accession number, <b>ANO</b> :	
Identification number, <b>INO</b> :	
Document type, <b>DT</b> :	monographic publication
Type of record, <b>TR</b> :	textual material
Contents code, <b>CC</b> :	bachelor thesis
Author, <b>AU</b> :	Maja Varga
Mentor, <b>MN</b> :	Milan Vidaković, full professor, PhD
Title, <b>TI</b> :	Web application for smart lights management
Language of text, <b>LT</b> :	Serbian
Language of abstract, <b>LA</b> :	Serbian / English
Country of publication, <b>CP</b> :	Serbia
Locality of publication, <b>LP</b> :	Vojvodina
Publication year, <b>PY</b> :	2024
Publisher, <b>PB</b> :	author's reprint
Publication place, <b>PP</b> :	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, <b>PD</b> :	7 / 60 / 0 / 0 / 42 / 0 / 0
Scientific field, <b>SF</b> :	Software Engineering and Information Technologies
Scientific discipline, <b>SD</b> :	Software Engineering
Subject / Keywords, <b>S/KW</b> :	WiZ, smart lights, React, Flask, pywizlight
<b>UDC</b>	
Holding data, <b>HD</b> :	Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad
Note, <b>N</b> :	
Abstract, <b>AB</b> :	The thesis deals with implementing a web application designed to manage smart lights produced by <i>WiZ Connected</i> . The backend is implemented in <i>Python</i> programming language using <i>Flask</i> framework. The front end is implemented using the <i>React</i> framework. The application supports controlling, grouping, and organizing lights into homes and rooms.
Accepted by sci. Board on, <b>ASB</b> :	
Defended on, <b>DE</b> :	
Defense board, <b>DB</b> :	
president	Milan Segedinac, associate professor, PhD
member	Marko Marković, associate professor, PhD
mentor	Milan Vidaković, full professor, PhD
Mentor's signature	

