



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У  
НОВОМ САДУ

---




Предраг Главаш

# **Имплементација дистрибуираних трансакција и оркестрације микросервисног система за управљање радним налозима**

ЗАВРШНИ РАД  
- Основне академске студије -

Нови Сад, 2021



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

(Податке уноси предметни наставник - ментор)

|                                  |  |
|----------------------------------|--|
| Врста студија:                   | а) Основне академске студије<br>б) Основне струковне студије |
| Студијски програм:               | Примењено софтверско инжењерство                             |
| Руководилац студијског програма: | доц. др Александар Селаков                                   |

|          |                              |               |            |
|----------|------------------------------|---------------|------------|
| Студент: | Предраг Главаш               | Број индекса: | ПР 44/2017 |
| Област:  | Електротехника и рачунарство |               |            |
| Ментор:  | доц. др Никола Далчековић    |               |            |

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

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

### НАСЛОВ (BACHELOR) РАДА:

Имплементација дистрибуираних трансакција и оркестрације микросервисног система за управљање радним налозима

### ТЕКСТ ЗАДАТКА:

Извршити преглед литературе, идентификовати могуће начине унапређења постојећег решења паметних електроенергетских мрежа тако да одговори на нефункционалне захтеве решења као што су лакше и брже испоруке, скалабилности као и омогућавање парцијалног отказа система. Одабрати најпогоднији радни оквир и имплементирати решење. Приказати резултате и кроз један од најкомплекснијих случајева коришћења приказати начин имплементације решења.

|                                  |              |
|----------------------------------|--------------|
| Руководилац студијског програма: | Ментор рада: |
|                                  |              |

Примерак за: О - Студента; О - Студентску службу факултета

## Садржај

|           |   |           |
|-----------|---|-----------|
| <b>1.</b> | <b>УВОД .....</b>                                   | <b>1</b>  |
| 1.1       | ОПИС ПРОБЛЕМА .....                                 | 2         |
| <b>2.</b> | <b>ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА .....</b>            | <b>3</b>  |
| 2.1       | ТЕХНОЛОГИЈЕ И АЛАТИ ПРЕДЊЕ СТРАНЕ .....             | 3         |
| 2.2       | ТЕХНОЛОГИЈЕ И АЛАТИ ЗАДЊЕ СТРАНЕ .....              | 4         |
| 2.3       | ТЕХНОЛОГИЈЕ И АЛАТИ ЗА ОРКЕСТРАЦИЈУ .....           | 5         |
| <b>3.</b> | <b>ОПИС МОНОЛИТНОГ РЕШЕЊА .....</b>                 | <b>7</b>  |
| 3.1       | ДОМЕН И КОРИСНИЦИ АПЛИКАЦИЈЕ .....                  | 7         |
| 3.2       | НАЧИН УПОТРЕБЕ АПЛИКАЦИЈЕ .....                     | 9         |
| 3.3       | ОРГАНИЗАЦИЈА ПРЕДЊЕ СТРАНЕ АПЛИКАЦИЈЕ .....         | 12        |
| 3.4       | ОРГАНИЗАЦИЈА ЗАДЊЕ СТРАНЕ АПЛИКАЦИЈЕ .....          | 14        |
| 3.5       | БЕЗБЕДНОСТ АПЛИКАЦИЈЕ .....                         | 15        |
| <b>4.</b> | <b>МИГРАЦИЈА НА МИКРОСЕРВИСНУ АРХИТЕКТУРУ .....</b> | <b>17</b> |
| 4.1       | ПОДЕЛА ДОМЕНСКОГ МОДЕЛА ПОДАТАКА .....              | 17        |
| 4.2       | БАЗА ПОДАТАКА У МИКРОСЕРВИСНОЈ АРХИТЕКТУРИ .....    | 18        |
| 4.3       | КОМУНИКАЦИЈА ИЗМЕЂУ МИКРОСЕРВИСА .....              | 19        |
| 4.4       | ДИСТРИБУИРАНЕ ТРАНСАКЦИЈЕ .....                     | 20        |
| 4.5       | ПРЕГЛЕД МИКРОСЕРВИСНОГ РЕШЕЊА .....                 | 21        |
| 4.6       | ОРКЕСТРАЦИЈА РЕШЕЊА .....                           | 23        |
| <b>5.</b> | <b>ЗАКЉУЧАК .....</b>                               | <b>25</b> |
|           | <b>ЛИТЕРАТУРА .....</b>                             | <b>27</b> |
|           | <b>БИОГРАФИЈА .....</b>                             | <b>29</b> |

# 1.УВОД

Једна од главних дизајнерских одлука при пројектовању модерних веб апликација јесте који стил архитектуре употребити. Избор адекватног стила је битан при дугорочном планирању развоја софтвера јер одређује правац у ком се софтвер може развијати у будућности. При одлучивању у обзир се морају узети сви релевантни чиниоци као што су: величина развојног тима, могући број корисника апликације, време за које је потребно изградити апликацију, намена апликације, могући правци унапређивања. Неретко, међутим, развојни тимови су суочени са ситуацијом где је потребно већ постојећу апликацију подмладити и рефакторисати како би адекватно одговорила повећаном броју корисника и потреби за унапређивањем. Управо у оваквим случајевима се прибегава миграцији са старих монолитних решења на дистрибуирана микросервисна решења.

Монолитна решења енкапсулирају све функционалности у оквиру једног пројекта. Њихова главна предност је (уколико нису компликована) у томе што се лакше развијају, тестирају и постављају. Код оваквих решења обично постоји једна база података и користи се једна технологија, што ако је апликација мала гарантује бржи развој и лакше радикално мењање апликације у перспективи. Постојање једне базе података такође елиминише потребу за управљањем дистрибуираним трансакцијама и у том смислу олакшава развој. У прошлости, монолитна архитектура је била примарни избор великих компанија при пројектовању софтвера. Међутим како су се захтеви у погледу скалабилности апликација повећавали, а софтверска заједница постепено окретала облачном рачунарству појавила се потреба за радикалном променом у филозофији дизајна апликација. Ту на сцену ступа микросервисна архитектура која врло ефикасно користи све предности облачног рачунарства. [1]

Микросервисна решења би се могла описати као скуп слабо повезаних сервиса, који у сарадњи чине једну целину. Сваки сервис енкапсулира свој скуп функционалности и у технолошком смислу може бити урађен независно од технологија других сервиса. Сервиси се развијају, тестирају и постављају независно један од другог, а отказ једног не урушава потпуно функционалност целе апликације. У случају великих монолитних апликација база кода са којом сваки софтверски инжењер мора бити упознат како би учествовао у изради пројекта је велика, док код микросервисних решења сваком тиму може бити додељено развијање једног сервиса те инжењери морају бити упознати само са базом кода тог решења. Са протоком времена и порастом базе кода монолитних решења појављује се потреба за миграцијом на микросервисну архитектуру, те је при томе кључно упознати се са адекватним техникама које олакшавају овај процес.[1][2]

Циљ овог рада је да опише на конкретном примеру миграцију једне монолитне веб апликације на дистрибуирану микросервисну архитектуру, описујући овај процес и са архитектуралног и са имплементационог становишта. Главни акценат при миграцији ће бити на адекватној подели задатака апликације на мање, међусобно што независније целине. Поред саме поделе на мање целине, биће речи и о начинима на које међусобно могу комуницирати као и техникама одржавања конзистентности података у дистрибуираним базама података. Проћиће се кроз процес контејнеризације микросервиса и њихове оркестрације, као и оркестрације помоћних компоненти.

У првом поглављу биће описан проблем који се решава са освртом на могуће правце истраживања и имплементације решења, такође ће бити предочене предности и мане потенцијалних приступа решењу. У другом поглављу биће описане технологије и алати коришћени при изради софтверског решења. У трећем поглављу ће се описивати постојеће монолитно решење са акцентом на намену апликације, кориснике и њихова задужења, архитектуру и безбедност. У четвртном поглављу ће се говорити о корацима предузетим при миграцији решења на микросервисну архитектуру, у сваком кораку биће описано шта је мотивација за предузети корак и шта су изазови на које се наишло. У петом поглављу ће се извршити резиме урађеног уз опис карактеристика решења и могућих начина за његово побољшање.

## 1.1 Опис проблема

Потребно је мигрирати монолитну веб апликацију за надгледање радова над дистрибутивном електроенергетском мрежом на микросервисну архитектуру. При миграцији потребно је потпуно очувати постојеће функционалности монолитне апликације, контејнеризовати микросервисе и оркестрирати помоћу неког од оркестратора. Поред очувања функционалности потребно је имплементирати адекватне механизме за комуникацију између микросервиса а да при том они остану довољно независни једни од других. Размотрити од колико база података ће се састојати решење, те имплементирати шаблоне за очување конзистентности података у случају дистрибуираних трансакција.

Постојеће решење има велики потенцијал за проширење због постојања све веће потребе за стабилним електроенергетским системима. Монолитна архитектура апликације може представљати препреку за њен развој у будућности и скалирање према броју корисника. Управо из тог разлога ова апликација је идеалан кандидат за миграцију на микросервисну архитектуру како би се адекватно одговорило новим изазовима. Пре свега је неопходно истражити методе за поделу функционалности апликације на мање целине и утврдити које целине унутар ње су довољно независне да могу свој посао обавити без превише ослањања на остале сервисе. Доменски модел података апликације је потребно поделити тако да одговара функционалности коју ће пружити сваки нови микросервис.

При одабиру приступа за моделовање базе података пружају се два избора, први је дељење једне базе података од стране свих микросервиса, а други дистрибуирање података у оквиру више база. Први приступ гарантује конзистентност података увођењем ограничења на нивоу базе података без потребе за додатним усклађивањем на апликативном нивоу. Међутим дељење базе података између микросервиса резултује њиховом чвршћом везаношћу јер инжењери који развијају индивидуалне микросервисе морају додатно координисати са осталим тимовима како би изменили шему базе података у делу који се њих тиче. Јединствена база података значи и да сваки микросервис има приступ подацима који доменски не припадају њему, те их може додати директно из базе уместо позива ка другом сервису. Други приступ је дистрибуирање базе података тако да сваки микросервис има своју базу која је физички гледано потпуно независна од база других сервиса. Овакав приступ је више у духу микросервиса али доноси са собом проблем одржања конзистентности података који се уместо нивоа базе података пребацује на апликативни ниво.[3] Потребно је идентификовати, у складу са наведеним предностима и манана најбољи приступ при миграцији овог решења на микросервисну архитектуру.

Децентрализација функционалности намеће и потребу да сервиси међусобно комуницирају како би обавили своје задатке. Уколико је гранулација сервиса превелика то ће узроковати слабије перформансе решења због превелике потребе за разменом података. Сама размена података подразумева коришћење комуникационих канала и протокола са различитим кашњењем узрокованим брзином мреже па је самим тим и брзина дистрибуираних операција непредвидива. Поред комуникације између сервиса, додатан проблем је и серијализација и десеријализација података у и из транспортног облика која такође уноси додатно кашњење.[3] При имплементацији комуникације између микросервиса треба узети у обзир све потенцијалне мане које доноси свака од технологија и одабрати ону која најбоље одговара домену ове апликације.

Пошто је микросервисна архитектура настала из тежње да се максимално искористе погодности облачног рачунарства [1] природно постоји потреба да се те погодности искористе и у овом случају. Како би се апликација касније могла адекватно скалирати [1][2], неопходно је извршити контејнеризацију свих микросервиса и њихових пратећих компоненти. Технологија која се намеће као природан избор је Докер (енг. *Docker*)[4] чије ће карактеристике бити описане у наредном поглављу. Уз контејнеризацију потребно је изабрати и адекватан оркестратор контејнера, најпопуларнији кандидати су Кјубернетис (енг. *Kubernetes*)[5] и Докер-компоуз (енг. *Docker-compose*)[6].

## 2. ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА

### 2.1 Технологије и алати предње стране

**Ангулар** (енг. *Angular*) је развојна платформа изграђена на Тајпскрипту намењена изради веб апликација. Састоји се од компонентно базираног радног оквира, колекције библиотеке са различитим функционалностима и алата за израду, тестирање и допуну веб апликација. Веб странице израђене у Ангулару састоје се од компоненти где свака компонента чини један део странице заједно са његовим изгледом и функционалностима. Ангулар компоненте се састоје од:

- Шаблона изгледа
- Силова изгледа
- Функционалности компоненте

Компоненте се могу инстанцирати више пута у оквиру једног приказа, а о инјекцији њихових зависности брине Ангуларов инјектор зависности. У Ангулару је могуће дефинисати приказе на основу путања и мапирати која компонента ће бити инстанцирана на ком приказу. Компоненте могу бити сачињене од једне или више других компоненти.[7]

**Тајпскрипт** (енг. *Typescript*) је строго типизирана екстензија Јаваскрипт скриптног језика. За разлику од Јаваскрипта, Тајпскрипт уводи елементе статичких језика и олакшава транзицију са програмских језика на које су програмери већ навикли. Пошто је Тајпскрипт надскуп Јаваскрипта, сваки Јаваскрипт код је уједно и легалан Тајпскрипт код, док се Тајпскрипт код у фази превођења преводи у Јаваскрипт код.[8]

**Бутстреп** (енг. *Bootstrap*) је пројекат отвореног кода за стандардизацију дизајна предњих страна веб апликација. У својој бази омогућава респонзиван дизајн веб страница и пружа могућност коришћења мреже од 12 колона ширине 940 пиксела. Поред мреже подржава стилизовање компоненти додавањем предефинисаних маргина, процентуалних ширина и дужина и фонтова.[9]

**Ангулар материјал** (енг. *Angular material*) је Гуголова (енг. *Google*) библиотека готових компоненти за предње стране веб апликација израђених у Ангулару. Пружа сет компоненти за израду форми, менија, навигационих трака, нотификација итд... Компоненте се могу до одређеног нивоа стилизовати прослеђујући параметре при инстанцирању.[10]

**Апекс чартс** (енг. *Apex charts*) је библиотека са готовим графиконима који се могу интегрисати у веб апликације. Подржане су интеграције са свим модерним радним оквирима предњих и задњих страна веб апликација.[11]

**Јаваскрипт** (енг. *Javascript*) је програмски језик веб прегледача. Језик је објектно оријентисан, али има врло специфичан начин наслеђивања кроз прототипе где објекти наслеђују карактеристике директно од других објеката. Користи се претежно за манипулисање објектним моделом документа који чини структуру веб страница. Овај језик је карактеристичан по томе што није строго типизиран.[12]

**ХТМЛ** (енг. *HyperText Markup Language, HTML*) је језик за дефинисање веб страница. Пружа начине за одвајање садржаја на веб страницама и посебно дефинисање слика, наслова, заглавља, цитата и слично.[13]

**Визуал студио код** (енг. *Visual Studio Code*) је лако развојно окружење за писање, покретање и тестирање изворног кода програма. У изворном облику подржава Јаваскрипт и Тајпскрипт али се могу инсталирати екстензије и за друге програмске језике попут Пајтона (енг. *Python*), Јаве и сличних. Пружа инжењерима подршку у виду сугестија при писању изворног кода кроз свој Интели-сенс (енг. *Intelli sense*) систем, што убрзава процес писања кода. Данас се доминантно користи као алат за израду предњих страна веб апликација.[14]

## 2.2 Технологије и алати задње стране

**С# (Си-шарп)** је вишенаменски, строго типизиран, објектно оријентисан програмски језик развијен од стране Мајкрософта (енг. *Microsoft*). Платформски је неутралан и може радити са различитим преводиоцима и радним оквирима, где је најпознатији и најпопуларнији оквир *.NET*. Подржава све постулате објектног програмирања, омогућава детекцију грешака у коду у фази превођења и елиминише потребу да се о управљању меморијом брину софтверски инжењери. О заузимању и ослобађању радне меморије се брине Колектор смећа (енг. *Garbage Collector*), који купи и уништава објекте у програму који више нису у употреби а заузету меморију враћа оперативном систему.[15]

**Дот нет кор** (енг. *.NET Core*) је лаки вишеплатформски радни оквир настао од *.NET* радног оквира. У почетку је био подскуп радног оквира од ког је настао да би с временом добио функционалности својствене само њему. Претежно се користи за израду задњих страна веб апликација и подржава прављење рестфул веб сервиса. Пошто је реч о лаком радном оквиру његов меморијски отисак је мали те је и брзина извршавања апликација већа. Апликације се могу поставити да раде са одређеном верзијом Дот нет кор радног оквира не нужно најновијом, те је развој апликација флексибилнији. За веб апликације је могуће подесити инјекцију зависности, крајње тачке на којима су сервиси, путање сервиса, операције које се извршавају у средњем слоју (мидлверу, енг. *Middleware*) и још много тога.[16]

**Ентити фрејмворк кор** (енг. *Entity Framework Core*) је објект-релациони мапер за Дот нет кор. Омогућава интеракцију са базом података апликације из С# кода без писања *SQL* упита. База података је представљена преко контекстне класе која има један или више сетова који представљају табеле у бази. Могуће је интегрисати већ постојећу базу података или направити нову дефинишући њена ограничења и табеле. Код првог приступа ће Ентити фрејмворк кор генерисати С# класе са адекватним пољима на основу постојеће базе података, а код другог ће се С# класе и ограничења дефинисана у њима претворити у код за генерисање базе података.[16]

**РЕСТ АПИ** (енг. *Representational State Transfer Application Programming Interface, REST API*) Концепт за моделовање и приступ подацима веб апликације као објектима и колекцијама објеката. Подацима се приступа преко јасно дефинисаних и стандардизованих путања а подаци се шаљу и примају у текстуалном облику што омогућава слабу повезаност између клијентских и серверских апликација и високу флексибилност.[17]

**Свагер** (енг. *Swagger*) је технолошки агностична спецификација *REST API* тачака приступа која се користи за њихово документовање, тестирање и касније лакше конзумирање. За сваку тачку приступа дефинише везу за њено позивање, глагол који употребљава, улазне и излазне податке као и могуће повратне кодове. Поред спецификације могуће је и тестирати тачке уносећи податке и вршећи позиве ка њима посредством графичког интерфејса који се аутоматски генерише при покретању апликације.[18]

**Аутомапер** (енг. *Automapper*) је библиотека за мапирање података између објеката различитих класа према конвенцији. Поред мапирања према конвенцији допушта подешавање параметара неког произвољног мапирања.[19] У веб апликацијама се најчешће користи за мапирање између транспортног и доменског модела података.

**ДАПР** (енг. *Distributed Application Runtime, DAPR*) је пројекат отвореног кода, спонзорисан од стране Мајкрософта који омогућава асинхрону комуникацију између сервиса. Овај пројекат имплементира шаблон „приколице“ где је инстанца ДАПР сервиса у посебном меморијском простору у односу на главни сервис, а комуницира са главним сервисом преко портова 80 или 443. Микросервиси како би комуницирали међусобно обрађају се својим ДАПР приколицама које ту комуникацију обављају за њих.[20]

**Вижуал студио 2019** (енг. *Visual Studio 2019*) је Мајкрософтово радно окружење за израду, тестирање и постављање апликација као и управљање верзијама. Постоји могућност инсталације надоградњи у виду различитих алата за покретање тестова и вршење анализа изворног кода. За разлику Вижуал студио кода поседује мноштво додатних могућности које га чине погоднијим за израду захтевних задњих делова веб апликација.[21]



## 2.3 Технологије и алати за оркестрацију

**Докер** (енг. *Docker*) је платформа за развој и контејнеризацију апликација где се све зависности и извршни код једне апликације пакују у слику која се касније може покретати на различитим машинама у контејнеру. Предност у односу на виртуелне машине је што не мора имати свој оперативни систем већ користи оперативни систем рачунара „домаћина“ на ком је покренут. [4]

**Докер-компоуз** (енг. *Docker compose*) је оркестратор контејнера који омогућава подешавање аутоматског истовременог покретања више контејнера на истом рачунару. Помоћу њега је могуће подесити редослед покретања контејнера, слике које ће бити у њима, мапирање релевантних портова и слично. За разлику од Кјубернетис оркестратора нема могућност покретања контејнера на различитим рачунарима нити подржава балансирање оптерећења. [4][6]



## 3. ОПИС МОНОЛИТНОГ РЕШЕЊА

### 3.1 Домен и корисници апликације

Апликација је намењена надгледању радова над дистрибутивном електроенергетском мрежом. Њен задатак је да обезбеди начин потрошачима електричне енергије да пријаве сметње и кварове, а да компанијама за одржавање обезбеди начин да спроведу корективне акције над мрежом. Модел података који би адекватно могао репрезентовати домен апликације дат је на слици 3.1. Сврха и значај сваког од елемената доменског модела података биће појашњени у поглављима 3.1 и 3.2.



Слика 3.1. Доменски модел података апликације.

Корисници апликације су:

- Потрошачи електричне енергије
- Администратори система
- Диспечери
- Екипе за отклањање квара
- Радници електродистрибуције

**Потрошачи електричне енергије** пријављују квар или јављају поновно успостављање стабилног снабдевања електричном енергијом (слика 3.2.). Поред тога потрошачи могу видети и све пријављене кварове. Како би видели пријављене кварове потрошачи морају бити аутентификовани, што могу урадити пријавом на систем помоћу свог Гугл или Фејсбук (енг. *Facebook*) налога.

**Report Outage**

Address:

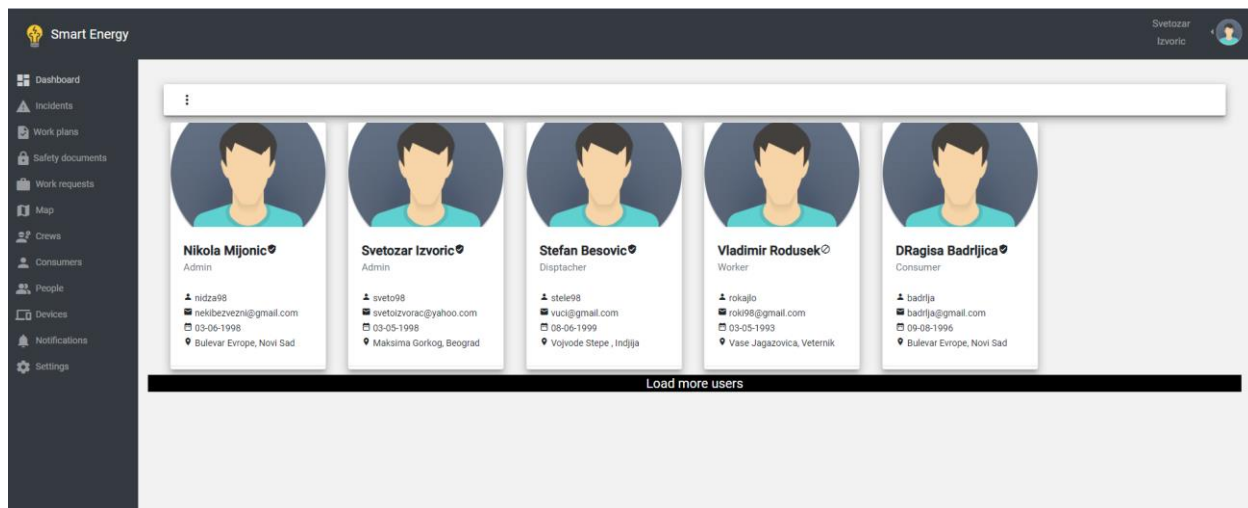
Reason:

Comment:

Hazard:

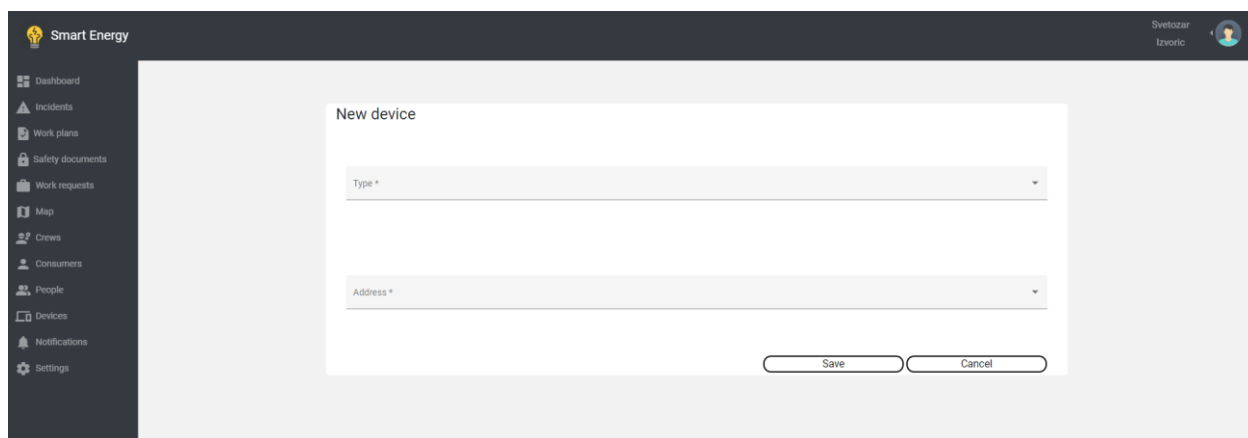
Слика 3.2. Форма за пријаву квара.

**Администратори система** управљају налозима корисника, уређајима у електроенергетском систему и екипама за отклањање кварова. Иако сви верификовани корисници могу видети списак осталих корисника апликације (слика 3.3.) једино администратори могу управљати њима.



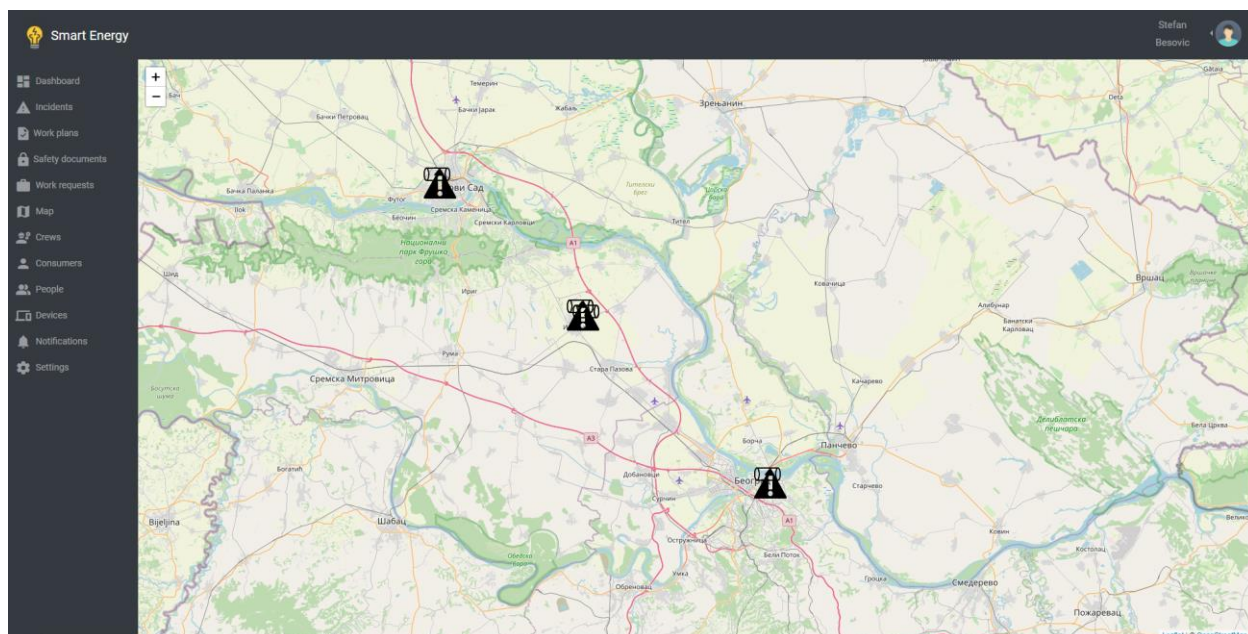
Слика 3.3. Преглед корисника система.

Администратори могу креирати или изменити екипе за отклањање кварова наводећи им назив и превлачећи мишем раднике у и из екипе. Уређаји се могу додати у систем бирањем типа уређаја и улице у којој се он налази. (слика 3.4.)



Слика 3.4. Додавање новог уређаја у систем.

**Диспечери** су у систему задужени за процес између пријаве квара и његовог физичког отклањања на терену. Они креирају инциденте на основу пријава квара и уређују њихове пропратне податке и мултимедијалне прилоге. Диспечери одређују која екипа ће изаћи на терен да отклања квар и праве додатну документацију која прати процес од пријаве до отклањања квара. Диспечер управља екипама прегледајући њихове позиције и позиције инцидента на мапи града (слика 3.5.).



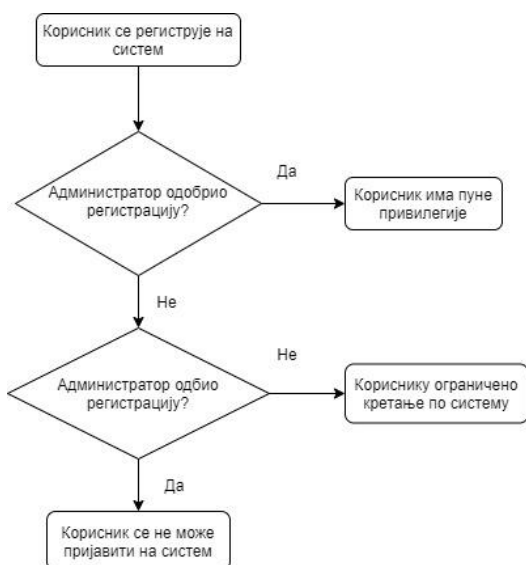
Слика 3.5. Преглед инцидента и уређаја на мапи.

**Екипе за отклањање квара** или тачније чланови тих екипа су задужени управо за физичко отклањање квара на терену. У систему веб апликације могу прегледати све податке, али смеју уређивати само она документа која су везана за посао који тренутно обављају или су обављали. Иако при отклањању квара екипе потенцијално могу заменити постојећи уређај новим, податке о новом уређају уносе искључиво администратори.

**Радници електродистрибуције** чине административно и нетехничко особље које у систему има право само да прегледа али не и мења било какве податке.

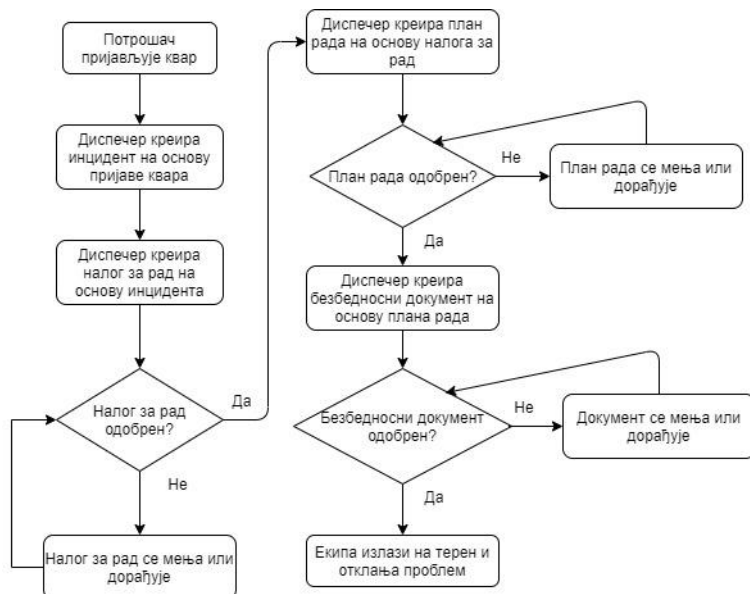
## 3.2 Начин употребе апликације

Пре употребе апликације од стране корисника потребно је да они направе налог региструјући се преко форме за регистрацију. Изузетак од овога су администратори система чији подаци већ постоје у бази података и потрошачи који се могу пријавити увезивањем свог налога са друштвених мрежа. Након регистрације корисници се могу пријавити на систем, међутим неће имати приступ свим деловима система док год их неко од администратора не одобри. По одобравању добијају мејл са обавештењем да им је налог одобрен и могућност да приступе деловима система сходно типу корисника. На слици 3.6. се може видети дијаграм тока регистрације на систем.



Слика 3.6. Дијаграм тока регистрације на систем.

Доминантан случај употребе апликације је решавање конкретног проблема пријављеног од стране неког или неких потрошача. На слици 3.7. се налази дијаграм целог процеса.



Слика 3.7. Дијаграм процеса пријаве и отклањања квара.

Први корак у процесу решавања проблема је пријава кvara, кvar пријављује потрошач користећи форму за пријаву кvara на почетној страници. Изглед форме је приказан раније на слици 3.1. Након пријаве кvara диспечер креира инцидент који описује кvar или нерегуларно функционисање система. При креирању инцидента диспечер уноси доступне информације, а затим прилаже додатке који боље описују инцидент и бира екипу која ће изаћи на терен. Међу додатке спадају пријаве кvara везане за инцидент, уређаји који су афектовани и евентуално мултимедијални документи који боље описују шта се десило. Након додавања уређаја афектованих инцидентом систем ће аутоматски увезати позиве са локације уређаја. Уколико је пријава кvara примљена телефонски диспечер може ручно унети податке које је примио помоћу форме са слике 3.8.

The screenshot shows a web form titled "New call". It contains several input fields: "Reason \*" (a dropdown menu), "Hazard \*" (a text field), "Comment" (a text area), and "Address \*" (a dropdown menu). Below these is a section titled "Customer info" which includes fields for "Name", "Last name", "Account", and "Address". At the bottom of the form are two buttons: "Choose customer" and "Save".

Слика 3.8. Форма за ручни унос детаља позива пријаве кvara.

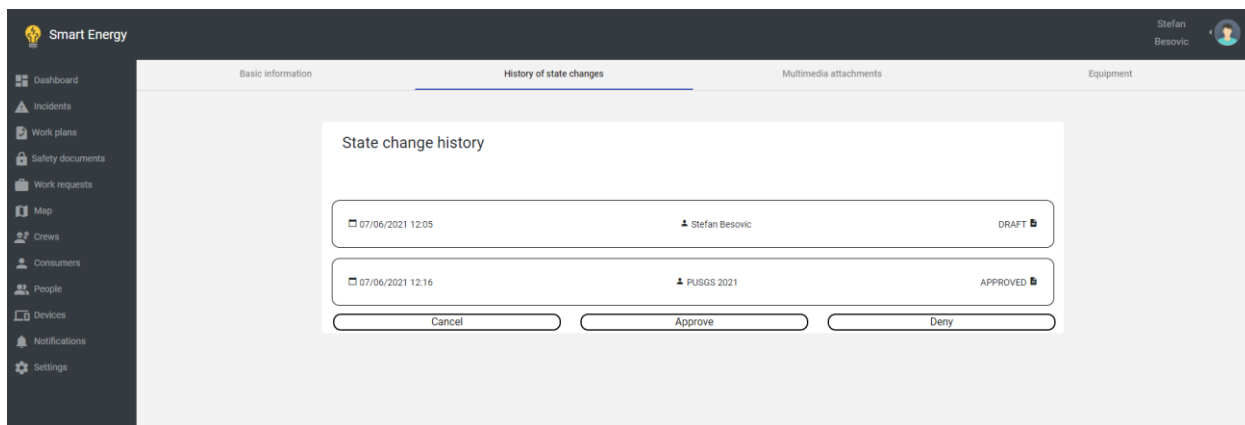
Наредни корак је креирање налога за рад на основу инцидента, налог креира диспечер бирајући одговарајући инцидент и попуњавајући одговарајуће податке из форме (слика 3.9.).

The screenshot shows a web application interface for "Smart Energy". The main content area is titled "New work request". A modal window titled "Choose incident" is open, displaying a table of incidents. The table has columns: "Type", "Priority", "Confirmed", "Status", "Incident Occurred", and "Voltage Level". The first row shows an incident with Type "UNPLANNED", Priority "3", Confirmed "No", Status "UNRESOLVED", Incident Occurred "22/07/2021", and Voltage Level "4". Below the table are "Items per page" (set to 5) and "1 - 1 of 1" with navigation arrows. A "Cancel" button is at the bottom of the modal. The background form has fields for "Document type", "Incident", "Start date/time", "Company", "Created on" (14/07/2021), "Created by" (Stefan Besovic), "Purpose", "Details", and "Notes". A "Save changes" button is at the bottom of the form.

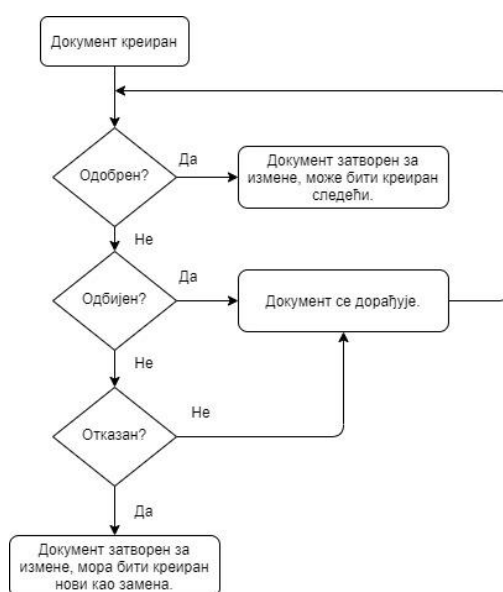
Слика 3.9. Форма за избор инцидента при креирању налога за рад.

Као и код инцидента могуће је окачити мултимедијалне прилоге у виду слика и докумената који ће помоћи при решавању кvara. Афектовани уређаји се копирају из инцидента на основу ког је налог креиран, те их је овде могуће само прегледати. Почев од налога за рад, сви наредни документи у овом процесу морају бити одобрени како би се процес могао наставити. При креирању сви документи су у стању нацрта, након чега их остали диспечери могу одобрити, одбити или отказати. Уколико је документ одобрен не може се више мењати а процес се наставља креирањем наредног документа у низу.

Уколико је документ одбијен значи да се уз одређене дораде касније може одобрити и процес може наставити. Ако је пак отказан то значи да се документ ни уз дораде не може одобрити и ту цео процес стаје док се не креира нови документ као адекватна замена. Изглед странице са променама стања документа и дијаграм промене стања документа могу се видети на сликама 3.10. и 3.11. респективно.

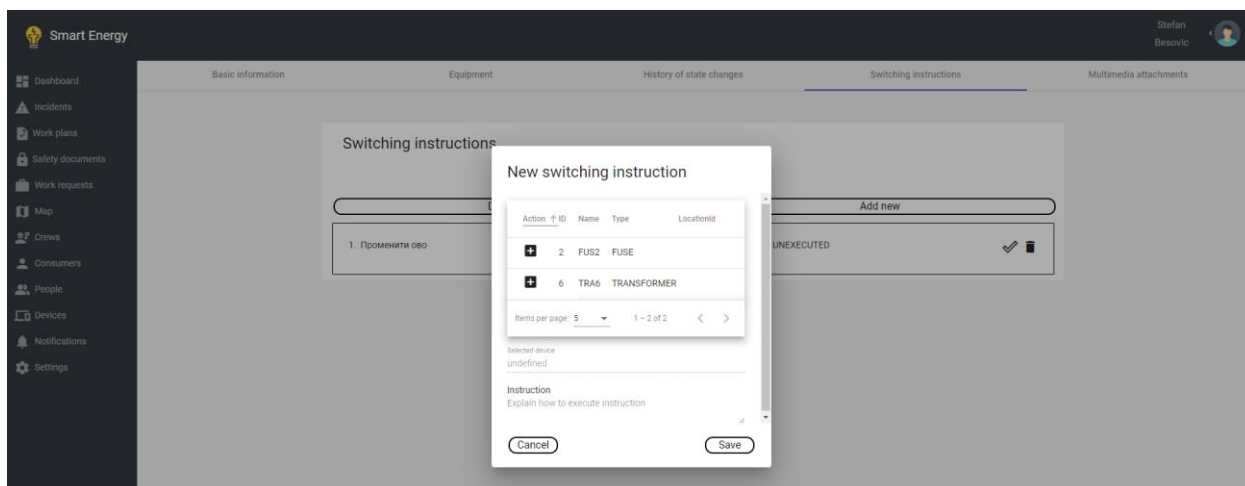


Слика 3.10. Страна са историјом промена стања налога за рад.



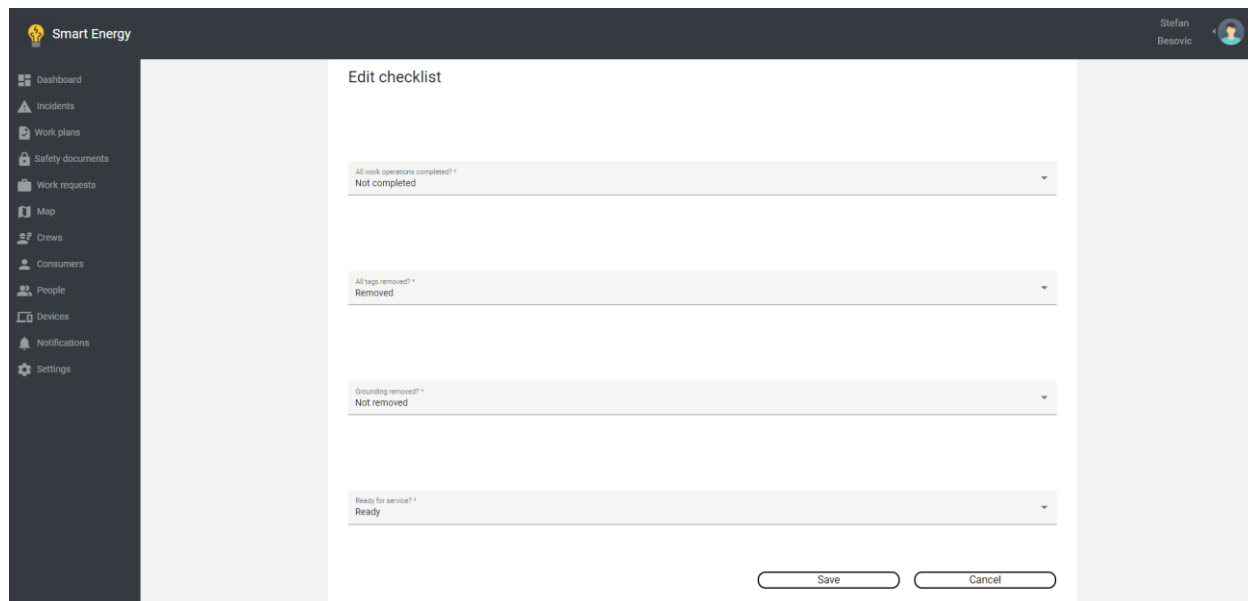
Слика 3.11. Дијаграм промене стања документа.

Након одобравања налога за рад креира се план рада на основу њега. План рада има све што има и налог за рад уз додатак радних инструкција. Афектовани уређаји се аутоматски копирају из изабраног налога за рад. Радне инструкције су везане за афектоване уређаје и текстуално описују какве операције је потребно извршити над конкретним уређајем. Инструкције могу додати и радници задужени за инцидент и диспечери путем форме са слике 3.12.



Слика 3.12. Форма за додавање радних инструкција над уређајем.

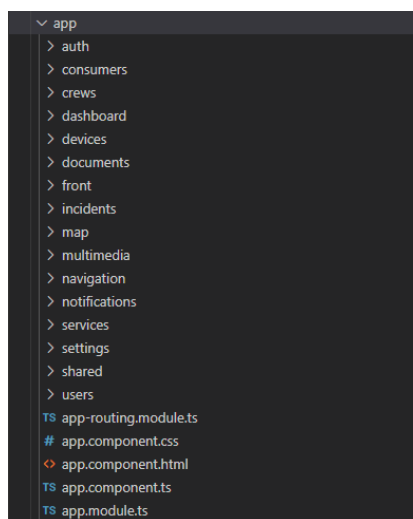
Безбедносни документ је последњи корак у решавању пријављеног проблема. Поседује све информације као и налог за рад уз додатак листе корака коју радници морају испоштовати како би се очувала њихова безбедност и безбедност околине током отклањања кvara. Након што је документ одобрен и екипа изашла на терен радници редом извршавају инструкције из плана рада над конкретним уређајима и означавају адекватно кораке из безбедносне листе. Безбедносна листа се може видети на слици 3.13.



Слика 3.13. Безбедносна листа.

### 3.3 Организација предње стране апликације

Предња страна апликације је урађена у Ангулар радном оквиру уз ослонац на Ангулар материјал компоненте за дизајн погледа. Компоненте предње стране су груписане логички по модулима где су укључене и њихове зависности, а сваки модул стоји у засебном директоријуму. Поред компоненти посебно су одвојени директоријуми где се налази модел података који се користи на предњој страни и сервиси преко којих предња страна комуницира са задњом страном апликације. На слици 3.14. се виде директоријуми са модулима предње стране апликације. Директоријум *shared* садржи модел података предње стране и интерфејсе који се користе на предњој страни, док *services* садржи сервисе за комуникацију са задњом страном апликације. Коренски модул апликације је *app-module.ts*, а модул за рутирање *app-routing.module.ts*. Коренска компонента апликације је *app-component* и у њој су инстанциран рутер који приказује компоненте на основу путања .



Слика 3.14. Модули предње стране апликације.



Поједине компоненте за које је уочено да се могу искористити на више места у апликацији због сличне или исте намене су издвојене у посебне модуле како би могле бити конзумиране од стране осталих модула независно. Један такав пример је мултимедијална компонента, која је неопходна на налозима за рад, плановима рада, безбедносним документима и инцидентима. Њен изглед и функционалност су апсолутно исти на сва четири места са изузетком контролера задње стране апликације са којим комуницира. Ова потешкоћа је превазиђена токен базираном инјекцијом сервиса у компоненту на основу путање. При инстанцирању компоненте у модулу за рутирање прослеђује јој се сервисни токен. На листингу 3.1. се види пример прослеђивања сервисног токена при инстанцирању мултимедијалне компоненте на страници са инцидентима.

```
{
  path: 'multimedia/:id',
  component: MultimediaAttachmentsComponent,
  canActivate: [AuthGuardService, AuthGuardApprovedService],
  data: {
    requiredService: INCIDENT_SERVICE_TOKEN,
    roles: ['CREW_MEMBER', 'DISPATCHER', 'WORKER']
  }
}
```

Листинг 3.1. Прослеђивање сервисног токена мултимедијалној компоненти.

Компонента затим у конструктору предаје сервисни токен инјектору зависности који враћа сервис на основу прослеђеног токена.

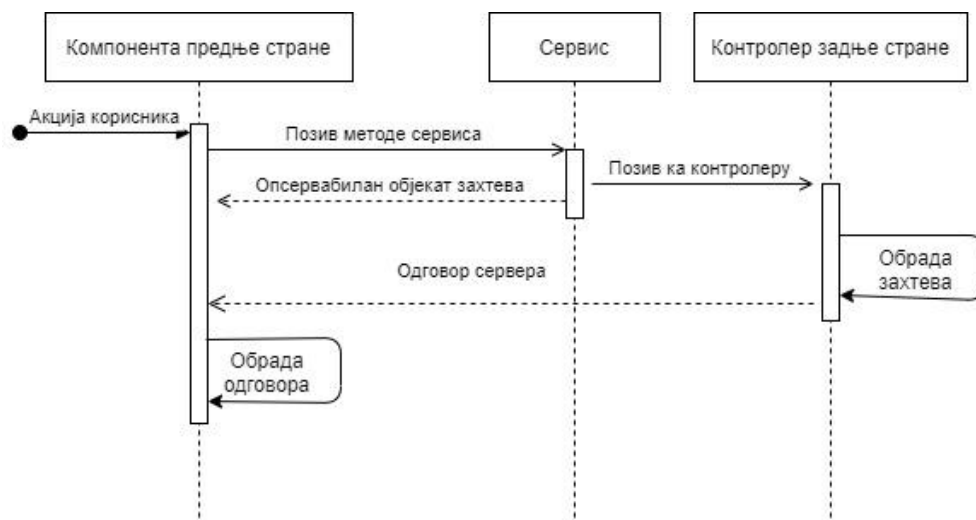
```
const serviceToken = route.snapshot.data['requiredService'];
this.multimediaService = injector.get<IMultimediaService>(serviceToken);
```

Мапирање између токена и сервиса се врши у коренском модулу апликације, секцији са пружаоцима зависности (листинг 3.2).

```
{
  provide: INCIDENT_SERVICE_TOKEN,
  useClass: IncidentService
}
```

Листинг 3.2. Мапирање токена на сервис.

Компоненте предње стране комуницирају са контролерима задње стране посредством сервиса. Компонента позива методу сервиса која затим шаље асинхрони захтев контролеру задње стране апликације на чији одговор се компонента претплаћује извршавајући предефинисан програмски код по добијању одговора. Дијаграм на слици 3.15. приказује ток комуникације.



Слика 3.15. Ток комуникације предње и задње стране апликације.

## 3.4 Организација задње стране апликације

Задња страна апликације је урађена у Дот нет кор радном оквиру уз коришћење Ентити фрејмворк кора као објект-релационог мапера за базу података. Архитектура задње стране је слојевита и изграђена од три слоја:

- Слој контролера
- Слој сервиса
- Слој података

**Слој контролера** је задужен за пријем ХТТП захтева и враћање адекватног одговора на захтев, а по потреби врши и ауторизацију позиваоца. Не врши обраду података и делегира је сервисном слоју позивајући методе адекватног сервиса. За сваку методу контролера је дефинисан ХТТП глагол и путања преко које се може позвати (анотација *HttpGet*) уз анотације које показују повратне вредности које метода може вратити као одговор (анотација *ProducesResponseType*) и листу типова корисника који су ауторизовани да је користе (анотација *Authorize*). Слој контролера ради искључиво са транспортним моделом података. Пример методе контролера за добављање свих инцидената је на листингу 3.3.

```
[HttpGet("all")]
[Authorize(Roles = "CREW_MEMBER, DISPATCHER, WORKER", Policy = "ApprovedOnly")]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(List<IncidentDto>))]
public IActionResult Get()
{
    return Ok(_incidentService.GetAll());
}
```

Листинг 3.3. Пример методе контролера инцидената за добављање свих инцидената.

**Слој сервиса** је дом бизнис логике апликације и ту се врше све обраде и агрегације података. Познаје и доменски и транспортни модел података и врши мапирање између њих помоћу Аутомапер библиотеке. Објекти улазних и повратних података су искључиво транспортног модела док се при комуникацији са слојем података користи доменски модел података. Валидације података се врше у методама слоја сервиса и у случају грешке генерише се изузетак који обрађује метода контролера и враћа одговарајући код одговора на основу њега.

**Слој података** чини Ентити фрејмворк кор са својим контекстом базе података и сетовима. Објект-релационо мапирање се врши аутоматски на нивоу радног оквира као и претварање C# кода у SQL упите.

База података је генерисана на основу класа доменског модела података и конфигурације за сваку класу. Конфигурације класа описују ограничења обележја и референцијалних интегритета на нивоу базе података и потпуно су одвојене од класа за које се пишу. Овакав приступ гарантује да је доменски модел технолошки агностичан и да су обележја и њихова ограничења јасно одвојена. Примери једне класе доменског модела података и њене конфигурације су на листинзима 3.4. и 3.5. Као што се види на примеру, класа садржи само поља са подацима и одређену доменску логику, у овом случају методу за ажурирање вредности. Конфигурација са друге стране садржи подешавање за поље *ID* које ће бити примарни кључ табеле а чија ће се вредност генерисати аутоматски при додавању и *CrewName* које је обавезно и има максималну дужину 50 карактера.

```
public class Crew
{
    public int ID { get; set; }
    public string CrewName { get; set; }
    public List<Incident> Incidents { get; set; }
    public List<User> CrewMembers { get; set; }
    public void UpdateCrew(Crew modified)
    {
        CrewName = modified.CrewName;
    }
}
```

Листинг 3.4. Класа домеског модела екипе *Crew*.

```

public class CrewConfiguration : IEntityConfiguration<Crew>
{
    public void Configure(EntityTypeBuilder<Crew> builder)
    {
        builder.HasKey(i => i.ID);

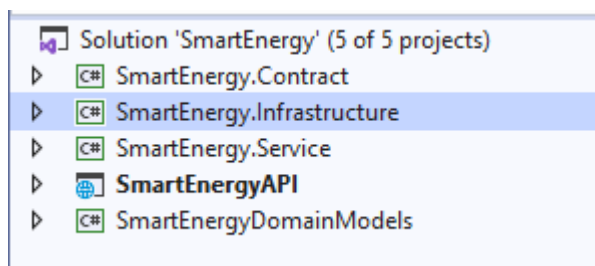
        builder.Property(i => i.ID)
            .ValueGeneratedOnAdd();

        builder.Property(i => i.CrewName)
            .IsRequired()
            .HasMaxLength(50);
    }
}

```

Листинг 3.5. Конфигурација доменског модела екипа *CrewConfiguration*.

Задња страна је сачињена од пет пројеката где је у сваком одвојена посебна група функционалности. Слика 3.16 показује пројекте од којих је сачињена задња страна.



Слика 3.16. Пројекти задње стране апликације.

**SmartEnergy.Contract** чине интерфејси, транспортни модели података, енумерације и изузеци. Сви остали пројекти референцирају овај пројекат.

**SmartEnergy.Infrastructure** чине конфигурације класа доменског модела података, миграције за базу података и контекст базе података са сетовима.

**SmartEnergy.Service** чине класе које имплементирају сервисни слој задњег дела, направљене су тако да имплементирају интерфејсе из *SmartEnergy.Contract* пројекта.

**SmartEnergyAPI** чини слој контролера, конфигурација Аутомапера и средњег слоја. Овде је дефинисано и мапирање између интерфејса и класа које их имплементирају како би радни оквир могао инјектовати зависности тамо где су потребне.

**SmartEnergy.DomainModels** чине класе доменског модела података. Као што се и види на листингу 3.4. класе немају анотације и чине их поља са подацима уз методе за њихову измену.

## 3.5 Безбедност апликације

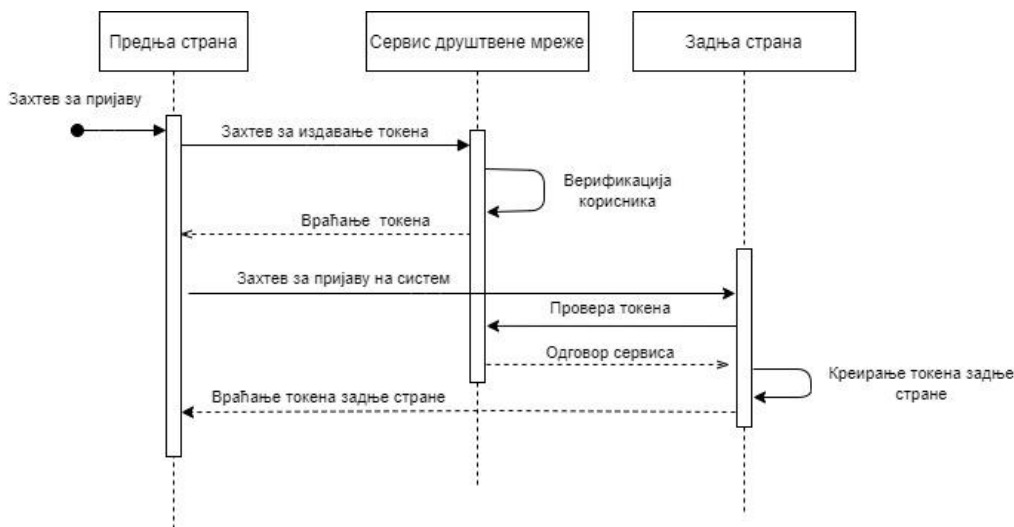
Аутентификација и ауторизација на предњој страни апликације заснива се на провери корисничких привилегија при покушају приступа путањама. За неке путање као што су почетна страна апликације није потребно да је корисник аутентификован, а за неке је потребно поред аутентификације да буде верификован од стране администратора и да буде у одређеној улози. Подаци о кориснику се добијају од токена са задње стране апликације при пријави на систем, а токен се чува у локалном стоваришту веб прегледача. При покушају корисника да приступи одређеној путањи, а уколико је та путања заштићена, проверава се да ли корисник поседује токен, да ли је токен истекао и која је корисникова улога. У случају истека важења токена, корисник ће бити преусмерен на почетну страницу уз поруку да је потребно поново да се пријави, а уколико нема адекватне привилегије да приступи некој путањи биће враћен на страну са контролном таблом. Битно је напоменути да се због природе Јаваскрипт кода, који се извршава на клијентској страни, не може ослонити на предњу страну апликације да брине о ауторизацији те су безбедносни механизми задње стране кључни за целу апликацију.

Аутентификација и ауторизација задње стране се заснива на валидацији токена. При слању ХТТП захтева ка задњој страни у захтев се умиће токен који се валидира на задњој страни. Валидација токена подразумева проверу потписа издавоца, проверу публице којој је намењен токен и проверу датума

истека. Поред тога, из токена се ваде информације о улози корисника и његовом статусу верификованости те се на основу тога одређује да ли ће захтев бити сервисан или одбијен као неауторизован. За интегритет и поверљивост информација током комуникације се користи ТЛС, тј. ХТТПС протокол.

Приликом регуларне пријаве на систем корисник уноси корисничко име и лозику у форму за пријаву (слика 3.17) који се затим шаљу задњој страни на проверу. Задња страна апликације по успешној провери генерише токен и враћа га кориснику те се он на основу њега може аутентификовати по систему. У случају пријаве преко друштвених мрежа процес је мало сложенији: предња страна апликације прво врши позив ка сервису друштвене мреже од које по успешној аутентификацији добија токен. Овако добијен токен није валидан начин аутентификације за задњу страну, те је потребно заменити га за токен издат од задње стране апликације. Замена се врши слањем токена задњој страни која га прво валидира позивом сервиса друштвене мреже која га је издала, а потом уколико је валидан издаје кориснику нови токен са његовим подацима. Новоиздати токен је исти као и онај који би се добио регуларном пријавом на систем те са те стране корисник нема препреке при аутентификацији. Дијаграм тока аутентификације налази се на слици 3.18.

Слика 3.17. Форма за пријаву на систем.



Слика 3.18. Дијаграм тока аутентификације преко друштвених мрежа.

Приликом качења мултимедијалних прилога на документе врши се скенирање прилога на вирусе помоћу антивируса на задњој страни апликације. Антивирус *ClamAV* се покреће у Докер контејнеру, а са њим је могуће комуницирати преко локалне машине на порту 3310. По слању низа бајтова мултимедијалног прилога са предње ка задњој страни апликације низ се прослеђује сервису антивируса. Сервис затим врши асинхронно скенирање прилога на вирусе и враћа одговарајући код спрам резултата скенирања. Уколико је прилог заражен одбацује се, а кориснику се враћа код грешке. Слика антивируса заједно са његовим сервисом јавно је доступна и није дело аутора ове апликације.

## 4. МИГРАЦИЈА НА МИКРОСЕРВИСНУ АРХИТЕКТУРУ

### 4.1 Подела доменског модела података

Први корак при миграцији монолитне апликације на микросервисну архитектуру је адекватна подела доменског модела података тако да је сваки добијени поддомен довољно независан од остатка података да се помоћу њега може репрезентовати скуп функционалности које се могу извршавати и у случају недоступности других података. У случају ове апликације то није сасвим могуће, али пажљивим разматрањем доменског модела (слика 3.1.) се могу издвојити три поддомена који приближно задовољавају ове критеријуме (слика 4.1.).



Слика 4.1. Идентификовани поддомени доменског модела података.

**Људски ресурси** чине поддомен везан за људе који користе, одржавају и управљају електроенергетским системом кроз апликацију. Задачи микросервиса представљеног овим моделом података су :

- Управљање корисничким налозима
- Аутентификација и ауторизација
- Управљање екипама
- Управљање потрошачима електричне енергије

Из друге тачке задатака овог микросервиса може се закључити да је од његовог функционисања зависан читав систем, јер без адекватне аутентификације и ауторизације цела апликација не може функционисати. Ово значи да би његовим испадом практично већина апликације била неупотребљива, међутим овај задатак се мора доделити неком од микросервиса како би се избегла комплексност дистрибуирања оваквих операција. Разлог за доделу тако важног задатка управо овом микросервису јесте потреба за приступом корисничким подацима сваки пут при аутентификацији и ауторизацији.

**Физички елементи** су поддомен који репрезентује уређаје електроенергетске мреже и локације које чине електроенергетски систем. Локације су везане и за потрошаче те се намеће питање због чега подаци о локацијама нису смештени заједно са подацима о потрошачима. Одговор лежи у већој потреби да се у неком моменту зна на којој тачно локацији и географским координатама се налази поковарени уређај, него да се зна адреса потрошача. Једноставније речено, ако би били познати поковарени уређаји, али не и њихове тачне локације (из разлога испода микросервиса са локацијама) квар никад не би могао бити отклоњен. Задачи микросервиса представљеног овим моделом података су:

- Управљање свим локацијама у систему
- Додавање, измена и брисање уређаја електроенергетске мреже

**Документи** су највећи поддомен монолитног модела података који представља скуп докумената и прилога потребних да се неометано одвијају радови над електроенергетском мрежом. Видно је већи од остала два скупа те је логично запитати се зашто није подељен на више подскупова. Даља подела овог поддомена не би била адекватна јер није уочена потреба у овом моменту за скалирањем апликације вертикално у правцу појединачних сервиса за сваки од типова докумената. Задаци микросервиса докумената су:

- Управљање позивима/пријавама квара
- Креирање, измена и брисање инцидента на основу пријава квара
- Креирање, измена и брисање налога за рад на основу инцидента
- Креирање, измена и брисање планова рада на основу налога за рад
- Креирање, измена и брисање безбедносних докумената на основу планова рада
- Управљање мултимедијалним прилозима
- Праћење историје промена докумената

## 4.2 База података у микросервисној архитектури

Након поделе доменског модела података на поддомене, а пре имплементације бизнис логике микросервисних апликација потребно је одлучити како ће изгледати и функционисати база података када се монолитна апликација мигрира на микросервисну архитектуру. Од два приступа описана у трећем пасусу поглавља 1.1 као боље решење се издваја креирање посебне базе података за сваки микросервис. Анализа других микросервисних решења отвореног кода и искустава инжењера који су радили на изради микросервисних апликација говори да је то доминантно коришћен приступ. Дистрибуирање монолитне базе података на више мањих база гарантује слабију повезаност микросервиса, већу скалабилност слоја података и бољу толерантност на отказе. Поред тога, креирање посебних база података ће омогућити да се у фази оркестрације решења може изабрати и различит сервер базе података за сваки микросервис уколико се увиди да постоје предности појединих типова сервера у односу на друге.[22]

Приликом креирања база података микросервиса морају бити уклоњена сва ограничења референцијалних интегритета која су у монолитном решењу постојала између табела које су сада у различитим микросервисима. Страни кључеви из којих произилазе ова ограничења се свде на обична обележја тако да практично на нивоу базе података не постоји никаква валидација за њих, валидација мора бити премештена на апликативни ниво. Листинзи 4.1 и 4.2 редом илуструју разлику између конфигурације локације корисника у монолитној и микросервисној архитектури. Разлика произилази из тога што се подаци о корисницима и подаци о локацијама у другом случају налазе у потпуно одвојеним базама података.

```
builder.HasOne(i => i.Location)
    .WithMany(p => p.Users)
    .HasForeignKey(i => i.LocationID)
    .IsRequired()
    .onDelete(DeleteBehavior.Restrict);
```

Листинг 4.1. Конфигурација корисникове локације у монолитној апликацији.

```
builder.Property(i => i.LocationID)
    .IsRequired();
```

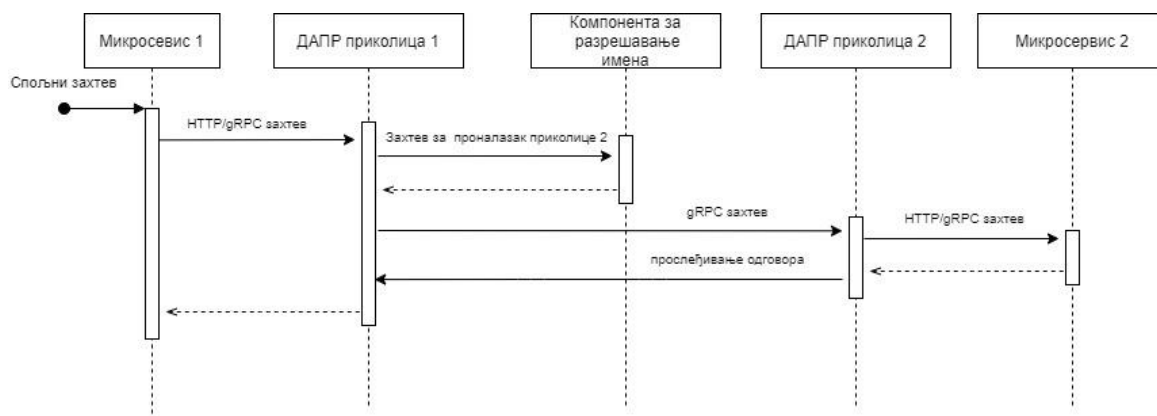
Листинг 4.2. Конфигурација корисникове локације у микросервисној апликацији.

Овако раздвојене базе података ће омогућити независан развој микросервиса и независно мењање појединачних шема база података. Међутим, јавља се проблем конзистентности података у случају да нека од трансакција обухвата више табела у различитим базама података. Такође уколико неки модел података или операција захтева агрегацију података из различитих микросервиса, њихово добављање и агрегирање мора бити урађено на апликативном нивоу јер Ентити фрејмворк кор не може приступати базама података у другим микросервисима. Овакви проблеми доводе и до питања шта треба урадити ако неки од микросервиса који су неопходни за агрегацију података падне.

## 4.3 Комуникација између микросервиса

Пошто миграција на микросервисну архитектуру мора очувати постојеће функционалности, нема потребе мењати бизнис логику апликације. Међутим, како би постојеће функционалности биле очуване, потребно је одредити помоћу чега и како ће микросервиси комуницирати и решити проблеме настале раздвајањем базе података који су наведени у претходном поглављу. Као релативно ново али због јаког развојног тима поуздано решење наметнуо се ДАПР пројекат. Начин на који ДАПР омогућава комуникацију између микросервиса је слање асинхроних ХТТП захтева ка другим микросервизима помоћу ДАПР приколица. Овако нешто је преко потребно како би се могле обавити неопходне агрегације података.

Сама архитектура комуникације је прилично једноставна и њен ток је представљен на дијаграму са слике 4.2. Микросервис 1 је споља добио захтев за враћање података које не поседује у својој бази података у целости па део мора прибавити од другог микросервиса и извршити агрегацију. Прво доступне податке добави из своје базе, па након тога зове други микросервис и затражује део података који сам не поседује. Позив ка другом микросервису се обавља тако што микросервис 1 захтев прво предаје својој ДАПР приколици (енг. *DAPR sidecar*) која онда врши позив ка компоненти за разрешавање имена која проналази локацију ДАПР приколице микросервиса 2. ДАПР приколица микросервиса 1 контактира ДАПР приколицу микросервиса 2 прослеђујући јој добијени захтев. ДАПР приколица микросервиса 2 затим позива адекватну методу контролера свог микросервиса и тражи податке. Микросервис 2 враћа одговор својој приколици која га онда прослеђује позивајући приколицу микросервиса 1. Подаци се на крају враћају микросервису 1 као одговор. У случају непостојања траженог ресурса, отказа позваног микросервиса или сличног проблема, генерисаће се изузетак.[20]



Слика 4.2. Ток комуникације између микросервиса.

Агрегација података на горенаведени начин је могућа само ако су оба микросервиса, њихове базе података и ДАПР приколице у функцији. Уколико неки од чинилаца није у функцији није могућа адекватна агрегација података, те је како би се колико-толико очувала функционалност апликације потребно уметнути неке податке уместо недостајућих који ће јасно означавати да праве податке није било могуће добавити. Пример овакве безбедне агрегације стоји на листингу 4.3. У датом примеру позива се подразумевани конструктор за објекат локације који поставља недостајуће податке на празне вредности што индицира да нису могле бити добављене.

```
foreach (UserDto us in returnValue.Users)
{
    try
    {
        us.Location = await
            _daprClient.InvokeMethodAsync<LocationDto>(HttpMethod.Get,
                "smartenergyphysical", $"/api/locations/{us.LocationID}");
    } catch
    {
        us.Location = new LocationDto();
    }
}
```

Листинг 4.3. Пример безбедне агрегације података при добављању локација корисника.

Модел података који микросервиси међусобно размењују сходно њиховој раздвојености не може бити доменски. Према томе, микросервисне апликације једна другој морају слати транспортни модел података као што је и случај у комуникацији предње и задње стране монолитне апликације.

## 4.4 Дистрибуиране трансакције

Уклањање ограничења референцијалних интегритета намеће додавање нових валидација у микросервисе како би се осигурала конзистентност података. Потребно је сваки пут проверити да ли вредност обележја страног кључа који се уписује у неку табелу, а који показује на табелу у другом микросервису заиста и постоји као примарни кључ табеле у другом микросервису. Овако нешто наравно није могуће уколико други микросервис од ког се зависи није доступан, што значи да се у том случају морају одбацити чак и потенцијално валидне операције над базом података и превентивно забранити на апликативном нивоу. На листингу 4.4. се може видети пример валидација референцијалног интегритета локације корисника и одбацавање трансакције у случају невалидности.

```
try
{
    LocationDto userLocation = await
        _daprClient.InvokeMethodAsync<LocationDto>(HttpMethod.Get,
            "smartenergyphysical", $"{api}/locations/{entity.Location.ID}");
}
catch
{
    throw new Exception("User does not have a valid location");
}
```

Листинг 4.4. Пример безбедне агрегације података при добављању локација корисника.

Одбацавање трансакције у случају немогућности утврђивања валидности вредности спољног кључа је на жалост решило само део проблема. Остало је питање шта радити у случају када дистрибуирана трансакција обухвата упис података у табеле у различитим микросервисима. Овакав проблем је доста комплекснији јер упис може бити успешан у једном микросервису али други може потенцијално бити у отказу тако да је потребно наћи начин да се координишу овакве трансакције. Софтверски шаблон који адекватно решава овај проблем је САГА шаблон.

САГА софтверски шаблон је настао као одговор на проблеме имплементације дистрибуираних трансакција у микросервисним апликацијама где су базе података физички одвојене. Сваки сервис који учествује у дистрибуираној трансакцији након успешног извршења свог дела посла генерише догађај којим сигнализира да други сервис може прећи на извршење свог дела трансакције. У случају грешке или немогућности правилног извршења дела трансакције емитују се догађаји који активирају низ компензационих операција које поништавају до тада извршене делове трансакције. Постоје два начина имплементације овог шаблона, први је кореографијска а други оркестрациона САГА. Кореографијска САГА подразумева да сваки сервис може емитовати догађај који покреће акције следећег сервиса, док код оркестрационе постоји централни оркестратор који помоћу догађаја координише трансакцијама.[23]

У случају ове апликације је због мање комплексности имплементације и малог броја места где је потребна САГА изабран приступ са оркестрацијом. За илустрацију функционисања оркестрационе САГЕ изабран је пример додавања уређаја на инцидент (листинг 4.5.). Додавање уређаја почиње позивањем методе контролера инцидента у микросервису докумената. Затим се прави захтев ка микросервису физичких елемената како би се у табели коришћених уређаја евидентирало да је уређај додат на инцидент. Након тога извршава се ажурирање приоритета уређаја и додавање позива у инцидент на основу локације уређаја у микросервису докумената.

```
var request =
    _daprClient.CreateInvokeMethodRequest<DeviceUsageDto>("smartenergyphysical",
        $"{api}/devices/device-usage",
        new DeviceUsageDto { IncidentID = incidentId, DeviceID = deviceId });

await _incidentService.AddDeviceToIncidentAsync(incidentId, deviceId);
await _daprClient.InvokeMethodAsync(request);

return Ok();
```

Листинг 4.5. Иницирање дистрибуиране трансакције при додавању уређаја на инцидент.

Уколико се током реализације оркестрације или извршења операција деси изузетак емитује се догађај који активира компензационе методе.

```
await _daprClient.PublishEventAsync<ReverseAddDeviceToIncidentEvent>("testsub",
    "AddDeviceToIncident", eventr);
```

Компензационе методе служе да врате стање оба микросервиса на онакво какво је било пре извршења трансакција. У овом случају то подразумева брисање уређаја из евиденције коришћених уређаја,



брисање везе између позива и инцидента и ресетовање приоритета инцидента. Листинзи 4.6. и 4.7. редом приказују компензационе методе у микросервисима физичких елемената и докумената. Анотација *Topic* у наведеним листинзима означава назив компоненте за асинхрону комуникацију на којој метода очекује да ће бити емитован догађај на који се она претплаћује као и који је то догађај. У овом случају то су редом *testsub* компонента и *AddDeviceToIncident* догађај.

```
[HttpPost("add-device-revert")]
[Topic("testsub", "AddDeviceToIncident")]
private async Task<IActionResult>
RevertAddDeviceToIncident(ReverseAddDeviceToIncidentEvent ev)
{
    _deviceUsageService.RemoveDeviceFromIncident(ev.IncidentID, ev.DeviceID);
    return Ok();
}
```

Листинг 4.6. Компензациона метода за брисање уређаја из евиденције.

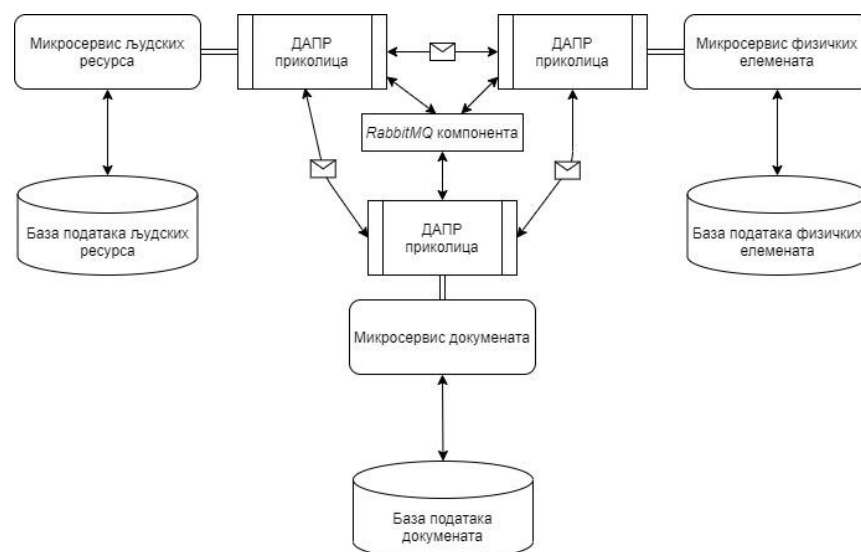
```
[HttpPost("add-device-revert")]
[Topic("testsub", "AddDeviceToIncident")]
private async Task<IActionResult>
RevertAddDeviceToIncident(ReverseAddDeviceToIncidentEvent ev)
{
    await _callService.RemoveCallsFromIncident(ev.IncidentID);
    await _incidentService.RevertIncidentToInitialState(ev.IncidentID);
    return Ok();
}
```

Листинг 4.7. Компензациона метода за ресетовање инцидента.

## 4.5 Преглед микросервисног решења

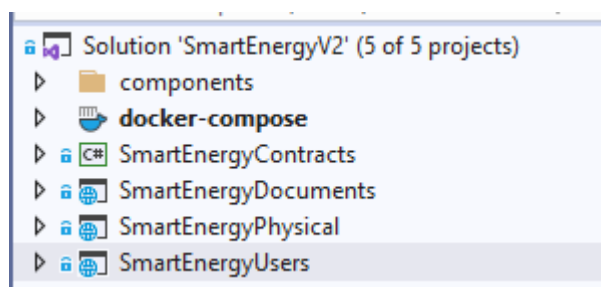
Након што су креиране базе података и решени проблеми дистрибуираних трансакција произашли из дистрибуирања базе података, а пре оркестрације решења, извршиће се преглед архитектуре и имплементације како би се стекао бољи утисак о изгледу решења. Преглед подразумева поглед на компоненте које чине решење и организацију кода микросервиса и предње стране.

Микросервисно решење се састоји од три микросервиса настала поделом монолитног доменског модела података како је описано у поглављу 4.1. Микросервиси су независне веб апликације које у софтверском смислу не референцирају једна другу, а класе и сервиси које деле налазе се у посебном пројекту који је референциран од стране сваког микросервиса. Њихове заједничке класе и сервиси су претежно класе транспортног модела података и сервиси за позивање екстерних функционалности као што је на пример слање мејла. Сваки микросервис има своју базу података са посебним контекстом базе података и својим, одвојеним доменским моделом података. При међусобној комуникацији микросервиси размењују транспортни модел података. Комуникација између микросервиса се одвија слањем асинхроних ХТТП захтева помоћу ДАПР модула где сваки микросервис има своју ДАПР приколицу. Скица архитектуре се налази на слици 4.3.



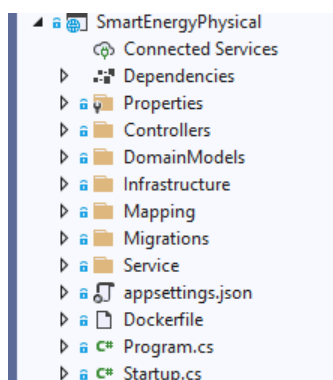
Слика 4.3. Скица архитектуре микросервисног решења.

Микросервисно решење, насупротив монолитном чине четири пројекта (слика 4.4.) где су три пројекта микросервисне веб апликације а један библиотека класа са заједничким класама и сервисима.



Слика 4.4. Пројекти микросервисног решења.

Додатна два директоријума видљива на слици под називима *components* и *docker-compose* су везана за оркестрацију решења и њихов значај ће бити накнадно описан. Сваки од микросервисних пројеката садржи исте директоријуме по којима су логички подељене софтверске целине унутар њега, изглед те поделе је дат на слици 4.5.



Слика 4.5. Подела софтверских целина унутар микросервиса физичких елемената.

Директоријум *Controllors* садржи контролере микросервисне веб апликације, *DomainModels* чине класе доменског модела података, *Infrastructure* садржи конфигурације табела у бази података и контекст базе података, а *Mapping* чини подешавање Аутомапера. *Migrations* садржи миграције базе података, а *Service* сервисни слој апликације.

Предња страна решења мора бити минимално измењена како би могла комуницирати са новоизмењеном задњом страном. Пошто је задња страна решења сачињена од више веб апликација логично је да ће се налазити на различитим веб портovima, с тога је на предњој страни потребно променити портове у путањама према којима сервиси предње стране шаљу ХТТП захтеве (листинг 4.8.).

```
export const environment = {
  production: false,
  usersServerURL : "http://localhost:44372/api/",
  physicalServerURL : "http://localhost:44373/api/",
  documentsServerURL : "http://localhost:44374/api/",
};
```

Листинг 4.8. Подешавања путања сервера задње стране на предњој страни апликације.

Поред измене портова у путањама још је потребно подесити и инјектор токена у захтеве да препозна нове портове/путање као валидне, у супротном токени не би били уметнути у ХТТП захтеве и микросервиси би их одбили. Подешавање инјектора се ради у коренском модулу предње стране апликације секцији са увозним зависностима и дато је на листингу 4.9.

```
JwtModule.forRoot({
  config: {
    tokenGetter: tokenGetter,
    allowedDomains:['localhost:44372', 'localhost:44373', 'localhost:44374']
  }
}),
```

Листинг 4.9. Подешавање инјектора токена предње стране.

## 4.6 Оркестрација решења

Последњи корак у миграцији ове монолитне апликације на микросервисну архитектуру је контејнеризација и оркестрација решења. Као оркестратор решења изабран је Докер-компоуз због мањих хардверских захтева и веће доступности адекватне документације за његову употребу у комбинацији са већ коришћеним технологијама за израду апликације. Према архитектури решења, јавља се потреба за следећим контејнерима:

- Контејнер са сликом микросервиса људских ресурса
- Контејнер са сликом микросервиса физичких елемената
- Контејнер са сликом микросервиса докумената
- Три контејнера са сликом *SQL* сервера базе података за сваки микросервис
- Контејнер са компонентом за емитовање и претплаћивање на догађаје *RabbitMQ*
- Три контејнера са сликом ДАПР сервиса као приколица за сваки од микросервиса

Директоријум *docker-compose* са слике 4.4. садржи датотеку неопходну за оркестрацију наведених контејнера а датотека *Dockerfile* са слике 4.5 конфигурацију за креирање слике микросервиса. Креирање слике подразумева копирање Дот нет кор радног оквира (кључне речи *AS base*), превођење решења (кључне речи *AS build*) и копирање извршне датотеке преведеног решења у слику (кључне речи *AS final*). Битно је напоменути да пошто се ради о Дот нет кор радном оквиру креирање слике се врши помоћу више међуслика како би се смањила величина крајње добијене слике. С тога се у фази превођења прави међуслика са свим алатима потребним како би се превело решење, док се ти алати не налазе у крајње добијеној слици.[24] Изглед датотеке за креирање слике је дат на листингу 4.10.

```
FROM mcr.microsoft.com/dotnet/aspnet:3.1 AS base
WORKDIR /app
EXPOSE 80
FROM mcr.microsoft.com/dotnet/sdk:3.1 AS build
WORKDIR /src
COPY ["SmartEnergyPhysical/SmartEnergyPhysical.csproj", "SmartEnergyPhysical/"]
RUN dotnet restore "SmartEnergyPhysical/SmartEnergyPhysical.csproj"
COPY . .
WORKDIR "/src/SmartEnergyPhysical"
RUN dotnet build "SmartEnergyPhysical.csproj" -c Release -o /app/build
FROM build AS publish
RUN dotnet publish "SmartEnergyPhysical.csproj" -c Release -o /app/publish
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "SmartEnergyPhysical.dll"]
```

Листинг 4.10. Датотека за креирање слике микросервиса физичких елемената.

Пошто је датотека за оркестрацију решења обимна, а већина подешавања за оркестрацију сваког од микросервиса иста за пример ће бити узето подешавање оркестрације микросервиса људских ресурса. Пре свега се подешава оркестрација микросервиса навођењем путање на којој се налази креирана слика и путање Докер датотеке микросервиса, поред тога врши се мапирање портова 50001 и 80 унутар контејнера на портове локалне машине како би се избегла колизија због више микросервиса који ће се касније подићи. Портови 50001 и 80 су битни јер преко њих ДАПР приколица комуницира са микросервисом за који је закачена, а пошто ће приколица бити у посебном контејнеру потребно је навести на ком порту локалне машине може контактирати свој микросервис. Последње подешавање за микросервис је навођење зависности, у овом случају микросервис зависи од контејнера базе података и контејнера компоненте за емитовање и претплаћивање на догађаје *rabbitmq*. Листинг 4.11 приказује подешавања за оркестрацију контејнера микросервиса људских ресурса.

```
smartenergyusers:
  image: ${DOCKER_REGISTRY-}smartenergyusers
  build:
    context: .
    dockerfile: SmartEnergyUsers/Dockerfile
  ports:
    - "54000:50001"
    - "44372:80"
  depends_on:
    - users-db
    - rabbitmq
```

Листинг 4.11. Подешавање оркестрације контејнера микросервиса људских ресурса.

Подешавање оркестрације контејнера базе података људских ресурса почиње такође навођењем путање на којој се налази слика, што у овом случају није локална машина већ путања на интернету јер се скида слика сервера базе података из Докер регистра. Подешавање се наставља мапирањем порта 1433 преко кога се комуницира са сервером базе података на порт локалне машине и завршава навођењем варијабли окружења неопходних за покретање сервера базе података. Те варијабле у случају *SQL Server* сервера базе података су лозинка администраторског корисника и експлицитно прихватање услова коришћења навођењем “Y”. Подешавање контејнера базе података је на листингу 4.12.

```
users-db:
  image: mcr.microsoft.com/mssql/server
  ports:
    - "1401:1433"
  environment:
    ACCEPT_EULA: "Y"
    SA_PASSWORD: "Your+password123"
```

Листинг 4.12. Подешавање оркестрације контејнера базе података људских ресурса.

Оркестрација контејнера ДАПР приколице је доста сложенија, али почиње такође навођењем путање слике. Наставља се навођењем команди које се извршавају при покретању ДАПР контејнера а које дефинишу назив инстанце, ХТТП порт и путању на којој се налазе додатне комуникационе компоненте. Назив инстанце се наводи након „-app-id“ и битан је јер се његовим навођењем може контактирати ДАПР приколица. ХТТП порт се наводи након „-app-port“, а ДАПР приколица ће на том порту пробати да контактира микросервис. Путања са додатним компонентама је у општем случају опциони параметар а у овом случају неопходан јер микросервисно решење користи компоненту за емитовање и претплаћивање на догађаје. Контејнери ДАПР приколица зависе од контејнера микросервиса и покушаће да их контактирају на подешеним портovima након покретања. У наставку се фасцикла са додатним компонентама монтира у подразумевани радни директоријум навођењем мапирања путање. На крају се наводи да ДАПР приколица дели мрежни именски простор са микросервисом за који је закачена. Подешавање контејнера ДАПР приколице је на листингу 4.13.[25]

```
smartenergyusers-dapr:
  image: "daprio/daprd:latest"
  command: [ "./daprd", "-app-id", "smartenergyusers", "-app-port", "80", "-components-path", "/components" ]
  depends_on:
    - smartenergyusers
  volumes:
    - "./components/:/components"
  network_mode: "service:smartenergyusers"
```

Листинг 4.13. Подешавање оркестрације контејнера ДАПР приколице људских ресурса.

Подешавање компоненте за емитовање догађаја није дело аутора већ је у целости преузето из извора [26] јавно доступног без накнаде.

При оркестрацији овако конфигурисаних контејнера може се јавити проблем повезивања микросервиса са базом података при покретању на слабијем хардверу. Сам контејнер микросервиса ће се покренути након покретања контејнера базе података, али то не значи да ће у том моменту и сам сервер базе података бити потпуно иницијализован и спреман за употребу. Сходно овоме, при раном покушају повезивања на базу података због миграције модела јавиће се изузетак и оркестрација се неће адекватно извршити. Овај проблем је решен на апликативном нивоу додавањем аутоматских узастопних покушаја извршавања миграције. Број покушаја је постављен на три, а паузе између покушаја су 60 секунди. Број покушаја и дужина паузе су експериментално одређени.

## 5.3 АКЉУЧАК

У овом раду је представљен један пример имплементације дистрибуираних трансакција и оркестрације у микросервисној апликацији за управљање радним налозима. Миграција апликација са монолитне на микросервисну је вршена по корацима, пре свега пажљивом поделом доменског модела података како би микросервиси који произађу из подскупова добијених том поделом имали логичког смисла. Размотрена су и одређена задужења које сваки микросервис треба да има заједно са делом бизнис логике из монолитне апликације коју треба да наследи. Прегледом литературе утврђен је и најбољи приступ за креирање базе података те је монолитна база података подељена на више независних база за сваки микросервис. При оваквој подели раскинута су нека ограничења референцијалних интегритета и сведена на обична обележја што је створило потребу за проналажењем адекватне замене за та ограничења. Замена је пронађена пребацивањем валидација референцијалног интегритета на апликативни ниво тј. сервисни слој микросервисне апликације. Прегледом опција за имплементацију асинхроне комуникације између микросервиса идентификован је ДАПР пројекат отвореног кода као валидан начин за реализацију исте. Поред могућности слања асинхроних ХТТП захтева између микросервиса ДАПР је подржао и комуникацију базирану на догађајима што је пресудило да избор падне на њега. Комуникација базирана на догађајима је употребљена као срж имплементације САГА софтверског шаблона помоћу кога је премошћен проблем одржавања конзистентности података при дистрибуираним трансакцијама. Као начин за имплементацију САГА софтверског шаблона изабрана је САГА оркестрација. Добијено микросервисно решење је на крају контејнеризовано помоћу Докер технологије и оркестрирано заједно са пратећим компонентама помоћу Докер-компоуз оркестратора. Због недостатка одговарајућег хардвера откривени су проблеми у оркестрацији сервера базе података и микросервиса, који иначе не би били детектовани. Због робусности решења и стабилности на различитом хардверу одлучено је да се ови проблеми ипак реше на апликативном нивоу додавањем узастопних покушаја извршавања миграција.

Добијено микросервисно решење у односу на монолитно карактерише већа флексибилност и већи развојни потенцијал. Повећањем броја корисника апликација се у оваквој архитектури може пребацивати у окружење облака, а индивидуални микросервиси скалирати вертикално по потреби. Услед неумитног проширења захтева за новим функционалностима апликације који прате повећање броја корисника јавиће се и потреба за проширењем развојног тима. Довођење већег броја нових људи на пројекат биће олакшано јер ће се даљи развој апликације моћи поделити по тимовима тако да сваки тим развија по један микросервис. Упознавање са архитектуром једног микросервиса и базом кода иза њега је свакако лакше него се упознати са целом монолитног апликацијом.

У случају исправљања недостатака или додавања нових функционалности на једном од микросервиса остатак функционалности ће бити доступан, а уколико постоје недостајући подаци који би се добављали из неактивног микросервиса уместо њих биће уметнуте одговарајуће замене како би било могуће нешто приказати кориснику. Агрегација података између табела које су у различитим микросервисима је знатно спорија у односу спајање табела исте базе података те је у овом смислу микросервисно решење спорије у односу на монолитно.

Пошто је микросервис људских ресурса уједно и идентитетски његов испад би довео до неупотребљивости већине система. Овај проблем је слаба тачка и монолитног и микросервисног решења. Могући правац решења овог проблема је коришћење токена друштвених мрежа као валидне замене за токен издат од стране ове апликације. Потребно је сагледати све предности и мане оваквог приступа у светлу безбедности и доступности апликације.

Аутентификација и ауторизација у монолитној апликацији је токен базирана, где су се при том у сваки токен уметали подаци о валидним конзументима токена. У микросервисном решењу, тако нешто није урађено и не постоји никаква валидација у том смислу. Било која трећа страна која дође у посед овако издатог токена, може га потенцијално злоупотребити да се представи као валидан конзумент. Поред тога, нигде се не врши инвалидација токена након одјављивања корисника из апликације па су могуће злоупотребе и у том погледу.

Приликом контејнеризације микросервиса са документима није било могуће успешно подесити постојећи сервис за скенирање датотека на вирусе, па у односу на монолитно решење постоји озбиљан безбедносни ризик доспевања нежељеног софтвера или рачунарског вируса на сервер. Потребно је у наредним итерацијама побољшавања софтвера дати приоритет проналажењу адекватног, лако конфигурабилног антивирусног решења које може функционисати у контејнерском окружењу.

Приликом регистрације нових корисника, нуди се избор свих постојећих локација у система на основу ког корисник бира своју адресу. Такође, при регистрацији нових чланова екипа нуди им се избор једне од постојећих екипа у систему. Овакав приступ открива податке о унутрашњости система неауторизованим корисницима и може довести до злоупотреба у физичком домену. Уместо избора локације приликом регистрације, кориснику се може дозволити да упише своју адресу ручно. Наравно, валидација корисничких адреса мора бити адекватна и стриктна на апликативном нивоу, али се такође мора захтевати и потврда од стране администратора система као додатна сигурност. Избор екипа при регистрацији је потребно пребацити на време након потврде корисникове регистрације од стране администратора система, тиме би се спречило да непознати корисници стекну увид у екипе у систему.

Додатни безбедносни пропуст представља недостатак чувања записа о акцијама које су корисници покушали у систему или догађајима који су се десили. Записивање акција би, поред информација о потенцијалним неовлашћеним покушајима приступа систему допринело и лакшем поправљању грешака у софтверу и побољшању корисничког искуства.

Микросервисно решење за разлику од монолитног мора на апликативном нивоу вршити агрегацију података. Такав приступ је у општем случају спорији него спајање табела у бази података те су перформансе оваквог решења слабије. У будућности је потребно истражити могућности побољшања перформанси дистрибуираног спајања табела кроз ефикасне алгоритме.

Тренутно решење не подразумева репликацију микросервиса, те у случају отказа једног од њих не постоји резерва која га може заменити. Потребно је са растом апликације размислити о преласку на другачију технологију оркестрације контејнера. Кјубернетис је валидна замена за тренутни оркестратор и пружа могућност креирања више реплика микросервиса, као и балансирање оптерећења између њих.

- [1] L. De Lauretis, "From Monolithic Architecture to Microservices Architecture," *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 93-96, doi: 10.1109/ISSREW.2019.00050.
- [2] O. Al-Debagy and P. Martinek, "A Comparative Review of Microservices and Monolithic Architectures," *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, 2018, pp. 000149-000154, doi: 10.1109/CINTI.2018.8928192.
- [3] Kalske M., Mäkitalo N., Mikkonen T. (2018) Challenges When Moving from Monolith to Microservice Architecture. In: Garrigós I., Wimmer M. (eds) *Current Trends in Web Engineering. ICWE 2017. Lecture Notes in Computer Science*, vol 10544. Springer, Cham. [https://doi.org/10.1007/978-3-319-74433-9\\_3](https://doi.org/10.1007/978-3-319-74433-9_3)
- [4] Докер <https://www.docker.com>, приступљено јула 2021.
- [5] Кјубернетис <https://kubernetes.io>, приступљено јула 2021.
- [6] Докер-компоуз <https://docs.docker.com/compose>, приступљено јула 2021.
- [7] Ангулар <https://angular.io/docs>, приступљено јула 2021.
- [8] Тајпскрипт <https://www.typescriptlang.org/docs/>, приступљено јула 2021.
- [9] Spurlock, Jake. *Bootstrap: responsive web development*. " O'Reilly Media, Inc.", 2013.
- [10] Clow M. (2018) Angular and UI Widgets. In: *Angular 5 Projects*. Apress, Berkeley, CA. [https://doi.org/10.1007/978-1-4842-3279-8\\_14](https://doi.org/10.1007/978-1-4842-3279-8_14)
- [11] Апекс чартс <https://apexcharts.com/>, приступљено јула 2021.
- [12] Jensen, Simon Holm, Anders Möller, and Peter Thiemann. "Type analysis for JavaScript." *International Static Analysis Symposium*. Springer, Berlin, Heidelberg, 2009.
- [13] Raggett, Dave, Arnaud Le Hors, and Ian Jacobs. "HTML 4.01 Specification." *W3C recommendation* 24 (1999).
- [14] "Вижуал студио код" <https://code.visualstudio.com/>, приступљено јула 2021.
- [15] Albahari, Joseph, and Ben Albahari. *C# 7.0 in a nutshell: The definitive reference*. " O'Reilly Media, Inc.", 2017.
- [16] Chowdhuri, Shahed. *ASP. NET Core Essentials*. Packt Publishing Ltd, 2016.
- [17] Masse, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O'Reilly Media, Inc.", 2011.
- [18] "Свагер" <https://swagger.io/>, приступљено јула 2021.
- [19] "Аутомапер" <https://automapper.org/>, приступљено јула 2021.
- [20] "ДАПР" <https://docs.microsoft.com/en-us/dotnet/architecture/dapr-for-net-developers/>, приступљено јула 2021.
- [21] "Вижуал студио 2019" <https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019/>, приступљено јула 2021.
- [22] Laigner, Rodrigo, et al. "Data Management in Microservices: State of the Practice, Challenges, and Research Directions." *arXiv preprint arXiv:2103.00170* (2021).
- [23] Rudrabhatla, Chaitanya K. "Comparison of event choreography and orchestration techniques in microservice architecture." *International Journal of Advanced Computer Science and Applications* 9.8 (2018): 18-22.
- [24] Докер контејнери <https://docs.microsoft.com/en-us/visualstudio/containers/container-build?view=vs-2019>, приступљено јула 2021.
- [25] Оркестрација помоћу Докер-компоуза <https://github.com/dapr/samples/tree/master/hello-docker-compose/>, приступљено јула 2021.

[26] ДАПР компонента за емитовање догађаја <https://docs.dapr.io/developing-applications/building-blocks/pubsub/howto-publish-subscribe/>, приступљено јула 2021.



## БИОГРАФИЈА

Предраг Главаш је рођен 3.5.1998. године у Београду. Основну школу „Јован Јовановић Змај“ завршио је 2013. године у Сремској Каменици. Средњу електротехничку школу „Михајло Пупин“ у Новом Саду завршио је 2017. године. Исте године уписао је Факултет Техничких Наука у Новом Саду, смер Примењено софтверско инжењерство. Положио је све испите у предвиђеном року просечном оценом 9,38.



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

## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

|   |  |
|---|--|
| Редни број, <b>РБР:</b>   |  |
| Идентификациони број, <b>ИБР:</b>   |  |
| Тип документације, <b>ТД:</b>   | Монографска документација  |
| Тип записа, <b>ТЗ:</b>  | Текстуални штампани материјал  |
| Врста рада, <b>ВР:</b>  | Завршни (Bachelor) рад   |
| Аутор, <b>АУ:</b>   | Предраг Главаш   |
| Ментор, <b>МН:</b>  | др Никола Далчековић, доцент   |
| Наслов рада, <b>НР:</b>   | Имплементација дистрибуираних трансакција и оркестрације микросервисног система за управљање радним налозима   |
| Језик публикације, <b>ЈП:</b>   | Српски / ћирилица  |
| Језик извода, <b>ЈИ:</b>  | Српски   |
| Земља публикавања, <b>ЗП:</b>   | Република Србија   |
| Уже географско подручје, <b>УГП:</b>  | Војводина  |
| Година, <b>ГО:</b>  | 2021.  |
| Издавач, <b>ИЗ:</b>   | Ауторски репринт   |
| Место и адреса, <b>МА:</b>  | Нови Сад; трг Доситеја Обрадовића 6  |
| Физички опис рада, <b>ФО:</b><br>(поглавља/страна/ цитата/табела/слика/графика/прилога) | 7/31/26/0/23/18/0  |
| Научна област, <b>НО:</b>   | Електротехника и рачунарство   |
| Научна дисциплина, <b>НД:</b>   | Примењени софтверски инжењеринг  |
| Предметна одредница/Кључне речи, <b>ПО:</b>   | Микросервиси, оптимизација, архитектура задње стране, САГА   |
| <b>УДК</b>  |  |
| Чува се, <b>ЧУ:</b>   | У библиотеци Факултета техничких наука, Нови Сад   |
| Важна напомена, <b>ВН:</b>  | Нема   |
| Извод, <b>ИЗ:</b>   | У раду је представљен један пример имплементације дистрибуираних трансакција и оркестрације микросервисне апликације за надгледање радова над дистрибутивном електроенергетском мрежом. Појашњена је намена и начин коришћења монолитне апликације и разлози за прелазак на микросервисну архитектуру. Имплементација је урађена по корацима где је у сваком кораку објашњен његов значај и потенцијални проблеми на које се наишло. На крају је урађен преглед архитектуре финалног решења уз дискусију о његовим dobrim и лошим странама и потенцијалним унапређењима. |
| Датум прихватања теме, <b>ДП:</b>   | 30.08.2021   |
| Датум одбране, <b>ДО:</b>   | 09.09.2021.  |
| Чланови комисије, <b>КО:</b>  | Председник: др Срђан Вукмировић, ванредни професор   |
|   | Члан: др Немања Недић, доцент  |
|   | Члан, ментор: др Никола Далчековић, доцент   |
|   | Потпис ментора   |



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

## 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 printed material  |   |               |
| Contents code, <b>CC</b> :   | Bachelor Thesis   |   |               |
| Author, <b>AU</b> :  | Predrag Glavaš  |   |               |
| Mentor, <b>MN</b> :  | Nikola Dalčeković, assistant professor  |   |               |
| Title, <b>TI</b> :   | Implementation of distributed transactions and orchestration in microservice system for work request management   |   |               |
| Language of text, <b>LT</b> :  | Serbian   |   |               |
| Language of abstract, <b>LA</b> :  | Serbian/English   |   |               |
| Country of publication, <b>CP</b> :  | Republic of Serbia  |   |               |
| Locality of publication, <b>LP</b> :   | Vojvodina   |   |               |
| Publication year, <b>PY</b> :  | 2021.   |   |               |
| Publisher, <b>PB</b> :   | Author's reprint  |   |               |
| Publication place, <b>PP</b> :   | Novi Sad, Dositeja Obradovica sq. 6   |   |               |
| Physical description, <b>PD</b> :<br>(chapters/pages/ref./tables/pictures/graphs/appendixes) | 7/31/26/0/23/18/0   |   |               |
| Scientific field, <b>SF</b> :  | Electrical Engineering  |   |               |
| Scientific discipline, <b>SD</b> :   | Applied software engineering  |   |               |
| Subject/Key words, <b>S/KW</b> :   | Microservices, Optimization, Back end architecture, SAGA  |   |               |
| <b>UC</b>  |   |   |               |
| Holding data, <b>HD</b> :  | The Library of Faculty of Technical Sciences, Novi Sad, Serbia  |   |               |
| Note, <b>N</b> :   | None  |   |               |
| Abstract, <b>AB</b> :  | This thesis discusses an example implementation of distributed transactions and orchestration in a microservice web application for monitoring of work in distributing power systems. It explains purpose and usage of monolithic application and reasons for migration to microservice architecture. Implementation is done in steps where significance of each step is explained along with potential problems that were ran into. In the end overview of final solution's architecture was given along with discussion about it's upsides, downsides and potential improvements. |   |               |
| Accepted by the Scientific Board on, <b>ASB</b> :  | 30.08.2021.   |   |               |
| Defended on, <b>DE</b> :   | 09.09.2021.   |   |               |
| Defended Board, <b>DB</b> :  | President:  | Srđan Vukmirović, PhD, Associate professor  |               |
|  | Member:   | Nemanja Nedić, PhD, Assistant professor     | Mentor's sign |
|  | Member, Mentor:   | Nikola Dalčeković, PhD, Assistant professor |               |