Факултет техничких наука у Новом Саду

Елементи развоја софтвера

Техничка документација за пројекат: ‘Asset Management’ - Апликација за управљање имовном

Професор:
Варга Ервин

Асистент-ментор:
Бабић Зорана

Пројекат радили студенти:
ПР42/2020 Орестијевић Јована
ПР47/2020 Делић Стефан
ПР54/2020 Буха Тамара
ПР59/2020 Стаменковић Сара

Јануар 2023. године

Садржај техничке документације

1. Захтев за пројектовање

1.1. Опис проблема

Потребно је направити дизајн система, архитектуру система, имплементирати и истестирати решење који симулира рад и комуникацију ‘Asset Management’ система. AMC води рачуна о свим уређајима у систему као што су на пример: прекидачи, трансформатори, осигурачи, вентили, генератори… и осигурава њихов стабилан рад пратећи број извршених операција и број радних сати.

1.2. Циљеви пројекта и захтеви

Основни циљ овог система је правилно одржавање опреме. Пројекат се састоји из дизајна система, архитектуре система, имплементације и тестирања решења који симулира рад и комуникацију управљача имовине (енг .‘Asset Management’). Управљач имовином представља систематично коришћење и праћење чинилаца имовине неког система, организације, декларације, скупине, простора или објекта. Обједињује ентитете од важности за уређаје, који исте прате читав њихов радни век (или док се не замене неком оптимизованијом варијантом - новим моделом/генерацијом уређаја). Постоји више типова AMC - ова, овај конкретан пројекат симулира рад простих и сложених електромагнетичких и електротехничких уређаја, као и уређаја за регулацију протока течности и/или гаса, које често видимо и користимо у свакодневном животу, у домаћинству. Неки од њих су: прекидачи, утикачи, трансформатори, генератори, вентили, осигурачи...
Циљ пројекта је веродостојно, једноставно, једнознчно и модерно симулирање AMC-а за једноставне уређаје у домаћинству.

Захтеви:

Систем садржи 3 компоненте:

1. Локални уређај
2. Локални контролер
3. Asset Management (AMS)

Локални уређај је једно мерно место у електроенергетском систему. Локални уређај може да мења стање на два начина:

- Дигитално (‘ON/OFF’, ‘OPEN/CLOSE’...) - прекидачи, осигурачи, вентили итд.
- Аналогно (‘setpoint’) - генератори, батерије итд.

Локални уређај сваку промену шаље локалном контролеру или директно AMC, у зависности од подешавања локалног уређаја:

- ‘Local device code’ - јединствено име уређаја имплементирано као ‘hash code’
- ‘Timestamp’ (‘UNIX’ тиместамп формат)

- ‘Actual value’ (тренутна вредност…)

Апликација Локалног уређаја је засебна конзолна апликација. Локални уређај се пали ручно из апликације и може бити угашен у сваком моменту, како плански из апликације тако и непланski гашењем саме апликације (тиме се симулира отказ опреме). Додавање новог Локалног уређаја се ради по принципу ‘plug-and-play’, што значи да када се нови Локални уређај упали (упали се нова инстанца конзолне апликације), почиње слање својих података и мора бити прихваћено од стране Локалног контролера или АМС осим у случају ако то име већ постоји у систему. Слање података је периодично а број секунди трајања циклуса дефинише се у ‘XML’ конгфигурационом фајлу.

Локални контролер чува све промене која долазе од стране свих локалних уређаја пријављених на контролер и на сваких 5 минута (време је конфигурабилно у ‘XML’ фајлу) их прослеђује АМС-у. У случају успешног слања Локални контролер брише своју бафер базу (‘XML’), а у случају неуспешног чува бафер до успешног слања. Ако се апликација насилно угаси пре слања бафера, приликом иницијализације учитаће се вредности из фајла.
Локални контролер може бити упаљен у сваком моменту, али може бити и угашен исто као и локални уређај.

Апликација ЛК-а је засебна конзолна апликација која своју базу чува у ‘XML’ фајлу. Додавање новог ЛК-а се ради по истом принципу као и додавање уређаја. У систему може постајати више ЛК апликација.

АСМ чува све промене у систему у својој бази која је јединствена за цео систем и служи за прављење извештаја:

- Детаљи промена за изабрани период за изабрани локални уређај (све промене сумарно);
- Број радних сати за изабрани уређај за изабрани временски период (од - до календарски по сатима);
- Излиставање свих уређаја чији је број радних сати преко конфигурисане вредности (алармирати и обојити у црвену боју оне уређаје за које је број радних сати већи од границе дефинисане у опцијама апликације);
- Листање свих постојећих уређаја у систему.

АМС апликација је засебна апликација која има свој кориснички интерфејс (може бити и терминал) и своје податке чува у ‘SQL’ бази.
Када се направи нови локални уређај, у конфигурацији се бира ком локалном контролеру или АМС припада, па стога мора да се излиста списак свих ЛК и конкретног система приликом креирања уређаја.

1.3. Очекивани напредак

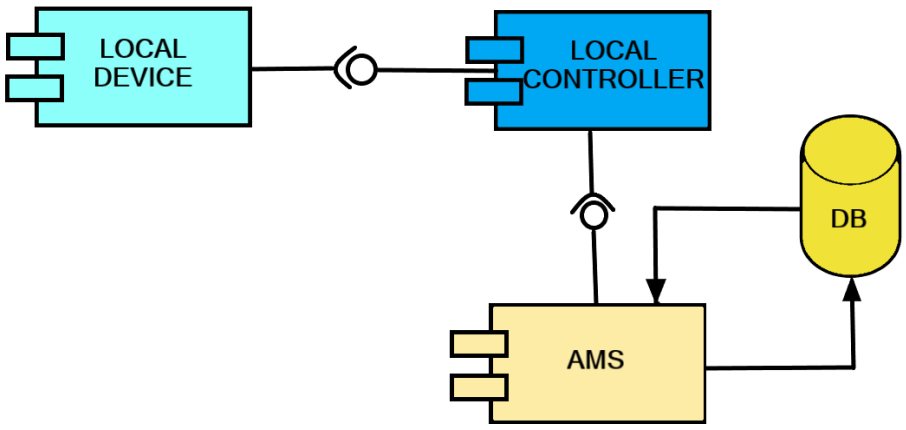
Очекивани напредак израде пројекта и подела послова представљени су кроз 4 спринта, постављена на ‘Azure DevOps’, који прате техничко решење. Техничко решење ће бити објашњено у наредним поглављима.

Тим који ради на пројекту састоји се од четири члана (студената који су претходно наведени као аутори документације).

Како је овај проблем уско везан са реализацијом самог курса ‘Елементи развоја софтвера’, пројекат прати последњих пар недеља курса (семестра) и почетну испину недељу - у којој је планирана одбрана истог.

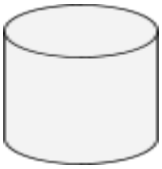
2. Идејно решење

2.1. Дијаграм компоненти



Компонента је модуларни део система. Представља „паковање“ логичких апстракција (класа) у имплементацији најчешће кроз једну кључну класу. Компонента имплементира један или више интерфејса, тј. један или више посредства путем ког комуницирају два елемента (у нашем случају програм и корисник али само са по једним интрефејсом за сваку компоненту). Свака има јединствено име како би се у моделу (програму) оне разликовале и сачувала њихова јединственост. Компоненте морају бити усаглашене у јединствени систем - све морају бити повезане како би веродостојно приказале слику модела (програма).

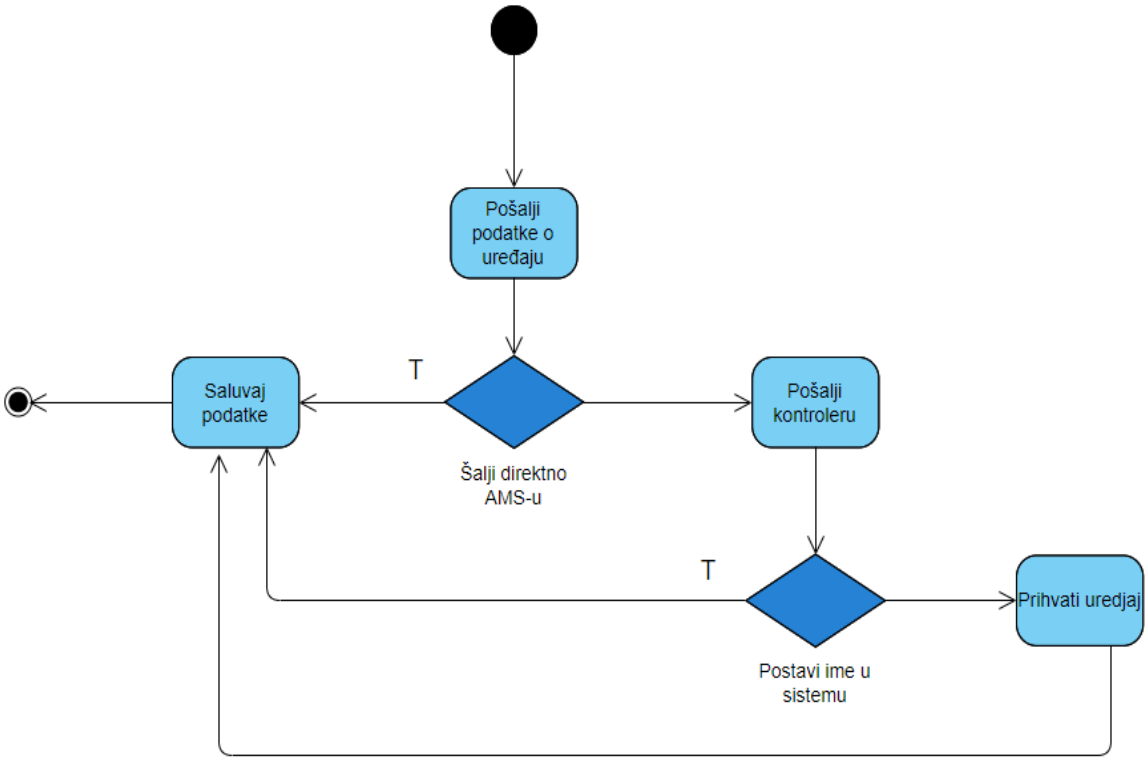
Интерфејс показује како су компоненте повезане и како међусобно делују. У овом конкретком примеру приказана је интеракција преко интерфејса. Разликујемо интерфејс који реализује компонетна(извозни) и интерфејс који компонента користи(увозни). Они су кључни за смисленост модела јер дају класама другу димензијалност - димензијалност повезаности и међусобних односа у моделу.



База података је колекција података организованих за брзо претраживање и приступ. Из угла корисника подаци су на неки логички начин повезани.

Објашњење дијаграма: Три основне компоненте у дијаграму компонетни су локални уређај, локални контролер и АМС. Локални уређај шаље податке локалном контолеру, а локални контролер АМС-у. Спрам затева корисника веома је битан проток информација кроз компоненте јер корисник жели потпуну аутономију над избором тока података. АМС је повезан са базом података и само и једино он комуницира са њом. Локланих контролера и локалних уређаја може да буде више - кроз истанце исте класе која је представљена модуларним елементом виртуално се прави више елемената које уствари обрађује и описује иста класа.

2.2. Дијаграм активности слање подтака о уређају



Легенда елемената дијаграма активности:

Почетак - почетна тачка - почетни чвор
Почетни чвор представља покретницу у дијаграму од које почиње обједињавање акција за испуњење одређене активности. У почетни чвор не може ући ни једна компонента а из њега мора изаћи тачно једна! Почетни чвор може и не мора до носи име, у колико носи неко име то име представља име почетне акције и самим тим се дијаграм смањује за једну компоненту активности.

Псеудо стање - грана - повезивање стања (елемената)

У овој нотацији кориштен је усмерени повезник који релевантно приказује однос преноса података у односу на стања између којих се нађе. Назив псеудо стање потиче од тога што ми тачно знамо шта се дешава од података (у ком су они стању) али не можемо привремене прелазе назвати конкретним стањима. Повезници су кључни за дијаграм активности како би омогућили реалнији приказ извршавања алгоритма или програма.

Гранања - одлука и стапање

Гранање специфицира алтернативне путање којима ће се ићи у зависности од услова. Исти симбол се користи за гранање и спајање секвенцијалног тока контроле. Више грана може излазити из симбола секвенцијалног гранања. Постоји тачно једна долазна ивица и произвољан број одлазних ивица, од којих свака има услов. Он индукује одлуку која треба да се донесе како би се активност одвијала даље у неком од смерова, тј. отишла у неки елемент путем гране.

Акције и активности

Акција је основна јединица спецификације понашања која репрезентује неку трансформацију или обраду у моделарном систему. Акција такође представља основни извршни елемент активности, као и један корак у активности који се обично даље не декомпонује. То значи да акција има један излаз, а може имати више улаза који могу а не морају бити спојени спојним (интеграрним чворем). Стањем акције се представља извршење акције, тј. стање самих података програма или алгоритма који се описују.
Активност - контекст акције је спецификација моделованог понашања које се изражава кроз ток извршења преко секвенцирања подактивности. Поједине акције су елементарне јединице податкивности. Стање активности има неко трајање. Извршење активности може да обухвати више акција или чекање да се деси неки догађај. Написано је у форми коју човек може да разуме - речима се описује тренутно стање података, тј. даје се смисленост самој акцији. У овој нотацији активност и акција су обједињени - у квадрату који представља акцију речима је објашњена активност.

Завршетак - крајња тачка - контролни чвор (преудочвор)

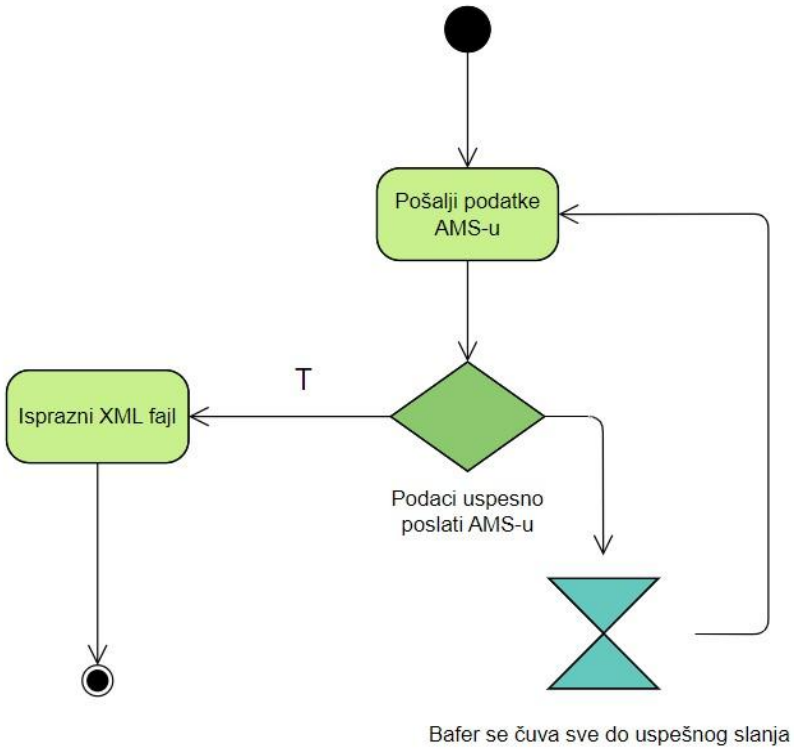
Постоје две врсте завршних чворова: завршни чвор активности и завршни чвор тока. Коначни чвор тока означава крај све контроле тока унутар активности. Коначни чвор активности означава крај једне контроле тока. Дијаграм активности може имати више коначних чворова - пошто наш дијаграм активности представља једну активност он самим тим има један крајњи чвор. Такође је дозвољено да нема завршни чвор - након чега најчешће иде надовезивање на цео један нови дијаграм активности - чест пример у алгоритмима и моделима. Ако дијаграм активности нема коначни чвор, он моделира бесконачну активност као што је бесконачна петља. Ако активност има више од једног коначног чвора први постигнут зауставља све токове у активности. Токен (подаци) који достигне завршни чвор активности

прекида активност - представља коначан резултат дела или целог модела. Конкретно, зауставља сва извршавања радњи у активности и уништава све токене у чворовима објекта, осим у чворовима параметара излазне активности (оно решење које ће нам приказати). Из ових стања се закључује да из финалног чвора више нема излаза, тј. гранања, и да у финални чвор може да улази више грана, које могу а не морају бити спојене спојницом или спојним чворем.

Објашњење дијаграма:

Како је приликом уноса/промене локалног уређаја по захтевима пројекта потребно одабрати да ли се он везује директно за АМС или за локални контролер, акција слања података о уређају иде у чвор одлучивања (‘Шаљи директно АМС-у’) где се пита да ли се подаци шаљу директно АМС-у. Ако је одлука да (математичко логичко ‘да’ на дијаграму) подаци се шаљу АМС-у и чувају (шаљу у базу података којој приступ има само АМС - није експлицитно наведено у дијаграму као одвојена акција већ се везује за акцију ‘Сачувај податке’ као контекст акције) и акција ‘Сачувај податке’ се завршава - долазимо у финални чвор. Ако то није случај то значи да се локални уређај везује за локални контролер и податке о локалном уређају шаљемо локалном контролеру - то представља акцију десно од чвора одлуке (тј. математичко логичко ‘не’ на одлуку). Како би поставили име локалног уређаја у локални контролер питамо се да ли исти већ поседује уређај са тим именом и ако га не поседује постављамо име, а ако га поседује у наредно слању извештаја локалног контролера АМСу шаљмо и податке о том локалном уређају које АМС смешта у базу (математичко логичко ‘да’ чвора одлуке). Све док се не пошаљу подаци АМС-у они се чувају у ‘XML’ фајлу што није конкретно наведено као акција већ се обједињава у склопу активности ‘Постављање имена’. Након тога долазимо до краја активности и завршавањем у финалном чвору, приказујемо даљи рад програма за неке нове захтеве, акције и активности. У овом дијаграму занемарена су поједина пуцања до којих у стварности рада програма наравно може доћи, такође занемарени су нагли прекиди, чекања, погрешни уноси и руковање програмом.

2.3. Дијаграм активности слање подтака од контролера ка АМС-у



Легенда елемената дијаграма активности:

Временски сигнал - прихвата временске догађаје
Сигнал указује на то да активност прима догађаје из спољног процеса. Временским сигналима приказују се одређена чекања у процесу или могу бити окидачи неких активности. Временски сигнал настаје протоком времена. Ако је појава временска појава, вредност резултата садржи време у којем се догађај десио. Таква акција се неформално назива радња времена чекања. Активност непрекидно ослушкује сигнале, а на дијаграму је приказано реаговање. Такође означава да ли је могуће даље специјализовати акцију. Ако је вредност тачна, онда није могуће даље специјализовати акцију. Временским сигналима приказују се одређена чекања у процесу или могу бити окидачи неких активности.

Објашњење дијаграма:
Овај систем може се посматрати као хијарархијски по питању слања података - што је и показано у дијаграму компоненти претходно наведеном. АМС је ‘надређен’ конторлеру и ‘захтева’ од њега да му исти на сваких пет минута шаље извештај стања то представља почетак овог дијаграма, тачно испод почетног чвора налази се акција ‘Пошаљи податке

АМСу’ подаци представљају извештај у ком се налази листа локалних уређаја који су везани за дат контролер. Чвор одлуке ‘Подаци успешно послати АСМ-у’ као ‘да’ грана (математичко логичко ‘да’) иде у акцију ‘Испразни ‘XML’ фајл’ која се односи на брисање података из ‘XML’ фајла локалног контролера након што је слање АМС-у успешно. Тиме се завршава читава активност и одлази се у крајњи чвор који и представља крај исте. Са десне стране чвора одлуке ‘Подаци успешно послати АСМ-у’ налази се ‘не’ грана која иде у временски сигнал под називом ‘Бафер се чува све до успешног слања’ који представља тренутке времена у којима се дешава акција поновног слања све док оно не буде успешно. Исто се не дешава конкретно у временском сигналу већ он само симулира разлог враћања на кључну акцију слања, како би смисленост дијаграма била конкретнија и изглед лепши.

Овај дијаграм може да се гледа и као унутрашњост претходног дијаграма између акција ‘Пошаљи контролеру’ и ‘Сачувај податке’.

3. Техничко решење

3. 1. Концепт израде техничког решења

Овај пројекат је планирано радити у '**Visual Studio Enterprise 2022**' програмском окружењу, уз помоћ 'workloads' - а: '**.NET desktop development**'. Због '**Code Coverage**' - а била је неопходна 'Enterprise' врста која нам уз помоћ '**PowerShell**'-а омогућује приказ покривености кода. Покривеност кода је метрика која може помоћи да разумемо колико је извора кода тестирано. То је веома корисна метрика која омогућава процену квалитета тестног пакета.

Апликације су рађене у '**C#**' језику - то је програмски језик опште намене високог нивоа који подржава више парадигми. Погодан је за прављење десктоп апликација које симулирају рад различитих типова модела из реалног света. Закључно са дескриптивнимјезицима омогућава прављење изузетно лепих апликација, лаких за коришћење. Програмско окружење нам такође уз пар додатих 'workloads' - ва омогућава и брз и лак рад са базама података - која је била задата у захтевима.

Како су у захтевима постављене три компоненте система тако ће и техничко решење то да испрати, као и повезаност једног од њих са базом уз помоћ објекта '**SQLConnection**' класе из '**Microsoft.Data.SqlClient**' нејмспејса.

У захтевима је написано да кориснички интерфејси апликација могу бити обједињени само у конзоли тако да посебно прављење прозора њихових интерфејса није урађено.

3. 2. Списак активности

Списак активности обједињује технички изглед апликација имплементираних у C# језику кроз Visual Studio програмско окружење у виду њихових класа.

- **'LocalDeviceProject' апликација**
    - Укључује интерфејсе: **ILocalDevice** и **IMyNetworkStream**
    - Испис активности (додавање, да ли је послато, тренутно стање…) у конзоли
    - Креирање конструктора класе **LocalDevice**
    - Оверрајдовање **ToString()** методе
    - Креирање хеш вредности
    - **NetworkStream Stream** објекат који обезбеђује методе за слање и пријем података преко Стреам сокета у режиму блокирања. Уз помоћ њега може се вршити и синхрона и асинхрона комуникација.
    - **TCP (client-server)** комуникација са AMS-ом и са локалним контролером
        - Покретање комуникације - **Startup()**
        - Слање података - **SendData()**

- **‘LocalControllerProject’ апликација**
  - Укључује интерфејс **ILocalController**
  - Након успешне конекције прихвата и креира сопствен објекат клијента **ReceiveData()**
    - Бинарно форматира и меморишки стримује податке уз помоћ функција из **BinaryFormatter**
  - Прављење клијента са којим ћемо даље радити - **Startup()**
    - **MyTcpClient**
  - Покретање новог направљеног сервера - **StartServer()**
    - Укључује два ‘**MyNetworkStream**’ - један за локални контролер један за AMC
    - Ослушкује долазак потенцијалних клијената - **MyTcpListener**
    - Започиње рад - **StartServer()** → све док стартовани сервер ради прихватају се подаци и уписују у XmlWriter
  - У комуникацији са AMCом шаље податке **SendToAMS()**
    - Прави листу уређаја од прочитаних података из XmlReader (уз помоћ **ReadData()** функције)
    - Бинарно форматира и меморишки стримује податке уз помоћ функција из BinaryFormatter
    - Уписује податке у ‘**MyNetworkStream**’ објекту направљеном у **StartServer()** за AMC
  - Функционалности и имплементацију за **MyTcpListener** и **MyTcpClient** класе кориштене у претходим класама
  - Функционалности за **XmlReader** и **XmlReader** класе кориштене у претходим класама

- **‘’AssetManagement’’ апликација**
  - AMS
    - Покретање сервера - AMS апликација представља крајњи сервер чији клијент могу бити уређај или локални контролер (или више њих) - **StartServer()**
    - **KomunicirajSaUredjajima()** → привата листу коју ја претходно направљена у **IspisiSveUredjaje()** → користи се као интерактивни пролазак до краја листе која је послата као аргумент, који након тога прави нову листу (која постоји само у тој функцији) прошлих уређаја из прослеђене листе. Тако добијамо тренутни објекат који принтујемо у конзоли проласком кроз тренутну листу и исписом елемената на конзоли итеративним путем.
    - **DaLiPostoji()**
      - проверава да ли прослеђен уређај постоји у листи
      - Користи се у **IspisiSveUredjaje()** како би иницирао прескакање у проласку кроз листу → зато што исти већ постоји
    - Функција која прихвата податке од клијента, даје обавештење о томе и исписује их → **ReceiveData()**

            - Креира клијента и проверава успешност конекције са тренутним клијентом
            - Креира празну листу која ће се у наставку користи за лакши рад и чување базе
            - Бинарно форматира и меморишки стримује податке уз помоћ функција из BinaryFormatter и MemoryStream
    - DataBase
        - Слање команди
        - Унос података у базу
        - Виртуалне методе sa креиранје и чување
            - У овом пројекту SQL бази приступа и за њу се директно везује само AMC, а ‘’AssetManagement’’ пројекат обједињује све остале пројекте, ове методе су виртуалне да се не би бунио пројекат приликом рада програма → ово није најбоље решење али без овога тестови би можда правили проблем.
            - Виртуелне методе дозвољавају подкласама типа да замене метод. Користе се за имплементацију полиморфизма времена извршавања или касног везивања. Када је метода декларисана као виртуелна у основној класи, а иста дефиниција постоји у изведеној класи, нема потребе за заменом, али другачија дефиниција ће функционисати само ако је метода замењена у изведеној класи.

- **‘’AssetManagementTest’’ апликација**
    - Тестирање претходне три апликације → биће обрађено у наредном поглављу
    - Укључује:
        - **AMSTests**
        - **LocalControllerTest**
        - **LocalDeviceTest**

4. Анализа извођења радова - Пројектовање изведених стања и провера

Анализа извођења радова извршава се на основу главног пројекта - у овом случају завршеног програма (апликације) који испуњава све наведене захтеве. Радови на апликацији су представљњни кроз примере коришћења и тест план који употпуњују техничку документацију програма тако што дају идеје и правила о коришћењу кориснику који ће убудуће радити на истом, уз круцијалне провере које служе као одгледало тачности и јединствености апликације. Како је сам циљ ове апликације веродостојна симулација управља имовине, од великог је значаја разумети детаље који чине апликацију а могу бити показани и објашњени само уз коршћење и тестирање исте.

4.1. Тест план

Шта је тестирање?

> Динамичка провера понашања програма избоћењем коначног броја тестова и упоређивање са очекиваним понашањем програма.

Задатак тестирања:

> Откривање што више програмским мана извршавањем и провером што већег дела програма да би се мане манифестовале као грешке.

Циљ тестирања:

> Стварање одређеног нивоа сигурности да програм ради оно што је описано захтевима.

Тестови побољшавају архитектуру решења и дефинишу функционалности, такође убрзавају развој. Постоји више категорија тестова. Према врсти захтева разликују се **тестови за проверу фукционалних карактеристика** и **тестови за проверу нефункционланих карактеристика** (брзина рада, безбедност, оптерећење…).
Тестови за проверу функционалних карактеристика:

- ‘**unit**’ - појединачни тестови, тестирање једне класе
- ‘**integration**’ - тестирање групе повезаних класа
- ‘**system**’ - тестирање целог програма
- ‘**acceptance**’ - тестирање целог програма од стране крајњег корисника

Шта је јединствено тестирање?

> Јединствено тестирање је тестирање мале логике или кода како би се потврдило да је излаз кода онакав какав се очекује на уносу одређених података и/или на задовољавању одређених услова. Обично би јединични тестови требало да буду независни од осталих тестова. Јединствени тест циља само малу јединицу кода

која може бити само метода или класа. **Овај пројекат ради само јединствена тестирања.**

Шта је JUnit?

То је један од најпознатијих оквира за јединствено тестирање. JUnit тестирање представља развојни процес у којем се најмање јединице апликације које се могу тестирати појединачно и независно проверавају у зависности од жељеног понашања. JUnit тестове пише програмер који имплементира функционалности.

Тестови који су рађени у овом пројекту:

Шта је NUnit?

NUnit представља “оквир” (**framework**) за JUnit тестирање које се може користити за све **.NET** jezike. **NUnit** је отвореног кода.
Тестови су означени анотацијама:

- **[TextFixture]** - означава класу која садржи тестове, и опционо, методе за поставку и чишћење тестова.
- **[Test]** - означава одређену методу унутар тест класе

C# Unit Tests with Mocks

Шта је лажно тестирање?

Креирање лажне верзије екстерне или интерне услуге која може да замени праву, помажући да тестови раде брже и поузданије. Када имплементација ступа у интеракцију са својствима објекта, а не његовом функцијом или понашањем, може се користити имитација.

Mock object - лажни објекат

Користи се за верификацију "индиректног излаза" тестираног кода, првим дефинисање очекивања пре него што се тестирани код изврши. У објектно оријентисаном програмирању, лажни објекти су симулирани објекти који имитирају понашање стварних објеката на контролисане начине, најчешће као део иницијативе за тестирање софтвера. Програмер обично креира лажни објекат да тестира понашање неког другог објекта.

AMSTests.cs

Компонента	Назив теста	Метода која се тестира	Врста	Пролазност	Време трајања теста
AMS	AMSConstructor	AMS()	unit	✔	386 ms
	DaLiPostoji_ReturnsFalse	DaLiPostoji()	unit	✔	6 ms

	DaLiPostoji_ReturnsTrue	DaLiPostoji()	unit	✔	1 ms
	ReceiveData_ReturnsTrue	ReceiveData()	mock	✔	195 ms
	StartServerTest	StartServer()	unit	✔	3 ms

LocalControllerTest.cs

Компонента	Назив теста	Метода која се тестира	Врста	Пролазност	Време трајања теста
LC	LocalControlerConstructor	LocalControler()	unit	✔	2 ms
	ReceiveData_ReturnsFalse	ReceiveData()	mock	✔	15 ms
	SendToAMS_ReturnFalse	SendToAMS()	mock	✔	20 ms
	SendToAMS_ReturnTrue	SendToAMS()	mock	✔	4 ms
	TestFailRand	Startup() StartServer() ObrisiXml() KomunicirajSaUredjajima() KomunicirajSaAMS() ReceiveData()	unit	✔	9 ms
	TestOkRand	WriteData() SacuvajController()	unit	✔	5 ms

LocalDeviceTest

Компонента	Назив теста	Метода која се тестира	Врста	Пролазност	Време трајања теста
LD	CreateHash_ReturnsHash	CreateHash	unit	✔	11
	LocalDeviceConstructorTest	LocalDevice	unit	✔	3
	LocalDevicetest2	DataBase()	unit	✔	2

	OtherTests	Write, Read, Close	unit	✔	<1
	SendData_ReturnFalse	SendData	mock	✔	3
	SendData_ReturnTrue	SendData	mock	✔	2
	TestDbController	ControllerDb()	unit	✔	1