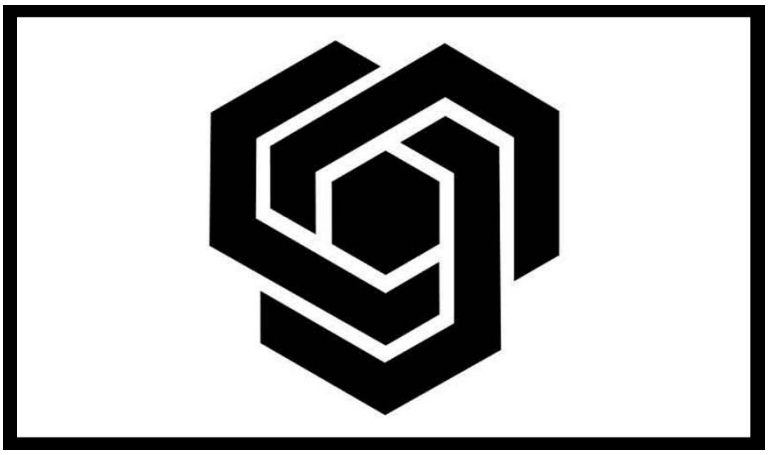
ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Курсов проект по Софтуерни архитектури



Изготвила: Даяна Димитрова Димитрова
Фак. номер: 471 219 010
Специалност: Информатика и софтуерни науки
Група: 77
Курс: II

Възложил: Гл. ас. Виктор Главев;

I. Въведение

Този документ представя архитектурата като поредица от изгледи; използвайки use-case изглед, логически изглед, process изглед и deployment изглед. Всички те са представени с помощта на подходящи UML диаграми и нужното описание на сценариите в тях.

Цел на архитектурата

Разработка на софтуерна архитектура на система тип “One stop shop услуги за граждани и бизнеса”, в контекста на предоставяне на лиценз за управление на МПС, избягвайки бюрокрацията. Този софтуер ще позволи информационните системи на отделните администрации да обменят данни по между си, като тази информация ще се съхранява на централен сървър, до който всяко звено ще има достъп само и единствено чрез посочване на ЕГН. По този начин ще се избегне чакането на дълги опашки и минаването през няколко институции за събиране на съответните документи, което само по себе си отнема много време.

Така не само гражданите ще са облагодетелствани, но и служителите на отделните звена, тъй като вече няма да им се налага да издават квитанции, бележки и декларации писмено. По този начин дори може да се твърди, че се осигурява по-голяма сигурност и съответно по-висок контрол върху личните данни. Подобна архитектура би спомогнала и за редуцирането на фалшиви документи, защото на КАТ ще му бъде предоставен директен достъп до нужните институции, които от своя страна ще му предоставят независими данни.

Самият факт, че много западни страни прилагат подобен подход в голяма част от държавните си администрации, е едно доказателство, че подобна стъпка е полезна и нужна за преминаването към един по-съвременен и модерен европейски стандарт.

Дефиниции, акроними и абревиатури

- МПС – моторно превозно средство
- ЕГН – единен граждански номер
- КАТ – контрол на автомобилния транспорт
- БД – база данни
- LAN – local area network
- WAN – wide area network

Други документи, цитирани в документа

1. Barbacci Mario, Klein H. Mark, Longstaff A. Thomas, Weinstock B. Charles “Quality Attributes”, Software Engineering Institute, December 1995
2. Roberto Verdecchia Philippe Kruchten Patricia Lago Ivano Malavolta “Building and evaluating a theory of architectural technical debt in software-intensive systems”

Общ изглед на документа

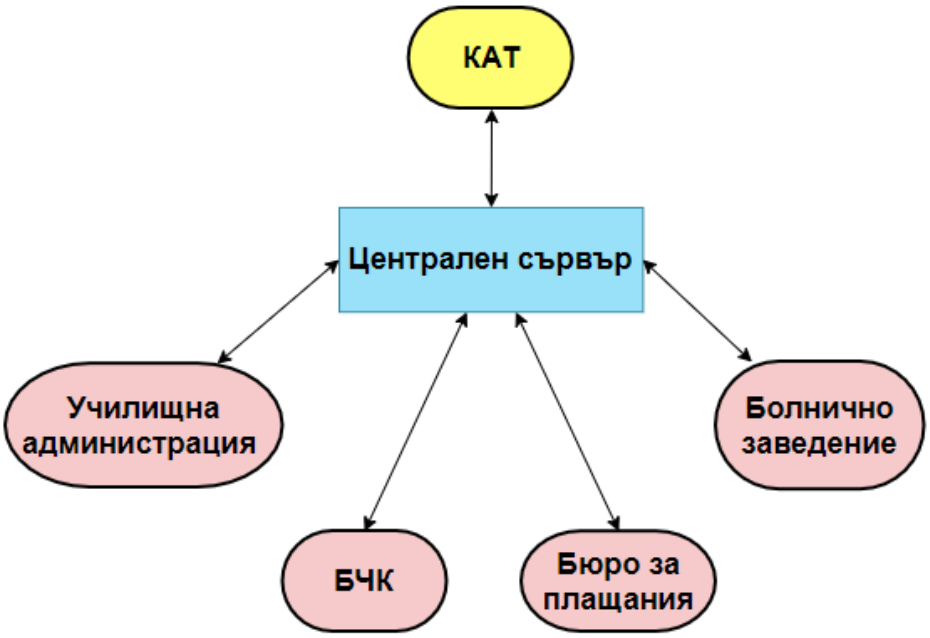
- **Функционални изисквания**

Системата ще се състои от централен сървър, който ще играе ролята на разпределител, четири администрации, а именно БЧК, Бюро за плащания, Училищно

администрация, Болнично заведение. Тези администрации ще бъдат от полза на КАТ, която ще борави с определени данни и информация от тях.

Всяка администрация ще трябва да дефинира какви данни би могла да предостави. Отделно сървърът ще трябва да поддържа шаблон на справки и техните отговори, така че да може да преразпределя изпълнението им между отделните администрации.

Обменените данни ще бъдат представени във вид, достъпен както за вещи лица в тази сфера, така и за не толкова компетентни. Системата ще поддържа няколко нива на потребност като се започне от първичния потребител, т.е. гражданина и се стигне до крайния – служителите на конкретните звена.



Системата ще използва тип архитектура Client-server, изградена с централен комуникационен възел. В суров вид са представени основните действия, които приложението ще поддържа:

- КАТ изпращат заявка до централния сървър за получаване на лиценз за управление на МПС;
- Централният сървър обработва заявката и я разделя на части;
- Централният сървър пренасочва отделните части под формата на заявки към отговорните администрации;
- В централния сървър се отчита изпратената заявка (дата на изпращане на заявката и **вид** на изисквани данни);
- Отговорните администрации обработват заявките и връщат отговор до централния сървър;
- В централния сървър се отчита полученият отговор (дата на получаване и **вид** на предоставени данни за определено лице);
- Централният сървър обработва получената информация в общ отговор, който връща в КАТ;

- За да се гарантира защитата на личните данни, след получаването на лицензa за управление на МПС от конкретното лице, неговите данни се заличават от администрацията на КАТ и от централния сървър;

Подобен тип архитектура е много рационално решение за конкретната цел на “One stop shop услуги за граждани и бизнеса“, имайки предвид следните предимства:

- Данните са централизирани в системата, която се поддържа на едно място; *(в централния сървър)*
- Моделът е ефективен при предоставянето на ресурси на клиента и също така изисква евтина поддръжка;
- Управлява се лесно и данните могат лесно да бъдат доставени на клиента; *(благодарение на разпределителната роля на централния сървър)*
- Тъй като данните са централизирани, тази система е по-сигурна и служи за допълнителна сигурност на данните; *(именно от тази сигурност се нуждае подобна система, работеща с личните данни на гражданите)*
- В рамките на този тип модели в сървъра могат да бъдат вградени повече клиенти и сървъри, което прави производителността изключителна и увеличава общата гъвкавост на модела; *(т.е. след време ако има законодателни промени относно нужните документи за издаване на шофьорска книжка, то би било лесно да се добави още една администрация)*
- По-лесна за администриране, когато мрежата е по-голяма, защото администрацията е централизирана.

- **Атрибути за качеството**

Качествените атрибути заемат важно място във взаимната координация между бизнес, информационните и инфраструктурните дисциплини в архитектурния процес.

Атрибут за качество е свойство на функция. Той не описва самата функция, а аспект от начина, по който функцията се държи, и какви изисквания се изпълняват в този контекст. Следователно качествен атрибут предоставя желанието да се посочи пригодността на дадена работа.

От взаимната координация в рамките на всяка дисциплина могат да бъдат разработени решения, които не са изолирани, а са последователни и подходящи за цялата архитектура. Разбира се, във всяка една система отделните атрибути носят различна тежест. Ако за архитектурата на една банкова система сигурността е от изключителна важност, то това не значи, че за архитектура от рода на „Софтуер за управление на кризисни ситуации“ този атрибут играе толкова важна роля. Относно “One stop shop услуги за граждани и бизнеса”, в контекста на предоставяне на лиценз за управление на МПС, има няколко основни атрибута, чието внедряване е от голямо значение:

- *Производителност* - системата трябва извършва обработка на потребителски заявки в рамките на 2 сек; многократното нарастване на броя едновременно работещи с нея потребители не трябва да дава значение върху бързината на отговора;
- *Ефективност* – системата да преизползва определени ресурси, за които това е допустимо, за да не се получава излишно препращане на заявки и препокриване на информация;
- *Скалируемост* – когато потребителят има нужда, той може да увеличи броя на ресурсите като клиенти и сървъри; по този начин се увеличава размера на

сървъра без много прекъсвания; дори ако размерът се увеличи, няма колебание относно разрешението за мрежови ресурси, тъй като сървърът е централизиран; системната архитектура позволява лесно преконфигуриране на отделните модули при нарастване на обема на потребителски заявки с цел запазване на определената производителност;

- *Сигурност* – защита на обменяните данни; в клиентската сървърна мрежа данните са добре защитени поради централизираната си архитектура, нещо повече, системата гарантира заличаването на данните след като веднъж си е свършила работата с тях;
- *Възможност за разширяемост* – системата гарантира лесно добавяне на още възли (администрации) при подобна нужда без да се наруши цялостната и работата на другите звена в архитектурата;
- *Възможност за поддръжка* – системата гарантира независимост на своите компоненти, което от своя страна води до по-лесна поддръжка и изисква по-малко ресурси;

II. Описание на основните сценарии и актьори

Тази точка разглежда 5 основни Use-case диаграми, благодарение на които ще се онагледят от един птичи поглед функционалностите, с които ще разполагат потребителите на системата. Като техника, ориентирана към потребителя, Use-case диаграмите помагат да се гарантира, че е разработена правилната система, като отчитат изискванията от гледна точка на потребителя. Тъй като са написани на natural език, use-case диаграмите са лесни за разбиране и осигуряват отличен начин за комуникация с клиенти и потребители. Също така могат да помогнат за управлението на сложността на проекта, като декомпозират проблема на основни функции, както ще се види и в примерните диаграми по-долу.

Основна диаграма

MainUseCase представя най-базовия изглед на системата, който включва нейните основни функционалности и актьори, в последствие представени по отделно в останалите четири Use-case диаграми.

Системи (systems):

- One stop shop
- Hospital
- BRC
- Payment Service
- School Administration

Случаи на употреба (use cases):

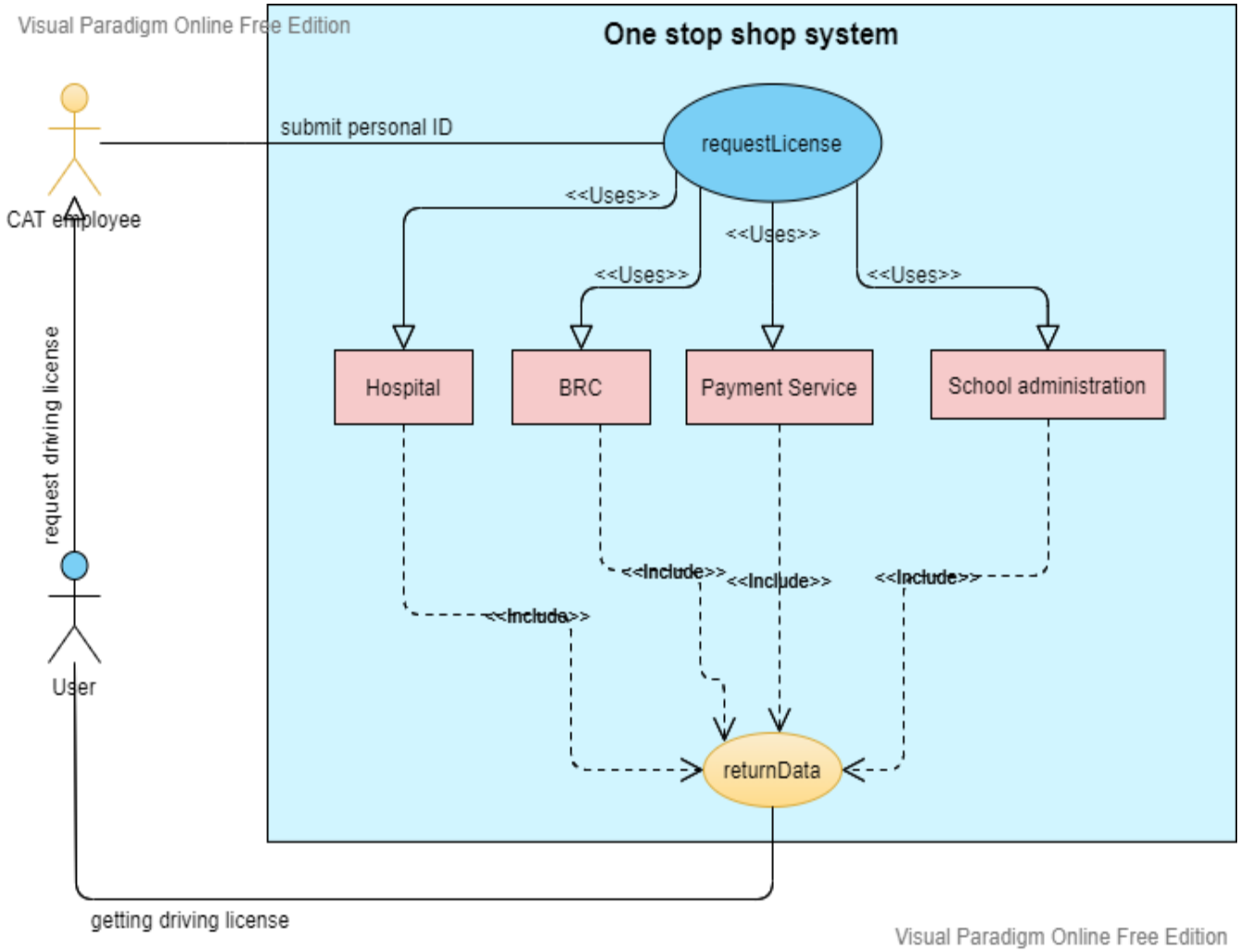
- requestLicense
- returnData

Актьори (actors):

- CAT employee
- User

Hospital, BRC, Payment Service и School Administration същност са отделните администрации, с които КАТ ще взаимодейства, за да изпълни своите задачи. Именно поради тази причина те не са класифицирани като случаи на употреба или актьори (някой

или нещо, което извършва дейност/поведение). Идеята е да се покаже от къде трябва да мине заявката на служителя в КАТ, така че крайният потребител, а именно гражданина да получи нужната информация и съответно лиценза си за управление на МПС – използва се връзка <<Uses>> с цел да се онагледи, че заявката requestLicense не може да бъде изпълнена без участието и на четирите компонента под нея.



1. Системи:

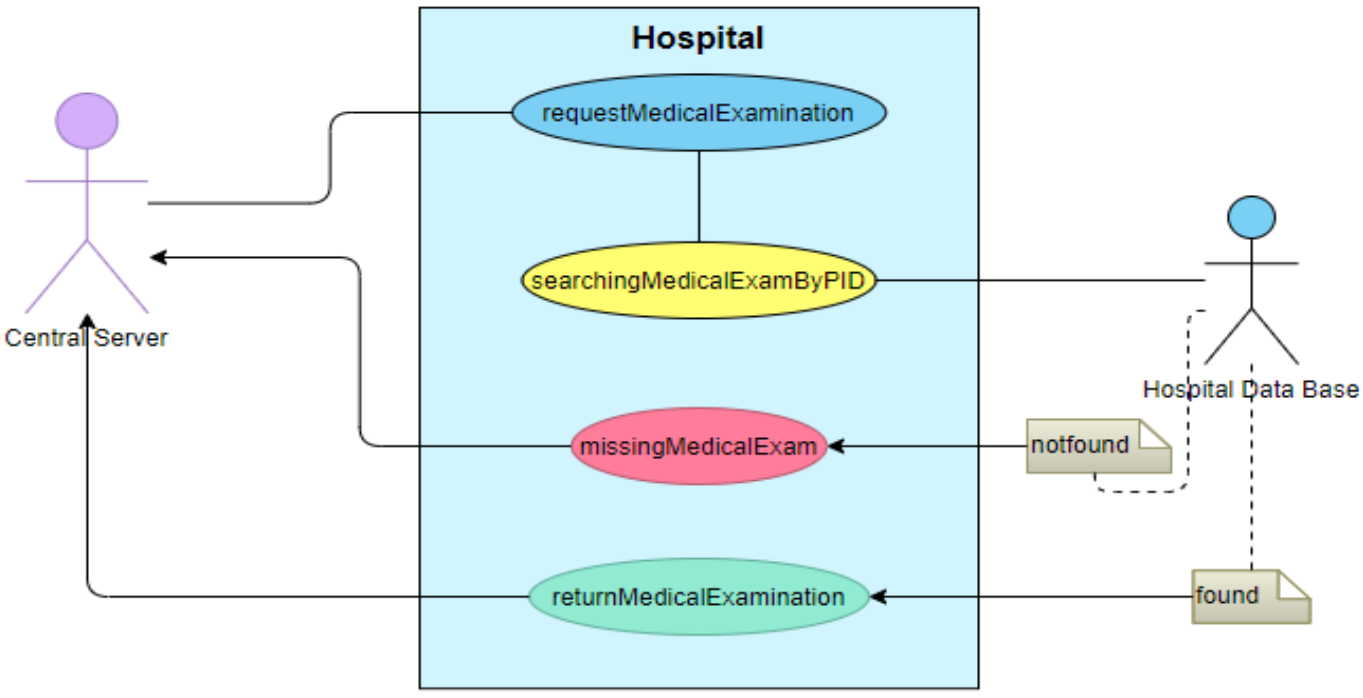
- *One stop shop* – това е цялата система, която улеснява предоставянето на лиценз за управление на МПС на гражданите;
- *Hospital* – администрацията, от която КАТ черпи информация относно статуса на картата за медицински преглед на водач на МПС;
- *BRC* – администрацията, от която КАТ черпи информация относно съществуването на удостоверение за завършен курс за първична до лекарска помощ / БЧК;
- *Payment Service* - администрацията, от която КАТ черпи информация относно (не)платени такси (в случай, че не са платени се заплащат на място);
- *School Administration* - администрацията, от която КАТ черпи информация относно съществуването на диплома за завършено най-малко основно образование;

2. **Случаи на употреба:**
   - *requestLicense* – позволява на служителя на КАТ (CAT employee) да подаде заявка относно получаването на лиценз за управление на МПС на конкретен гражданин (User);
   - *returnData* - връща информация – получава на лиценза за управление на МПС или неуспешна заявка, описваща причините за неполучаването на лиценза за управление на МПС;
3. **Актьори:**
   - *CAT employee* – представител на служител в КАТ, който обработва желанието на гражданите (User);
   - *User* - представител на гражданите, които изискват получването на лиценза им за управление на МПС;

Hospital диаграма

HospitalUseCase представлява връзката между централния сървър и болничното заведение с цел да се провери статуса на картата за медицински преглед на водача на МПС, т.е. дали прегледът е успешен, дали не е успешен или дали въобще съществува такъв.
Случаи на употреба:
- requestMedicalExamination
- searchingMedicalExamByPID
- missingMedicalExam
- returnMedicalExamination

Актьори:
- Central Server
- Hospital Data Base

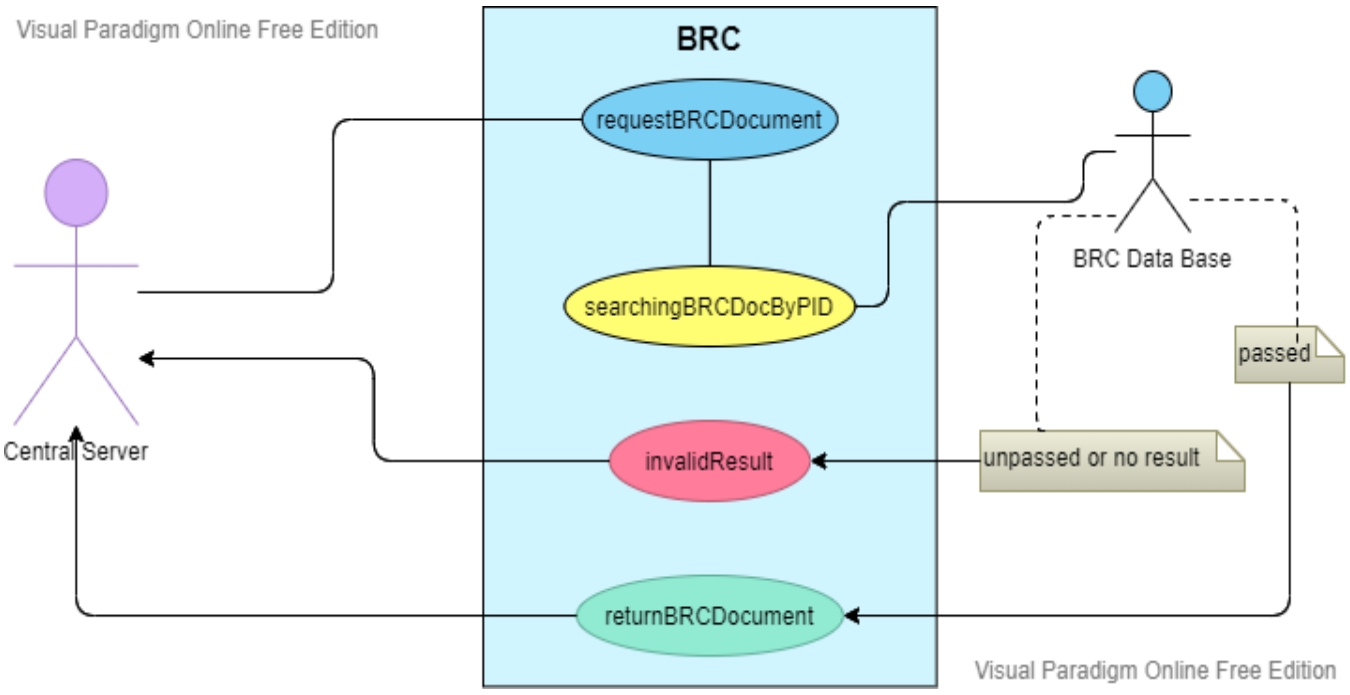


Заявката подадена от MainUseCase попада в Central Server, който взаимодейства с Hospital, за да извлече необходимите данни, които първоначалната заявка на служителя от КАТ изисква.

1. **Случаи на употреба:**
   - *requestMedicalExamination* - след като централният сървър получи заявка за получаване на лиценз за управление на МПС, той раздробява тази заявка на четири подзаявки, едната от които е requestMedicalExamination; тази заявка пуска запитване относно статуса на картата за медицински преглед на водач на МПС;
   - *searchingMedicalExamByPID* – приема като параметър ЕГН-то на конкретен гражданин и го използва като ключ, по който да търси медицинския преглед в Medical Data Base – има два статуса на връщане found и notfound;
   - *missingMedicalExam* – в случай, че searchingMedicalExamByPID върне notfound, тази заявка обработва причините за статуса на връщане като ги препраща обратно към заявителя в подходящ формат, в случая Central Server;
   - *returnMedicalExamination* - в случай, че searchingMedicalExamByPID върне found, то тази заявка връща към Central Server картата за медицинския преглед на водача на МПС;
2. **Актьори:**
   - *Central Server* – представлява централния сървър, който играе роля на разпределител на заявки към конкретните администрации, които са в състояние да му дадат нужната информация в подходящ формат;
   - *Hospital Data Base* – представлява базата данни, в която се търси картата за медицински преглед на водача на МПС по ЕГН;

BRC диаграма

BRCUseCase представлява връзката между централния сървър и БЧК с цел да се предостави информация относно дали курсът по БЧК е минат или въобще не е изкаран.



Случаи на употреба:
- requestBRCDocument
- searchingBRCDocByPID
- invalidResult
- returnBRCDocument

Актьори:
- Central Server
- BRS Data Base

Заявката подадена от MainUseCase попада в Central Server, който взаимодейства с BRC, за да извлече необходимите данни, които първоначалната заявка на служителя от КАТ изисква.

1. **Случаи на употреба:**
   - *requestBRCDocument* - след като централният сървър получи заявка за получаване на лиценз за управление на МПС, той раздробява тази заявка на четири подзаявки, едната от които е requestBRCDocument; тази заявка пуска запитване относно съществуването на удостоверение за завършен курс за първична до лекарска помощ / БЧК;
   - *searchingBRCDocByPID* – приема като параметър ЕГН-то на конкретен гражданин и го използва като ключ, по който да търси удостоверението за завършен курс за първична до лекарска помощ в BRC Data Base – има два статуса на връщане passed и unpassed or no result;
   - *invalidResult* - в случай, че searchingBRCDocByPID върне unpassed or no result, тази заявка обработва причините за статуса на връщане като ги препраща обратно към заявителя в подходящ формат, в случая Central Server;
   - *returnBRCDocument* - в случай, че searchingBRCDocByPID върне passed, то тази заявка връща към Central Server удостоверението за завършен курс за първична до лекарска помощ / БЧК на водача на МПС;
2. **Актьори:**
   - *Central Server* – представлява централния сървър, който играе роля на разпределител на заявки към конкретните администрации, които са в състояние да му дадат нужната информация в подходящ формат;
   - *BRS Data Base* - представлява базата данни, в която се търси удостоверението за завършен курс за първична до лекарска помощ на водача на МПС по ЕГН;

Payment Service Диаграма

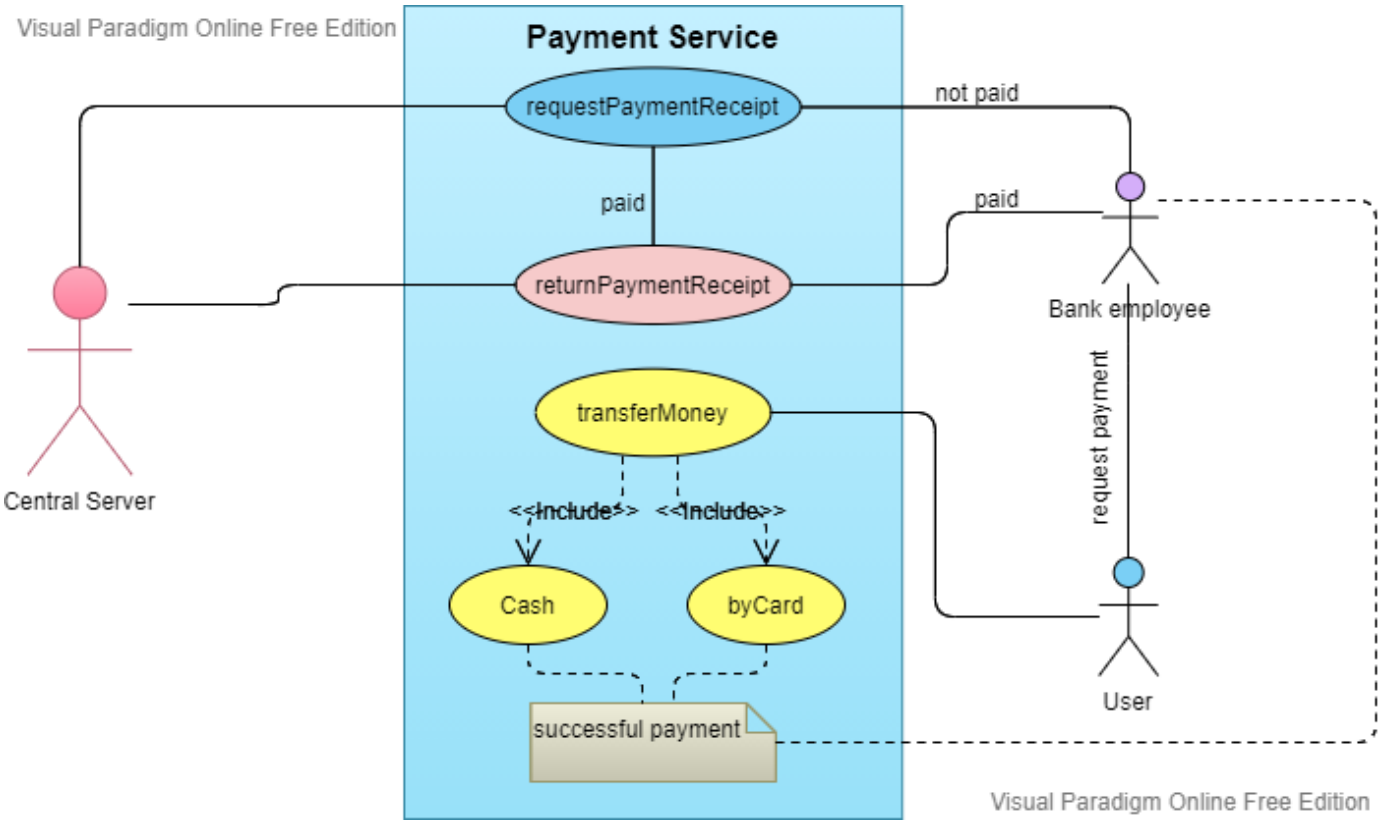
PaymentServiceUseCase представлява връзката между централния сървър и Бюрото за плащания с цел да се предостави информация относно това дали са заплатени нужните такси или не – в случай, че не са е възможно заплащане на момента.

Случаи на употреба:
- requestPaymentReceipt
- returnPaymentReceipt
- transferMoney
- Cash
- ByCard

Актьори:
- Central Server

- Bank employee
- User

Заявката подадена от MainUseCase попада в Central Server, който взаимодейства с Payment Service, за да извлече необходимите данни, които първоначалната заявка на служителя от КАТ изисква.



1. Случаи на употреба:

- *requestPaymentReceipt* - след като централният сървър получи заявка за получаване на лиценз за управление на МПС, той раздробява тази заявка на четири подзаявки, едната от които е requestPaymentReceipt; тази заявка пуска запитване относно това дали са заплатени нужните такси или не; само чрез проверка по ЕГН изкарва чакащите плащане такси и съответно успешно платените - има два статуса на връщане paid и not paid;
- *returnPaymentReceipt* – независимо дали гражданинът е заплатил предварително нужните такси или на място в КАТ с помощта на Payment Service-а, returnPaymentReceipt връща на Central Server бележка за успешно платени такси в подходящ формат;
- *transferMoney* – в случай, че гражданинът не е заплатил предварително нужните такси, то transferMoney му дава възможност да го направи на място в КАТ – има статус на връщане successful payment; съществуват два варианта:
  - *Cach* – осъществява плащане в кеш;
  - *byCard* – осъществява плащане чрез карта на POS терминал;

2. Актьори

- *Central Server* – представлява централния сървър, който играе роля на разпределител на заявки към конкретните администрации, които са в състояние да му дадат нужната информация в подходящ формат;
- *Bank employee* – в случай, че гражданинът (User) не е заплатил нужните такси предварително, то неговото плащане може да бъде обработено на момента от съответния банков служител, който праща заявка за плащане към гражданина (User); след като заплати (независимо чрез Cash или byCard) успешно, Bank employee обработва това плащане и го връща към returnPaymentReceipt в подходящ формат;
- *User* – представител на гражданите, които изискват получаването на лиценза им за управление на МПС; има право да заплати на място в кеш или чрез карта на POS терминал;

School administration диаграма

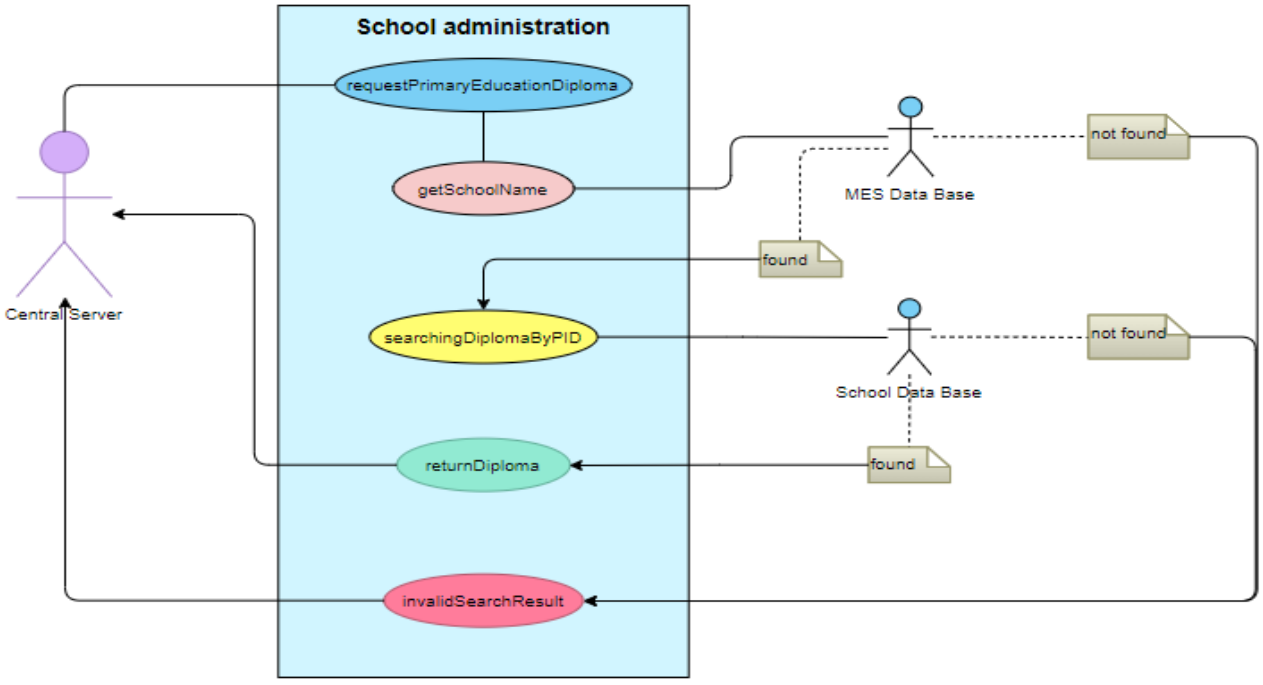
SchoolAdministrationUseCase представлява връзката между централния сървър и Училищната администрация с цел да се предостави информация относно съществуването на диплома за завършено най-малко основно образование;

Случаи на употреба:

- requestPrimaryEducationDiploma
- getSchoolName
- searchingDiplomaByPID
- returnDiploma
- invalidSearchResult

Актьори:

- Central Server
- MES Data Base
- School Data Base



Заявката подадена от MainUseCase попада в Central Server, който взаимодейства с Payment Service, за да извлече необходимите данни, които първоначалната заявка на служителя от КАТ изисква

1. **Случаи на употреба:**
   - *requestPrimaryEducationDiploma* - след като централният сървър получи заявка за получаване на лиценз за управление на МПС, той раздробява тази заявка на четири подзаявки, едната от които е requestPrimaryEducationDiploma; тази заявка има за цел да подаде в подходящ вид данните на getSchoolName, за да се осъществи по-нататашното търсене;
   - *getSchoolName* – търси в базата данни на МОН училищното заведение, в което конкретното лице, подадено от requestPrimaryEducationDiploma е учило – има два статуса на връща, съответно found и not found;
   - *searchingDiplomaByPID* – в случай, че getSchoolName върне статус found се задейства searchingDiplomaByPID, който търси в базата на съответното училище дали съществува диплома за завършено най-малко основно образование по ЕГН – има два статуса на връща, съответно found и not found;
   - *returnDiploma* – в случай, че searchingDiplomaByPID върне статус found се задейства returnDiploma, който в подходящ формат връща потвърждение за съществуването на най-малко основно образование към Central Server;
   - *invalidResult* – връща на Central Server неуспешната заявка, като подава причините за това в подходящ формат; неуспешна заявка може да възникне на две места в сценария – 1. Ако в базата на МОН не се открие училище, в което съответния гражданин е бил, 2. Ако в базата на училищната администрация не се открие диплома за най-малко основно образование;
2. **Актьори**:
   - *Central Server* – представлява централния сървър, който играе роля на разпределител на заявки към конкретните администрации, които са в състояние да му дадат нужната информация в подходящ формат;
   - *School Data Base* - представлява базата данни, в която се търси най-малко основно образование на гражданина, изискващ лиценза за управление на МПС;
   - *MES Data Base* - представлява базата данни, в която се търси училищното заведение, което е посещавал гражданина, изискващ лиценза за управление на МПС;

III. Описание на логическия изглед на архитектурата

В тази точка се разглеждат 4 UML диаграми, които включват в себе си една state machine диаграма, две class диаграми и една компонентна диаграма. Благодарение на тяхното реализиране и описание ще представим логическия изглед на софтуерния продукт. Основната цел на логическия изглед е да се дефинират компонентите, които изграждат системата и да се определят интерфейсите, чрез които те ще комуникират и взаимодействат помежду си. Всеки компонент е отговорен за една ясна цел в цялата система и взаимодейства само с други, които също имат конкретна функционалност и роля. Логическият изглед е предназначен за проектантите на системата и неговите разработчици.

Основна компонентна диаграма

Основната компонентна диаграма предоставя компонентите, които са малко по-висока абстракция от класовете, интерфейсите, които са операции използвани от компонентите и третият елемент са връзките, които са необходими за дадени зависимости. Това е по-базов изглед, в сравнение с class диаграмата, която ще бъде описана по-долу.

Компоненти (Components):

- CAT
- Central Server
- Queries
- References
- BRC
- Hospital
- Payment Service
- School Administration
- BRC Data Base
- Hospital Data Base
- MES Data Base
- School Data Base

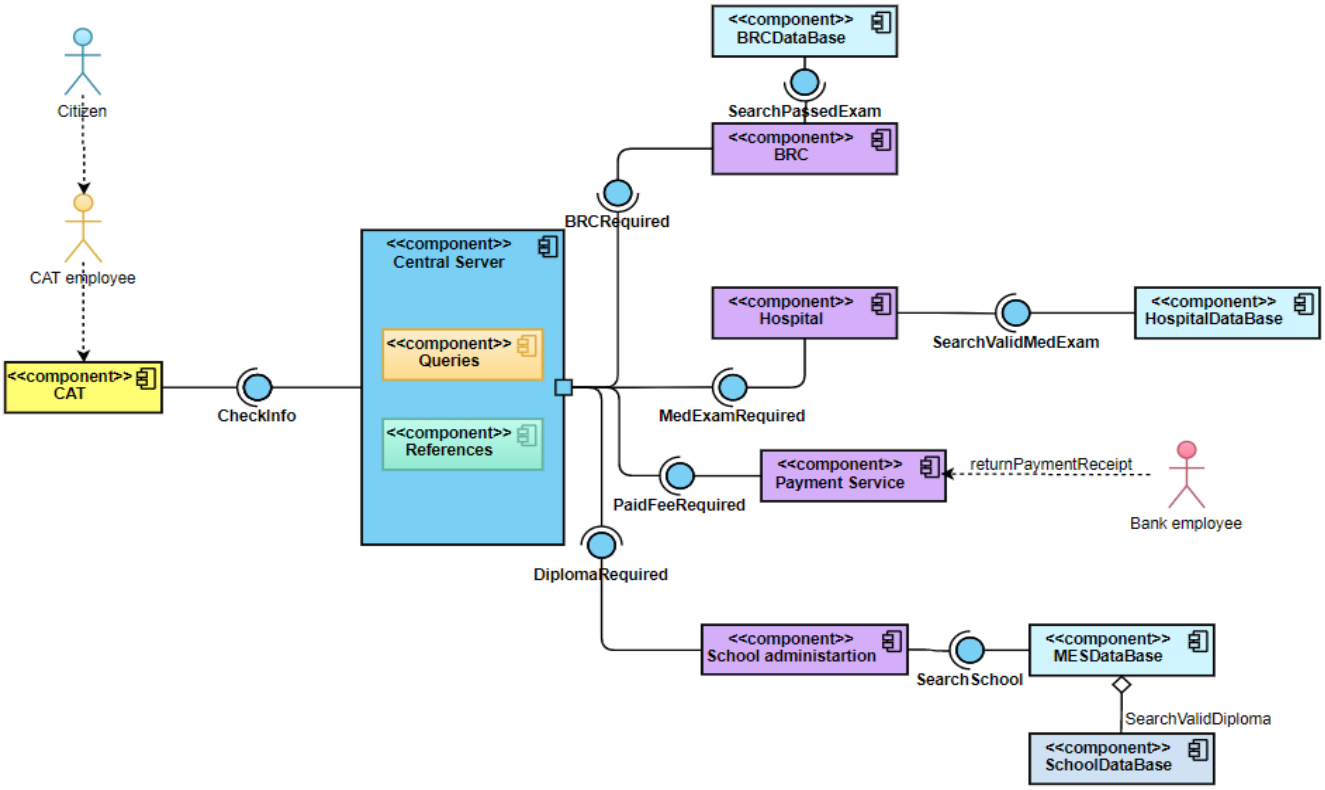
Интерфейси (Interfaces):

- CheckInfo
- BRCRequired
- MedExamRequired
- PaidFeeRequired
- DiplomaRequired
- SearchPassedExam
- SearchValidMedExam
- SearchSchool

Връзки (Relationships):

- Агрегация

Първият компонент, до който трябва да се осъществи връзка е именно КАТ, до който негов служител изпраща запитване, което е поискано от гражданин. С интерфейса CheckInfo се изпраща запитване до Central Server за оформяне на необходимите заявки за предоставяне на информация. BRC, Hospital, Payment Service и School Administration са компонентите, до които Central Server изпраща необходимите заявки, представени с интерфейсите BRCRequired, MedExamRequired, PaidFeeRequired и DiplomaRequired, за да изпрати запитване, с което да получи необходимата информация. Всяка отделна администрация разполага със собствена база данни: BRC Data Base, Hospital Data Base, Mes Data Base и School Data Base, чрез която се извършва справката за всеки един компонент, за който тя е поискана. Изключение прави компонента Payment Service, който има връзка единствено с банков служител за получаване на платежно. Връзката до отделните бази се осъществява с интефейсите: SearchPassedExam, SearchValidMedExam и SearchSchool. Изключение прави компонента SchoolDataBase, до която връзката се осъществява чрез агрегация.



1. Компоненти:

- *KAT* – компонента, от който започва изпращането на запитване за получаване на необходимите данни за управление на МПС;
- *Central Server* – главният сървър, в който се обработват заявки, връщат се отговори на тези заявки и се пази временно необходимата информация на гражданина, като сам по себе си той съдържа още два компонента Queries, който е базов и нужен за осъществяване на всяка една заявка, и References, който разполага с подадените заявки и получените отговори за тях в определен период от време;
- *BRC*, *Hospital*, *Payment Service* и *School Administration* – администрациите, до които се изпраща запитване от Central Server;
- *BRC Data Base* – базата данни, с която БЧК пази информацията за това кои граждани имат успешно положен изпит;
- *Hospital Data Base* – базата данни, с която болнично заведение пази информацията на гражданите, които притежават валидно медицинско;
- *MES Data Base* – базата данни, с която МОН пази информация за това гражданите в кое училище са завършили, ако са посещавали въобще училищно заведение;
- *School Data Base* – базата данни, с която всяко едно училище разполага и пази дипломите за завършено образование;

2. Интерфейси

- *CheckInfo* – интерфейсът, чрез който се осъществява връзката между КАТ и централния сървър. Чрез кратък XML е демонстрирано как би изглеждал целия процес от запитване до получаване на информацията от гледна точка на централния сървър.

```
<query>
        <to>Central Server</to>
        <from>CAT employee</from>
        <heading>Get Information</heading>
        <body>
                <Employee>
                        <name>Petyr Ivanov</name>
                        <city>Lovech</city>
                        <data>dd-mm-yyyy</data>
                </Employee>
                <Citizen Data>
                        <PID>**********</PID>
                </Citizen Data>
        </body>
</query>

<responce>
        <to>CAT employee></to>
        <from>Central Server</from>
        <heading>Driving License</heading>
        <data>dd-mm-yyyy</data>
        <body>
                <Citizen Data>
                        <name>Mariq Vekova</name>
                        <city>Teteven</city>
                        <PID>**********</PID>
                        <BRCDoc>passed</BRCDoc>                 //unpassed or no result
                        <MedExam>found</MedExam>                //not found
                        <Payment>paid</Payment>                 //not paid
                        <PrimDiploma>found</PrimDiploma>        //not found
                </Citizen Data>
        </body>
</responce>
```

- *BRCRequired* и *SearchPassedExam* – двата интерфейса, които са необходими за получаване на пълната информация за успешно положен изпит в БЧК, BRCRequired е интерфейсът от централния сървър към компонента БЧК, а SearchPassedExam е интерфейсът между компонента БЧК и неговата база данни, в която трябва да се направи необходимата справка. Чрез кратък XML е демонстрирано как би изглеждал целия процес от запитване до получаване на информацията от гледна точка на централния сървър.

```
<query>
        <to>BRC</to>
        <from>Central Server</from>
        <heading>Get BRCDoc</heading>
        <data>dd-mm-yyyy</data>
        <body>
                <given data>
                        <PID>**********</PID>
                </given data>
                <required>
                        <name>Mariq Vekova</name>
                        <PID>**********</PID>
                        <BRCDoc>'status'</BRCDoc>
                </required>
        </body>
<query>

<response>
        <to>Central Server></to>
        <from>BRC</from>
        <heading>BRCDoc</heading>
        <data>dd-mm-yyyy</data>
        <body>
                <Citizen Data>
                        <name>Mariq Vekova</name>
                        <city>Teteven</city>
                        <PID>**********</PID>
                        <BRCDoc>passed</BRCDoc>                 //unpassed or not found
                </Citizen Data>
        </body>
</response>
```

- *MedExamRequired и SearchValidMedExam* – двата интерфейса, които са необходими за получаване на пълната информация за валидно медицинско. MedExamRequired е интерфейсът между централния сървър и компонента Hospital, а SearchValidMedExam е интерфейсът между Hospital и неговата база данни. Чрез кратък XML е демонстрирано как би изглеждал целия процес от запитване до получаване на информацията от гледна точка на централния сървър.

```
<query>
       <to>Hospital</to>
       <from>Central Server</from>
       <heading>Get Medical Examination</heading>
       <data>dd-mm-yyyy</data>
       <body>
              <given data>
                     <PID>**********</PID>
              </given data>
              <requided>
                    <name>Mariq Vekova</name>
                    <PID>**********</PID>
                    <MedExam>'status'</<MedExam>
              </required>
        </body>
<query>

<response>
      <to>Central Server></to>
      <from>Hospital</from>
      <heading>Medical Examination</heading>
      <data>dd-mm-yyyy</data>
      <body>
             <Citizen Data>
                    <name>Mariq Vekova</name>
                    <city>Teteven</city>
                    <PID>**********</PID>
                    <MedExam>found</MedExam>                    //not found
             </Citizen Data>
      </body>
</response>
```

- *PaidFeeRequired* – интерфейса, който осъществява връзката между централния сървър и компонента Payment Service. Чрез кратък XML е демонстрирано как би изглеждал целия процес от запитване до получаване на информацията от гледна точка на централния сървър.

```
kquery>
       <to>Payment Service</to>
       <from>Central Server</from>
       <heading>Get Payment Receipt</heading>
       <data>dd-mm-yyyy</data>
       <body>
              <given data>
                     <PID>**********</PID>
              </given data>
              <requided>
                    <name>Mariq Vekova</name>
                    <PID>**********</PID>
                    <Payment>'status'</Payment>
              </required>
        </body>
<query>

<response>
     <to>Central Server></to>
     <from>Payment Service</from>
     <heading>Payment Receipt</heading>
     <data>dd-mm-yyyy</data>
     <body>
            <Citizen Data>
                    <name>Mariq Vekova</name>
                    <city>Teteven</city>
                    <PID>**********</PID>
                    <Payment>paid</Payment>                 //not paid
            </Citizen Data>
     </body>
</response>
```

- *DiplomaRequired* и *SearchSchool* – двата интерфейса, които са необходими за получаване на информацията дали гражданинът имат завършено средно образование. DiplomaRequired е връзката между централния сървър и компонента School Administration, а SearchSchool е връзката между School Administration и неговата база данни. Чрез кратък XML е демонстрирано как би изглеждал целия процес от запитване до получаване на информацията от гледна точка на централния сървър.

```
<query>
      <to>School Administration</to>
      <from>Central Server</from>
      <heading>Get PrimaryDiploma</heading>
      <data>dd-mm-yyyy</data>
      <body>
             <given data>
                    <PID>**********</PID>
             </given data>
             <requided>
                  <name>Mariq Vekova</name>
                  <PID>**********</PID>
                  <PrimDiploma>'status'</PrimDiploma>
             </required>
      </body>
<query>

<response>
     <to>Central Server></to>
     <from>School Administration</from>
     <heading>Primary Diploma</heading>
     <data>dd-mm-yyyy</data>
     <body>
            <Citizen Data>
                  <name>Mariq Vekova</name>
                  <city>Teteven</city>
                  <PID>**********</PID>
                  <PrimDiploma>found</PrimDiploma>                  //not found
            </Citizen Data>
     </body>
</response>
```

3. Връзки

- *Агрегация* – това е връзката между базата данни на МОН и конкретната база данни за определено училище, избрана е тази тип връзка, защото базата данни на конкретното училището е „част“ от тази на МОН

Клас диаграма от гледна точка на централния сървър към отделните администрации за съставяне на заявки

Диаграмата на класовете е основният градивен елемент за конструиране и визуализиране на обектно-ориентираните системи. Компонентите, които ги изграждат са класове, атрибути, операции (методи) и връзките между обектите.

Класове:

- <<abstract>> Query
- Central Server
- <<enumeration>> Response
- BRC Query
- Payment Query
- School Administration Query
- Hospital Query

- CAT Query
- Citizen

Атрибути:

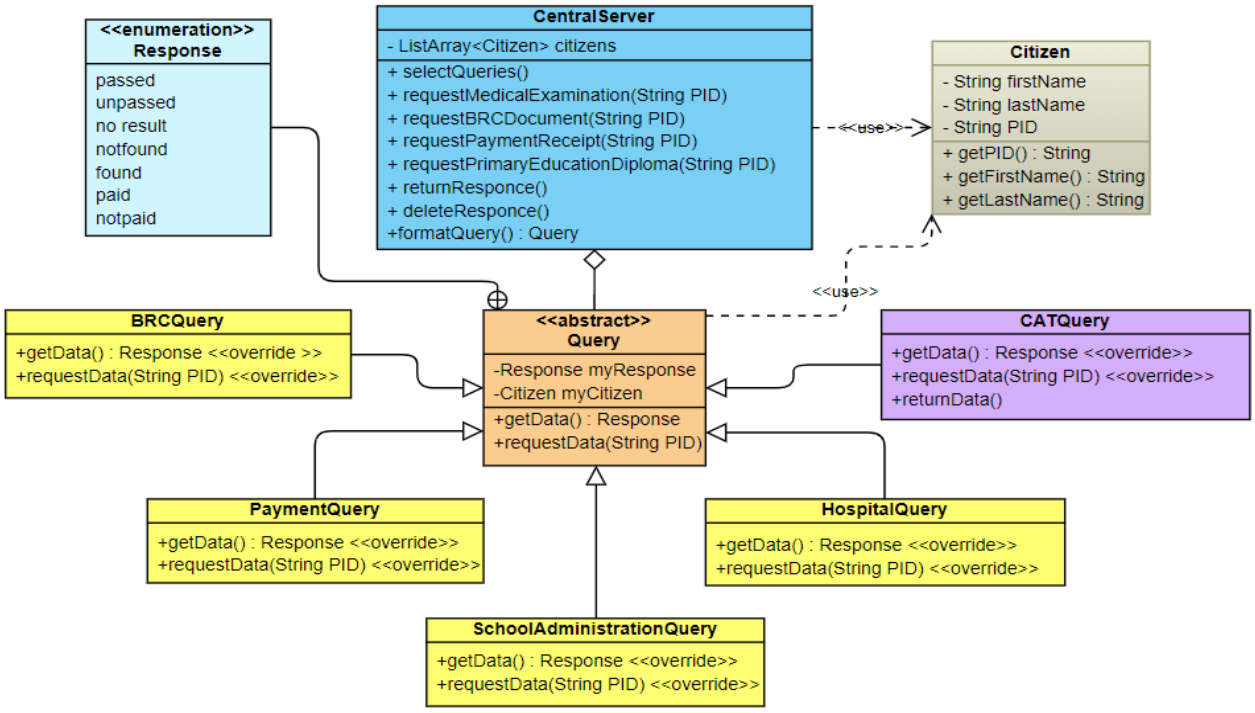
- Response myResponce
- Citizen myCitizen
- ListArray <Citizen> citizens
- String firstName
- String lastName
- String PID

Операции (методи):

- requestData()
- selectQueries()
- requestMedicalExamination(String PID)
- requestBRCDocument(String PID)
- requestPaymentReceipt(String PID)
- requestPrimaryEducationDiploma(String PID)
- returnResponse()
- deleteResponse()
- formatQuery() : Query
- getPID() : String
- getFirstName() : String
- getLastName() : String

Връзки:

- Наследяване (Inheritance)
- Агрегация (Aggregation)
- Зависимост (Dependency)
- Ограничаване (Containment)



Реализацията на тази клас диаграма започва от класа Central Server, който осъществява връзка чрез агрегация с абстрактния клас Query и зависимост с класа Citizen, който визуализира зависимост и със самия клас Query. Абстрактния клас се явява базов за останалите 5 класа, които са: BRCQuery, PaymentQuery, SchoolAdministrationQuery, HospitalQuery и CATQuery, които се нуждаят от методите му, за да могат да реализират своите заявки, които да бъдат изпратени до отделните администрации.

1. Класове

- *CentralServer* – основния клас, благодарение, на който става възможна реализацията и на останалите класове;
- Citizen – клас, в който се пази основна информация за конкретния човек;
- *CATQuery* – в него се съдържат методите необходими за изпращането на request към КАТ за предоставяне на информация и върнатия отговор от службата на КАТ;
- *HospitalQuery* – в него се съдържат методи за request за получаване на информация и върнатия отговор от болничното заведение;
- *SchoolAdministrationQuery* - в него се съдържат методи за request за получаване на информация и върнатия отговор от училищната администрация;
- *PaymentQuery* - в него се съдържат методи за request за получаване на информация и върнатия отговор от бюрото за плащания;
- *BRCQuery* - в него се съдържат методи за request за получаване на информация и върнатия отговор от БЧК;

2. Атрибути

- *Response myResponce и Citizen myCitizen* – private атрибути в класа Query, които са необходими за съхраняване на информация при извършването на заявки;
- *ListArray <Citizen> citizens* – private списък от тип класа Citizen, като този списък се намира в класа за централния сървър, в него се съхраняват временно данните за отделните граждани, които са били в КАТ;
- *String firstName* – private атрибут, който съхранява първото име на гражданина, като този атрибут се намира в класа Citizen;
- *String lastName* – private атрибут, който съхранява второто име на гражданина, като този атрибут се намира в класа Citizen;

3. Операции (методи)

- *requestData()* – публичен метод, който се наследява и override-ва от всичките Query наследници, той бива пренаписан във всеки клас поотделно в зависимост от заявката, за която ще бива използван, чрез него се прави запитване до отделните администрации за предоставяне на информация;
- *getData()* – публичен метод, който се наследява и override-ва от всичките Query наследници, типът на връщана стойност му е класът Response, използва се за връщане на резултат;
- *selectQueries()* – публичен метод в централния сървър, който избира подходящите заявки за всяка една отделна операция;
- *requestMedicalExamination(String PID), requestBRCDocument(String PID), requestPaymentReceipt(String PID), requestPrimaryEdicationDiploma(String*

PID) – публични методи в централния сървър, които съдържат заявките, които се изпращат като запитване за получаване на информация до всяка една администрация;

- *returnResponse()* – публичен метод в централния сървър, който връща информацията получена за даден гражданин;
- *deleteResponse()* – публичен метод в централния сървър, който след определен период от време изтрива информацията получена за даден гражданин;
- *formatQuery()* – публичен метод в централния сървър, който форматира получените заявки с информация;
- *getPID()* – публичен метод в класа Citizen, връщаната стойност му е String, като неговата роля е да вземе ЕГН на определен гражданин;
- *getFirstName() и getLastName()* – публични методи в класа Citizen, връщаната им стойност е от тип String, като тяхната роля е да вземат съответно първото име и фамилното име на определен гражданин;

4. Връзки

- *Наследяване* – този вид връзка е реализирана с класовете BRCQuery, PaymentQuery, SchoolAdministrationQuery, HospitalQuery, CatQuery, които наследяват абстрактния клас Query;
- *Агрегация* – този вид връзка е реализирана с класа за централния сървър и абстрактния клас Query ;
- *Зависимост* – този вид връзка е реализирана между класовете за централния сървър и Query с класа Citizen, защото те използват параметрите в него;
- *Ограничаване* – това е връзката между класът от тип enum и Query, използвана е именно тази връзка, защото резултат от дадена заявка може да бъде само една фиксирана константа от тип enum;

Клас диаграма от гледна точка на централния сървър към отделните администрации и техните база данни

Класове:

- Central Server
- Hospital
- School Administration
- Payment Service
- BRC
- Hospital Data Base
- BRC Data Base
- MES Data Base
- School Data Base

Атрибути:

- ListArray <Citizen> citizens
- Doctor doctors
- Patient patients
- Staff staff
- Department departments
- Room rooms
- Documentations document

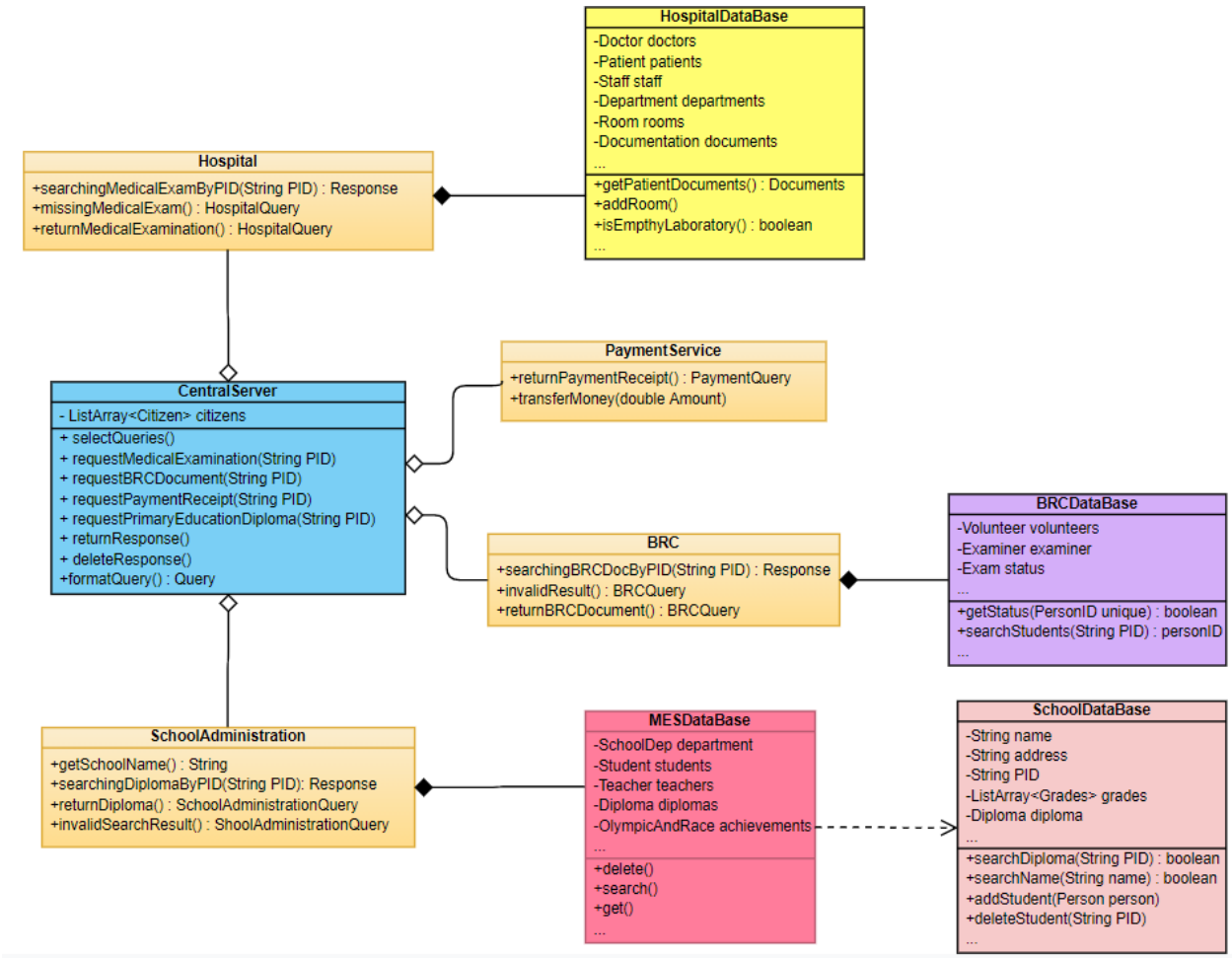
- Volunteer volunteers
- Examiner examiner
- Exam status
- SchoolDep department
- Student students
- Teacher teachers
- Diploma diplomas
- OlympicAndRace achievements
- String name
- String address
- String PID
- ListArray <Grades> grades

Операции (методи):

- selectQueries()
- requestMedicalExamination(String PID)
- requestBRCDocument(String PID)
- requestPaymentReceipt(String PID)
- requestPrimaryEducationDiploma(String PID)
- returnResponse()
- deleteResponse()
- formatQuery() : Query
- searchingMedExamByPID(String PID) : Response
- missingMedicalExam() : HospitalQuery
- returnMedicalExamination() : HospitalQuery
- getSchoolName() : String
- searchingDiplomaByPID(String PID) : Response
- returnDiploma() : SchoolAdministrationQuery
- invalidSearchResult() : SchoolAdministrationQuery
- getPatientDocuments() : Documents
- addRoom()
- isEmptyLaboratory() : Boolean
- returnPaymentReceipt() : PaymentQuery
- transferMoney(double amount)
- searchingBRCDocByPID(String PID) : Response
- invalidResult() : BRCQuery
- returnBRCDocument() : BRCQuery
- delete()
- search()
- get()
- getStatus(PersonalId unique) : Boolean
- searchStudents(String PID) : PersonalId
- searchDiploma(String PID) : Boolean
- searchName(String PID) : Boolean
- addStudent(Person person)
- deleteStudent(String PID)

Връзки:

- Агрегация
- Композиция
- Зависимост



1. Класове

- *CentralServer* – сървърът, който стои в основата на целия процес;
- *Hospital, SchoolAdministration, PaymentService и BRC* – отделните администрации;
- *HospitalDataBase, BRCDataBase,MESDataBase и SchoolDataBase* – база данни за отделните администрации;

2. Атрибути

- *ListArray<Citizens> citizens* – списък с гражданите, който се пази временно в централния сървър;
- *Doctor doctors, Patient patients, Staff staff, Department departments, Room rooms, Documentation document* – атрибути, които пазят информация в класа HospitalDataBase;
- *Volunteer volunteers, Examiner examiner, Exam status* – атрибути, които пазят информация в класа BRCDataBase;

- *SchoolDep department, Student students, Teacher teachers, Diploma diplomas, OlympicAndRace achievements* – атрибути, които пазят информация в класа MESDataBase;
- *String name, String address, String PID, ListArray<Grades> grades, Diploma diploma* – атрибути, които пазят информация в класа SchoolDataBase;

3. Операции (методи)

- *searchingMedicalExamByPID(String PID) : Response, missingMedicalExam() : HospitalQuery, returnMedicalExamination() : HospitalQuery* – публични методи в класа Hospital, като всеки един от тях извършва определена функция: търси медицинско, връща резултат при липса на такова, връща медицинско;
- *getSchoolName() :String, searchingDiplomaByPID(String PID) : Response, returnDiploma() : SchoolAdministrationQuery, invalidSearchResult() : SchoolAdministrationQuery* – публични методи в класа SchoolAdministration, като всеки един от тях извършва определена функция – взимане на името на съответното училищно заведение, търсене на диплома по ЕГН, връщане на диплома при валиден резултат и функция за връщане на невалиден резултат при неоткрита диплома;
- *returnPaymentReceipt() : PaymentQuery, transferMoney(double amount)* – публични методи в класа PaymentService, като първия метод връща квитанция за платена такса, а вторият предоставя начин за осъществяване на плащане, ако тя липсва;
- *searchingBRCDocByPID(String PID) :Response, invalidResult() : BRCQuery, returnBRCDocument() : BRCQuery* – публични методи в класа BRC, като тяхната роля е : търсене на документ за преминат курс по БЧК, връщане на резултат при неположен или неуспешен изпит и връщане на документ за преминат курс за БЧК;
- *getPatientDocuments() : Documents, addRoom(), isEmptyLaboratory() : Boolean* – публични методи в класа HospitalDataBase, които са характерни за всяка база данни, която притежава болнично заведение
- *getStatus(PersonalID unique) : Boolean, searchStudents(String PID) : personalID* – публични методи в класа BRCDataBase, като тяхната роля е : взимане на информация за успешно преминат курс и търсене на граждани, които са се записали;
- *delete(), search(), get()* – публични методи в класа MESDataBase, тяхната роля е на изтриване, търсене и взимаме на информация от данни на МОН;
- *serachDiploma(String PID) : Boolean, searchName(String name) : Boolean, addStudent(Person person), deleteStudent(String PID)* – публични методи в класа SchoolDataBase, тяхната роля е : търсене на диплома, търсене по име, добавяне на нови ученици, премахване на ученици;

4. Връзки

- *Агрегация* - осъществява се между класовете Hospital, PaymentService, BRC, SchoolAdministration и CentralServer;
- *Композиция* – осъществява се между Hospital, BRC и SchoolAdministratin и техните база данни съответно HospitalDataBase, BRCDateBase и SchoolAdministrainDataBase;
- *Зависимост* – осъществява се между класове MESDataBase и SchoolDataBase;

State Machine диаграма

Диаграма на състоянието се използва за описание на поведението на системата. Това поведение се анализира и представя от поредица от събития, които могат да се случат в едно или повече възможни състояния.

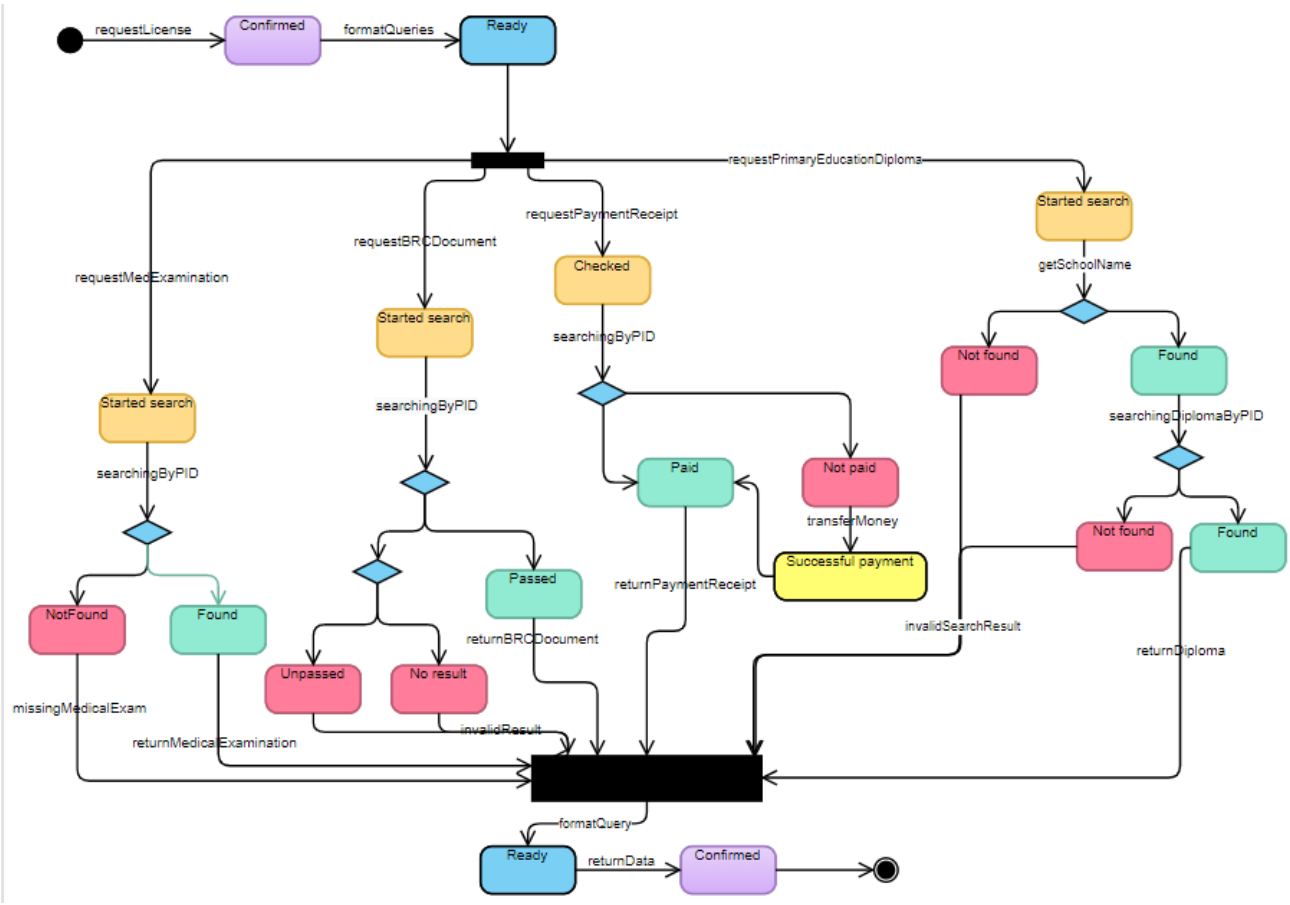
Състояния:

- Confirmed
- Ready
- Started search
- Checked
- Not found
- Found
- Final State
- Passed
- No result
- Unpassed
- Paid
- Unpaid
- Successful payment

Събития:

- requestLicense
- formatQueries
- requestMedExamination
- requestBRCDocument
- requestPaymentReceipt
- requestPrimaryEdicationDiploma
- searchingByPID
- serachingDiplomaByPID
- getSchoolName
- missingMedicalExam
- returnMedicalExamination
- invalidResult
- returnBRCDocument
- returnPaymentReceipt
- transferMoney
- invalidSearchResult
- returnDiploma
- formatQuery
- returnData

Началното състояние, от което започва представянето на системата е Initial State. Всяко едно състояние е предхождано и водено от определено събитие, което определя резултата в следващия етап на диаграмата. Когато имаме дадено условие, което ще определи следващото състояние, го изобразяваме със син обърнат ромб. Когато преминем през всички необходими състояния и събития ще стигнем до края на нашата диаграма, а именно finalState, който бележи края на тази реализация.



1. Състояния

- *Confirmed* – състояние на приет request;
- *Ready* – състояние, в което вече е прието запитването и системата е готова да изпълни своята задача;
- *Started search* – състояние на търсене на информация за всяка една администрация;
- *Checked* – състояние, оказващо, че е проверено за съществуването на бележка за платени такси;
- *Not found* – състояние, което ни дава информация, че търсения документ от определена администрация не е открит;
- *Found* – състояние, което ни дава информация, че търсения документ от определена администрация е открит;
- *Final State* – крайното състояние на системата, която тя вече е изпълните всичките си заявки, върнала е отговор и ги е потвърдила;
- *Passed* – състояние, показващо взет БЧК изпит;
- *Unpassed* – състояние, показващо не взет БЧК изпит;
- *No result* – състояние, показващо, че няма резултат за полаган БЧК изпит;
- *Successful payment* – състояние, потвърждаващо успешно плащане на касата в КАТ;

- *Paid* – състояние, показващо дали са платени нужните такси преди отиване в КАТ;
- *Unpaid* – състояние, показващо, че таксите не са платени преди отиване в КАТ;

2. Събития

- *requestLicense* – първото събитие, което настъпва, когато гражданин изпраща запитване за получаване на свидетелство за управление на МПС;
- *formarQueries* – второто събитие, което форматира заявките, които трябва да бъдат изпратени до всяка една отделна администрация;
- *requestMedExam, requestBRCDocument, requestPaymentReceipt, requestPrimaryEducationDiploma* – събития, които изпращат запитване до своите администрации за получаване на необходимата информация;
- *searchingByPID* – търсенето на информация във всяка една администрация се осъществява чрез търсене по ЕГН;
- *getSchoolName* – търсене на училището, в което гражданинът, който желае да получи свидетелство за МПС е положил своето образование;
- *searchingDiplomaByPID* – търсене на диплома за завършено средно образование на гражданина;
- *missingMedicalExam* – събитие настъпващо при липса на медицински документ на гражданина;
- *returnMedicalExamination* – събитие, което настъпва ако гражданина има издаден медицински документ;
- *invalidResult* – събитие, което настъпва при невалиден положен изпит в БЧК;
- *returnBRCDocument* – събитие, което настъпва при успешно издържан изпит в БЧК и предоставяне на документ за изпълнението му;
- *returnPaymentReceipt* – събитие, което настъпва при платена такса и предоставяне на документ за изпълнението му;
- *transferMoney* – събитие, в което се превеждат пари, за да се осъществи таксата, която се изисква;
- *invalidSearchResult* – събитие, което настъпва, ако гражданина няма завършено средно образование;
- *returnDiploma* – събитие, което настъпва при открита диплома за средно образование;
- *formatQuery* – форматиране на всички върнати заявки;
- *returnData* – предоставяне на цялата нужна информация на КАТ за съответното лице;

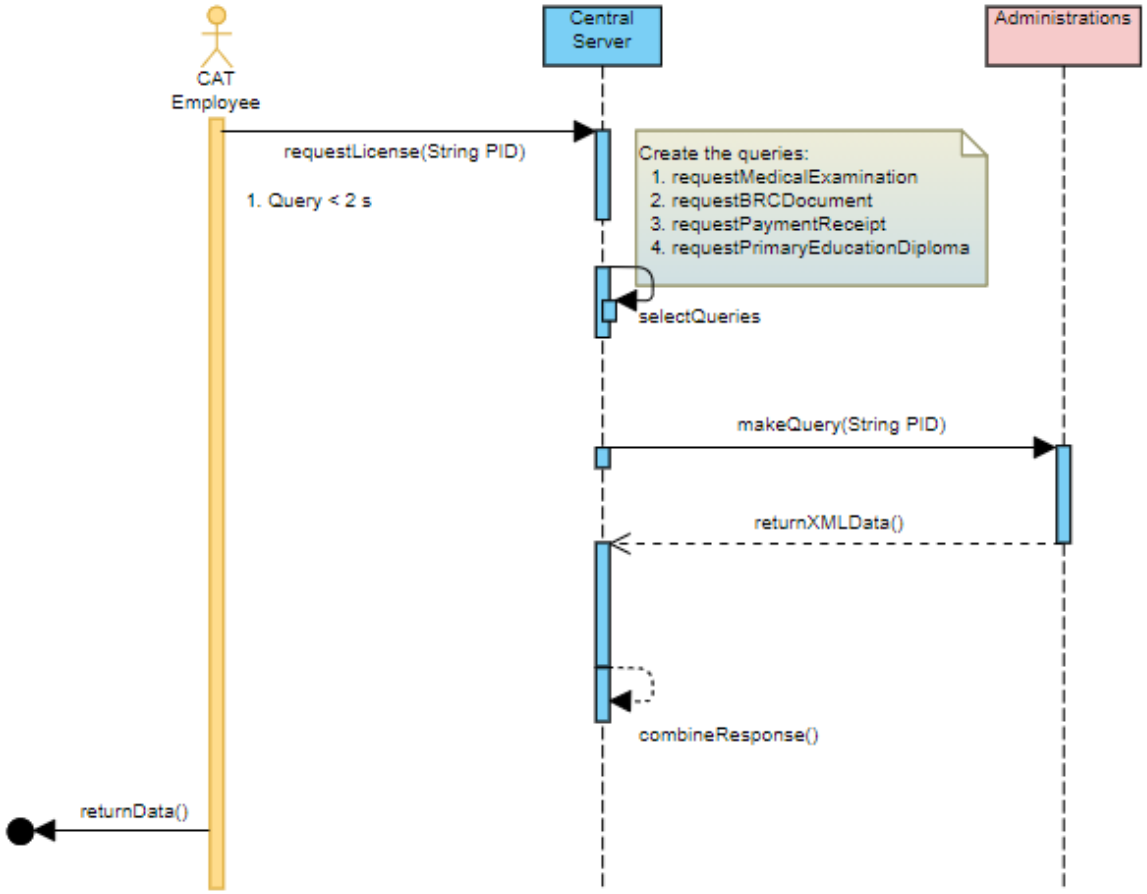
IV. Изглед на процесите

На този етап от документацията се разглеждат 2 основни sequence диаграми. Изгледът на процесите е отговорен за ясното представяне на системните процеси, начина, по който се извършва комуникацията, поведението на самата система по време на работа и други. Този изглед служи пред системните интегратори и има много предимства като:
- Простота на моделиране
- Лесно четене и интерпретация на изгледа
- Лесно поддържане

- Дава ясен „поглед“ на решенията на системата и др.

Basic sequence диаграма

Basic sequence представя „поглед“ отгоре на последователността и времето, по което се изпълняват заявките към системата и съответно техните отговори – синхронни и асинхронни.



Актьори:

- CAT employee

Времеви линии:

- Central server
- Administrations

Актьорът CAT employee е съответният работник в КАТ, който пуска заявка за получаване на шофьорска книжка, поискана от гражданин на място в КАТ. Тази заявка е синхронна, т.е. работникът трябва да изчака нейния отговор преди да премине към друга дейност. Заявката трябва да бъде обработена за по-малко от 2 секунди, защото в противен случай това би нарушило някои от нефункционалните изисквания, които системата трябва да поддържа. Нещо повече, забавяне повече от 2 секунди би накарало потребителите на системата да си мислят, че не са въвели правилен формат на данните или пък заявката им въобще не е постъпила за обработване.

След като заявката requestLicense(String PID) постъпи в централния сървър, той извиква selectQueries 4 пъти, за да формира съответно 4 различни заявки към БЧК, болнично заведение, бюро за плащания и училищна администрация. Тези заявки и съответните им отговори поддържат формата, описан в предходната точка – Логически изглед.

Времевата линия Administrations обединява четирите администрации, към които се пуска запитване за съществуването на конкретен документ. makeQuery(String PID) поддържа асинхронна връзка с администрациите. Това е така, защото централният сървър не трябва да чака отговор от конкретно звено, за да прати запитване към следващото. Веднъж получил всички отговори той вика сам в себе си combineResponse(), който оформя нова заявка спрямо получените, която да върне в готов вид на работника в КАТ.

Payment Service sequence диаграма

Payment service описва начина, по който се осъществява проверка за платени такси. В случай, че не се намери платежно, то тази диаграма описва последователността, през която може да се премине, за да се извърши плащане на момента.

Актьори:

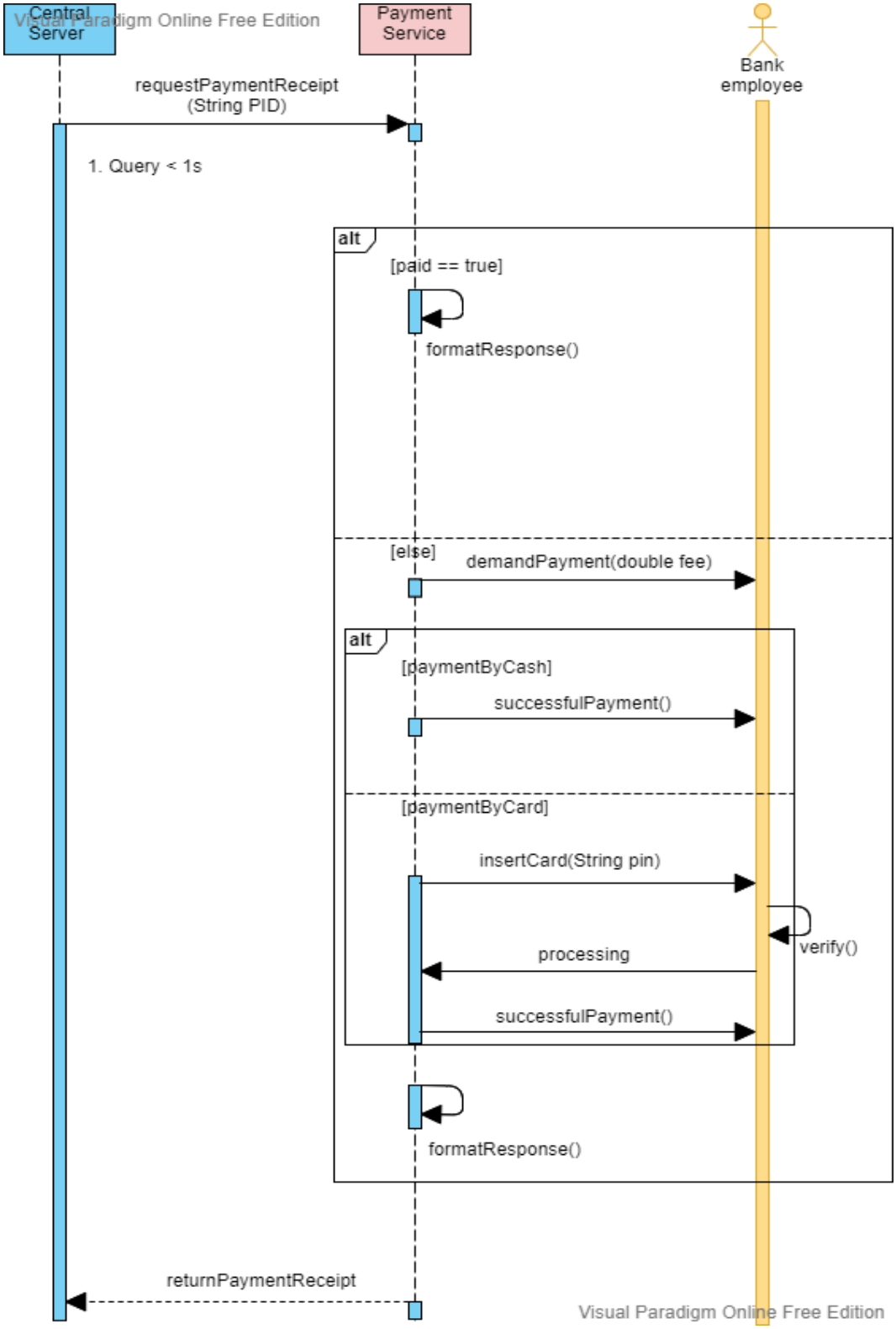
- Bank employee

Времеви линии:

- Central server
- Payment service

Както стана ясно от предходната диаграма, централният сървър формира различни заявки за всяка администрация. В случая requestPaymentReceipt(String PID) е една от тези заявки. Централният сървър я праща към Payment Service асинхронно, тъй като той продължава със своята работа по изпращане на запитвания към останалите звена без да изчаква отговора от Payment Service.

Веднъж постъпила във втората времева линия обработването на заявката започва като се проверява дали съществува бележка за платени такси, кореспондираща с подаденото ЕГН. Ако съществува такава директно се извиква методът formatResponse(), прескачат се останалите условия и проверки и директно се връща готовият отговор за успешно намиране на бележката към централния сървър чрез returnPaymentReceipt(). В случай, че подобно платежно не се открие и гражданина има възможност да осъществи плащането на момента, то се преминава към demandPayment(double fee), който се изпраща към актьора Bank employee. Съществуват два варианта за плащане. Единият е в кеш, след превеждането на въпросната сума по successfulPayment() следва одобрение от банковия служител, formatResponce() за обработване на отговора към сървъра и съответното му връщане чрез returnPaymentReceipt(). В случай че гражданинът иска да заплати чрез карта, то той я използва на терминал като insertCard(String PIN) се свързва с банковия служител и проверява пина. Банковия служител верифицира плащането и чрез processing() и successfulPayment() осведомява Payment Service за обработката на преведените пари. Вика се formatResponce() за обработване на отговора към сървъра и съответното му връщане чрез returnPaymentReceipt(). Всички съобщения в else блока на първия alt са синхронни.


Central Server
Visual Paradigm Online Free Edition
Payment Service
Bank employee
requestPaymentReceipt
(String PID)
1. Query < 1s
alt
[paid == true]
formatResponse()
[else]
demandPayment(double fee)
alt
[paymentByCash]
successfulPayment()
[paymentByCard]
insertCard(String pin)
verify()
processing
successfulPayment()
formatResponse()
returnPaymentReceipt
Visual Paradigm Online Free Edition

V. Изглед на внедряването

Изгледът на внедряването осигурява основа за разбиране на физическото разпределение на системата в набор от възли за обработка в работния поток. Той илюстрира разпределението на обработката в набор от възли в системата, включително физическото разпределение на процеси и нишки. Усъвършенства се по време на всяка итерация от процеса на документиране на системата.

Разпределението на компонентите на системата е представено чрез Deployment диаграма. Тя е тип UML диаграма, която показва архитектурата на изпълнение на системата, включително възли като хардуерни или софтуерни среди за изпълнение, както и свързващия ги междинен софтуер. Диаграмите за внедряване обикновено се използват за визуализиране на физическия хардуер и софтуер на системата.

Актьори:

- CAT employee

Възли:

- CAT PC
- Local Network
- Central Server
- WAN
- BRC
- Payment Service
- Hospital
- School Administration
- Bank

Компоненти:

- One stop shop application

Артефакти:

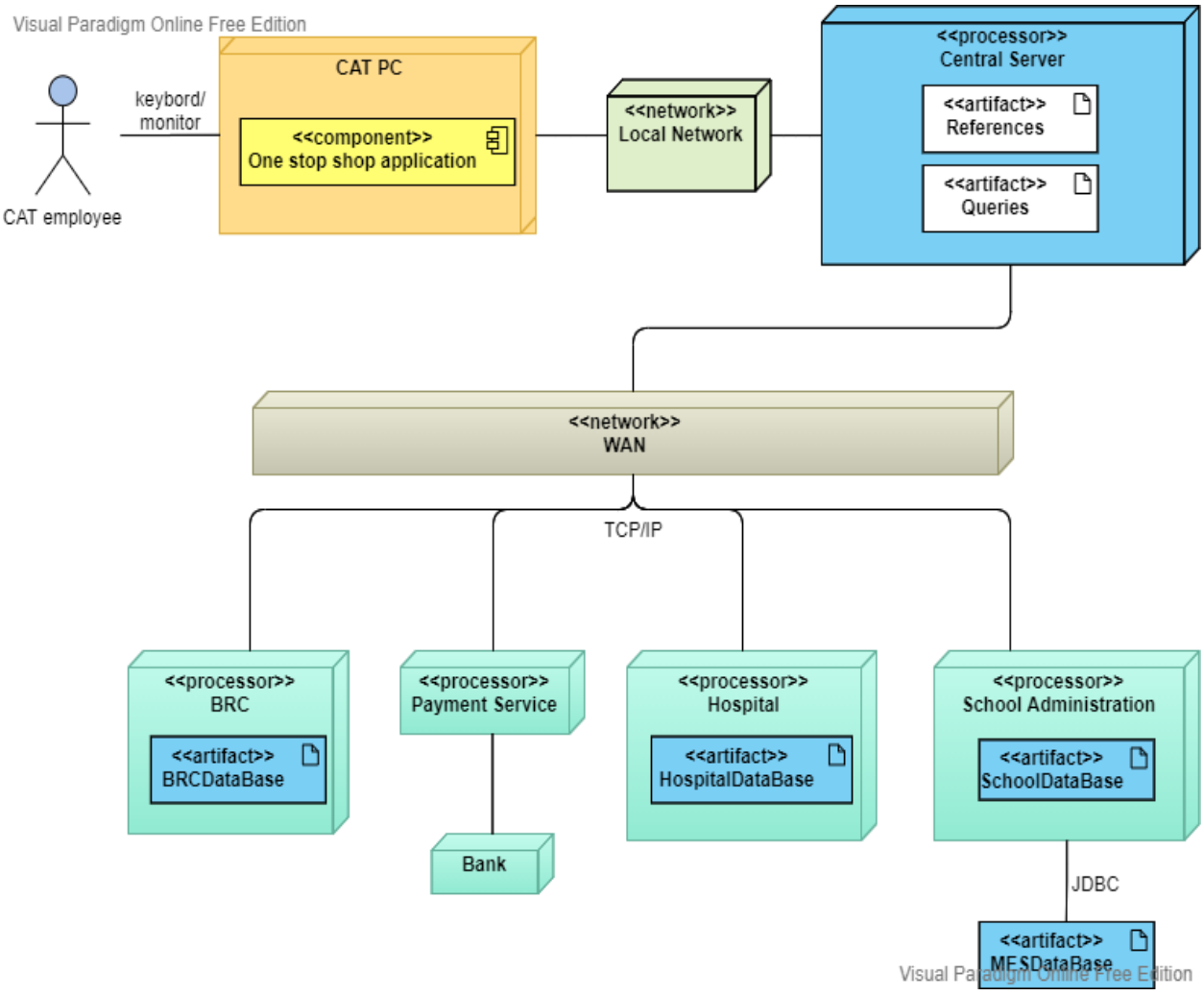
- References
- Queries
- BRCDataBase
- HospitalDataBase
- MESDataBase
- SchoolDataBase

Връзки:

- keyboard/monitor
- TCP/IP
- JDBC

За да работи със системата CAT employee използва съответно keyboard/monitor връзката, за да осъществи поисканата заявка. CAT PC разполага с компонент, наречен One stop shop application, което именно е системата за получаване на лиценз за управление на МПС. Използва се Local network, тъй като работната станция на служителя и Central Server се намират в сградата на КАТ. LAN е вид малка компютърна мрежа, обслужваща компютри и други устройства (напр. мрежови принтери или скенери), свързани помежду си. За разлика от големите (международни) WAN мрежи, локалната мрежа се разполага обикновено в една сграда, в случая – сградата на КАТ. От своя страна Central Server има два артефакта – References, където се държи информацията за обработените заявки и върнатите отговори от тях и Queries за формата на отговор и заявка към администрациите. Връзката с BRC,

Payment Service, Hospital и School Administration се осъществява с WAN. WAN мрежите се използват за да свързват много LAN (в случая всяка една отделна администрация сама по себе си е LAN) и други видове мрежи, така че потребителите и устройствата разположени в една от тези мрежи да могат да достъпват, да обменят информация и да използват ресурси от отдалечените мрежи. Връзката е TCP/IP. В модела TCP/IP информацията се пренася под формата на пакети.



VI. Заключение

Всяка система трябва да спазва някакви критерии, които да гарантират нейното качество. В зависимост конкретното предназначение и цел някои изисквания са от по-голяма важност, а други не. Следователно е невъзможно да бъдат спазени всички атрибути за качество. Именно поради тази причина съществуват така наречените нефункционални изисквания, които наблягат на най-важните „рамки“ относно конкретната система. One stop shop системата е съсредоточена около следните атрибути: **Производителност, Ефективност, Скалируемост, Сигурност, Възможност за разширяемост, Възможност за поддръжка**.

Производителността на системата се гарантира от самата реализация и обединение на различните компоненти. Client server шаблонът е подходящ избор, тъй като системата не би била подложена на твърде голямо едновременно натоварване предвид начина, по който ще се използва. Една много бърза и приблизителна сметка е следната: в България има 28 областни градове (в това число са София-град и София-област), във всеки град има по една служба на КАТ, ако предположим, че средно всяка служба разполага с 3 гишета за получаване на лиценз за управление на МПС (имайки предвид, че в по-малките областни градове дори е само едно гише, но за сметка на по-големите), то в най-лошия случай системата би била подложена на 84 едновременни заявки, като осигуряването на един стабилен и мощен сървър, който да обработва тези заявки, не е непосилно. Нашият софтуер цели преминаването към един по-модерен свят, затова осигуряването на необходимите ресурси няма да бъде проблем. Ако системата беше предназначена за ползване директно от гражданите, то тогава със сигурност щеше да съществува проблем от претоварване, но в случая такъв няма.

Ефективността на системата има за цел да осигури начини за по-ефикасна работа на софтуера. Способността да изпълнява задачите си с възможно най-адекватното използване на всички необходими заявки се осъществява като се избягва препокриването на информация с цел пестене на време и ресурси. Софтуерът има за цел да улесни гражданите от гледна точка на време, което те губят, за да им се предостави свидетелство за управление на МПС, затова целта е да се спести тяхното време дори от излишно чакане пред един служител на КАТ. Именно заради това заявките се изпращат, обработват и връщат за отрицателно време, благодарение на премахването на излишни ресурси.

Скалируемостта на системата ни позволява добавяне на нови сървъри или клиенти, което е очакван резултат, имайки предвид, че се добавят нови администрации, всяка, от които със собствена база данни, като следователно трябва да имат достъп до централния сървър. Клиентите, които ще се възползват от тази услуга също ще нараства, защото всяка година хората, които ще трябва да изпълнят всичките условия за предоставяне на свидетелство за управление на МПС, ще бъдат записвани съответно във всяка база данни на отделните администрации. Ресурсите стават по-големи всяка година. Проблем няма да се забележи от нито една страна, която използва централния сървър, благодарение на неговата централизация и лесна преконфигурация на отделните модули.

Сигурност, тя е от изключителна важност за софтуерния продукт, който пази предоставените лични данни на отделните граждани само за определен период от време, след това всичко бива заличено, за да не може външно лице да разполага с лични данни за някого друг и да се получи изтичане на информация. Тъй като данните са централизирани, тази система е по-надеждна и служи за допълнителна сигурност на данните. Единственият достъп, който се осъществява към сървъра до личните данни е от страна на КАТ, което намалява видимостта им и увеличава сигурността.

Пред системата със сигурност има поле за изява относно бъдещето й разширяване. Например може да се реши в даден момент, че КАТ иска да добави някаква функционалност, свързана с по-бързо регистриране на автомобил или пък след определени законови промени няма нужда от някой от документите за получаване на лиценз за управление на МПС, или пък-точно обратното – трябват справки с още администрации. Списъкът с възможности е дълъг, но в случая важното е, че системата позволява много лесно интегрирането на подобни промени. Благодарение на шаблона въз основа, на който тя е проектирана, добавянето на някакъв модул или пък премахването на вече съществуващ

е лесно, тъй като подобна промяна би довела само до „настройки“ в централния сървър, тъй като останалите администрации не споделят никакви данни или операции по между си.

Всяка система в даден момент „крашва“, получават се непредвидени ситуации, има грешки и т.н. В тези случаи трябва да се реагира бързо и адекватно, но освен това и самата система трябва да бъде разработена по такъв начин, че да предразполага за лесната си поддръжка. В случая на One stop shop подобни моменти може да са много опасни, но това по никакъв начин не трябва да нарушава в голяма степен нормалния темп на работа с нея. Поддръжката включва и някои от горе споменатите нефункционални изисквания като надеждност, ефективност и други, които вече са описани по какъв начин се осъществяват. Нещо повече, продуктът е съвместим, тъй като е до голяма степен независим от ОС. Позволява лесно разширяване и намиране на възникнал проблем, благодарение на ключовата роля, която играе централния сървър и данните, с които той разполага.