1. Тема:

SimpleNet

(Проект 622)

2. Автори:

Калин Петков Кръстев

0045026563

Ж.К.КРАСНО СЕЛО КВ.БОРОВО 232 вх.Б ет.5 ап.35

0888744980

kalin.p.krustev@gmail.com

Първа Частна Математическа Гимназия

11^{а} клас

3. Ръководител:

Диана Петрова

dianapetrova@abv.bg

Учител

4. Резюме

4.1 Цели

Проектът е предназначен за разработчици, а не за потребители. Написан е на *Java* и може да бъде използван само от програми написани на същия език. Целта му е да улесни неопитни разработчици, които не са напълно запознати с *java.net* пакета. Прилкожното приложение предоставя методи за: осъществяване на клиент-сървър връзка, изпращане на съобщения от клиента до сървъра и от сървъра до всеки един клент както и за прекъсването на тази връзка. В нея се съдържат класове, които могат да бъдат използвани като основа за двете страни на клиент-сървър мрежовата архитектура. Проектът предоствя опцията да криптира комуникацията между устройствата. Преносът на данни и криптирането са напълно интергрирани в приложението и не представляват пречка за неопитният разработчик. За него остава само да реши какво да изпрща и как да третира получените съобщения. Съществуващи решения на този проблем не се намират рядко, но не предоставят опция за криптиране или ако предоставят не е написана в лесен за разбиране и използване вариант.

4.2. Основни етапи в реализирането на проекта

4.2.1 Реализирането на библиотеката

Първият етап от развитието беше да бъде написана основната фукционалност – класът *Client,* класът *Server* и класът *SerializableMessage*. Около тях биват написани всички други касове в главния пакет. Първият горе споменат клас представлява клиентът в клиент-сървър архитектурата, втория – сървъра, а третия представлява общият вид на съобщенията, които ще бъдят изпращани. Вторият етап представлява адаптирането на вече написанети класове да изпозлват Интернет протокола. Тази адаптация бива отделена в подпакет на име *internet,* който бива разделян на *udp* и *tcp*. В тези пакети се съдържат неабстрактни класове, който разширяват вече съществуващите основни класове в главния пакет – *Client* и *Server*. Това са класовете предназначени за използване от разработчиците, който искат да работят с Интернет протокола. Всеки друг протокол може да бъде написан използвайки двата основни класове. Следващият етап, след като основната фукнционалност беше готова, беше да се подсигури коминикацията чрез криптиране и така се роди пакетът *encryption*.

Единственият в него клас е *EncryptedMessage*, който разширява основният клас *SerializeableMessage*, като предоставя възможността информацията в него да бъде криптирана. Имплементираните видовете криптиране са два – един симетричен и един несиметричен. Симетричният е отделен в пакет *aes*, а асиметричният в *rsa*.

4.2.1 Реализирането на сървърното приложение

След като библиотеката беше написана, остана да се напише графичен интерфейс. Целта му е да улесни още повече разработчиците използващи библиотеката. Графичният интерфейс бива отделен в друга апликация наречена „сървърното приложение“. Първият етап в създавеното и беше да се имплементира вече написаната библиотека и имплементацията бива отделена в пакета *networking*, а графичният дизайн в пакета *ui*. Елементите, които съдържа са: две текстови полета – конзола и поле за вход на данни, списък с всички свързани клиенти и меню, разположено в северната част на панела. Следващата стъпка беше да се напише фунционалността на полето за вход на данни. Целта на това поле е на дава наставления на програта и да я командва докато работи. След въвеждането на текста в полето за вход на данни се натиска клавиетурният клавиш *Enter* или графичният бутон *Execute* и командата бива екзекутирана. Управляването на командите бива отделено в пакет *command*. Първият клас написан в този пакет е *Command* и той представлява абстрактен клас, който е основата на всяка една команда. Следващият написан клас е *CommandMap,* който отговаря да управлява всички команди. За да може една команда да бъде вкарана в ефект тя трябва да бъде регистрирана в инстанция на горе споменатия клас. Следващият етап представлява написването на всяка една комнанда, нужна за основното функциониране на сървърното приложение – налагане на забрана на осъществяването на връзка с определен интернет адрес, отмяна на тази забрана и настройки на самото приложение. Тези команди биват отделенеи в подпакет *defaults*. След като цялостта на сървърното приложение беше написана остана само да се напише интеракцията на разработчика със сървърното приложение. Пътят по който приложението пое за осъществяването на тази интеракция беше имплементирането на плъгин система. Синоним на плъгин е думата „приставка“ и по дефиниции от *Wikipedia.org* това представлява “софтуерен компонент, който се инсталира в допълнение към съществуващо софтуерно приложение (компютърна програма), за да предостави на потребителя допълнителна функционалност, която не е част от базовото

приложение.” В конкретният случай тази приставка е кодът написан от разработчикът. За да може да бъде имплементиран в сървъра трябва да изпълнява няколко условия:

1. Да бъде jar файл.
2. Да съдържа клас, който да имплементира всички методи на написания в сървърното приложение файл *Plugin.java,* наречен накратко главен клас.
3. Да съдържа текстов файл, спазващ определен формат, който да навигира четеца на сървърното приложение къде се намира главният клас на приставката, какво е името ѝ, и каква е версията ѝ.

Всеки файл, който изпълнява тези условия бива наречен плъгин. Начинът на зареждане на този файл в сървърното приложение бива следния. Със стартирането на сървъра се търси директория на име *plugins* и ако не бива намерена се създава. Тази директория е предназначена да съхранява плъгините. Ако при стартирането биват открити плъгини те се зареждат. В случай, когато файл бива добавен към вече стартирало приложение неговото зареждане трябва да бъде зададено с команда от командното поле в графичния интерфейс. Имплементацията на тази фунционалност е отделена в пакета *plugin*. Там се намира основата на всички плъгини - абстрактният клас *Plugin*. Това е класът, който прави връзката между разработчикът и приложението. В него има методи, с които уведомява разработчикът когато:

- Плъгинът бива активиран
- Плъгинът бива деактивиран
- Клиент се присъдини към сървъра
- Сървърът получи съобщение от клиент
- Клиент прекрати връзката си със сървъра

Останалите класове с пакета са написани да помогнат на плъгинът да се интегрира.

4.3 Ниво на сложност на проекта - основни проблеми при реализация

4.3.1 Поддържането на връзка с повече от един клиент

4.3.2 Приемане и изпращане на съобщения. Сериализация и десериализация

4.3.3 Подсигуряване на съобщенията. Криптиране и декриптиране

4.4 Логическо и функционално описание на решението

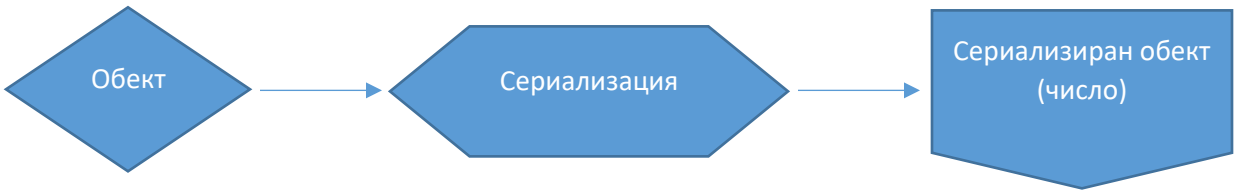
4.4.1 Поддържането на връзка с повече от един клиент

Когато сървърното приложение осъществи връзка с клиент, то постоянно чете дали има нови данни подадени от този клиент. Тази проверка ангажира цялата апликация и прекратява всякакви други операции докато работи. За да се отсрани този проблем апликацията бита разделена на *Thread*-ове. *Thread* буквално преведено означава „нишка“ и идеята на тази нишка е да изпълнява командите, които и биват зададени. По подразбиране всяка апликация има по една нишка, която изпълянва зададените инструкции. Нишките нямат заделена памет,а споделят ресурсите на процеса, който обитават. Когато сървърът приеме нов клиент, той създава нова нишка, която е предназначена само за клиентът и която бива терминирана, когато клиентът прекъсне връзката. С поддържането на отделна нишка за всеки клиент, апликацията работи без забавяния.

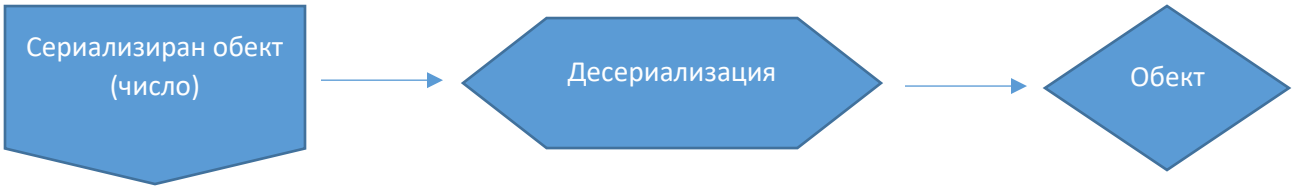
4.4.2 Приемане и изпращане на съобщения. Сериализация и десериализация

За да може едно съобщение да бъде изпратено, то трябва да бъде приведено в един общ вид. Този общ вид представлява резултата от сериализирането на това съобщение. По дефиниция от *Wikipedia.org* сериализацитя представлява „процес на преобразуване на структури от данни или обекти до поток от байтове запазвайки състоянието на техните полета и свойства.“. Използването на тази техника позволява да бъдат изпращани цели класове. Когато обектът бива сериализиран, той бива изпращан. Обратното действие на сериализацията е десериализацията и чрез прилагането на този процес се пресъздава идентично копие на оригиналното съобщение.

Сериализация:

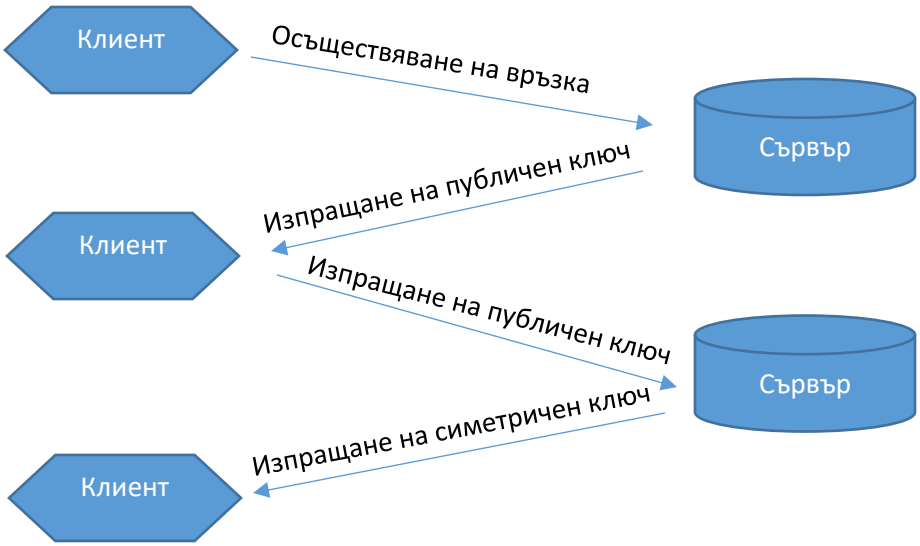


Десериализация:



4.4.3 Подсигуряване на съобщенията. Криптиране и декриптиране

За да може едно съобщение да бъде криптирано от библиотеката то трябва да бъде под формата на число. Това условие се изпълнява от сериализацията на това съобщение, тъй като винаги когато едно съобщение бива сериализирано то бива преобразувано в число. Библиотеката поддържа два вида криптиране: симетрично под формата на AES и асиметрично – RSA. Подсигуряването на съобщенията става като и клинетът и сървърът разменят публичните си асиметрични ключове и след това като сървърът изпрати симетричен ключ. Използването на този ключ позволява на двете страни да криптират съобщения бързо и ефикасно.



4.5 Реализация

За създаването на сървъра беше необходими два ресурса: *Java Development Kit 9 и IntelliJ IDEA*. Първият горе споменат ресурс е необходим за да се използва езикът за програмиране *Java*, а вторият е софтуерът, който представлява средата на писане. За демонстрация на

функционалност на приложението се използва софтуерът *WireShark*. За написването на документацията и изготвяне на презентацията се изполва *MS Office*.

4.6. Описание на приложението

Приложението се стартира като се отвори файлът *SimpleNet.jar*. За да може да фунционира правилно приложението трябва да има правомощия да чете и да пише в директорията, в която се намира. Приложението няма опция да се инсталира или деинсталира. Поддръжката не е нужда, дори някой от файловете да бъде изтрит, той ще бъде заменен от нов, генериран от приложението.

4.7. Заключение

Основният резултат е продукт, не напълно готов за ползване от други разработчици. Единственото приложение до момента е софтуер, който е написан за демонстрация като част от презентацията. Приложението още не е публикувано като завършен проект. Възможностите за развитие и усъвършенстване включват:

- Имплементация на интернет протоколът *UDP*, защото в момента разполага само с *TCP*
- Добавяне на още методи за криптиране, както и настройки на криптирането
- Добавяне на още функционалност на приставките(плъгините)
- Добавяне на режим на работа без графичен интерфейс