проект на тема:

Система за обработка и управление на клиентска бановка информация

колектив:
F74277 - Мартин Драганов
F76731 - Илиян Костов
F78397 - Николай Николов

част: мрежова комуникация

разработена от Илиян Костов (F76731)
https://github.com/iliyan-kostov/Bank-System

Проектът включва:

- Създаване на графичен потребителски интерфейс за клиентите на банката, генериращ заявки и интерпретиращ отговорите - разработен от Мартин Драганов (F74277)
- Осигуряване на мрежова фукнционалност, управление на сървъра и защита на преноса на данни - разработена от Илиян Костов (F76731)
- Предоставяне на връзка от сървъра към система за управление на база данни (MySQL) за съхранение и обработка на транзакциите - разработена от Николай Николов (F78397)

Разработената от студента част включва:

- Анализ и документиране на изискваната функционалност.
- Създаване на протокол за обмен на данни (съобщения) в рамките на системата.
- Осигуряване на мрежовата комуникация съгласно протокола.
- Автентикация и управление на връзките с клиенти от страна на сървъра.
- Защита на преноса на данни чрез използване на SSL - частни и публични ключове (сертификати).
- Графичен интерфейс за управление на връзките между сървъра и клиентите.

Съдържание:

Всички схеми са налични на адрес:
https://github.com/iliyan-kostov/Bank-System/documentation

Списък на изображенията:

1. Описание на задачата

Разработеното от колектива приложение представлява:

- система за електронно банкиране, поддържаща и управляваща данните за множество клиенти и техните банкови сметки
- е разработено на език за програмиране Java с актуалната версия на Java Development Kit (JDK v1.8)
- за поддържане на базата данни се използва MySQL
- за връзка между системата за управление на базата данни и приложението се използва JDBC driver

Индивидуалната задача на студента (F76731) се състои в:

- анализ и документиране на необходимата функционалност чрез UML диаграми
- създаването на протокол за обмен на данни (съобщения) в рамките на системата
- осигуряване на мрежовата комуникация съгласно протокола
- автентикация и управление на връзките с клиенти от страна на сървъра
- защита на преноса на данни чрез използване на SSL - частни и публични ключове (сертификати)
- създаване на графичен интерфейс за управление на връзките между сървъра и клиентите

2. Основни функционалности (анализ)

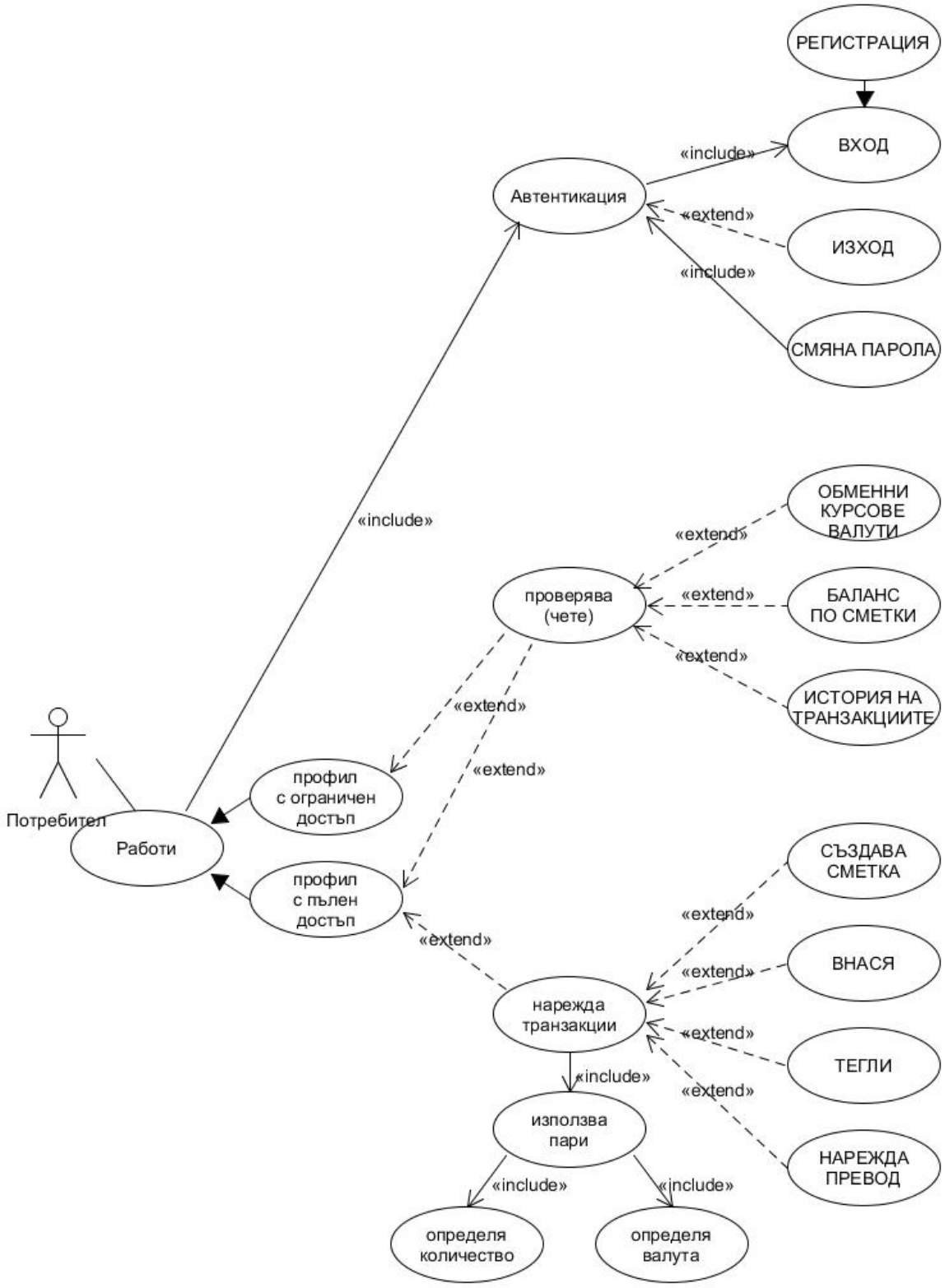
(виж Фиг. 1 Основни функционалности (UML Use Case Diagram))

Системата предлага на потребителя следните функционалности:

- да може да създава профил в системата
- да може да проверява за наличност и последни транзакции
- да тегли и да внася пари в зависимост от типа профил
- да нарежда банкови преводи
- да работи с различни видове валути
- да работи с различни видове банкови сметки

Системата осигурява:

- идентификация на потребителя
- защита на обменяната по мрежата информация (чрез SSL)
- удобен графичен потребителски интерфейс - както за клиентите, така и за управление на сървъра



Фиг. 1 Основни функционалности (UML Use Case Diagram)

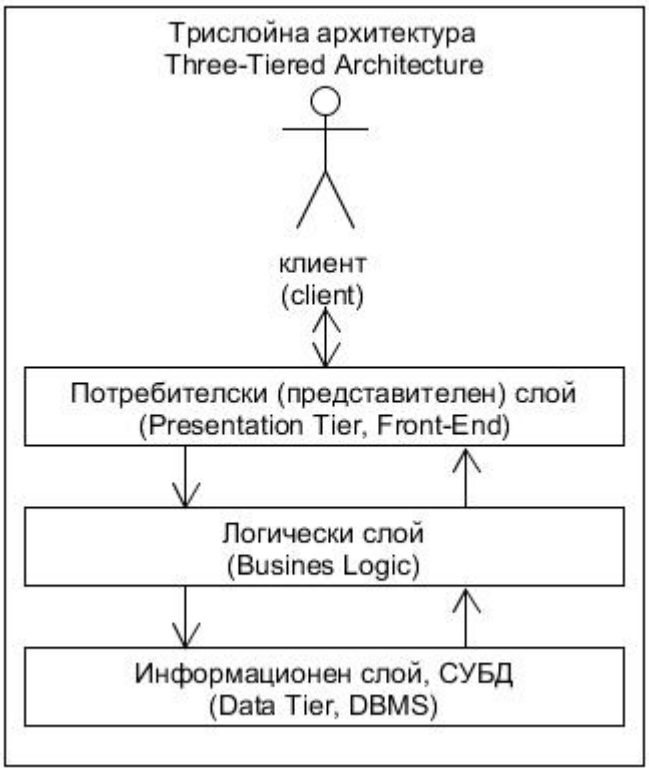
Съгласно функционалноста е разработен протоколът за обмен на данни (виж Мрежова комуникация и протокол за обмен на данни (съобщения)).

3. Архитектура на системата

Системата използва т.нар. "трислоен модел" ("трислойна архитектура", "three-tier architecture"). В системи, използващи трислойния модел, се различават три слоя (нива на организация):

- потребителски - предоставя достъп (подходящ интерфейс) на клиентите до системата;
- логически - управлява връзките с клиентите, интерпретира и обработва заявките, взаимодейства със СУБД, изпраща отговори;
- информационен - най-често представлява система за управление на база данни (СУБД). Не е предмет на текущата разработка - проектът използва MySQL СУБД и JDBC drvier за връзка.

Всеки слой взаимодейства само със съседните му (не е разрешен например пряк достъп на клиента до СУБД):



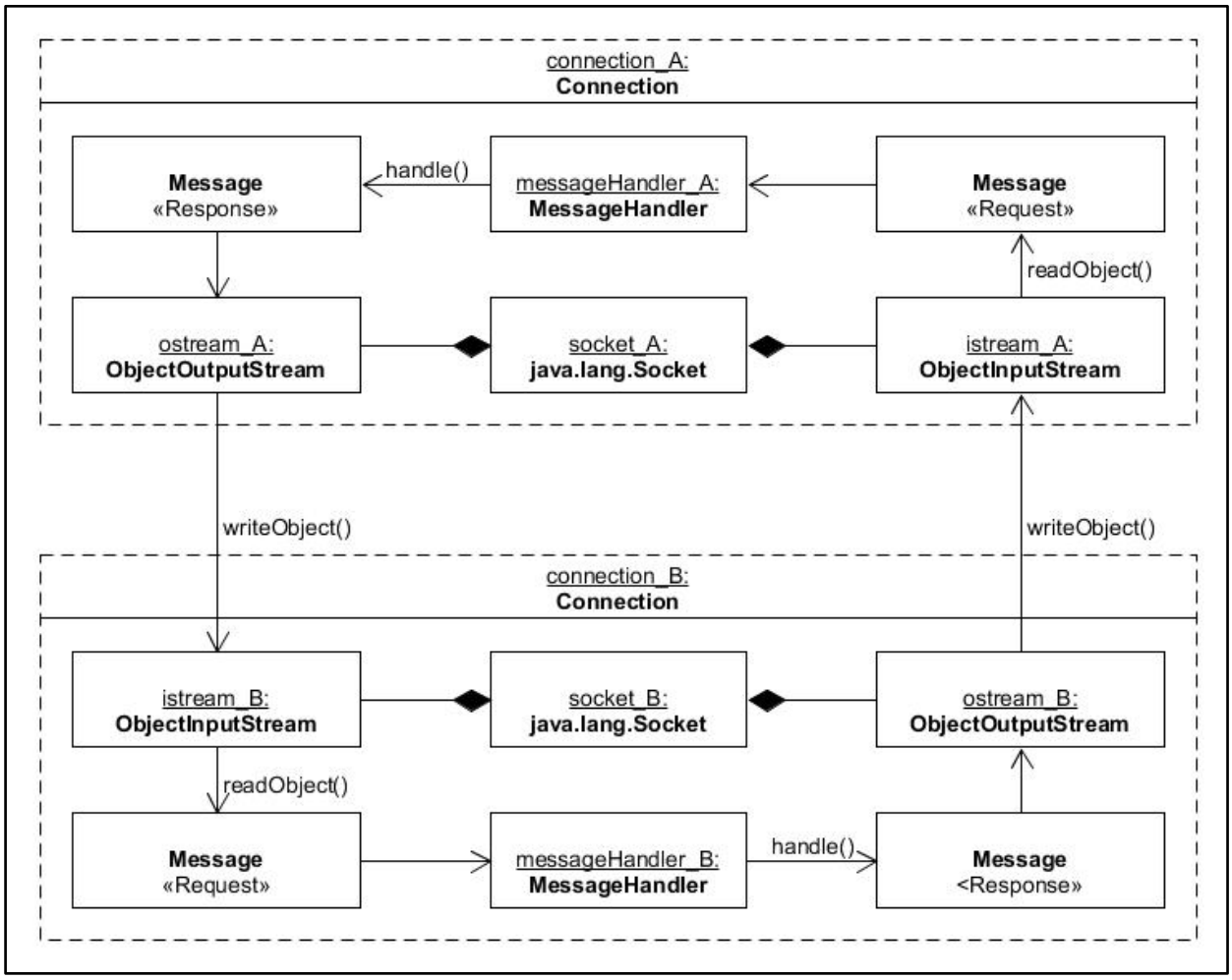
Фиг. 2 Трислоен модел

4. Структурна схема на комуникацията клиент-сървър

Сървърът осигурява възможност за свързване и обработка на заявки и връщане на отговори към множество клиенти едновременно. Това се осигурява чрез многонишкова организация на сървърната част.

Сървърът управлява множество връзки едновременно чрез обект от класа "ConnectionManager". Всяка входяща заявка за връзка на клиент се поема и обработва от самостоятелна нишка от класа "Serverside" (връзка от страна на сървъра). Съобщенията, които клиентите изпращата, се предават за обработка от обект, имплементиращ интерфейса "Messagehandler" ("обработчик на съобщения"), който връща резултат - съобщение, което при необходимост се изпраща обратно до клиента.

Клиентската част също разполага с обект, имплементиращ интерфейса "MessageHandler", който автоматично обработва съобщението според вида му, и връщз отговор - съобщение, което при необходимост се изпраща обратно до сървъра.



Фиг. 3 Обработка на съобщения

5. Мрежова комуникация и протокол за обмен на данни (съобщения)

Съгласно разработения протокол, клиентската и сървърната страна общуват чрез обмен на съобщения (производни на базовия абстрактен клас "Message" - съобщение). Класът "Message" имплементира интерфейса "java.io.Serializable", което позволява обекти от класа и неговите наследници да бъдат пренасяни по мрежата, използвайки потоци от обекти ("java.io.ObjectInputStream", " java.io.ObjectOutputStream"). Това позволява преноса на капсулирани данни.

(виж Фиг. 5 Комуникация клиент-сървър-СУБД)

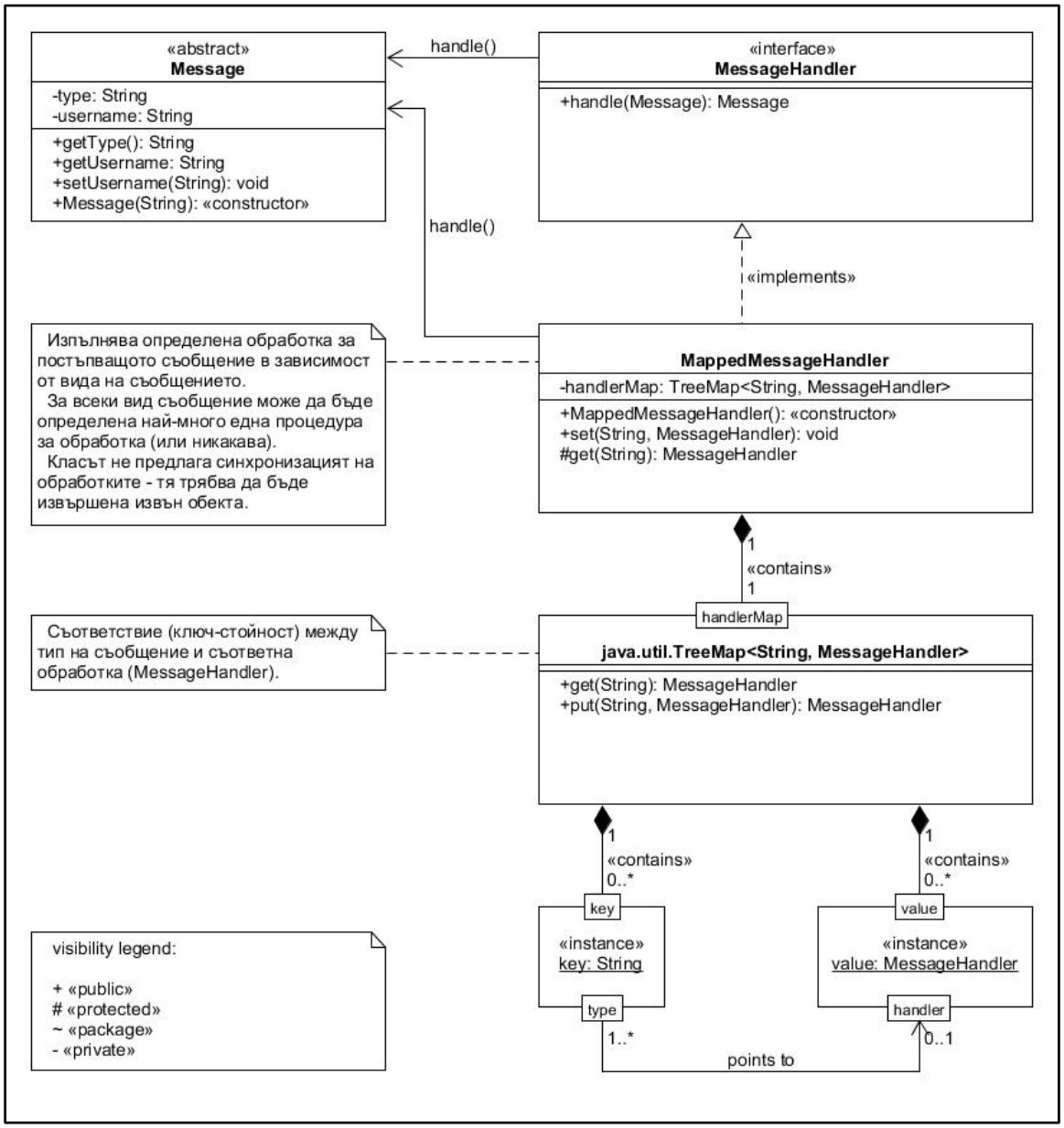
Получаваните през "java.io.ObjectInputStream" входящ поток сериализирани обекти са от базов клас "Object" и е необходимо те да бъдат явно преобразувани в действителния си клас. Разпознаването на типа обект се осигурява от полето "Message.type", налично в създадения базов клас. За всеки клас-наследник на "Message" е определена уникална стойност за това поле, в зависимост от функционалността. Чрез преобразуване на получения обект към класа "Message" и прочитане на стойността на полето е възможно сигурното преобразуване на обекта към реалния му тип (клас) и успешната последваща обработка.

За обработка на съобщенията е създаден интерфейса "MessageHandler", който декларира единствен метод "handle(Message)". Този метод връща обработва входящото съобщение и при необходимост връща резултат - изходящо съобщение (или стойност NULL, ако отговор не се изисква). Чрез създаването на класове, имплементиращи интерфейса "MessageHandler", за всеки специфичен вид съобщение (съответна функционалност) и тяхното инстанциране в обекти е възможно обработването на съобщения от разлини видове (класове). Този подход позволява разширение на функционалността на системата чрез използване на отделни класове и пакети, без необходимост от еднократно създаване на целия програмен продукт със статично определена функционалност.

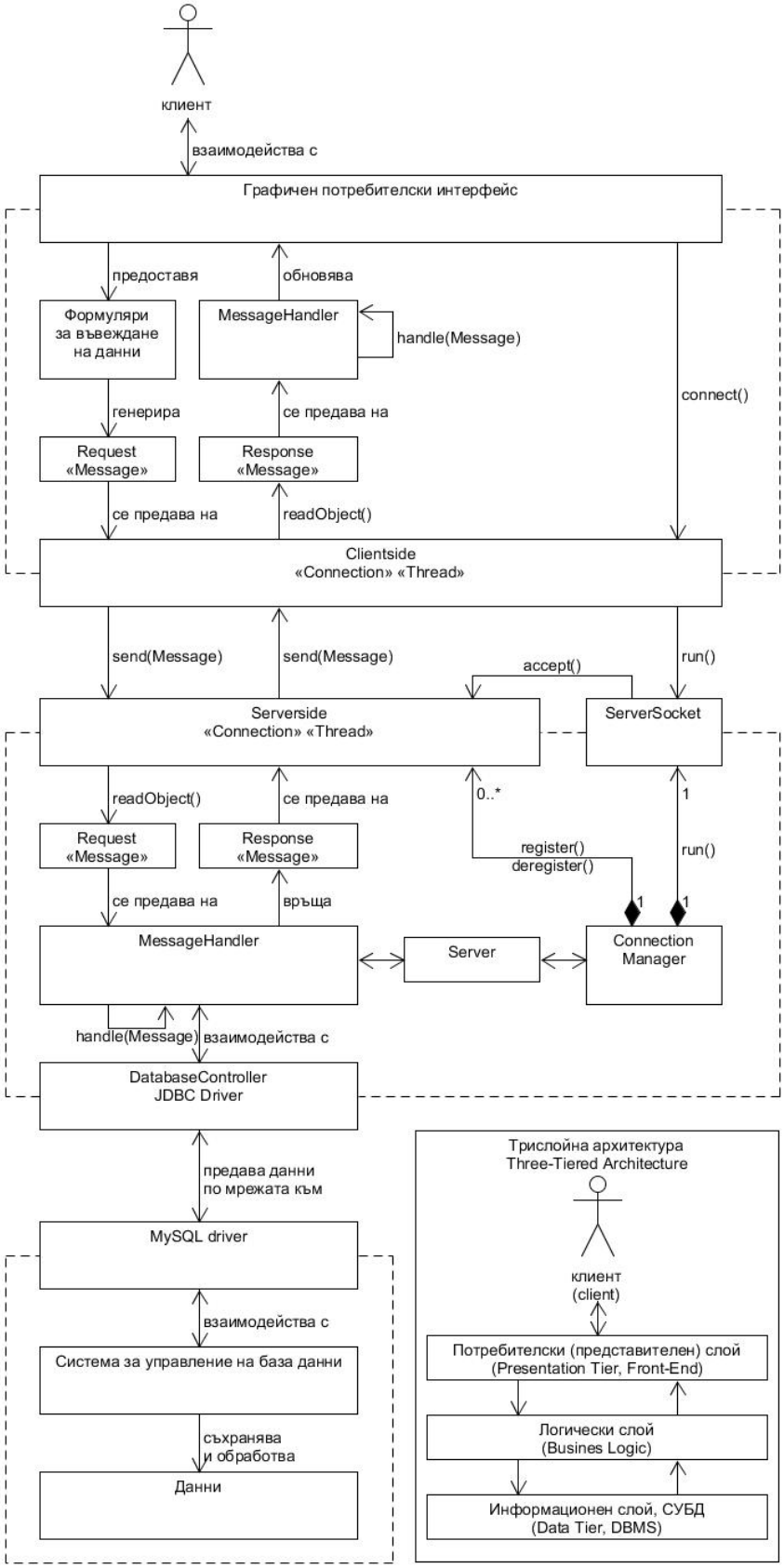
(виж Фиг. 4 Клас MappedMessageHandler и обработка на специфични заявки)

За обработката на специфични съобщения (различаващи се по стойността на полето "Message.type") се използват различни обработки. Това е осигурено от класа "MappedMessageHandler". Той представлява

набор от обекти - инстанции на класове, имплементиращи интерфейса "MessageHandler" за конкретен тип съобщение. И клиентскат, и сървърната страна използват такъв обект за обработката на входящи съобщения (заявка от клиента или отговор от сървъра). Чрез прочитане на типа съобщение се извиква уникално съпоставения обект за обработка. Класът използва "java.util.TreeMap" като вътрешна структура за съпоставянето. Разширението на функционалността се осъществява чрез добавянето на двойки "тип съобщение" - "обект за обработката му".

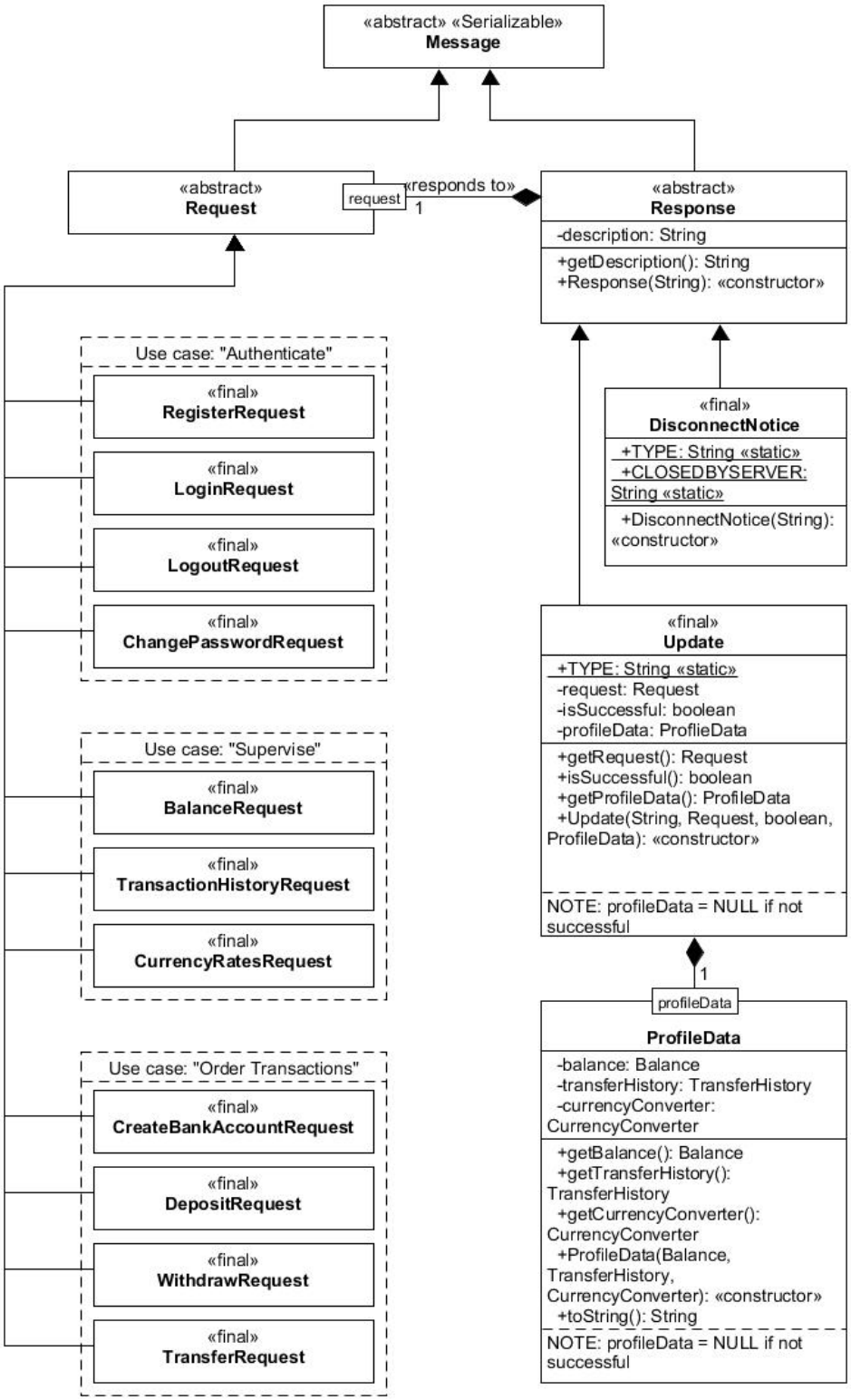


Фиг. 4 Клас MappedMessageHandler и обработка на специфични заявки

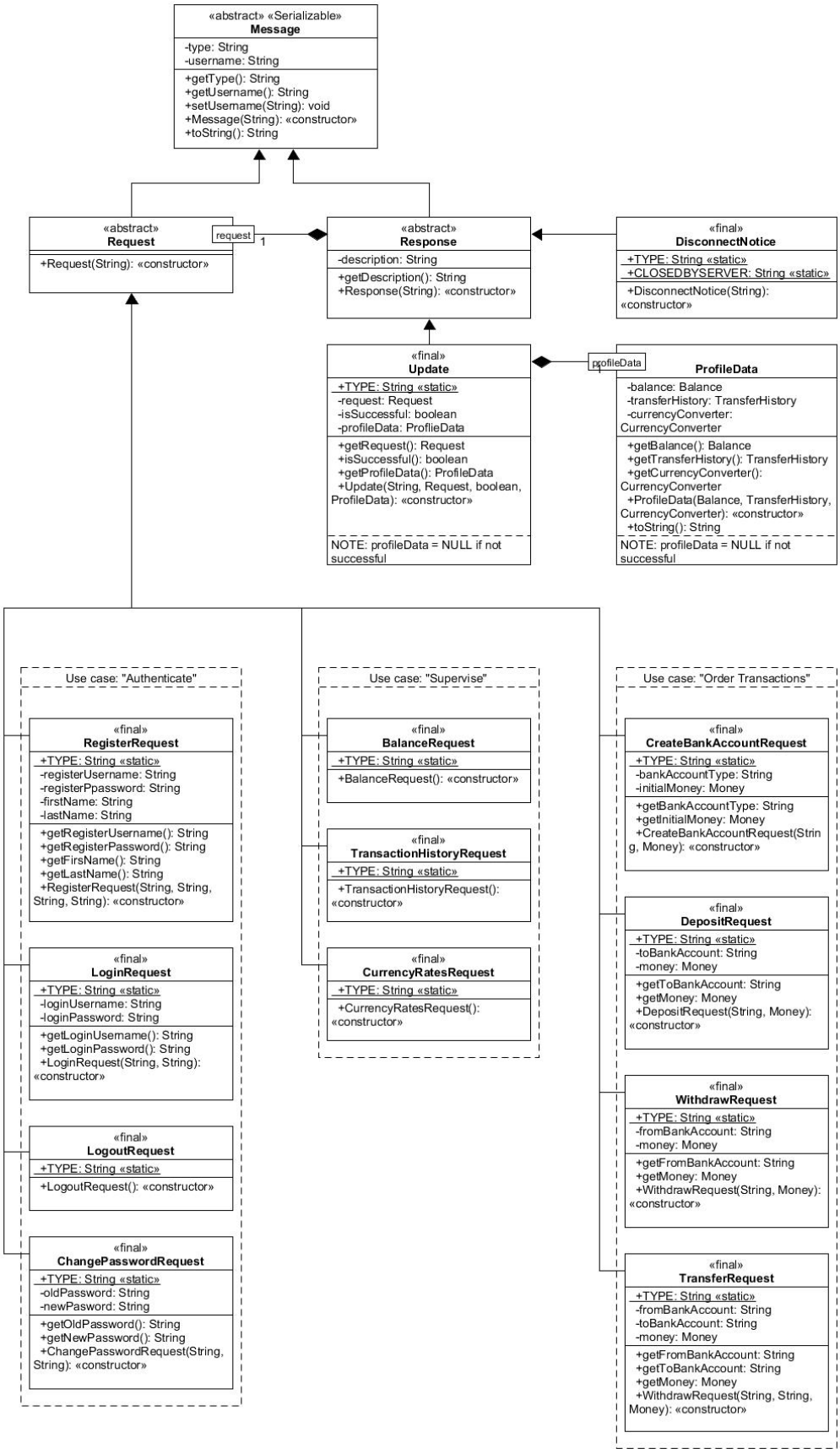


Фиг. 5 Комуникация клиент-сървър-СУБД

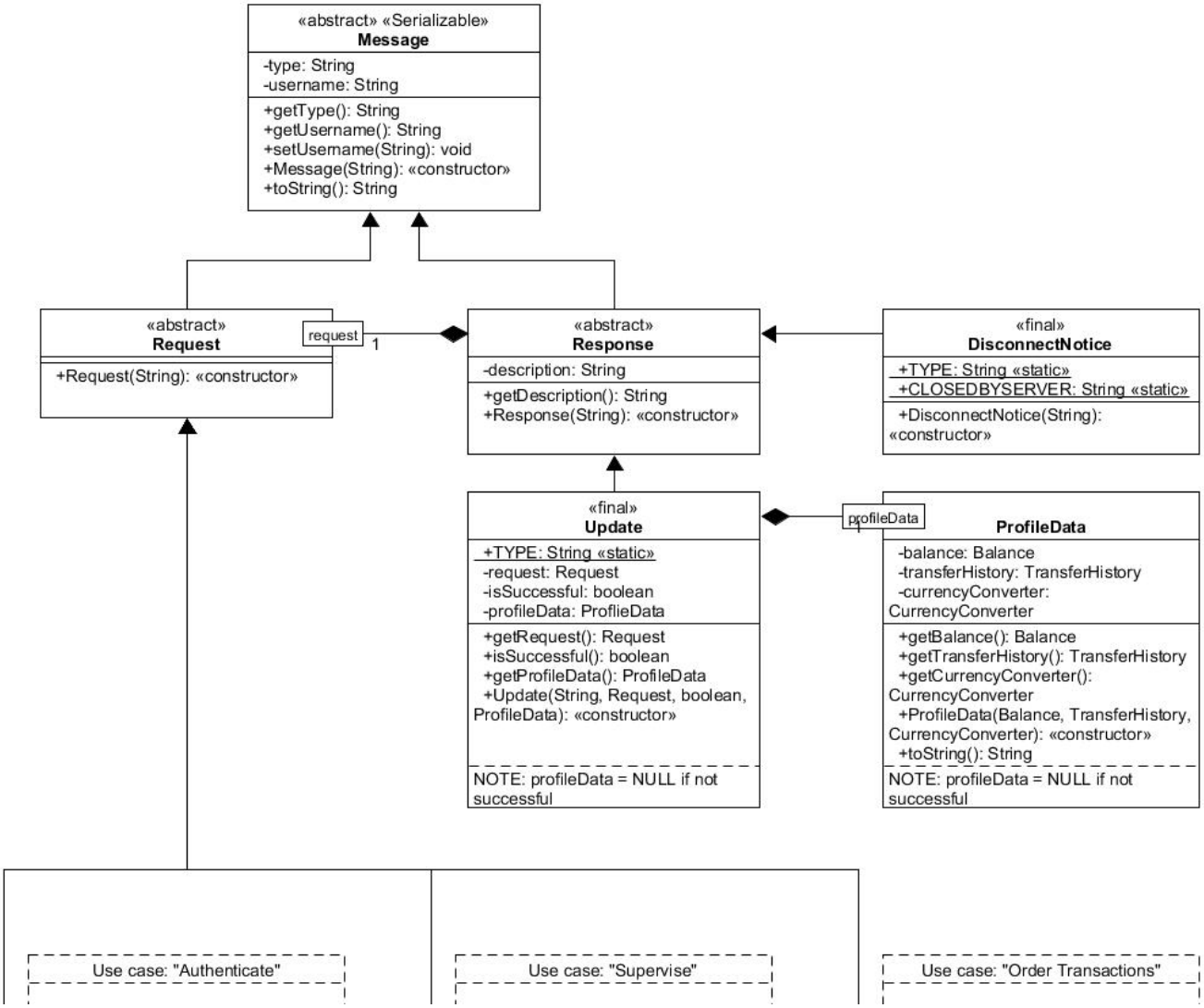
Съгласно анализа на функционалността на системата (виж Фиг. 1 Основни функционалности (UML Use Case Diagram)) са разработени специализирани класове-съобщения, капсулиращи необходимите данни.

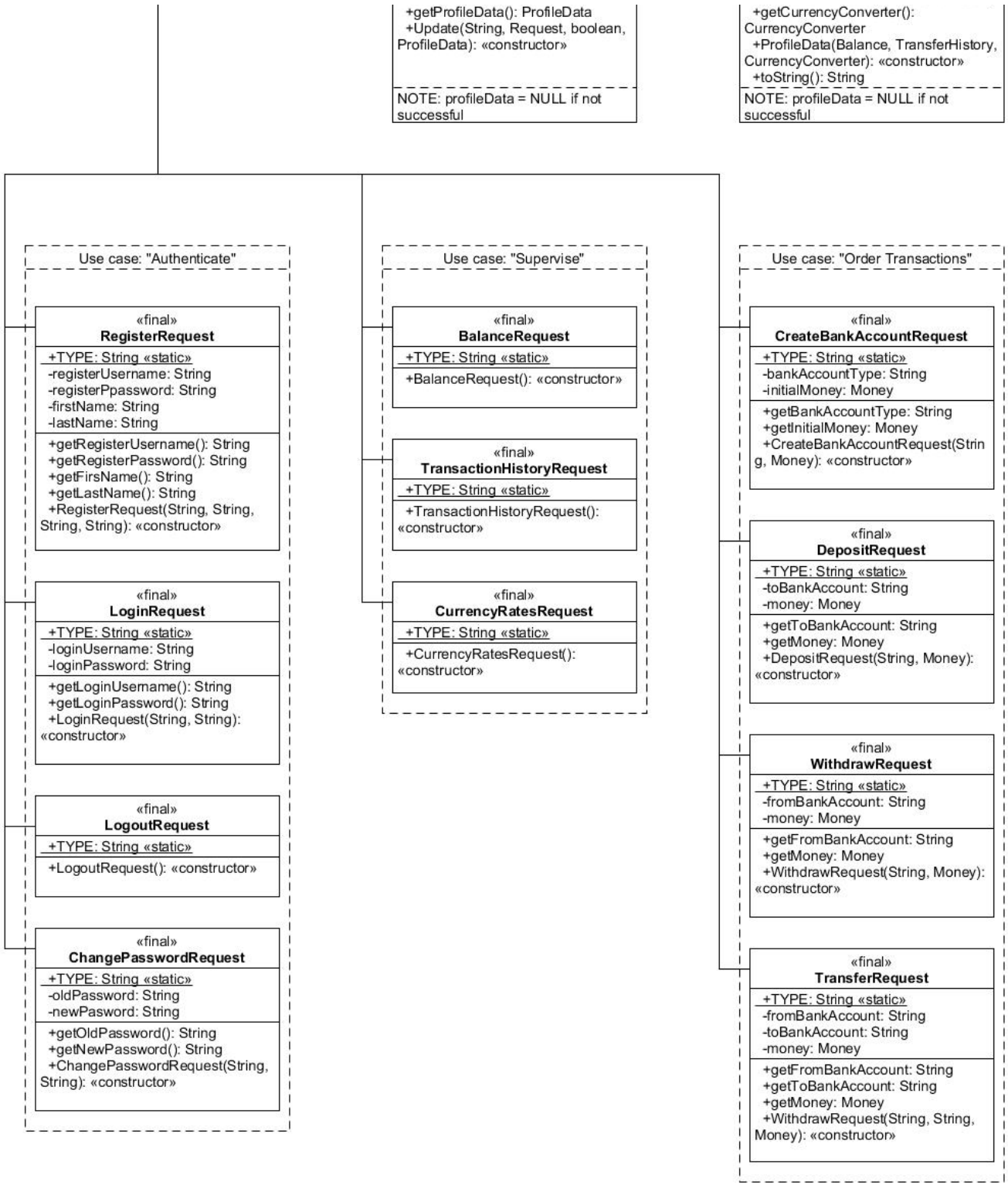


Фиг. 6 Йерархия на съобщенията (пакета networking.messages)



Фиг. 7 Съобщения (UML Class Diagram)


«abstract» «Serializable»
Message
-type: String
-username: String
+getType(): String
+getUsername(): String
+setUsername(String): void
+Message(String): «constructor»
+toString(): String
«abstract»
Request
+Request(String): «constructor»
request
1
«abstract»
Response
-description: String
+getDescription(): String
+Response(String): «constructor»
«final»
DisconnectNotice
+TYPE: String «static»
+CLOSEDBYSERVER: String «static»
+DisconnectNotice(String): «constructor»
«final»
Update
+TYPE: String «static»
-request: Request
-isSuccessful: boolean
-profileData: ProflieData
+getRequest(): Request
+isSuccessful(): boolean
+getProfileData(): ProfileData
+Update(String, Request, boolean, ProfileData): «constructor»
NOTE: profileData = NULL if not successful
profileData
1
ProfileData
-balance: Balance
-transferHistory: TransferHistory
-currencyConverter: CurrencyConverter
+getBalance(): Balance
+getTransferHistory(): TransferHistory
+getCurrencyConverter(): CurrencyConverter
+ProfileData(Balance, TransferHistory, CurrencyConverter): «constructor»
+toString(): String
NOTE: profileData = NULL if not successful
Use case: "Authenticate"
Use case: "Supervise"
Use case: "Order Transactions"


+getProfileData(): ProfileData
+Update(String, Request, boolean, ProfileData): «constructor»
NOTE: profileData = NULL if not successful
+getCurrencyConverter(): CurrencyConverter
+ProfileData(Balance, TransferHistory, CurrencyConverter): «constructor»
+toString(): String
NOTE: profileData = NULL if not successful
Use case: "Authenticate"
«final»
RegisterRequest
+TYPE: String «static»
-registerUsername: String
-registerPpassword: String
-firstName: String
-lastName: String
+getRegisterUsername(): String
+getRegisterPassword(): String
+getFirsName(): String
+getLastName(): String
+RegisterRequest(String, String, String, String): «constructor»
«final»
LoginRequest
+TYPE: String «static»
-loginUsername: String
-loginPassword: String
+getLoginUsername(): String
+getLoginPassword(): String
+LoginRequest(String, String): «constructor»
«final»
LogoutRequest
+TYPE: String «static»
+LogoutRequest(): «constructor»
«final»
ChangePasswordRequest
+TYPE: String «static»
-oldPassword: String
-newPasword: String
+getOldPassword(): String
+getNewPassword(): String
+ChangePasswordRequest(String, String): «constructor»
Use case: "Supervise"
«final»
BalanceRequest
+TYPE: String «static»
+BalanceRequest(): «constructor»
«final»
TransactionHistoryRequest
+TYPE: String «static»
+TransactionHistoryRequest(): «constructor»
«final»
CurrencyRatesRequest
+TYPE: String «static»
+CurrencyRatesRequest(): «constructor»
Use case: "Order Transactions"
«final»
CreateBankAccountRequest
+TYPE: String «static»
-bankAccountType: String
-initialMoney: Money
+getBankAccountType: String
+getInitialMoney: Money
+CreateBankAccountRequest(String, Money): «constructor»
«final»
DepositRequest
+TYPE: String «static»
-toBankAccount: String
-money: Money
+getToBankAccount: String
+getMoney: Money
+DepositRequest(String, Money): «constructor»
«final»
WithdrawRequest
+TYPE: String «static»
-fromBankAccount: String
-money: Money
+getFromBankAccount: String
+getMoney: Money
+WithdrawRequest(String, Money): «constructor»
«final»
TransferRequest
+TYPE: String «static»
-fromBankAccount: String
-toBankAccount: String
-money: Money
+getFromBankAccount: String
+getToBankAccount: String
+getMoney: Money
+WithdrawRequest(String, String, Money): «constructor»

6. Автентикация и управление на връзките с клиенти от страна на сървъра

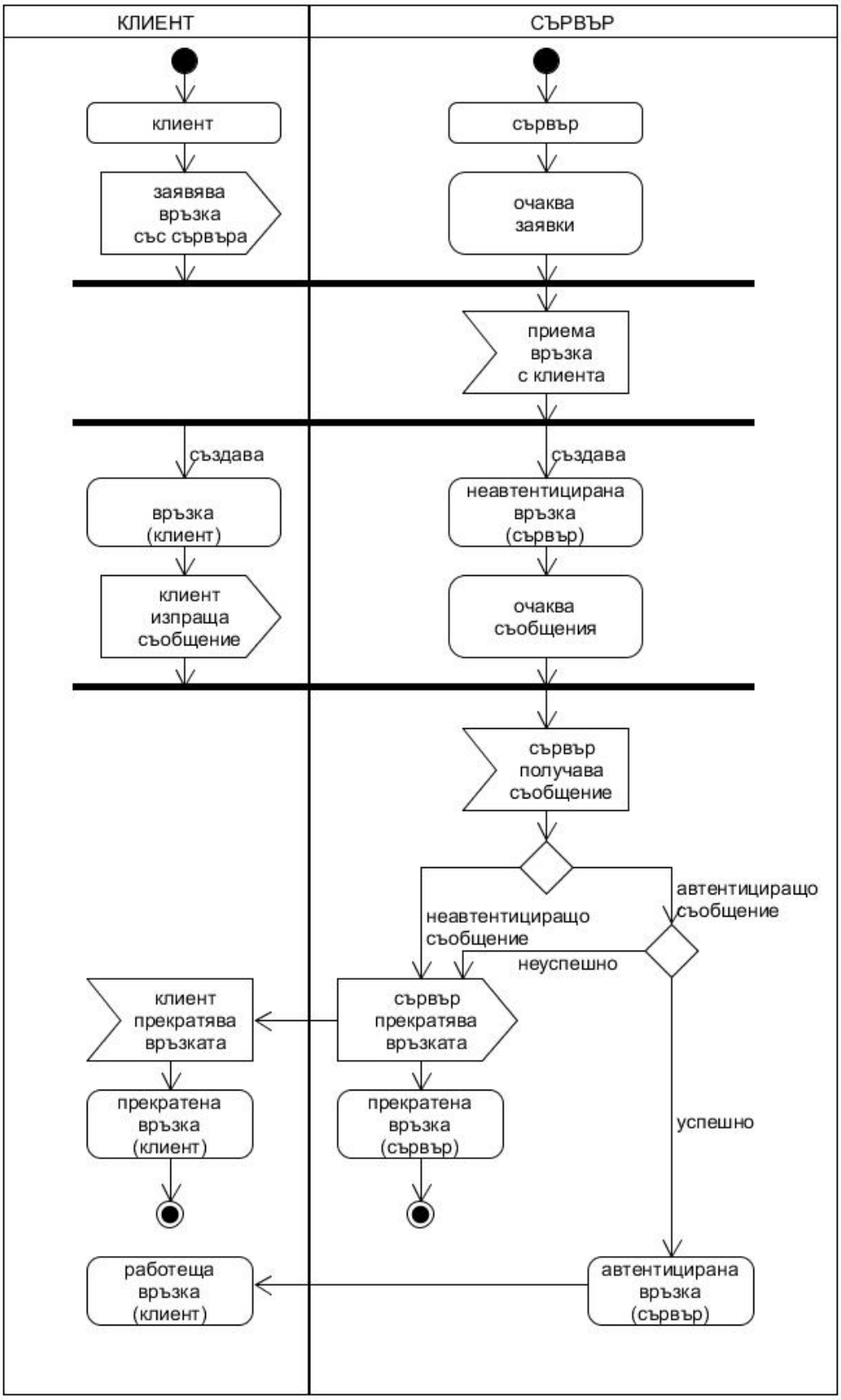
(виж Фиг. 8 Автентикация на клиентите от сървъра)

След като се е свързал със сървъра, клиентът може да изпраща данни (съобщения). Автентикацията на клиента от страна на сървъра се извършва, като първото изпратено от клиента съобщение след установяването на връзката се проверява от сървъра:

- ако съобщението не е от автентициращ тип (не се заявява регистрация на клиент или не се прави опит за влизане (log in) в системата), връзката незабавно се прекратява;
- ако съобщението е от автентициращ тип, то се проверява чрез наличните в базата данни (или се създава нов профил, ако това е заявено чрез съобщението). Успешно изпълнение на проверката позволява на клиента да изпраща нови заявки към сървъра, а при неуспех връзката незабавно се прекратява.

Сървърът (чрез обект от класа "ConnectionManager") поддържа списък на автентицираните връзки (еднозначно съответствие между име на профил в системата (username) и активна връзка от страна на сървъра), както и списък на всички активни неавтентицирани (току-що стартирани) връзки. При успешно включване на потребител в системата сървъра незабавно прекратява всяка активна връзка към същия потребител, ако такава е налична. По този начин се осигурява единствена позволена връзка на даден потребителски профил към системата.

Прекратяване на работата на сървъра води до неприемане на нови връзки и прекратяване на всички активни връзки, автентицирани или не.



Фиг. 8 Автентикация на клиентите от сървъра

7. Защита на данните чрез използване на SSL - частни и публични ключове

За защита на преноса на данни се използва криптографски протокол за връзка SSL (Secure Socket Layer). Използването на протокола SSL осигурява комуникация, при която данните нито могат да бъдат "прочетени", нито променени от недобронамерени звена по мрежата. Протоколът SSL използва несиметрично криптиране, при което се използват двойки частен и публичен ключ. Единият ключ се използва за криптиране (кодиране) на данните, а съответстващият му - за декодиране:

- клиентът кодира и декодира чрез публичния ключ на сървъра
- сървърът кодира и декодира чрез частния си ключ

"Несиметрично криптиране" означава, че познаването на единия ключ от двойката е недостатъчно (в реално време) да бъде открит съответстващия му ключ от двойката, и че за работата на системата са необходими и двата ключа от двойката. Публичният ключ често е наричан "сертификат".

Задаването на двойка ключове за сървъра се извършва чрез графичния му интерфейс (класът "ServerGUI_SSL"). За клиента това се осъществява чрез създадения клас "networking.security.SSLContextFactory", който има статичен метод-фабрика за генериране на SSLContext обекти, от които се създават SSLSocketFactory обекти, използващи зададените ключове.

Двойки SSL ключове могат да бъдат генерирани чрез използване на инструмента "Java keytool", включен в Java Runtime Environment и Java Development Kit пакетите. Генерирането на SSL ключове не е обект на заданието, но работещи ключове са предоставени в папката "\documentation\example_certificates" за проверка на функционалността:

- във файла "server.keystore" са включени частен и публичен ключ на сървъра, както и публичния ключ (сертификат) на клиента
- във файла "client.keystore" са включени частен и публичен ключ на клиента, както и публичния ключ (сертификат) на сървъра

Паролите за достъп до данните във файловете са съответно:

- "server" за сървъра
- "client" за клиента

8. Графичен интерфейс за управление на връзките между сървъра и клиентите

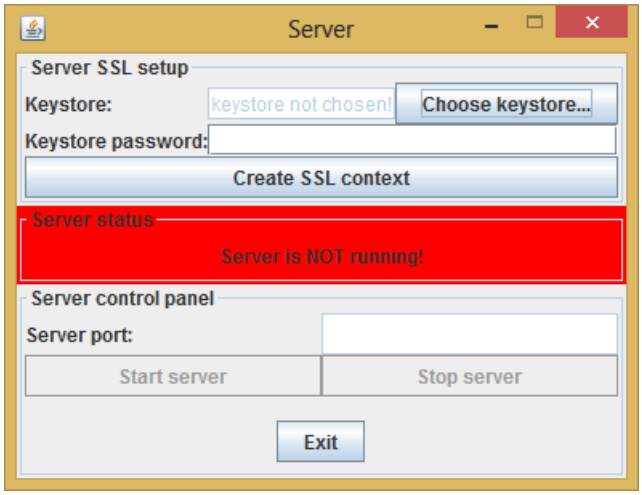
(виж Фиг. 9 Интерфейс сървър - начален)
(виж Фиг. 10 Интерфейс сървър - с настроен SSL контект)
(виж Фиг. 11 Интерфейс сървър - работещ)

Графичният интерфейс за управление на сървъра предлага следните функции:

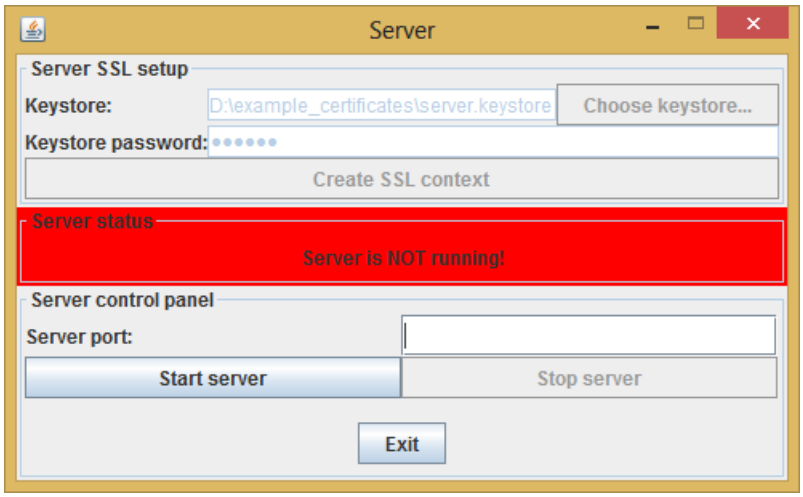
- настройване на SSL контекста (двойка публичен и частен ключ за сървъра и набор от "доверени" или "достоверни" (trusted) публични ключове на клиенти, от които да бъдат приемани връзки по мрежата
- задаване на локален порт, на който да работи сървъра
- стартиране на работата на сървъра
- прекратяване на работата на сървъра, с евентуална промяна на порт и ново стартиране

Графичният интерфейс е разделен на три панела:

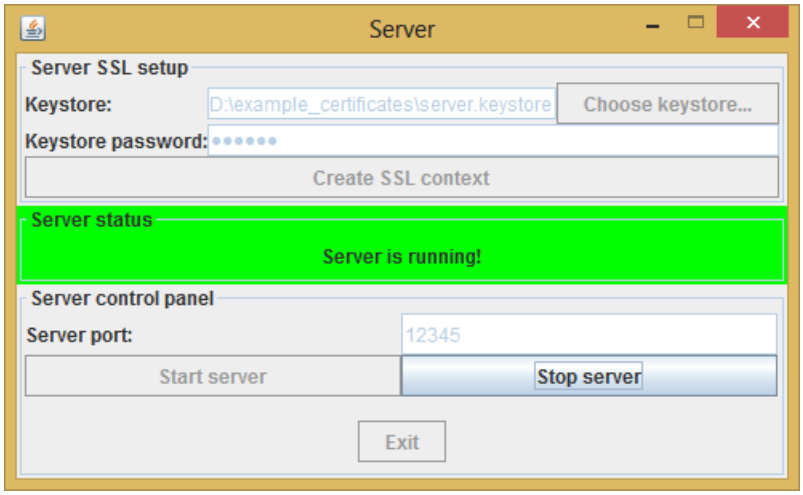
- панел "Server SSL setup", предлагащ избор на keystore файл за сървъра и въвеждане на паролата за данните във файла, както и бутон за генерирането на SSLContext. След успешното генериране на SSLContext панелът става неактивен, предотвратявайки промени в SSL контекста по време на изпълнение.
- панел "Server status", показващ състоянието на сървъра - работещ или не, със съответен цвят (зелен или червен).
- панел "Server control panel", който се активира след успешно създаване на SSL контекст. Панелът предлага задаване на локален порт, на който сървъра да работи. Чрез бутон "Start server" и "Stop server" се стартира или спира работата на сървъра. При спиране на работата на сървъра се забранява приемането на нови и се прекратяват всички активни връзки към клиенти.



Фиг. 9 Интерфейс сървър - начален



Фиг. 10 Интерфейс сървър - с настроен SSL контекст



Фиг. 11 Интерфейс сървър - работещ