Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Специальность: «Программное обеспечение информационных технологий»

**КОНТРОЛЬНАЯ РАБОТА**

По курсу: «Технология разработки программного обеспечения»

На тему: «Социальная сеть. Клиентская часть на мобильном устройстве»

Студент-заочник 2 курса

Группы № 581072

Вонвов Валерий Валерьевич

Адрес: Минский р-н, Ждановичи 2, РИВВИТ 3, кв15.

[vovnovvalera@gmail.com](mailto:vovnovvalera@gmail.com)

Тел. +375(29)834-40-57

Проверила: Бакунова О.М.

Минск, 2017

**СОДЕРЖАНИЕ**

Введение 3

1 Обоснование выбора языка программирования 4

2 Системное проектирование программного средства 5

3 Логическое моделирование 7

3.1 Диаграмма вариантов использования 7

3.2 Диаграмма классов 9

3.3 Диаграмма деятельности 10

3.4 Диаграмма коопераций 12

3.5 Диаграмма последовательности 13

3.6 Диаграмма развёртывания 15

3.7 Диаграмма компонентов 16

3.8 Сгенерированный код в Rational Rose 17

Список использованных источников 33

**ВВЕДЕНИЕ**

Социальные сети, с момента их появления все больше проникают в нашу жизнь. Для многих они уже сейчас являются основным местом проведения времени в интернете. Это происходит потому, что всем нам хочется общения, но не все способны общаться в реальной жизни, так как существует множество психологических барьеров. С каждым годом количество сервисов из персональных компьютеров переносят на мобильные устройства, социальные сети в этом лидируют, пользователи получают мгновенно сообщения либо какой-то контент. Впервые за последнее время количество пользователей через мобильное устройство в социальных сетях посещает больше чем с персонального компьютера. Наличием в мобильном телефоне интернета сегодня никого не удивишь. Стоит ли удивляться появлению новых мобильных сообществ, клубов и чатов. С развитием технологии 3g связи мобильный телефон стал еще и персональным, хотя и малогабаритным компьютером. Видео звонок, который раньше можно было встретить только в книжках писателей-фантастов, сейчас является чем-то обыденным. А про выход в интернет с мобильного телефона в середине 90-х годов прошлого века вообще разговор не шел. С развитием современных средств связи обывателям стали доступны услуги WAP-сети, приходилось искать в сети сайты, которые были специально адаптированы под сотовые телефоны. Про скорость говорить не приходилось.   
Но даже при наличии интернета в мобильном телефоне он остается, прежде всего, средством общения. Сегодня самыми популярными сайтами в интернете являются социальные сети. Мы же рассмотрим относительно новое для Рунета понятие мобильные социальные сети.

В рамках контрольной работы была выбрана тема, социальная сеть на мобильном устройстве. Клиент – серверное приложение, где клиент это приложение, установленное на мобильном устройстве, а сервер это точка связи между клиентами. Сервер имеет свою собственную архитектуру и имеет свою базу данных. На мобильных устройствах тоже есть собственная база данных, кэш загруженный с сервера. В данной контрольной работе мы разработаем бизнес процессы и какие задачи наш сервис будет решать. Разработка такого комплекса очень перспективна в будущем, разив данный сервис, мы тем самым создадим мощную площадку для рекламы, для любых слоев людей, что в итоге данный сервис будет зарабатывать деньги и развиваться, наполняя сам себя контентом.

1. **Обоснование выбора языка программирования**

В рамках генерации кода, был выбран язык программирования Java. В среде разработки Rational Rose.

Java — объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения

Отличительной особенностью Java в сравнении с другими языками программирования общего назначения является обеспечение высокой продуктивности программирования, нежели производительность работы приложения или эффективность использования им памяти.

В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++.

Три ключевых элемента объединились в технологии языка Java.

Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML.

Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты.

Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода.

Огромное преимущество Java заключается в том, что на этом языке можно создавать приложения, способные работать на различных платформах. К сети Internet подключены компьютеры самых разных типов - Pentium PC, Macintosh, рабочие станции Sun и так далее. Даже в рамках компьютеров, созданных на базе процессоров Intel, существует несколько платформ, например, Microsoft Windows версии 3.1, Windows 95, Windows NT, OS/2, Solaris, различные разновидности операционной системы UNIX с графической оболочкой X­Windows. Между тем, создавая сервер Web в сети Internet, хотелось бы, чтобы им могло пользоваться как можно большее число людей. В этом случае выручат приложения Java, предназначенные для работы на различных платформах и не зависящие от конкретного типа процессора и операционной системы.

1. **Системное проектирование программного средства**

IDEF0 — методология функционального моделирования (англ. function modeling) и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность (поток работ).

Стандарт IDEF0 представляет организацию как набор модулей, здесь существует правило — наиболее важная функция находится в верхнем левом углу, кроме того есть правило стороны:

* стрелка входа приходит всегда в левую кромку активности,
* стрелка управления — в верхнюю кромку,
* стрелка механизма — нижняя кромка,
* стрелка выхода — правая кромка.

Описание выглядит как «чёрный ящик» с входами, выходами, управлением и механизмом, который постепенно детализируется до необходимого уровня. Также для того чтобы быть правильно понятым, существуют словари описания активностей и стрелок. В этих словарях можно дать описания того, какой смысл вы вкладываете в данную активность либо стрелку.

Также отображаются все сигналы управления, которые на DFD (диаграмме потоков данных) не отображались. Данная модель используется при организации бизнес-проектов и проектов, основанных на моделировании всех процессов: как административных, так и организационных.

Графический язык IDEF0 удивительно прост и гармоничен. В основе методологии лежат четыре основных понятия:

Первым из них является понятие функционального блока (Activity Box). Функциональный блок графически изображается в виде прямоугольника (см. рис. 2.1) и олицетворяет собой некоторую конкретную функцию в рамках рассматриваемой системы. По требованиям стандарта название каждого функционального блока должно быть сформулировано в глагольном наклонении (например, “производить услуги”, а не “производство услуг”).

Контекстная диаграмма модели показана на рисунке 2.1. Декомпозиция контекстной диаграммы функциональной модели показана на рис. 2.2. Для построения модели использовался продукт BPwin 4.0 фирмы Computer Associates.

Субъектом моделирования является программное средство «Социальная сеть», цель – описать функциональность работы программного средства, точка зрения– пользователь, сервер.

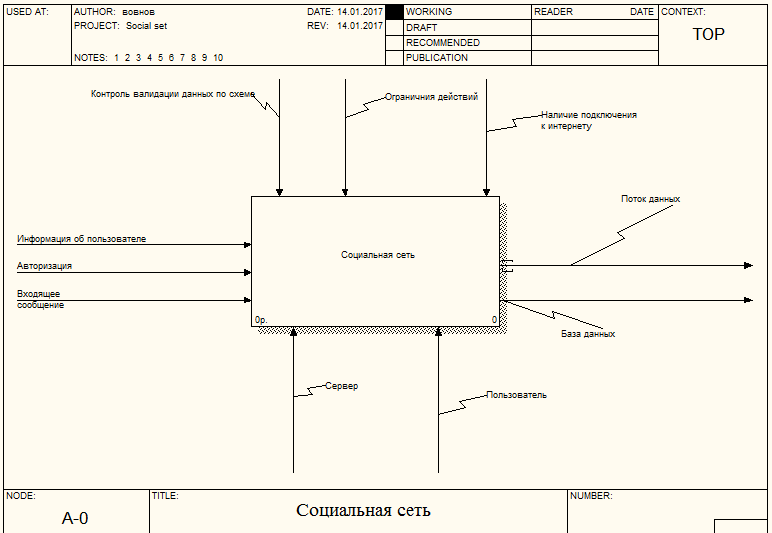


Рисунок 2.1 – Контекстная диаграмма модели

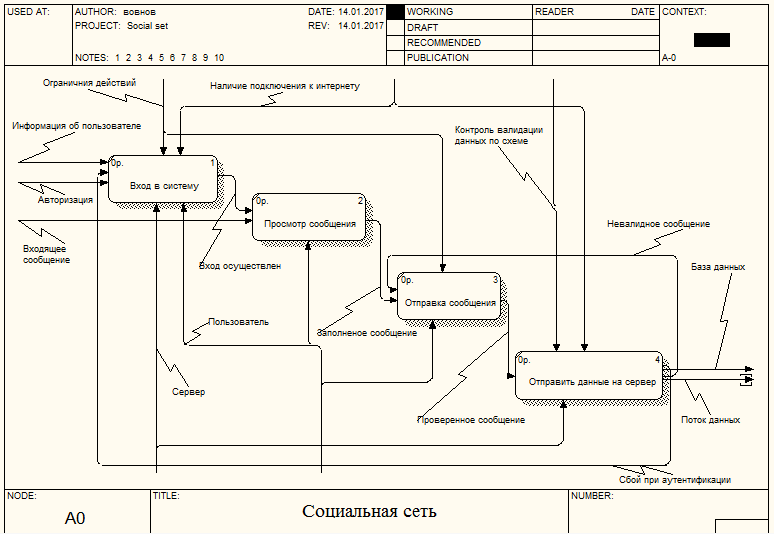


Рисунок 2.2– Декомпозиция контекстной диаграммы

функциональной модели

**3 Логическое моделирование**

Rational Rose - популярное средство визуального моделирования, которое считается стандартом де-факто среди средств визуального проектирования приложений. Этот продукт входит в состав пакета IBM Rational Suite и предназначен для моделирования программных систем с использованием широкого круга инструментальных средств и платформ. Инструментальное средство IBM Rational Rose расширяет возможности моделирования программных систем, выходящих за рамки платформы J2EE и инструментальных средств моделирования в составе IBM Rational Professional Bundle. Являясь простым и мощным решением для визуальной разработки информационных систем любого класса, Rational Rose позволяет создавать, изменять и проверять корректность модели. Rational Rose объединяет команду разработчиков на базе универсального языка моделирования UML, который определяет стандартную графическую символику для описания архитектуры ПО. Любые участники проекта - аналитики, специалисты по моделированию, разработчики и другие - могут использовать модели, построенные в Rational Rose, для большей эффективности создания конечного продукта.

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

### 3.1 Диаграмма вариантов использования

Модель вариантов использования предназначается для определения требований к системе. Она включает в себя актеров, варианты использования и связи между ними. Для отображения этой модели язык UML предлагает использовать диаграммы Use Case (вариант использования) совместно с моделями State Diagram (диаграммы состояний) и Activity Diagram (диаграммы деятельности/активности). Последние используются для конкретизации вариантов использования системы

Разработка диаграммы вариантов использования преследует цели:

* определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Представленные варианты использования имеют логический смысл, актер «Пользователь» имеет следующие возможности:

* регистрация, возможность регистрации в социальной сети;
* аутентификация, процесс распознавания пользователя в системе;
* сообщения, возможность отправлять, принимать и удалять сообщения, которые пришли от сервера;
* друзья, данная возможность позволяет формировать свой круг знакомых зарегистрированных в социальной сети, так же есть возможности оперирования с ними, удаление и добавления;
* черный лист, список людей, которые не могу вам писать и видеть ваши данные;
* группы, страницы, содержащие в себе какой либо информации, либо контента.

Диаграмма вариантов использования представлена на рисунке 3.1.

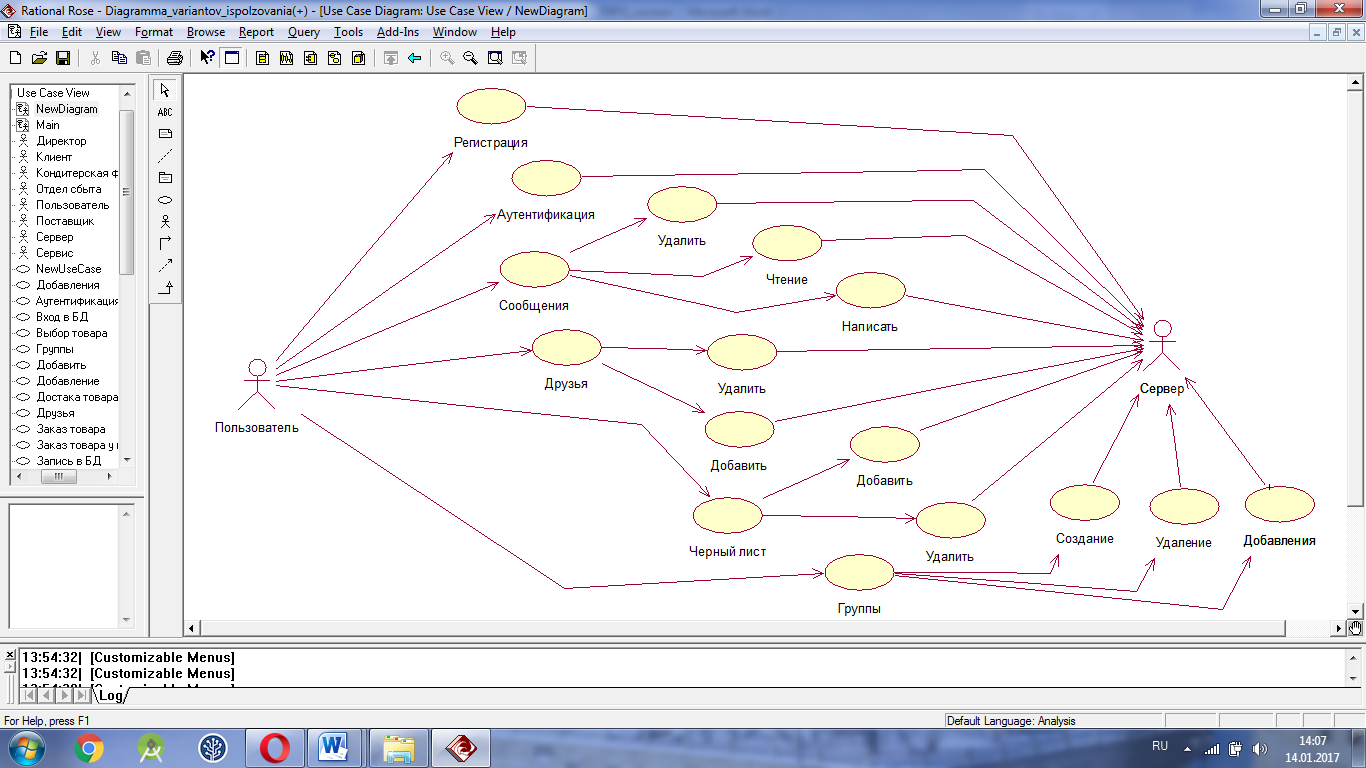


Рисунок 3.1 – Диаграмма вариантов использования

### 3.2 Диаграмма классов

Диаграмма классов — диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Входит в UML.

Существует два вида:

* статический вид диаграммы рассматривает логические взаимосвязи классов между собой;
* аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

* концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
* точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;
* точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

Диаграмма классов представлена на рисунке 3.2.

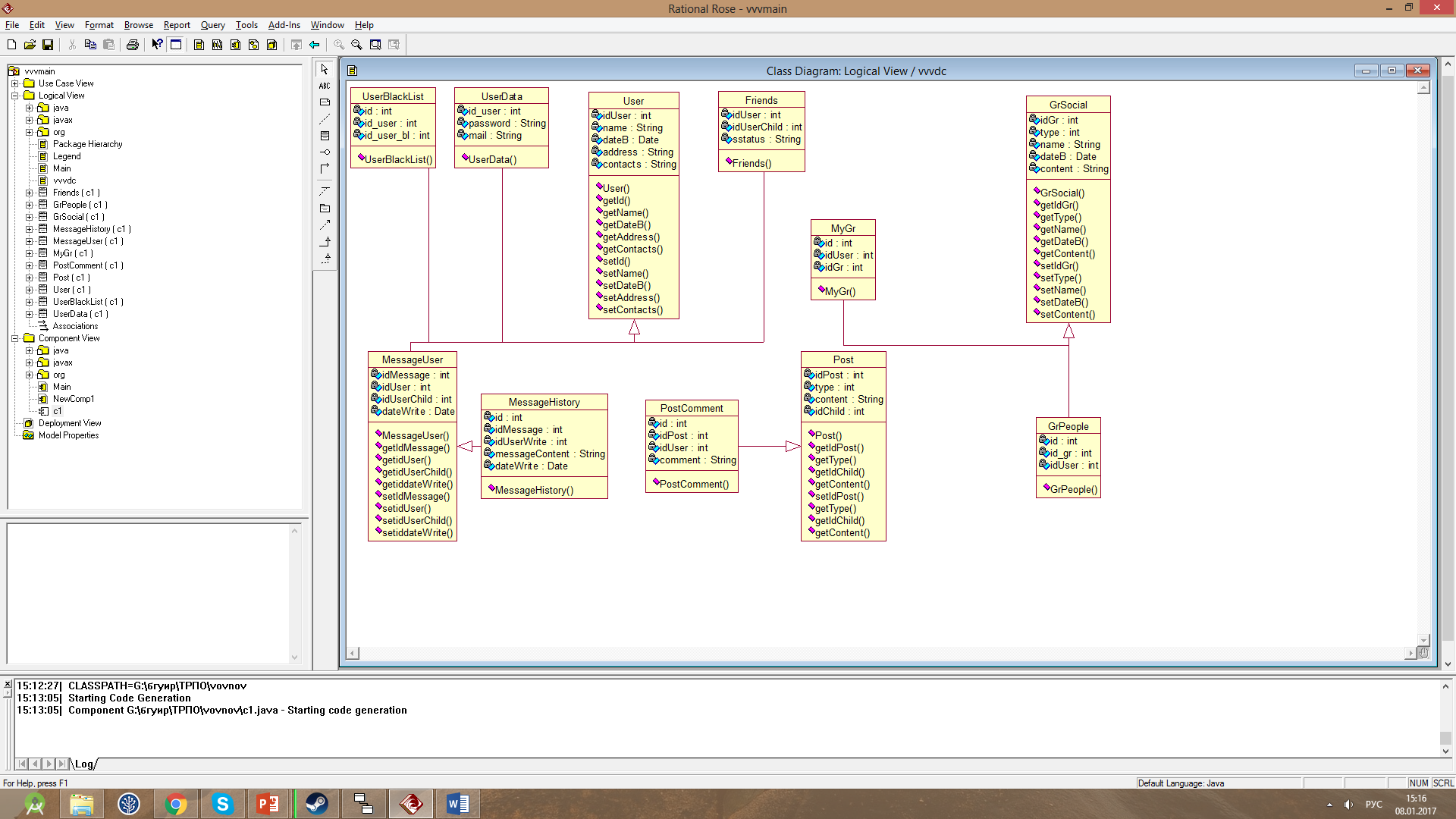


Рисунок 3.2 – Диаграмма классов

**3.3 Диаграмма деятельности**

Диаграмма деятельности — UML-диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью (англ. activity) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий англ. action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Диаграммы деятельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений.

На данной диаграмме виден весь бизнес-процесс как может вести себя клиент на мобильном устройстве. Диаграмма деятельности представлена на рисунке 3.3.

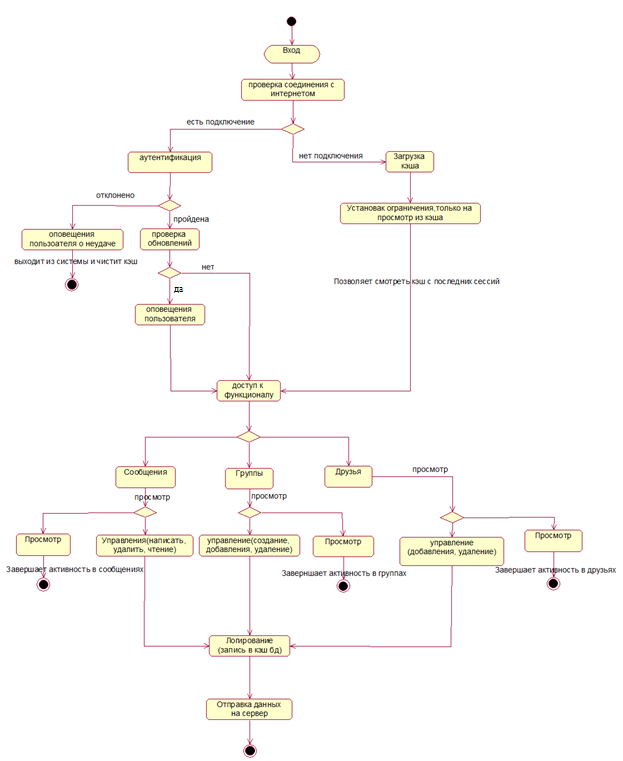


Рисунок 3.3 – Диаграмма деятельности

### 3.4 Диаграмма коопераций

Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

Прежде всего, на диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи - потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии. На этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров. Следовательно, если необходимо явно специфицировать взаимосвязи между объектами в реальном времени, лучше это делать на диаграмме последовательности.

На данной диаграмме видно взаимодействия пользователя(клиента) и сервера в работе. Диаграмма коопераций представлена на рисунке 3.4.

****

Рисунок 3.4 – Диаграмма коопераций

### 3.5 Диаграмма последовательности

Диаграмма последовательности — диаграмма, на которой показано взаимодействие объектов, упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления. Используется в языке UML.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

На данной диаграмме показано взаимодействия мобильного приложения со своей базой данных в виде кэша, само устройство с сервером и сервер со своей базой данных. Диаграмма последовательности представлена на рисунке 3.5.

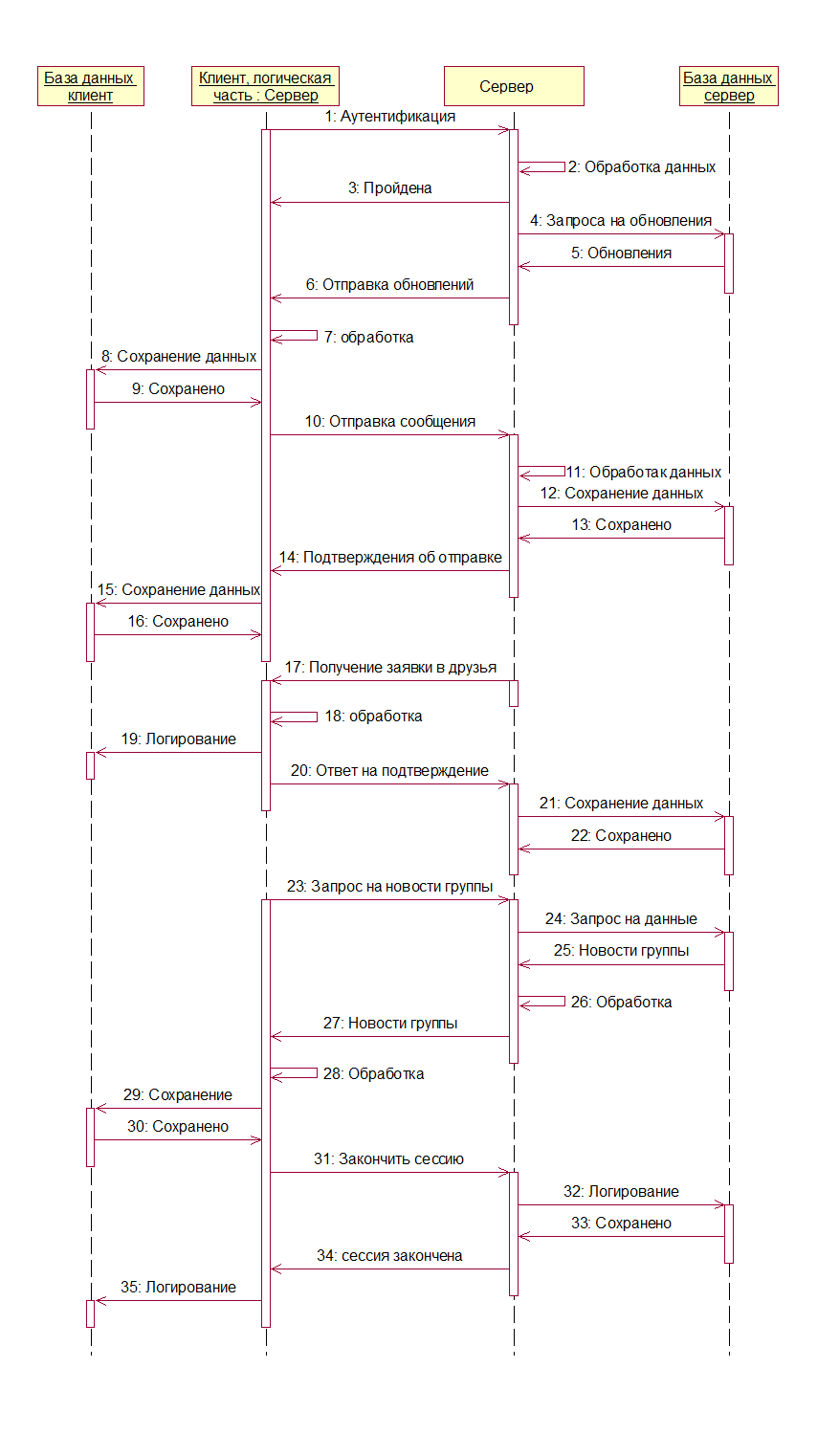


Рисунок 3.5 – Диаграмма последовательности

### Диаграмма развёртывания

Диаграмма развёртывания в UML моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, JDBC, REST, RMI).

Узлы представляются как прямоугольные параллелепипеды с артефактами, расположенными в них, изображенными в виде прямоугольников. Узлы могут иметь под узлы, которые представляются как вложенные прямоугольные параллелепипеды. Один узел диаграммы развертывания может концептуально представлять множество физических узлов, таких как кластер серверов баз данных.

Существует два типа узлов:

* узел устройства;
* узел среды выполнения.

Узлы устройств — это физические вычислительные ресурсы со своей памятью и сервисами для выполнения программного обеспечения, такие как обычные ПК, мобильные телефоны. Узел среды выполнения — это программный вычислительный ресурс, который работает внутри внешнего узла и который предоставляет собой сервис, выполняющий другие исполняемые программные элементы.

Диаграмма развёртывания представлена на рисунке 3.6.



Рисунок 3.6 – Диаграмма развёртывания

### Диаграмма компонентов

Диаграмма компонентов — статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Компоненты связываются через зависимости, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом иллюстрируются отношения клиент-источник между двумя компонентами.

Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту. Зависимость изображается стрелкой от интерфейса или порта клиента к импортируемому интерфейсу.

Когда диаграмма компонентов используется, чтобы показать внутреннюю структуру компонентов, предоставляемый и требуемый интерфейсы составного компонента могут делегироваться в соответствующие интерфейсы внутренних компонентов.

Делегация показывается связь внешнего контракта компонента с внутренней реализацией этого поведения внутренними компонентами.



Диаграмма компонентов представлена на рисунке 3.7

* 1. **Сгенерированный код в Rational Rose**

В рамках генерации кода, был выбран язык программирования Java, в среде разработке Rational Rose.

import java.sql.Date;

private class User

{

private int idUser;

private String name;

private Date dateB;

private String address;

private String contacts;

/\*\*

@roseuid 58721B310380

\*/

public User()

{

}

/\*\*

@return int

@roseuid 58721B6B0033

\*/

public int getId()

{

return idUser;

}

/\*\*

@return java.lang.String

@roseuid 58721D1B0030

\*/

public String getName()

{

return name;

}

/\*\*

@return java.sql.Date

@roseuid 58721D1B003C

\*/

public Date getDateB()

{

return dateB;

}

/\*\*

@return java.lang.String

@roseuid 58721D1B0049

\*/

public String getAddress()

{

return address;

}

/\*\*

@return java.lang.String

@roseuid 58721D1B0055

\*/

public String getContacts()

{

return contacts;

}

/\*\*

@param idUser

@roseuid 58721D89017B

\*/

public void setId(int idUser)

{

this.idUser = idUser;

}

/\*\*

@param name

@roseuid 58721DBE019B

\*/

public void setName(String name)

{

this.name = name;

}

/\*\*

@param dateB

@roseuid 58721EA1030F

\*/

public void setDateB(Date dateB)

{

this.dateB = dateB;

}

/\*\*

@param Address

@roseuid 58721EA10336

\*/

public void setAddress(String Address)

{

this.Address = Address;

}

/\*\*

@param Contacts

@roseuid 58721EA1035A

\*/

public void setContacts(String Contacts)

{

this.Contacts = Contacts;

}

}

private class UserData extends User

{

private int id\_user;

private String password;

private String mail;

/\*\*

@roseuid 58721B31038A

\*/

public UserData()

{

}

}

private class UserBlackList extends User

{

private int id;

private int id\_user;

private int id\_user\_bl;

/\*\*

@roseuid 58721B310394

\*/

public UserBlackList()

{

}

}

private class MessageUser extends User

{

private int idMessage;

private int idUser;

private int idUserChild;

private Date dateWrite;

/\*\*

@roseuid 5872208C0302

\*/

public MessageUser()

{

}

/\*\*

@return int

@roseuid 58722140004F

\*/

public int getIdMessage()

{

return idMessage;

}

/\*\*

@return int

@roseuid 587221400059

\*/

public int getidUser()

{

return idUser;

}

/\*\*

@return int

@roseuid 587221400063

\*/

public int getidUserChild()

{

return idUserChild;

}

/\*\*

@return java.sql.Date

@roseuid 58722140006D

\*/

public Date getiddateWrite()

{

return dateWrite;

}

/\*\*

@param idMessage

@roseuid 58722140007B

\*/

public void setIdMessage(int idMessage)

{

this.idMessage = idMessage;

}

/\*\*

@param idUser

@roseuid 587221400098

\*/

public void setidUser(int idUser)

{

this.idUser = idUser;

}

/\*\*

@param idUserChild

@roseuid 5872214000B5

\*/

public void setidUserChild(int idUserChild)

{

this.idUserChild = idUserChild;

}

/\*\*

@param dateWrite

@roseuid 5872214000D4

\*/

public void setiddateWrite(Date dateWrite)

{

this.dateWrite = dateWrite;

}

}

private class MessageHistory extends MessageUser

{

private int id;

private int idMessage;

private int idUserWrite;

private String messageContent;

private Date dateWrite;

/\*\*

@roseuid 5872225F0152

\*/

public MessageHistory()

{

}

}

private class Friends extends User

{

private int idUser;

private int idUserChild;

private String sstatus;

/\*\*

@roseuid 587226450253

\*/

public Friends()

{

}

}

private class GrSocial

{

private int idGr;

private int type;

private String name;

private Date dateB;

private String content;

/\*\*

@roseuid 5872276800F4

\*/

public GrSocial()

{

}

/\*\*

@return int

@roseuid 587228A30128

\*/

public int getIdGr()

{

return idGr;

}

/\*\*

@return int

@roseuid 587228A30132

\*/

public int getType()

{

return type;

}

/\*\*

@return java.lang.String

@roseuid 587228A3013C

\*/

public String getName()

{

return name;

}

/\*\*

@return java.sql.Date

@roseuid 587228A30148

\*/

public Date getDateB()

{

return dateB;

}

/\*\*

@return java.lang.String

@roseuid 587228A30157

\*/

public String getContent()

{

return content;

}

/\*\*

@param idGr

@roseuid 587228A30164

\*/

public void setIdGr(int idGr)

{

this.idGr = idGr;

}

/\*\*

@param type

@roseuid 587228A30183

\*/

public void setType(int type)

{

this.type = type;

}

/\*\*

@param name

@roseuid 587228A301A4

\*/

public void setName(String name)

{

this.name = name;

}

/\*\*

@param dateB

@roseuid 587228A301C6

\*/

public void setDateB(Date dateB)

{

this.dateB = dateB;

}

/\*\*

@param content

@roseuid 587228A301EB

\*/

public void setContent(String content)

{

this.content = content;

}

}

private class GrPeople extends GrSocial

{

private int id;

private int id\_gr;

private int idUser;

/\*\*

@roseuid 5872276800FF

\*/

public GrPeople()

{

}

}

private class MyGr extends GrSocial

{

private int id;

private int idUser;

private int idGr;

/\*\*

@roseuid 58722BE9029F

\*/

public MyGr()

{

}

}

private class PostComment extends Post

{

private int id;

private int idPost;

private int idUser;

private String comment;

/\*\*

@roseuid 58722BE902AC

\*/

public PostComment()

{

}

}

private class Post

{

private int idPost;

private int type;

private String content;

private int idChild;

/\*\*

@roseuid 58722BE902BF

\*/

public Post()

{

}

/\*\*

@return int

@roseuid 58722CAE01E9

\*/

public int getIdPost()

{

return idPost;

}

/\*\*

@return int

@roseuid 58722CAE01F3

\*/

public int getType()

{

return type;

}

/\*\*

@return int

@roseuid 58722CAE01FE

\*/

public int getIdChild()

{

return idChild;

}

/\*\*

@return java.lang.String

@roseuid 58722CAE020A

\*/

public String getContent()

{

return content;

}

/\*\*

@param idPost

@roseuid 58722CAE0217

\*/

public void setIdPost(int idPost)

{

this.idPost = idPost;

}

/\*\*

@param type

@roseuid 58722CAE0234

\*/

public void getType(int type)

{

this.type = type;

}

/\*\*

@param idChild

@roseuid 58722CAE0255

\*/

public void getIdChild(int idChild)

{

this.idChild = idChild;

}

/\*\*

@param content

@roseuid 58722CAE0276

\*/

public void getContent(String content)

{ this.content = content; }}

**СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ**

1. ГОСТ 2.105-95 ЕСКД. Введ. 1996–07–01. – М.: Изд-во стандартов, 1996.
2. ГОСТ 19.104–78 – Единая система программной документации. Основные надписи.
3. Маклаков С.В. BPwin и ERwin: CASE - средства для разработки информационных систем.
4. Федотов Д.Э., Семенов Ю.Д., Чижик К.Н. Практикум для высших учебных заведений. CASE-технологии.-157с.
5. Интернет обучение [Электронный ресурс] / Котляр Д.С. Использование CASE-средства ERwin для автоматизации проектирования и разработки базы данных – Режим доступа: http://royallib.com/read/ bezopasnost – Дата доступа: 15.12.2015.
6. Интернет обучение [Электронный ресурс] / Информационные системы и технологии – Режим доступа: <http://www.narfu.ru> – Дата доступа: 19.12.2015.
7. Интернет обучение [Электронный ресурс] / Нормализация структурны данных – Режим доступа: http://infostart.ru/public/269803/ – Дата доступа: 15.01.2016.
8. Интернет обучение [Электронный ресурс] / Программа компьютерного моделирования BpWin – Режим доступа: http://bourabai.kz/cm/bpwin.htm– Дата доступа: 11.11.2015.