Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»
Институт математики, информатики и информационных технологий
Кафедра информатики, информационных технологий
и методики обучения информатике

ТЕХНОЛОГИЯ РАЗРАБОТКИ КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ НА ЯЗЫКЕ JAVA

Выпускная квалификационная работа по направлению
«02.03.02 Фундаментальная информатика и информационные технологии»

Исполнитель: студент ИИМиИТ
группы Б-41 Баранский И.В.

Руководитель: к.п.н, доцент кафедры
ИИТиМОИ Газейкина А.И.

Работа допущена к защите
«___»_____2017 г.
Зав. кафедрой _____

Екатеринбург – 2017

Реферат

Баранский И.В. ТЕХНОЛОГИЯ РАЗРАБОТКИ КЛИЕТН-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ НА ЯЗЫКЕ JAVA, выпускная квалификационная работа: 50 стр., рис. 15, табл. 3, библ. названий 24.

Ключевые слова: объектно-ориентированный язык программирования Java, сокеты, архитектуры клиент-серверных приложений, алгоритмы шифрования.

Объект исследования: процесс разработки клиент-серверного приложения на Java.

Цель работы: создать технологию разработки клиент-сервера и клиент-серверное приложение на объектно-ориентированном языке Java.

Работа направлена на разработку приложения с двухуровневой клиент-серверной архитектурой по созданной технологии разработки клиент-серверного приложения. Рассматривается специфика разработки приложений на языке программирования Java.

В ходе работы было создано клиент-серверное приложение «Chat». Исходные коды написаны на языке программирования Java.

Оглавление

Введение

Возможность передачи информации на расстоянии интересовала людей всегда. Для того чтобы обмениваться информацией не только при личной встрече, но и на больших расстояниях, люди изобретали всё новые технические средства, протягивали структурированные кабельные сети по всему миру, организовывали почтовые системы, запускали спутники связи. С развитием информационных технологий стали возможным еще более глобальные коммуникации. Сегодня люди с легкостью по всему миру могут обмениваться не только сообщениями, но и графикой и звуковой и другими видами информации. Историческим «докомпьютерным» предшественником чатов, несомненно, являлся телефон. Ни телеграф, ни почта не могли позволить общаться в реальном времени и не были доступны в домашних условиях. Изобретение и распространение телефона вызвало настоящую революцию способах и средствах общения. Возможность поговорить с собеседником в реальном времени на другом континенте казалась настоящим чудом.

Язык Java является основой практически для всевозможных типов сетевых приложений и всеобщим стандартом для разработки встроенных и мобильных приложений, веб-контента, игр, и корпоративного программного обеспечения. В мире насчитывается более девяти миллионов специалистов, разрабатывающих приложения на Java, которая позволяет эффективно разрабатывать, тестировать, внедрять и использовать программное обеспечение [14].

Объект исследования: процесс разработки клиент-серверного приложения на Java.

Предмет исследования: технология разработки клиент-сервера с двухуровневой архитектурой.

Цель исследования: создать технологию разработки клиент-сервера и клиент-серверное приложение на объектно-ориентированном языке Java.

Задачи:

1. Рассмотреть подходы и принципы создания клиент-серверных приложений.
2. Проанализировать особенности разработки клиент-серверных приложений на языке Java.
3. Обосновать технологию разработки клиент-серверных приложений.
4. Разработать клиент-серверное приложение «Chat» на языке Java и протестировать его.
5. Провести экспертную оценку разработанного приложения.

Глава 1. Теоретические основы разработки клиент-серверных приложений на языке Java

1.1 Разработка клиент-серверных приложений

Процесс разработки программного обеспечения – набор правил, согласно которой построена разработка программного обеспечения. Приложение можно назвать клиент серверным если оно включает в себя клиент-серверную архитектуру. Разработку клиент-серверного приложения необходимо начинать с выбора архитектуры клиент-сервера.

Архитектура «клиент-сервер» характеризуется наличием по крайне мере двух взаимодействующих, самостоятельных процессов – клиента и сервера.

Процессы, осуществляющие некоторую службу, например, службу базы данных или файловой системы, называются серверами, а процессы, запрашивающие службы у серверов посредством посылки запроса и последующего ожидания ответа от сервера, называются клиентами. Фактически эти процессы – программное обеспечение, которое установлено на разных вычислительных машинах и взаимодействующее между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине.

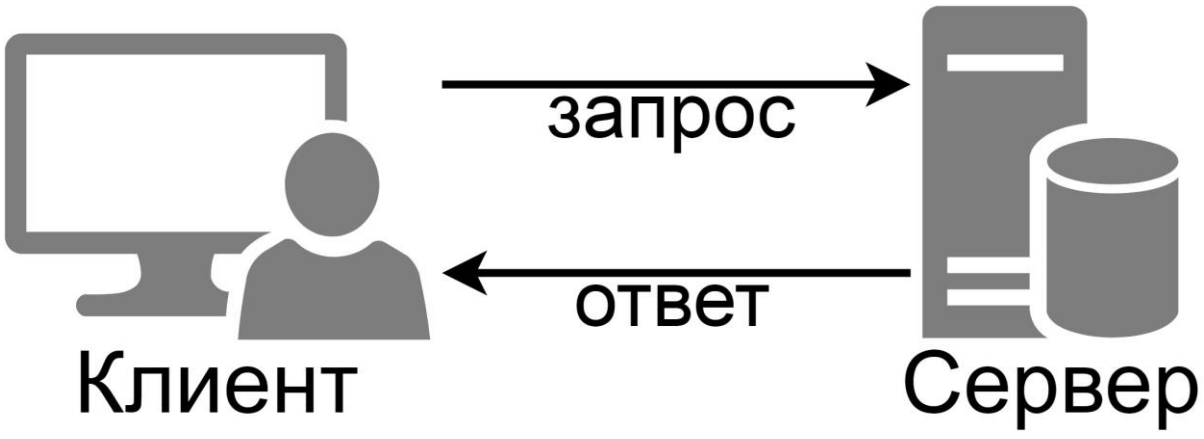


Рисунок 1. Двухуровневая архитектура клиент-сервер

Любая архитектура клиент-сервера должна включать в себя три основных компонента:

- ввод и отображение данных (интерфейс с пользователем);
- прикладные функции, характерные для данной предметной области;
- управление информационными ресурсами (базой даных или файловой системой и т.д).

В классической архитектуре клиент-сервер необходимо распределять три основные части приложения по двум физическим программным обеспечениям.

Компания Gartner Group, специализирующейся в области исследования информационных технологий, выводит следующие пять двухзвенных (two-tier, 2-tier) моделей взаимодействия клиента и сервера (двухзвенными они называются потому, что три компонента приложения различным образом распределяются между двумя узлами) [10].

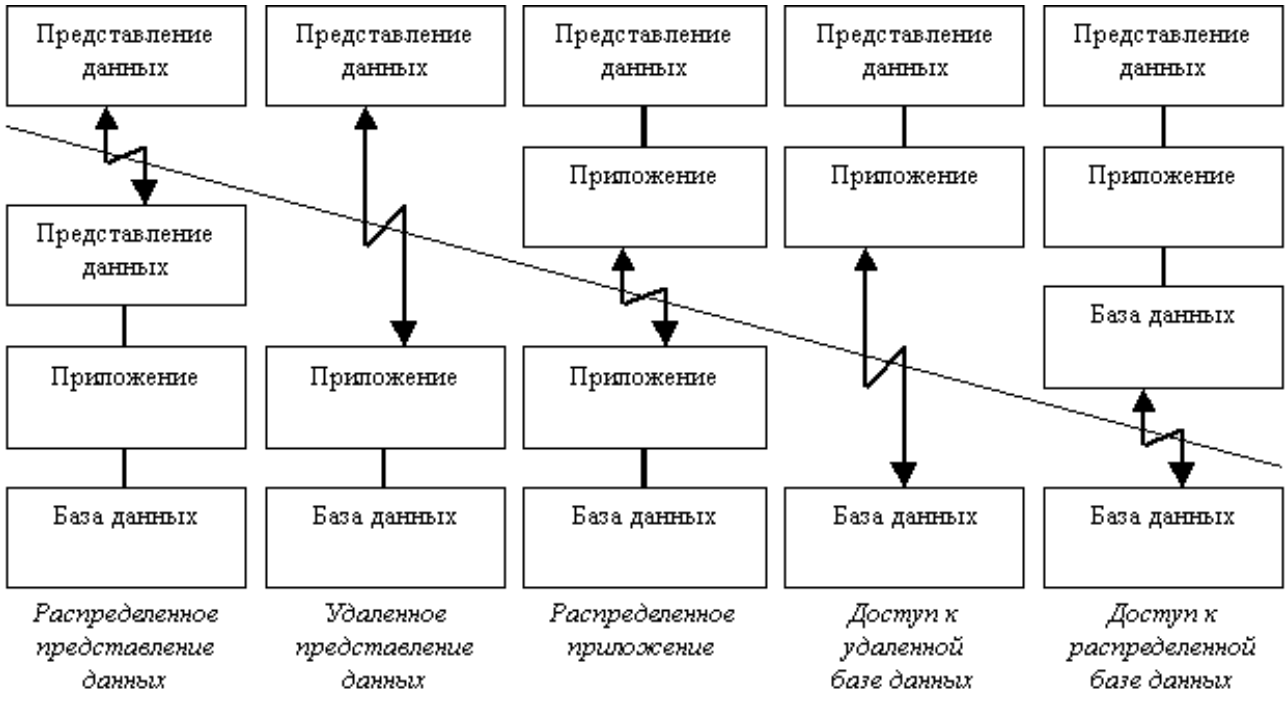


Рисунок 2. Модели взаимодействия двухзвенной (двухуровневой) архитектуры клиент-сервер

Плюсы двухуровневой архитектуры:

- Есть возможность для масштабирования

- Относительная экономия на оборудовании

Минусы:

- Масштабируется только вертикально (увеличение производительности компонентов системы);
- Сложность интеграции новых возможностей (в зависимости от реализации трех компонентов между клиентом и сервером);

Многоуровневая архитектура (N-tier или multi-tier, иногда его называют трехуровневая архитектура или трехзвенная архитектура, но это частный случай) представляет собой дальнейшее совершенствование технологии «клиент-сервер». В трехзвенной архитектуре вся обработка данных, ранее входившая в клиентские приложения или полностью была на сервере либо эти два звена делили её функции между собой, выделяется в отдельное звено, называемое сервером приложений.

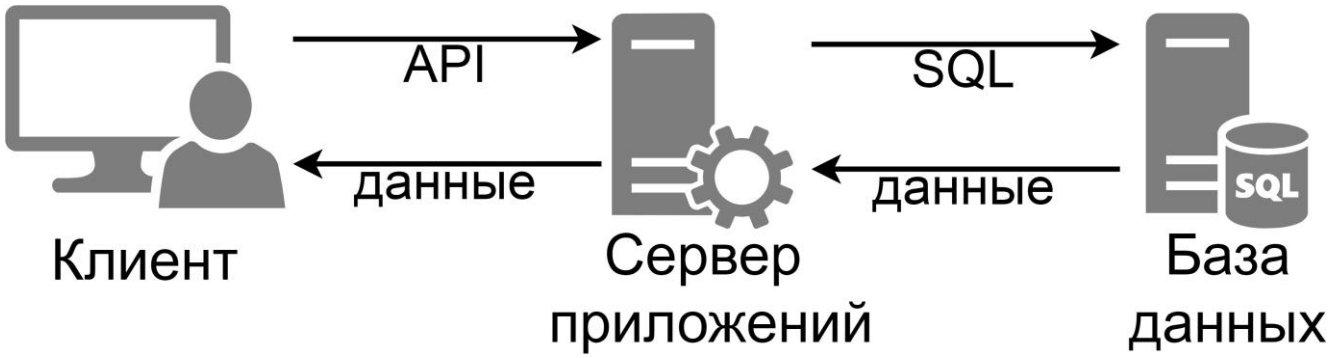


Рисунок 3. Многоуровневая архитектура клиент-сервер

Плюсы многоуровневой архитектуры:

- Масштабируемость (за счёт горизонтальной масштабируемости сервера приложений и мультиплексирования соединений);
- Интегрируемость (за счёт изолированности уровней друг от друга);
- Доступность из веб-браузера или из тонкого клиента (т.к клиент занимается только отображением информации пользователю в удобном виде);

- Высокая безопасность;

Минусы:

- Высокая стоимость, дополнительные расходы на администрирование;
- Разработка такой системы гораздо сложнее чем двухзвенной.

Клиент-серверная система должна иметь достойный уровень безопасности информации, хранимой и передаваемой между звеньями системы.

Разработчику такой системы нужно добиться согласованной работы средств защиты различных звеньев, и не усложнять жизнь рядовых пользователей, дабы не заставлять их в поисках нужных данных пробираться через множество слоев защиты отвечая на однотипные вопросы: «Стой, кто идет?».

Под безопасностью информационной системы понимается защищенность системы от случайного или преднамеренного вмешательства в нормальный процесс ее функционирования, от попыток хищения (несанкционированного получения) информации и её модификации.

При передаче информации по каналам связи используются криптографические алгоритмы шифрования.

Существует несколько классификаций КА алгоритмов одна из них делит по числу ключей, применяемых в конкретном алгоритме:

- бесключевые КА – не используют в вычислениях никаких ключей;
- одноключевые КА – работают с одним ключевым параметром (секретным ключом);
- двухключевые КА – на различных стадиях работы в них применяются два ключевых параметра: секретный и открытый ключи [1].

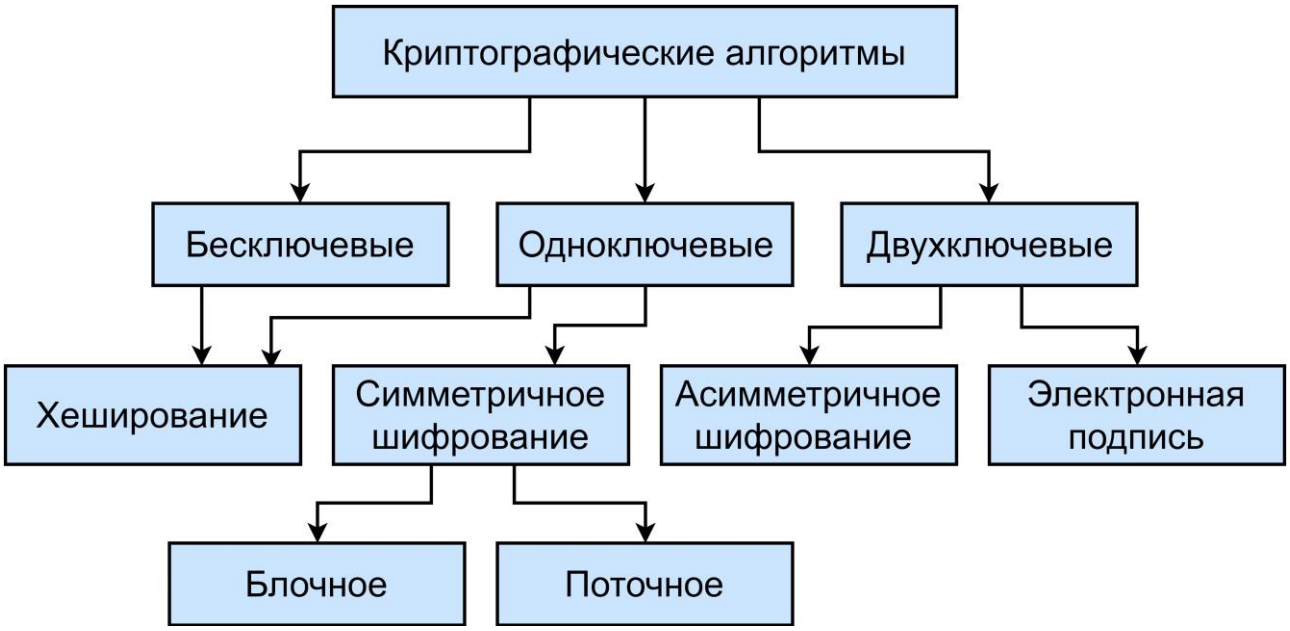


Рисунок 4. Классификация криптоалгоритмов защиты информации

Хеширование – это метод криптозащиты, представляющий собой контрольное преобразование информации: из данных неограниченного размера путем выполнения криптографических преобразований вычисляется хэш-значение фиксированной длины, однозначно соответствующее исходным данным.

Таблица 1. Алгоритмы криптографических хеш-функций

Алгоритм	Год создания	Особенности
MD2 (Message Digest 2 Algorithm)	1989	После MD2 были разработаны MD4, MD5 и MD6 в 1990, 1991 и 2008 годах соответственно. В 2011 году MD2 был официально списан из-за многочисленных успешных криптоатак. Сейчас использовать MD2 рекомендуют только в целях совместимости со старыми программами, которые до сих пор полагаются на него.
SHA-1 (Secure Hash Algorithm 1)	1995	В настоящее время SHA-1 больше нельзя считать надежным, большинство компаний переходят на SHA-2, которые построены на основе структуры Меркла – Дамгарда. После SHA-1 были разработаны SHA-2, SHA-3 в 2002 и 2008 годах соответственно.

Whirlpool	2000	Основан на схеме Миагучи-Пренеля, которая использует блочный шифр с симметричными ключами вместо функции сжатия. Блочный шифр в данном случае является измененным шифром AES, который был приспособлен для этой цели.
Стрибог	2012	Действующий российский криптографический стандарт, определяющий алгоритм и процедуру вычисления хеш-функции. Разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «ИнфоТеКС» и введенный в действие 1 января 2013 года.

Симметричное шифрование использует один ключ, с помощью которого производится как шифрование, так и расшифровка с использованием одного и того же алгоритма симметричного шифрования. Этот ключ передается двум участникам взаимодействия безопасным образом до передачи зашифрованных данных.

Симметричное шифрование можно разделить на два вида: блочное и поточное, в некоторых классификациях они не разделяются и считается, что поточное шифрование – это шифрование блоков единичной длины.

Блочное шифрование характеризуется тем, что информация предварительно разбивается на блоки зафиксированной длины (например, 64 или 128 бит). При этом в различных криптографических алгоритмах или даже в разных режимах работы одного и того же алгоритма блоки могут шифроваться как независимо друг от друга, так и «со сцеплением», т. е. когда результат шифрования текущего блока данных зависит от значения предыдущего блока или от результата шифрования предыдущего блока.

Таблица 2. Алгоритмы симметричного блочного шифрования

Алгоритм	Год создания	Размер ключа (бит)	Размер блока (бит)	Число раундов	Метод построение шифра

DES	1977	56 + 8 проверочных	64	16	Сеть Фейстеля
3DES	1978	168	64	48	Сеть Фейстеля
BassOmatic	1989	8 до 2048	2048	от 1 до 8	Подстановочно-перестановочная сеть
RC2	1989	8 до 1024	64	18	Сеть Фейстеля
IDEA	1991	128	64	9	модификация сети Фейстеля
Cobra	1996	8-576	128	12	Сеть Фейстеля
CRYPTON	1998	128/192/256	128	12	Подстановочно-перестановочная сеть
Camellia	2000	128/192 или 256	128	18/24	Сеть Фейстеля
Threefish	2008	256/512/1024	256/512/1024	72/80 – при ключе в 1024	Подстановочно-перестановочная сеть
RC5	1994	0-2040	32/64 или 128	1-255	Сеть Фейстеля

Поточное шифрование применяется, прежде всего, тогда, когда информацию невозможно разбить на блоки – например, есть некий поток данных, каждый символ которых нужно зашифровать и отправить, не дожидаясь остальных данных, достаточных для формирования блока.

Потоковые шифры на базе сдвиговых регистров активно использовались в годы войны, ещё задолго до появления электроники. Они были просты в проектировании и реализации. Алгоритмы поточного шифрования шифруют данные побитно или посимвольно.

Стойкость таких алгоритмов целиком зависит от внутренней структуры генератора ключевой последовательности. Если генератор выдает последовательность с небольшим периодом, то стойкость системы будет невелика. Напротив, если генератор будет выдавать бесконечную последовательность истинно случайных бит, то получится одноразовый блокнот с идеальной стойкостью [13].

К симметричным поточным алгоритмам относятся:

- A8 – используется для обеспечения конфиденциальности передаваемой по радиоканалу информации в стандарте мобильной сотовой связи GSM. A8 является одним из алгоритмов обеспечения секретности разговора в GSM вместе с A5 и A3.
- Decim – на основе регистра сдвига с линейной обратной связью, запатентован. Был представлен в проекте eSTREAM, где не прошёл дальше третьего этапа.
- Rabbit – разработан с целью использования в программном обеспечении, как обладающий высокой скоростью шифрования. вошл в портфолио eCRYPT, ориентированный на программную реализацию.
- MICKEY – использует нерегулярное тактирование сдвиговых регистров, а также новые методы, обеспечивающие достаточно большой период и псевдослучайность ключевой последовательности и устойчивость к атакам. Текущая версия

алгоритма – 2.0. Она вошла в портфолио eCRYPT, как потоковый шифр для аппаратной реализации.

- SEAL – описан как увеличивающая длину псевдослучайная функция, которая очевидным образом может быть использована для реализации поточного шифрования.
- SOSEMANUK – использует как основные принципы из потокового шифра SNOW 2.0, так и некоторые преобразования, выведенные из блочного шифра SERPENT. Как утверждается, любая длина ключа достигает 128-битной защиты.
- VMPC – разрабатывался в качестве усиленного варианта популярного шифра RC4. Основа шифра – генератор псевдослучайных чисел, базой которого является односторонняя необратимая функция VMPC.

Асимметричное шифрование характеризуется применением двух типов ключей: открытого (k1) – для зашифровывания информации и секретного (k2) – для ее расшифровывания. Открытый ключ вычисляется из секретного: k1=f(k2) Эти алгоритмы шифрования основаны на применении однонаправленных функций. Согласно определению, функция y=f(x) является однонаправленной, если: ее легко вычислить для всех возможных вариантов x и для большинства возможных значений y достаточно сложно вычислить такое значение x, при котором y=f(x) [19].

К асимметричным алгоритмам относятся:

- RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) – основывающийся на вычислительной сложности задачи факторизации больших целых чисел, стала первой системой, пригодной и для шифрования, и для цифровой подписи.
- ECC (Elliptic curve cryptography) – использует алгебраическую систему, которая описывается в терминах точек эллиптических

кривых, для реализации асимметричного алгоритма шифрования.

- Схема Эль-Гамаля (Elgamal) – вариант Диффи-Хеллмана, который может быть использован как для шифрования, так и для электронной подписи.

Электронная подпись (ЭП) – это особый реквизит документа, который позволяет установить отсутствие искажения информации в электронном документе с момента формирования ЭП и подтвердить принадлежность ЭП владельцу. Значение реквизита получается в результате криптографического преобразования информации.

Сертификат электронной подписи – документ, который подтверждает принадлежность открытого ключа (ключа проверки) ЭП владельцу сертификата. Выдаются сертификаты удостоверяющими центрами (УЦ) или их доверенными представителями.

Владелец сертификата ЭП – физическое лицо, на чье имя выдан сертификат ЭП в удостоверяющем центре. У каждого владельца сертификата на руках два ключа ЭП: закрытый и открытый.

Закрытый ключ электронной подписи (ключ ЭП) позволяет генерировать электронную подпись и подписывать электронный документ. Владелец сертификат обязан в тайне хранить свой закрытый ключ.

Согласно Федеральному закону №63-ФЗ «Об электронной подписи», имеет место деление на[23]:

- простую электронную подпись;
- усиленную неквалифицированную электронную подпись;
- усиленную квалифицированную электронную подпись.

Простая электронная подпись посредством использования кодов, паролей или иных средств подтверждает факт формирования ЭП определенным лицом.

К алгоритмам (схемам) использованием открытого ключа для создания электронной подписи относятся:

- DSA (Digital Signature Algorithm) – основан на вычислительной сложности взятия логарифмов в конечных полях.
- GHR (аббревиатура от фамилий Gennaro, Halevi и Rabin) – схема цифровой подписи с открытым ключом.
- Схема Шнорра (schnorr scheme) – безопасность алгоритма основывается на трудности вычисления дискретных логарифмов.

Уровень защиты всей системы определяется степенью защиты ее самого уязвимого звена, которым, как правило, являются включенные в сеть персональные (клиентские) компьютеры [18].

В данном параграфе были изложены основные теоретические сведенья о приложении с архитектурой клиент-сервера и алгоритмы шифрования т.к проблема защиты всегда носила реактивный характер: хакеры находили «дыры» в системе защиты, а компании спешно искали средства, чтобы закрыть обнаружившиеся бреши. Если поначалу понятие «хакер» ассоциировалось с образом способного молодого человека, который «шарит» по чужим сетям ради любопытства, то сегодня пиратский бизнес поставлен на профессиональную основу и им занимаются вполне взрослые и квалифицированные люди. Понятно, что и интерес у них к чужой информации исключительно коммерческий. Существуют хакерские электронные доски объявлений, где фигурируют расценки на добычу секретной информации, идет обмен известными (хакерам) паролями и кодами доступа к чужим информационным системам.

1.2 Особенности разработки клиент-серверных приложений на языке Java

В Java есть несколько способов взаимодействовать с сетью.

Пакет IO появился самым первым и долгое время оставался единственным возможным подходом к обеспечению сетевого взаимодействия. Суть его заключается в том, что со стороны сервера и клиента создается сокет

(со стороны клиента явно, со стороны сервера – в ответ на запрос соединения со стороны клиента). С каждым сокетом связаны два потока (InputStream и OutputStream): один служит для отправки сообщений, другой – для их приема. При этом все операции с данными потоками являются блокирующими, т.е. исполнение команд прерывается на время выполнения данных операций. Предположим, нам необходимо разработать индексатор сайтов (Web Crowler), являющийся частью поискового робота. Обращение к одному сайту занимает 300 мс. Если нужно проиндексировать 1000 сайтов, то в блокирующем режиме это займет 300 х 1000 мс, т.е. 300 секунд.

Путем ускорения работы приложений при использовании IO является многопоточность – задача делится на части, каждая из которых выполняется своим потоком управления. Например, чтобы обойти 1000 сайтов можно разбить их на четыре группы по 250 сайтов и обойти сайты из каждой группы параллельно. Самой популярной моделью взаимодействия является создание отдельного потока на каждое подключение. Таким образом все соединения обслуживаются параллельно и не мешают друг-другу. Недостатком такого подхода является линейный рост числа потоков и объема требуемой памяти при увеличении числа обслуживаемых соединений. Если потоков будет создано очень много (десятки тысяч), то операционная система сервера будет занята переключением контекстов потоков, а на полезную работу времени может не остаться.

Для решения проблем с блокирующим вводом-выводом был придуман механизм, основанный на мультиплексировании каналов. Данный механизм появился в Java 1.4.2 и был назван New IO (NIO). Суть механизма в следующем: существует мультиплексор (в терминах Java называемый селектором, java.nio.channels.Selector), который в одном потоке, последовательно, производит опрос каналов (в случае сетевого взаимодействия реализуемых классами java.nio.channels.SocketChannel и java.nio.channels.ServerSocketChannel). В результате каждого опроса селектор

возвращает идентификаторы каналов, готовых к выполнению операций ввода-вывода (т.е. канал соединился с удаленной системой и в него теперь можно отправлять запрос или, наоборот, удаленная система что-то записала в канал и из него теперь эти данные можно читать). Такие идентификаторы называются «ключами» (java.nio.channels.SelectionKey). Каждый ключ содержит информацию о том, к выполнению какой операции готов канал. Задача приложения – в цикле обойти все ключи и выполнить соответствующие операции.

При использовании данного подхода решение проблемы линейного роста числа потоков при увеличении числа соединений заключается в том, что все подключенные каналы обслуживаются в одном потоке. Однако, за такое решение приходится платить тем, что выполняемые над полученными в результате сетевого взаимодействия данными операции должны быть очень короткими. Любое блокирование обслуживания одного канала сказывается на всех остальных. Если блокировки длительны, то пока поток управления дойдет до последних готовых каналов, установленные ими соединения могут быть уже разорваны по причине бездействия. Решением данной проблемы может быть так называемая отложенная обработка – на основании принятых в одном NIO-потоке данных формируются команды, которые помещаются в неблокируемую очередь, а исполняются отдельным потоком или несколькими потоками, которые в свою очередь не отвлекаются на операции ввода-вывода.

Java NIO позволяет управлять несколькими каналами (сетевыми соединениями или файлами) используя минимальное число потоков выполнения. Однако ценой такого подхода является более сложный, чем при использовании блокирующих потоков, парсинг данных.

Если разработчику необходимо управлять тысячами открытых соединений одновременно, причем каждое из них передает лишь незначительный объем данных, выбор Java NIO может дать преимущество.

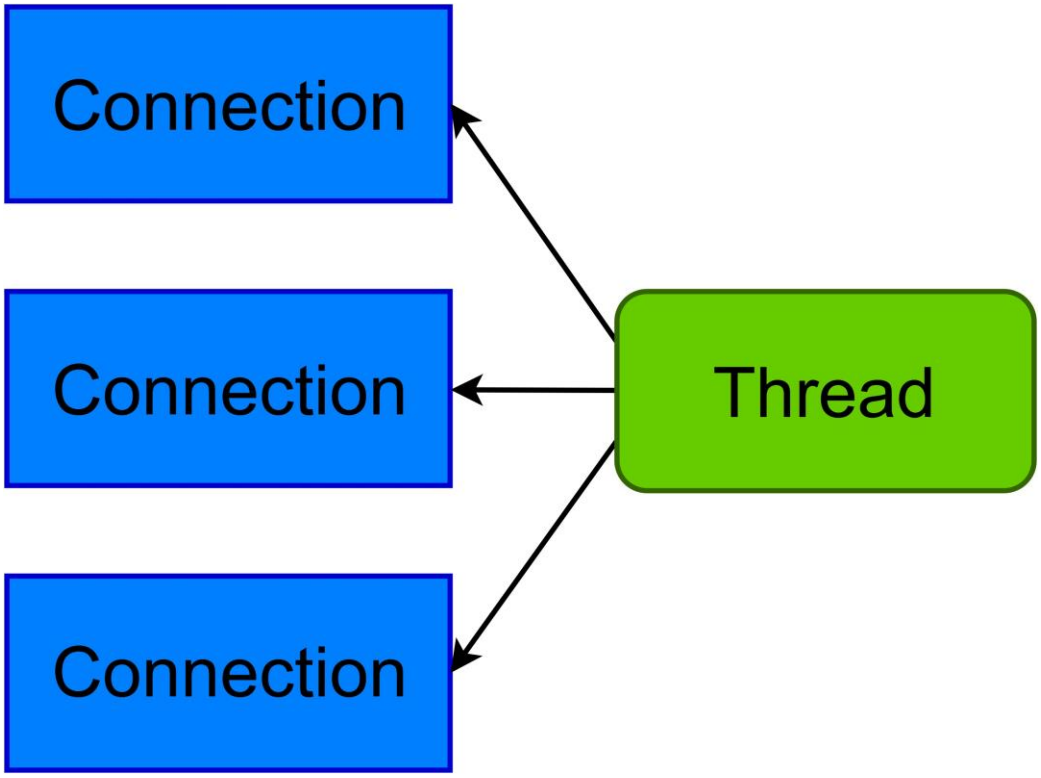


Рисунок 5. структура NIO

Если приложение имеете меньшее количество соединений, по которым передаются большие объемы данных, то лучшим выбором станет классический дизайн системы ввода/вывода:

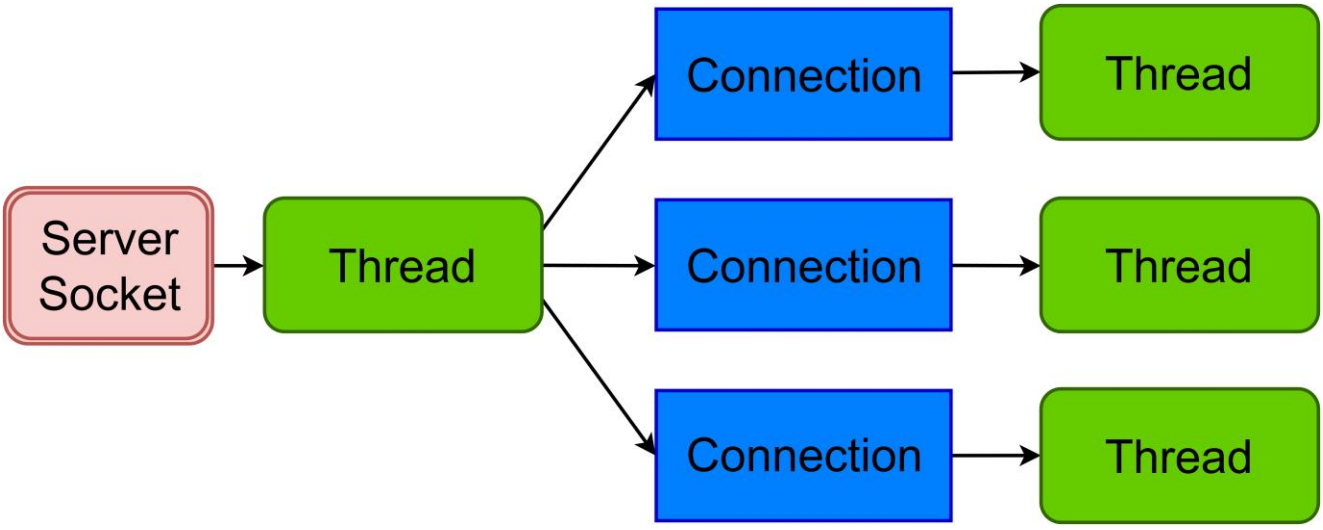


Рисунок 6. Структура IO

Важно понимать, что Java NIO отнюдь не является заменой Java IO. Его стоит рассматривать как усовершенствование – инструмент, позволяющий

значительно расширить возможности по организации ввода/вывода. Грамотное использование мощи обоих подходов позволит вам строить хорошие высокопроизводительные системы.

Язык Java определяет ряд API, охватывающих главные области безопасности:

- Криптографические операции (шифрование, цифровые подписи, обзоры сообщения, и т.д.);
- Генераторы или преобразователи криптографического материала (ключи и параметры алгоритма);
- Объекты (keystores или сертификаты), которые инкапсулируют криптографические данные и могут использоваться в более высоких уровнях абстракции.

JCA (Java Cryptography Architecture) предоставляет API который позволяют разработчикам легко интегрировать механизмы безопасности в свой код программы. JCA в пределах JDK (Java Development Kit) включает два компонента программного обеспечения:

1. Платформа, которая определяет и поддерживает криптографические службы, для который провайдеры предоставляют реализации. Эта платформа включает пакеты такой как java.security, javax.crypto, javax.crypto.spec, и javax.crypto.interfaces.
2. Фактические провайдеры такой как Sun, SunRsaSign, SunJCE, которые содержат фактические криптографические реализации.

Другие криптографические коммуникационные библиотеки, доступные в JDK, используют архитектуру провайдера JCA, но описываются в другом месте. Расширение Защищенного сокета Java™ (JSSE) обеспечивает доступ к Уровню защищенных сокетов (SSL) и Безопасность Транспортного уровня (TLS) реализации. Java Универсальные Службы безопасности (JGSS) (через Kerberos) API, и Простой Уровень Аутентификации и Безопасности (SASL)

может также использоваться для того, чтобы надежно обмениваться сообщениями между связывающимися приложениями [14].

JCA был разработан вокруг этих принципов:

- Независимость реализации и функциональная совместимость;
- Независимость алгоритма и расширяемость.

Независимость алгоритма достигается, определяя универсальный высокоуровневый прикладной программный интерфейс (API) что все использование приложений, чтобы получить доступ к типу службы. Независимость реализации достигается при наличии всех реализаций провайдера, соответствуют четко определенным интерфейсам[14].

API JCA содержит следующие классы:

- SecureRandom: используемый, чтобы генерировать случайные или псевдослучайные числа.
- MessageDigest: используемый, чтобы вычислить обзор сообщения (хеш) указанных данных.
- Signature: initilialized с ключами, они используются, чтобы подписать данные и проверить цифровые подписи.
- Cipher: инициализированный с ключами, они используемые для того, чтобы шифровать/дешифровать данные. Есть различные типы алгоритмов: симметричное шифрование больших объемов данных (например, AES, DES, DESede, Blowfish, IDEA), поточное шифрование (например. RC4), асимметричное шифрование (например. RSA), и основанное на пароле шифрование (PBE).
- Message Authentication Codes (MAC): как MessageDigests, они также генерируют значения хэш-функции, но сначала инициализируются с ключами, чтобы защитить целостность сообщений.

- KeyFactory: используемый, чтобы преобразовать существующие непрозрачные криптографические ключи типа Key в ключевые спецификации (прозрачные представления базового ключевого материала), и наоборот.
- SecretKeyFactory: используемый, чтобы преобразовать существующие непрозрачные криптографические ключи типа SecretKey в ключевые спецификации (прозрачные представления базового ключевого материала), и наоборот. SecretKeyFactorys специализируются KeyFactorys, которые создают секретные (симметричные) ключи только.
- KeyPairGenerator: используемый, чтобы генерировать новую пару открытых и закрытых ключей, подходящих для использования с указанным алгоритмом.
- KeyGenerator: используемый, чтобы генерировать новые секретные ключи для использования с указанным алгоритмом.
- KeyAgreement: используемый двумя или больше сторонами, чтобы согласовать и установить определенный ключ, чтобы использовать для определенной криптографической работы.
- AlgorithmParameters: используемый, чтобы сохранить параметры для определенного алгоритма, включая кодирование параметра и декодирование.
- AlgorithmParameterGenerator: используемый, чтобы генерировать ряд AlgorithmParameters, подходящего для указанного алгоритма.
- KeyStore: используемый, чтобы создать и управлять keystore. keystore является базой данных ключей. У закрытых ключей в keystore есть цепочка сертификата, связанная с ними, который

аутентифицирует соответствующий открытый ключ. keystore также содержит сертификаты от доверяемых объектов.

- CertificateFactory: используемый, чтобы создать сертификаты с открытым ключом и Списки аннулированных сертификатов (CRL).
- CertPathBuilder: используемый, чтобы создать цепочки сертификата (также известный как пути сертификации).
- CertPathValidator: используемый, чтобы проверить цепочек сертификата.
- CertStore: используемый, чтобы получить Certificates и CRLs от репозитария.

Класс Cipher, который является суперклассом всех классов, имеющих отношение к шифрованию, для создания объекта, реализующего алгоритм шифрования, используется метод getInstance().

Первым параметром задаётся имя алгоритма в виде строки, например, «AES» или «AES/CBC/PKCS5Padding». Поддерживаемые алгоритмы шифрования можно посмотреть в документации «Standard Algorithm Name Documentation» на официальном сайте oralce. Второй параметром указывается провайдер.

Криптографические реализации в JDK распределяются через несколько различных провайдеров (Sun, SunJSSE, SunJCE, SunRsaSign) и по историческим причинам и типами предоставленных услуг. Приложения общего назначения не ДОЛЖНЫ запрашивать криптографические службы от определенных провайдеров[14]. Это:

```
Cipher cipher = Cipher.getInstance("...", "SunJCE");  // not recommended

Cipher cipher = Cipher.getInstance("...");      // recommended
```

Иначе, приложения связываются к определенным провайдерам, которые, возможно, не доступны на других реализациях Java. Они также не могли бы быть в состоянии использовать в своих интересах доступных оптимизированных провайдеров (например, аппаратные акселераторы через PKCS11 или собственные реализации ОС, такие как MSCAPI Microsoft), у которых есть более высокий привилегированный порядок чем определенный требуемый провайдер.

Объектно-ориентированный язык Java содержит в себе развивающейся API передачи данных по всемерной сети, так же придает особое значение безопасности, включая безопасность языка, криптографию, инфраструктуру управления открытыми ключами, аутентификацию, безопасную передачу, и управление доступом. Эти качества очень важны для построения клиент-серверных приложений.

1.3 Техническое задание

1. Общие сведения.
   1.1. Название организации-заказчика: УрГПУ ИМИиИТ
   1.2. Название продукта разработки (проектирования): «Chat».
   1.3. Назначение продукта обеспечивать мгновенную связь двух и нескольких пользователей одновременно посредством передачи небольшого количества информации непосредственно друг другу.
   1.4. Плановые сроки начала и окончания работ:
   Начало работы: 16.01.2017.
   Окончание работы: 19.02.2017.
2. Область применения продукта.
   2.1. Процессы и структуры, в которых предполагается использование продукта разработки: посредством интернета данный чат может соединять двух людей в любой точке мира (где есть интернет). Он может применятся для личной переписки двух людей, и для небольшой группы

(предприятие или офис). Количество пользователей определяется техническими характеристиками сервера.

2.2. Характеристика персонала (количество, квалификация, степень готовности): для использования данного продукта требуются минимальные умения работы с ПК.

3. Требования к продукту разработки

3.1. Требования к продукту в целом: продукт должен представлять с собой Java-приложение с пользовательским интерфейсом, написанное для работы с виртуальной машиной Java.

3.1.1. Минимальные аппаратные требования

Клиент:

- Частота процессора: 1 ГГц;
- Количество ядер: 1;
- Разрешение экрана (минимальный размер окна): 400x300;
- Объем оперативной памяти: 128 Мб.
- Объём графической памяти 128 Мб
- Соединение с интернетом.

Сервер:

- Частота процессора: 1 ГГц;
- Количество ядер: 2;
- Разрешение экрана (минимальный размер окна): 300x400 пикселей;
- Объем оперативной памяти: 512 Мб (можно и ниже, но кол-во пользователей, которых можно будет подключить к серверу, будет ограниченным).
- Обьем графической памяти 128 Мб.
- Соединение с интернетом.

3.2. Указание системного программного обеспечения: Java Runtime Environment 7 или выше;

3.3. Указание программного обеспечения, используемого для реализации: Eclipse;

3.4. Особенности реализации серверной и клиентской частей: данные передающееся от клиента к серверу и обратно – это пакеты, состоящие из номера пакета и шифрованного массива байтов, при получении дешифруются и преобразуются в примитивные типы данных.

3.4.1. Форматы входных и выходных данных: входные данные – ip сервера и nickname, выходные данные – данные различного формата (текстовые и графические) от сервера (других клиентов).

3.4.2. Источники данных и порядок их ввода в систему (программу), порядок вывода, хранения: при запуске сервера в верхней части экрана отображается ip сервера, по нему происходит подключение клиентов.

3.5. Меры защиты информации: информация передающееся от клиента к серверу и наоборот, шифруется симметричным алгоритмом блочного шифрования “AES”, принятого в качестве стандарта шифрования правительством США в 2001г.

4. Требования к пользовательскому интерфейсу

4.1. Общая характеристика пользовательского интерфейса: интуитивно-понятный графический интерфейс.

4.2. Размещение информации на экране, дизайн экрана: за пример дизайна брались чаты на популярных ресурсах vk.com, www.twitch.tv.

4.3. Особенности ввода информации пользователем, представление выходных данных: клиент общается посредством текста и графики (смайлы, картинки).

5. Перечень сопроводительной документации: техническое задание.

Глава 2. Разработка клиент-серверного чата на языке Java

2.1 Технология разработки клиент-серверного приложения

Под технологией разработки ПО понимают оптимальный способ ведения разработки, который при определенных условиях обеспечит получение конечного продукта с заранее заданными свойствами.

В ходе исследования был структурирован существующий материал и на его основе создана технология разработки клиент-серверных приложений. Разработка осуществляется по следующему алгоритму:

1. Выбор архитектуры клиент-сервера (техническая часть). В начале разработки необходимо выбрать между двухуровневой и многоуровневой архитектурой. Выбор в данном случае зависит от количества пользователей, которые будут работать в данной системе, стоимости оборудования и дельнейшее обслуживания системы (например, доработка функционала). Подробное описание этих архитектур описаны в первой главе.

2.Выбор языка программирования (программная часть). Клиент-сервер можно написать на самых разных языках. У разных языков присутствуют свои особенности в реализации сетевых задач. Ниже приведен список десяти языков программирования по индексу TIOBE (“TIOBE programming community index” – индекс, оценивающий популярность языков программирования, на основе подсчета результатов поисковых запросов [12]. Для формирования индекса используется поиск в нескольких наиболее посещаемых (по данным Alexa) порталах: Google, Blogger, Wikipedia, YouTube, Baidu, Yahoo!, Bing, Amazon.

Таблица 3. Рейтинг языков программирования май 2017г

Место	Язык программирования	Рейтинг	Изменение %
1	Java	14.639%	-6.32%
2	C	7.002%	-6.22%
3	C++	4.751%	-1.95%
4	Python	3.548%	-0.24%
5	C#	3.457%	-1.02%
6	Visual Basic .NET	3.391%	+1.07%
7	JavaScript	3.071%	+0.73%
8	Assembly language	2.859%	+0.98%
9	PHP	2.693%	-0.30%
10	Perl	2.602%	+0.28%

Стоит учитывать, что Веб-программирование – это частный случай программирования клиент-серверного приложения [11]. Клиентом в веб-приложении выступает браузер, а сервером – веб-сервер что дает ряд плюсов:

- Низкая стоимость внедрения;
- Очень простая поддержка;
- Независимость от Операционной системы;
- Доступность из любой точки мира.

В веб-программировании обычно используют JavaScript на стороне клиента, а на стороне сервера могут быть как один или несколько языков программирования: PHP, Java, C++, C, JavaScript, ASP.NET, Python и другие.

3. Создание рабочего прототипа.

Процесс создания прототипа обычно состоит из шагов:

1. Определение начальных требований;

2. Разработка первого варианта прототипа, который содержит только пользовательский интерфейс системы;
3. Изучение прототипа заказчиком и конечными пользователями, получение обратной связи о необходимых изменениях и дополнениях;
4. Переработка и улучшение прототипа: с учетом полученных замечаний и предложений изменяются как спецификации, так и прототип, после этого шаги 3 и 4 могут повторяться.

Прототипы дают возможность глубже вникнуть в проблему и принять все необходимые проектные решения еще на ранних этапах проектирования [].

4. Тестирование. Для клиент-серверных приложений тестирование можно условно разделить на два уровня:

- Серверный;
- Клиентский.

На первом (серверном) уровне, тестируется взаимодействие выпускаемого программного обеспечения с окружением, в которое оно будет установлено:

1. Аппаратные средства (тип и количество процессоров, объем памяти, характеристики сети / сетевых адаптеров и т.д.);
2. Программные средства (ОС, драйвера и библиотеки, стороннее ПО, влияющее на работу приложения и т.д.);
3. Основной упор здесь делается на тестирование с целью определения оптимальной конфигурации оборудования, удовлетворяющего требуемым характеристикам качества (эффективность, портативность, удобство сопровождения, надежность).

На клиентском уровне, программное обеспечение тестируется с позиции его конечного пользователя и конфигурации его рабочей станции. На этом этапе будут протестированы следующие характеристики: удобство использования, функциональность. Для этого необходимо будет провести ряд тестов с различными конфигурациями рабочих станций:

1. Тип, версия и битность операционной системы (подобный вид тестирования называется кроссплатформенное тестирование);
2. Тип и версия Web барузера, в случае если тестируется Web приложение (подобный вид тестирования называется кросс-браузерное тестирование);
3. Тип и модель видео адаптера (при тестировании игр это очень важно);
4. Работа приложения при различных разрешениях экрана;
5. Версии драйверов, библиотек и т.д. (для JAVA приложений версия JAVA машины очень важна, тоже можно сказать и для .NET приложений касательно версии .NET библиотеки);
6. и т.д.

Уже на начальном этапе становится очевидно, что чем больше требований к работе приложения при различных конфигурациях рабочих станций, тем больше тестов нам необходимо будет провести. В связи с этим, рекомендуем, по возможности, автоматизировать этот процесс, так как именно при конфигурационном тестировании автоматизация реально помогает сэкономить время и ресурсы. Конечно же автоматизированное тестирование не является панацеей, но в данном случае оно окажется очень эффективным помощником.

5. Релиз и дальнейшая поддержка. Стабильная версия программы, готовая к использованию по её назначению, прошедшая тестирования, в которых исправлены основные ошибки, но существует вероятность появления новых, ранее не замеченных, ошибок.

2.2 Разработка клиент-серверного приложения

Для создания клиент-серверного приложения была выбрана двухуровневая архитектура и объектно-ориентированный язык программирования – Java, для написание всей программы. Вся обработка данных происходит на сервере. Двухуровневая архитектура легко реализуема и не требует каких-либо финансовых вложений.

Структурная модель строилась так что даже после релиза можно относительно легко добавлять новый функционал.

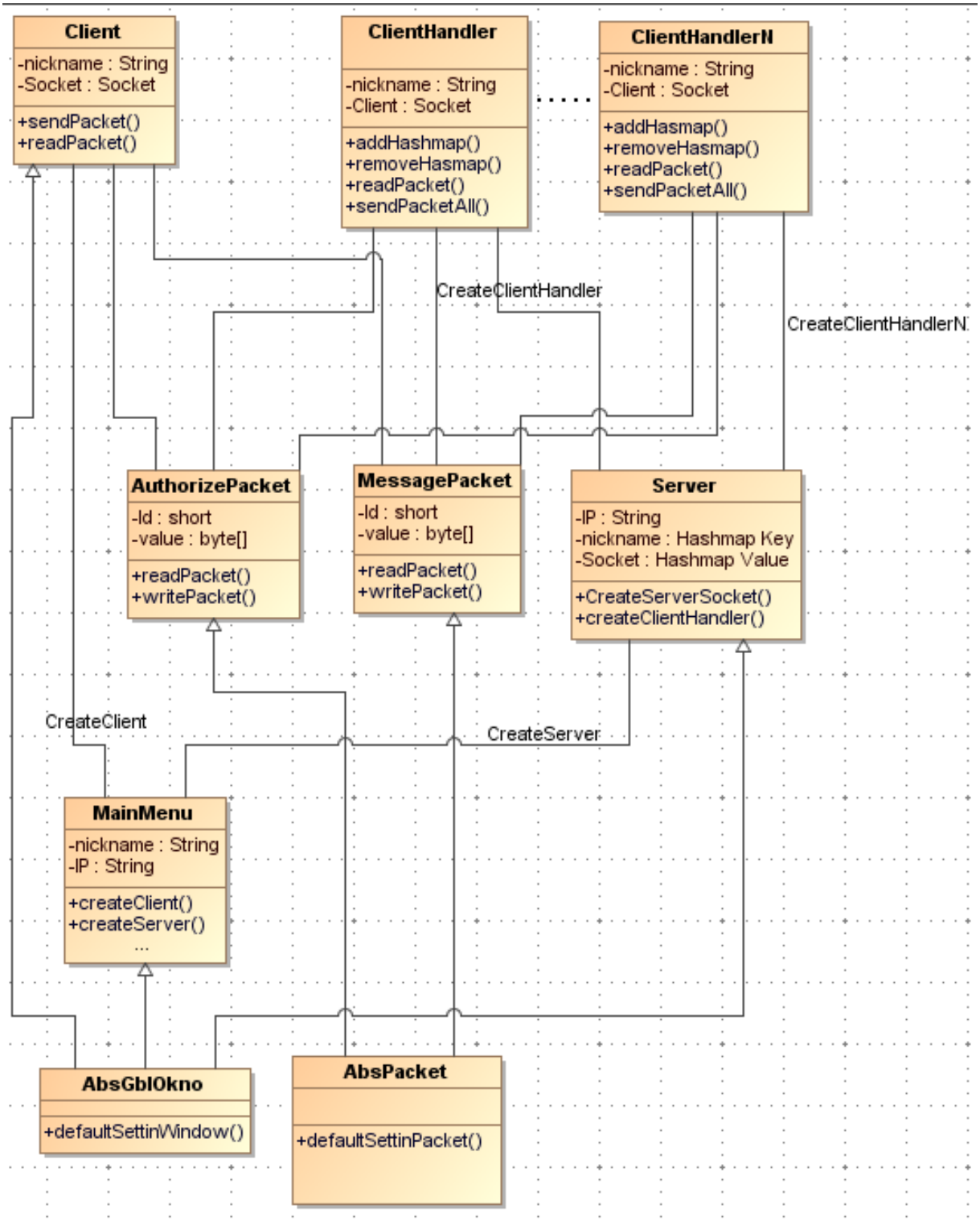


Рисунок 7. Структурная модель многопользовательского приложения «Chat»

Взаимодействие серверного интерфейса и клиентов происходит через потокозащищённый HashMap содержащий всю информацию о клиентах.

Наиболее значимыми являются код взаимодействия клиента и сервера.

```
private void Handling() {
                    while (true) {
                        try {
                            Socket client = server.accept();
                            new ClientHandler(client,KeyAES).start();
//Если клиент успешно установил соединение передаем его обработчику
//клиентов, а также ключ которым шифруются данные.
                        } catch (IOException e1) {
                            e1.printStackTrace();
                        }
                        try {
//после проверки данных в потоке засыпаем на 10 миллисекунд.
                            Thread.sleep(10);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
```

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.security.Key;
import java.security.KeyFactory;
import java.security.spec.X509EncodedKeySpec;
import javax.crypto.Cipher;
```

```
class ClientHandler extends Thread { //на сервере
        private final Socket client;
        private String name = "Неизвестный";
        private String value;
        private Cipher decrypter_AES;
        private Cipher encrypter_AES;
        private Cipher encrypter_RSA;
        private Key KeyAES;

        public ClientHandler(Socket client, Key KeyAES) {
                this.client = client;
                this.KeyAES = KeyAES;
                try {
                        //создаем фабрики шифрования/расшифровки
                        //алгоритмов
                        decrypter_AES = Cipher.getInstance("AES");
                        encrypter_AES = Cipher.getInstance("AES");
                        encrypter_RSA = Cipher.getInstance("RSA");
                        encrypter_AES.init(Cipher.ENCRYPT_MODE, KeyAES);
                        decrypter_AES.init(Cipher.DECRYPT_MODE, KeyAES);
                } catch (Exception e) {
                        System.err.println("Ошибка шифрования");

                }
        }

@Override
public void run() {
```

```
try {
        DataInputStream in = new DataInputStream(client.getInputStream());
        while (true) {
            if (in.available() > 0) { //Если в потоке есть данные читаем их.
                short id = in.readShort(); //id пакета

                switch (id) { // в зависимости он полученного id создаем
```
класс с реализацией чтения и записи данных данного пакета.
```
                case 1: {
                    MesPacket M;
                    int size = in.readInt();
                    int lenByte = in.readInt();
                    byte[] b = new byte[lenByte];
                    in.read(b);
                String message = new String (decrypter_AES.doFinal(b));
                    message = name + " : " + message;

                    if (size > 0) {
                        int[][] arrSmil = new int[size][2];
                        for (int i = 0; i < size; i++) {
                        arrSmil[i][0] = in.readInt()+3+name.length();
                        arrSmil[i][1] = in.readInt();
                                }
        M = new MesPacket(encrypter_AES.doFinal(message.getBytes()), arrSmil);
                                } else
        M = new MesPacket(encrypter_AES.doFinal(message.getBytes()));
                                M.readAll();
                                break;
```

```
                }

                case 3: {
                    AuthorizePacket AP = new AuthorizePacket();
                    AP.read(in);
                        switch (AP.getSost()) {
                            case 1: {
                    Cipher encrypter_RSA = Cipher.getInstance("RSA");
                    KeyFactory kf = KeyFactory.getInstance("RSA");
            encrypter_RSA.init(Cipher.ENCRYPT_MODE,kf.generatePublic(new
X509EncodedKeySpec(AP.getKey())));
            AP.setKey(encrypter_RSA.doFinal(KeyAES.getEncoded()));
                                sendPacket(AP,client);
                                break;
                            }
                            case 2:{
                                name = AP.getName();
                            if (AP.getCheck().equals("ВсеОк")){
                                        value = name+
" IP: "+client.getInetAddress().getLocalHost().getHostAddress().toString();
                                Server.csls.put(value, client);
                    MesPacket M = new
MesPacket(encrypter_AES.doFinal(("<font color=\"#002f55\"><b>"+name + "
вошел в чат"+"</b></font>").getBytes()));
                                    M.readAll();
                                    break;
                            }
                            AP.setSost((short)3);
                            sendPacket(AP,client);
```

```
                                        close();

                                    }
                                    default: {
                                        MesPacket M = new MesPacket(encrypter_AES.doFinal(("<font color=\"#002f55\"><b>"+name + " вышел из чата"+"</b></font>").getBytes()));
                                        M.readAll();
                                        close();
                                        break;
                                    }
                                    }
                                    break;
                                }
                                }
                            } else
                                Thread.sleep(10);
                        }
                    } catch (Exception e) {
                        System.err.println("Отключение"+name);
                        close();
                    }
                }
```

Подключение к серверу (на клиенте):

```
            private void Start() {
                try {
            TextChat.getDocument().insertString(TextChat.getDocument().getLength(), "Идет подключение..." + "\n", null);
                    socket = new Socket(sip, 9322);
```

```
        //обработка в случае успешного соединения с сервером.
                        handle();
                } catch (Exception e) {
                        try {
                TextChat.getDocument().insertString(TextChat.getDocument().getLengt
h(), "Не удалось подключиться к серверу, закройте приложение" + "\n", null);
                        } catch (BadLocationException e1) {
                                System.exit(0);
                        }
                }
        }

        public static void sendPacket (AbsPacket P,Socket s){
                DataOutputStream out;
                try {
                        out = new DataOutputStream(s.getOutputStream());
                        out.writeShort(P.GetId()); //считываем id пакета и пишем
                        P.write(out);
                        out.flush();
                } catch (IOException e) {
                        try {
                                s.close();
                        } catch (IOException e1) {
                        }
                        System.out.println("Закрытие потока");
                }
        }

        public void close() {
```

```
                Server.csls.remove(value);
                interrupt();
            }
        }
        public void readPacket() { //на клиенте
                try {
                    short id = in.readShort();
                    switch (id) {
                    case 1: {
                        try {
                            int size = in.readInt();
                            int lenByte = in.readInt();
                            byte[] b = new byte[lenByte];
                            in.read(b);
                            int sh = 0;
                        String a = new String (decrypter_AES.doFinal(b));
                        StringBuffer buf = new StringBuffer(a);
                HTMLEditorKit kit = (HTMLEditorKit) TextChat.getEditorKit();
        if (size > 0)
            for (int i = 0; i < size; i++) {
            int C = in.readInt() + sh;
            int S = in.readInt();

            buf.replace(C - 1, C, "<img src='file:Smilies/" + listPathIm[S] + "'>");
            System.out.println("buf = " + buf);
            sh += 24 + listPathIm[S].length();
        }
                                        kit.insertHTML((HTMLDocument)
TextChat.getDocument(), TextChat.getDocument().getLength(),
```

```
"[" + formating.format(System.currentTimeMillis()) + "] " + buf, 0, 0, null);
                                panel.updateUI();
                            } catch (BadLocationException e) {
                            }
                            break;
                        }
                        case 3: {
                            AuthorizePacket AP = new AuthorizePacket();
                            AP.read(in);
                            if (AP.getSost() == 1) {
                            Cipher decriptCipher = Cipher.getInstance("RSA");
                    decriptCipher.init(Cipher.DECRYPT_MODE, kP.getPrivate());
                                                    SecretKey Key = new
SecretKeySpec(decriptCipher.doFinal(AP.getKey()), "AES");
                        encrypter_AES.init(Cipher.ENCRYPT_MODE, Key);
                        decrypter_AES.init(Cipher.DECRYPT_MODE, Key);
                                AP.setSost((short) 2);
                                AP.setName(name);
                                sendPacket(AP);
                            } else {
                TextChat.getDocument().insertString(TextChat.getDocument().getLengt
h(), "Ошибка аутентификации, закройте приложение" + "\n", null);
                            }
                        }
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
        }
```

```
        private void handle() { //на клиенте
                try {
                        out = new DataOutputStream(socket.getOutputStream());
                        in = new DataInputStream(socket.getInputStream());
    sendPacket(new AuthorizePacket(kP.getPublic().getEncoded(),(short)1));
                        while (true) {
                                if (in.available() > 0)
                                        readPacket();
                                else
                                        try {
                                                Thread.sleep(10);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                        }
                } catch (Exception e) {
                        System.err.println("Ошибка чтения/передачи");
                        close();
                        e.printStackTrace();
                }
        }
```

Каждый пакет данных пересылающийся клиентом или сервером присваивается свой id. Это позволяет структурировать разного рода, типа информацию.

Java, строгий язык на котором можно решать в принципе любую задачу, благодаря кроссплатформенности может работать на разных платформах что очень важно для клиентской части.

Алгоритм работы системы программы выглядит следующим образом:

1. Сервер подключается к порту на хосте и ждет соединения с клиентом;
2. Клиент создает сокет и пытается соединить его с портом на хосте;
3. Если создание сокета прошло успешно, то сервер переходит в режим ожидания команд от клиента;
4. Клиент формирует команду и передает ее серверу, переходит в режим ожидания ответа;
5. Сервер принимает команду, выполняет ее и пересылает ответ клиенту;
6. Клиент обрабатывает полученную информацию и выводит её пользователю в удобном виде, для её восприятия.
7. Пока клиент или сервер не разорвет соединение смотреть пункт 4.

Приложение объединяет в себе два режима:

- Режим сервера: Запускает сервер на порту 9322. Представляет с собой окно со списком подключившихся к нему клиентов. В верхней части окна отображается IP адрес по которому клиенты имеют возможность подключиться;
- Режим клиента: открывается окно в котором нужно ввести IP сервера и “nickname” который должен состоять от 5 до 15 латинских, русских символов или цифр.

При создании сервера генерируется симметричный ключ алгоритмом AES этим ключом сервер шифрует всю информацию, передаваемую клиентам. Для получения этого ключа клиент при подключении к серверу генерирует у себя парный ключ (публичный и приватный) алгоритмом RSA.

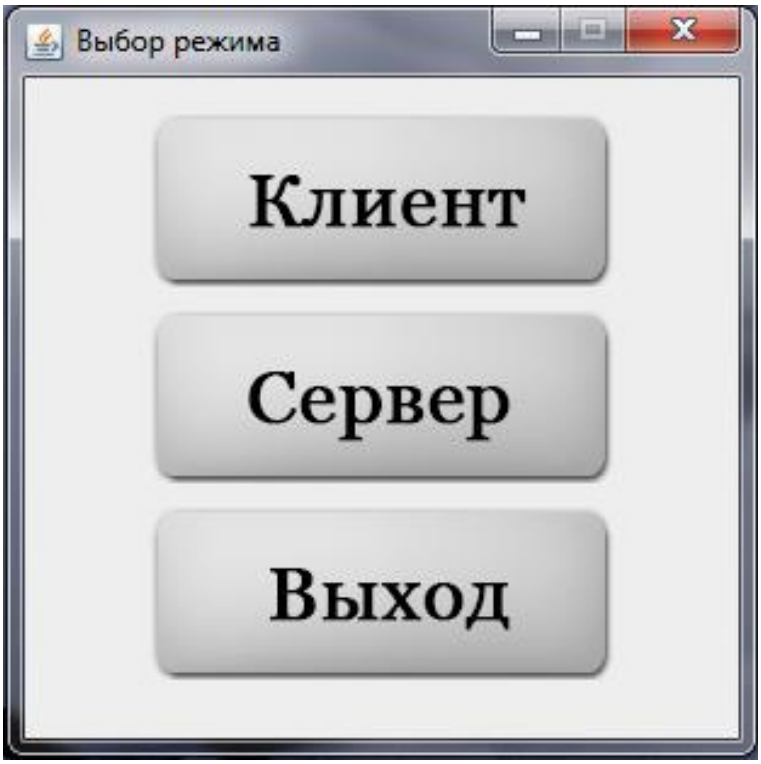


Рисунок 8. Главное меню

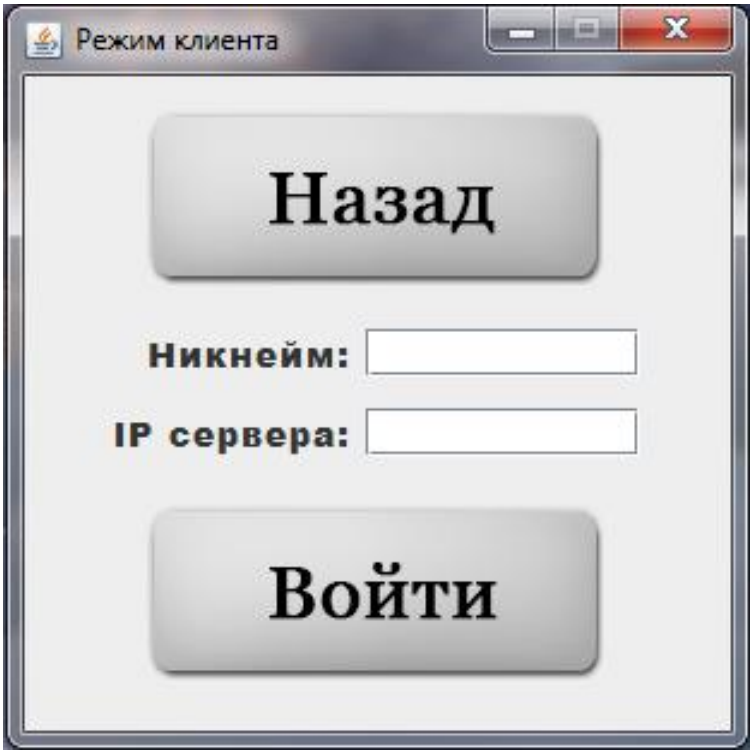


Рисунок 9. Подключение к серверу

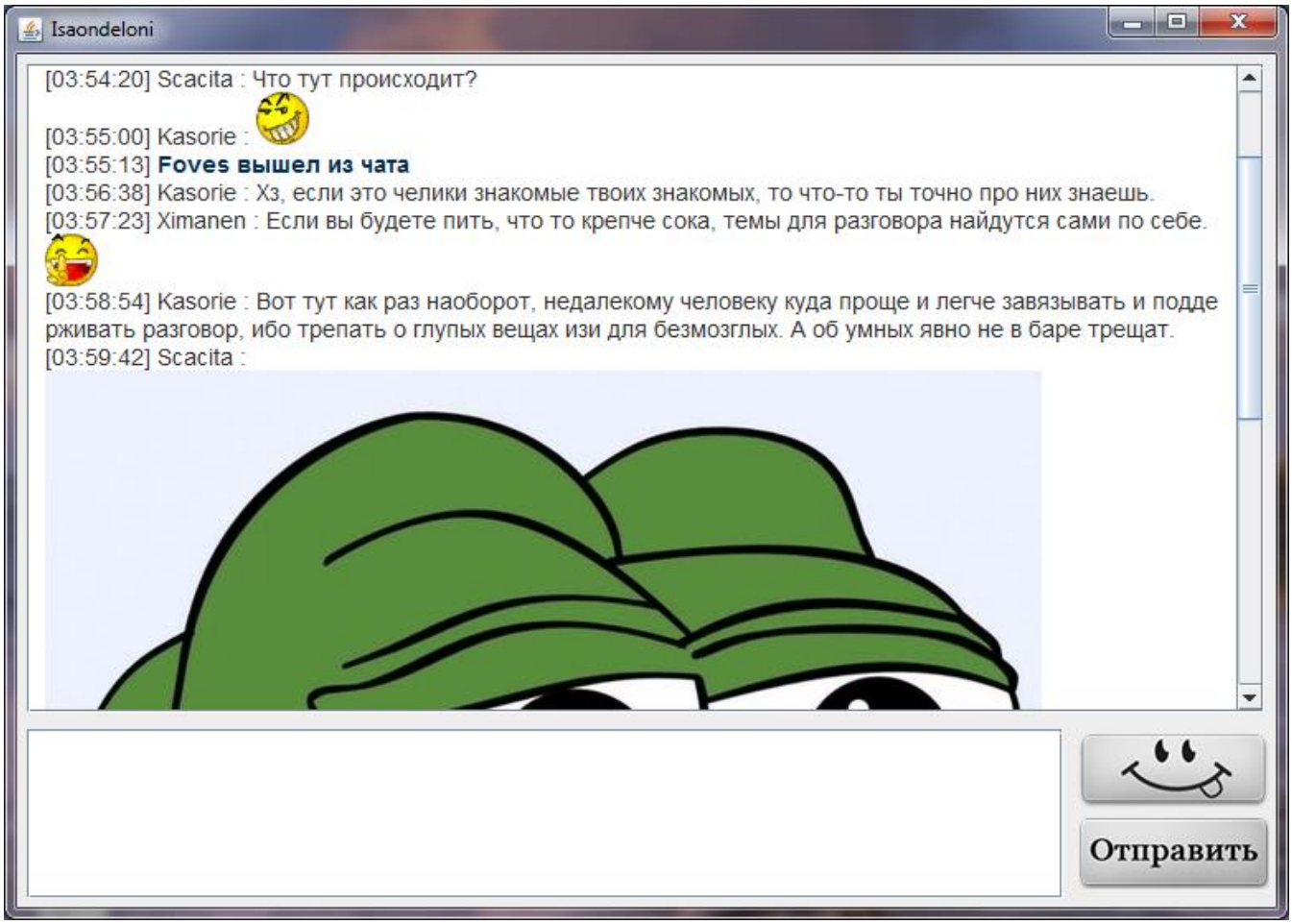


Рисунок 10. Режим клиента

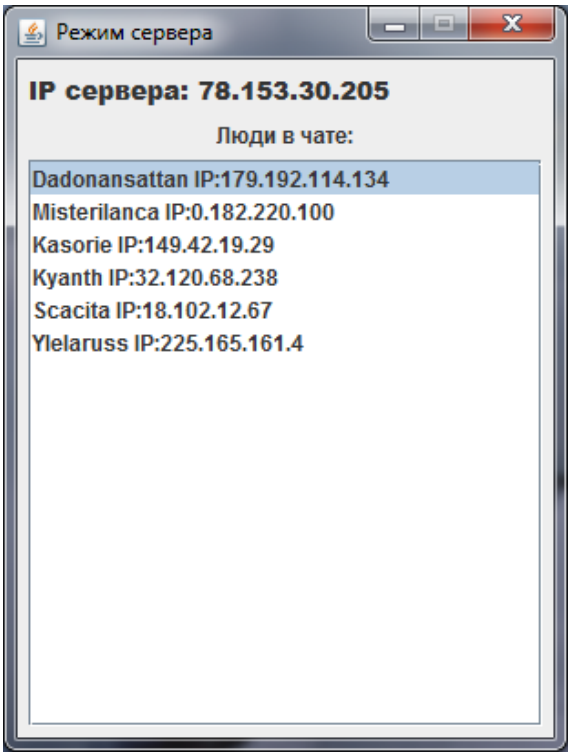


Рисунок 11. Режим сервера

Разработанное приложение работает везде где можно установить Java Runtime Environment 7 или выше. Имеет хорошую степень защиты информации т.к использует алгоритм AES, принятый в качестве стандарта шифрования правительством США в 2001 году.

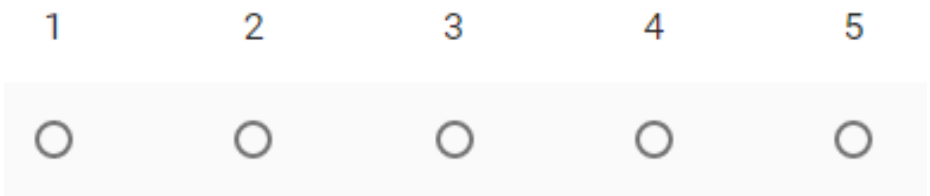
2.3 Результаты апробации

Для реализации апробации было проведено тестирование приложения в компании «ТехНик». Для выявления ошибок была создана анкета с вопросами, которая заполнялась после тестирования приложения.

Анкета 1.

Результаты тестирования приложения

1. ОС с который вы запускали данное приложение:
    a. Windows
    b. Linux
    c. Mac OS X
    d. Другое:_____
2. Были ошибки во время работы приложения, если да, то какие?
    Развернутый ответ:_____
3. Удобный интерфейс?
    a. Да
    b. Нет
4. Оцените приложение по шкале от 1 до 5



5. Пожелания для улучшения дизайна/работы приложения
    Развернутый ответ:_____

Результаты тестирования показали следующее:

1. Игра была установлена и работала под разными версиями операционных систем (Рисунок 14);
2. Ошибки во время работы приложения небыли обнаружены;
3. Интерфейс посчитали удобным 95,5% тестируемых (Рисунок 12);

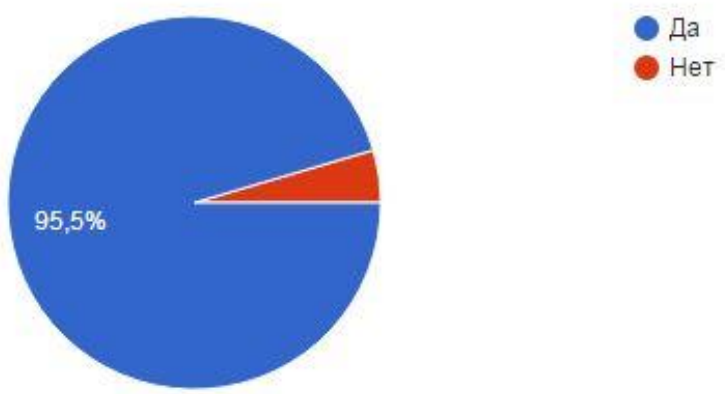


Рисунок 12. Оценка удобства интерфейса

4. Средняя оценка приложения составила: 4,59 (Рисунок 13);

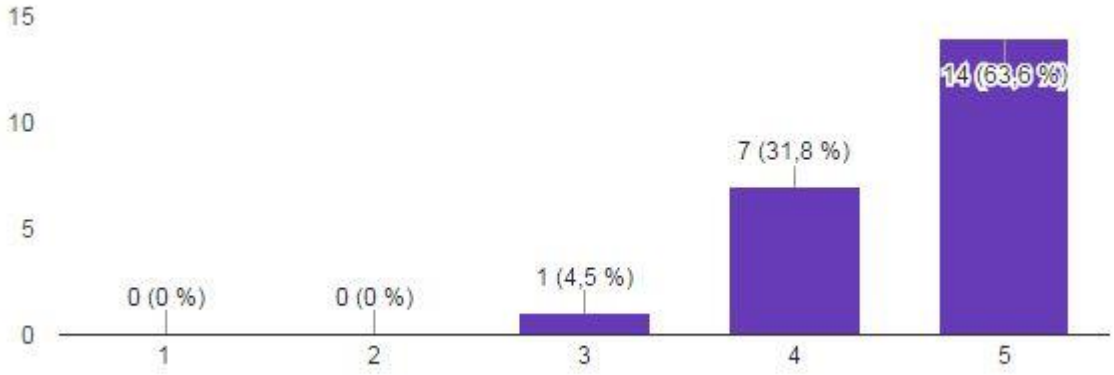


Рисунок 13. Оценка работоспособности и устойчивости приложения

5. Все пожелания были учтены, приложение доработано за несколько дней.

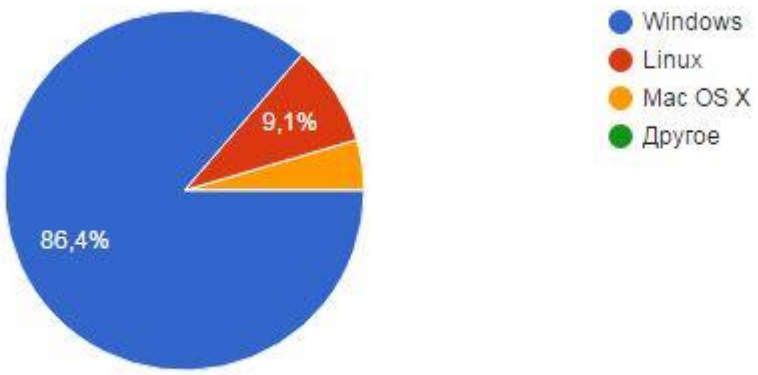


Рисунок 14. Платформы, на которых протестирована работа приложения

По результатам анкетирования критических багов не обнаружено. Найденные ошибки были исправлены в следующей версии, как пожелания и замечания тестировщиков.

Заключение

В результате исследования был получен материал, анализ которого позволил заключить, объектно-ориентированный язык программирования является наиболее популярным среди разработчиков.

В ходе исследования:

1. Проанализированы особенности написания клиент серверных приложений на языке Java, такие как:
    - Разные способы реализации потоков ввода и вывода информации, её передача посредством компьютерных сетей;
    - Защита и алгоритмы шифрования данных.
2. Произведено сравнение различных архитектур клиент-серверных приложений, а именно:
    - Рассмотрена модель и принципы построения данных архитектур;
    - Выделены три основных компонента которые должна включать в себя каждая архитектура;
    - Описаны плюсы и минусы данных архитектур.
3. Рассмотрены разные языки программирования для написания клиент-серверных приложений. После тщательного анализа программной реализации и популярности языков программирования среди программистов был выбран язык Java для создания клиент-серверного приложения.
4. Предложена технология создания клиент-серверных приложений.
5. Разработано клиент-серверное приложение «Chat»;
6. Проведена апробация разработанного клиент-серверного приложения в форме тестирования. Все замечания были учтены и доработаны в последующих версиях.

Библиографический список

1. Владимир Шаньгин, Защита информации в компьютерных системах и сетях, Москва 2012 – 595 с.
2. Вязовик Н., Жилин Е. Программирование на Java. Методическое руководство для преподавателей Текст. / Николай Вязовик ; Евгений Жилин. Москва: Центр Sun технологий МФТИ, 2003 г.
3. Григорьев Ю.А., Плутенко А.Д. Жизненный цикл проектирования распределенных баз данных. Благовещенск: изд-во Амурского гос. ун-та, 1999. – 266 с.
4. Герберт Шилдт. " Java 8. Полное руководство 9-е издание" 2015г
5. Х. М. Дейтел, П. Дж. Дейтел, С. И. Сантри. Технологии программирования на Java 2. Распределенные приложения. 2011г – 464г.
6. Домарев В.В. "Безопасность информационных технологий. Методология создания систем защиты" – К.: ООО "ТИД "ДС", 2002 – 688 с.
7. Г. Ладыженский. «Технология клиент-сервер и мониторы транзакций» – Открытые системы, Лето 1994.
8. Кей Хорстманн, Гари Корнелл "Java. Библиотека профессионала. Том 1 и 2". 9-е издание 2014г.
9. Пособие. Распределённые системы. Архитектура клиент-сервер URL: https://edu.vsu.ru/mod/book/view.php?id=12520 (дата обращения: 21.05.2017).
10. The world of Tomorrow // Hans Goedvolk, 1995г, URL: http://home.kpn.nl/daanrijsenbrij/vision/eng/vish4ex1.htm (дата обращения: 21.05.2017).
11. Веб-программирование // Олег Большаков, 31.7.2012г. URL: http://codingcraft.ru/web-programming.php (дата обращения: 21.05.2017).

12. TIOBE programming community index, May 2017.
URL: https://www.tiobe.com/tiobe-index/ (дата обращения: 21.05.2017).
13. Криптографическая защита информации : учебное пособие / А.В.Яковлев, А.А. Безбогов, В.В. Родин, В.Н. Шамкин. – Тамбов: Изд-во Тамб. гос. техн. ун-та, 2006. – 140 с.
14. Официальный сайт Oracle Платформа Java™, Standard Edition 8 Спецификация API URL: https://docs.oracle.com/javase/8/docs/api/
15. Спецификации и модели, справочник по MySQL
URL: http://www.webmaster.ee/MySQL/Chapter%208/1.htm (дата обращения: 21.05.2017).
16. Технические заметки IO vs NIO, Основные отличия.
URL: https://rayhon.gitbooks.io/tech-blog/content/nio.html (дата обращения: 21.05.2017).
17. Компоненты сетевого приложения. Клиент-серверное взаимодействие и роли серверов. // Анатольев А.Г., 06.12.2013.
URL: http://www.4stud.info/networking/lecture5.html (дата обращения: 19.09.2007).
18. Защита информации в архитектурах клиент/сервер
URL: http://www.infocity.kiev.ua/hack/content/hack054.phtml (дата обращения: 21.05.2017).
19. Фоксфорд. Информатика. Асимметричное шифрование. URL: https://foxford.ru/wiki/informatika/asimmetrichnoe-shifrovanie (дата обращения 11.06.2017).
20. Теория построения сетей – Технология клиент-сервер 13.1.2011г
URL: http://xnets.ru/plugins/content/content.php?content.217.6 (дата обращения: 21.05.2017).
21. Модели взаимодействия клиент-сервер // Антонов Кирилл, Июль 28, 2016. URL: http://zametkinapolyah.ru/servera-i-protokoly/o-modeli-

vzaimodejstviya-klient-server-prostymi-slovami-arxitektura-klient-server-s-primerami.html (дата обращения: 19.09.2007).

22. Поддержка разработки распределенных приложений в Microsoft .NET Framework // Сергей Горин, Всеволод Крищенко, 2006г URL: http://www.intuit.ru/studies/courses/1115/177/lecture/4778 (дата обращения: 19.09.2007).
23. Журнал о системах электронного документооборота URL: http://ecm-journal.ru/e-sign (дата обращения: 09.06.2017).
24. Обеспечение безопасности транзакций в среде клиент-сервер // Владимир Максимов, 1998г URL: http://www.interface.ru/magazine/tcs/Archive/198/maks.htm (дата обращения: 19.09.2007).