Оглавление

[Введение 3](#_Toc275326245)

[1. Исследовательская часть 5](#_Toc275326246)

[1.1. Анализ существующих систем 6](#_Toc275326247)

[1.1.1. ICQ 6](#_Toc275326248)

[1.1.2. Windows Live Messenger 6](#_Toc275326249)

[1.1.3. Skype 7](#_Toc275326250)

[1.1.4. Google Talk 7](#_Toc275326251)

[1.1.5. IRC 7](#_Toc275326252)

[1.1.6. Веб-чаты 8](#_Toc275326253)

[1.2. Подведение итогов обзорного анализа 9](#_Toc275326254)

[1.3. Вывод 10](#_Toc275326255)

[2. Конструкторская часть 12](#_Toc275326256)

[2.1. Терминология 12](#_Toc275326257)

[2.2. Общая структура системы 13](#_Toc275326258)

[2.3. Обзор протокола ENCP 14](#_Toc275326259)

[2.4. Структура клиентской программы 16](#_Toc275326260)

[2.5. Структура серверной программы 18](#_Toc275326261)

[2.6. Спецификация протокола ENCP 19](#_Toc275326262)

[2.7. Каналы управления 26](#_Toc275326263)

[2.7.1. Подключение нового участника конференции 26](#_Toc275326264)

[2.7.2. Обмен сообщениями 28](#_Toc275326265)

[2.7.3. Синхронизация клиентского списка 29](#_Toc275326266)

[3. Технологическая часть 31](#_Toc275326267)

[3.1. Выбор языка и среды программирования 31](#_Toc275326268)

[3.2. Дополнительные инструменты 31](#_Toc275326269)

[3.3. Обзор файлов исходного кода серверной части системы 32](#_Toc275326270)

[3.3.1. Модуль user.py 32](#_Toc275326271)

[3.3.2. Модуль utils.py 32](#_Toc275326272)

[3.3.3. Модуль websocket.py 32](#_Toc275326273)

[3.3.4. Модуль wsthread.py 32](#_Toc275326274)

[3.4. Обзор файлов исходного кода клиентской части системы 33](#_Toc275326275)

[3.4.1. Файл protocol.js 33](#_Toc275326276)

[3.4.2. Файл ui.js 33](#_Toc275326277)

[3.4.3. Файл utils.js 33](#_Toc275326278)

[3.4.4. Файл drawning.js 33](#_Toc275326279)

[3.4.5. Файл client.html 33](#_Toc275326280)

[3.4.6. Файл chat.css 33](#_Toc275326281)

[3.4.7. Файлы orange\_style.css, grey\_style.css, blue\_style.css 33](#_Toc275326282)

[3.5. Копии экранов системы 34](#_Toc275326283)

[4. Заключение 39](#_Toc275326284)

[5. Список литературы 40](#_Toc275326285)

[Приложение А 41](#_Toc275326286)

[1. Выборочный листинг исходного кода серверной части системы 41](#_Toc275326287)

[1.1. Файл utils.py 41](#_Toc275326288)

[1.2. Файл websocket.py 41](#_Toc275326289)

[1.3. Модуль wsthread.py 42](#_Toc275326290)

[2. Листинг исходного кода клиентской части системы 45](#_Toc275326291)

[2.1. Файл protocol.js 45](#_Toc275326292)

# Введение

Среди множества применений вычислительных сетей можно выделить *сетевое общение*. Электронная почта прочно вошла в современную жизнь и является одним из важнейших средств делового общения; растет число людей, пользующихся системами видеоконференций через Интернет; существует огромное количество интернет-форумов, эхоконференций и групп новостей по различным тематикам.

Средства сетевого общения можно разделить на две большие группы - *асинхронные* и *синхронные*. Асинхронные средства не требуют присутствия всех участников общения непосредственно в момент передачи сообщения; адресат может ознакомиться с сообщением в любое удобное для него время. К асинхронным средствам относятся интернет-форумы, новостные группы, электронная почта. Синхронные средства сетевого общения (обычно именуемые чатами или онлайн-конференциями), напротив, предполагают, что участники конференции общаются одновременно, и отправленное одним участником сообщение немедленно становится доступным другим участникам.

Трудно придумать что-либо новое в области асинхронного сетевого общения: в самом деле, в таком режиме общения можно передавать информацию практически любого вида. Другое дело - синхронное общение. Средства проведения онлайн-конференций постоянно совершенствуются; разработчики стремятся сделать сетевое общение доступнее, ближе к живому общению.

Целью данного курсового проекта является анализ существующих систем текстовых онлайн-конференций, разработка собственного протокола онлайн-конференции и реализация этого протокола в конкретном программном продукте.

# Исследовательская часть

Многопользовательские сетевые конференции появились достаточно давно (так, протокол IRC был создан в 1988 году), и к настоящему времени существует множество реализаций, с различными протоколами и функциональностью. Наиболее популярные системы общения включают:

* **Windows Live Messenger** (бывший MSN Messenger)
* **Skype**
* **Google Talk**
* **IRC** (Internet Relay Chat)
* **Веб-чаты**
* **Веб-чаты**
* и т.д

Общая функциональность, присутствующая в любой реализации онлайн-чата, включает:

* Список пользователей, отображающий всех людей, в данный момент подключенных к серверу;
* Возможность отправлять текстовые сообщения, видимые как всем пользователям, так и одному выбранному собеседнику (личные сообщения);
* Ведение истории сообщений, полученных в текущей сессии, и ее отображение в окне клиентской программы

Используя информацию, представленную в **[1]**, проведем обзорный анализ перечисленных выше реализаций.

## Анализ существующих систем

### ICQ

Являясь одной из самых популярной программ для моментального обмена сообщениями через Интернет (службой активно пользуется около 15 миллионов человек, а зарегистрированных учётных записей около 400 миллионов)[[1]](#footnote-2), программа **ICQ** реализует закрытый фирменный протокол **OSCAR**. Протокол обеспечивает такую функциональность, как многопользовательские беседы, передачу файлов и онлайн-игры. Хотя фирма-владелец **ICQ** старается пресечь попытки создать альтернативные клиенты для своего протокола, в частности, постоянно внося в него изменения, существует ряд альтернативных клиентов (программы **QIP**, **Miranda IM**), которые чуть ли не более популярны, чем официальный клиент (особенно на территории России и стран СНГ).

### Windows Live Messenger

Данный продукт является клиентским приложением для протокола **Microsoft Notificaion Protocol (MSNP)** от компании **Microsoft**. Приложение поддерживает как стандартный дуплексный режим общения двух пользователей, так и возможность объединения пользователей в группы и проведения многопользовательской текстовой конференции.

Одним из плюсов используемого протокола является возможность передачи нарисованных в области ввода текста изображений, что существенно облегчает общение, если необходимо показать собеседнику какую-нибудь формулу или схему. Кроме того, **MSNP**, в отличие от **OSCAR**, является открытым протоколом, что позволяет создавать для него альтернативные клиенты.

### Skype

**Skype** – приложение и протокол, предназначенные прежде всего для IP-телефонии. **Skype** очень популярен в России, так как  поддерживает возможность бесплатного видео- и голосового общения через Интернет. **Skype** поддерживает возможность создания многопользовательской конференции (аудио, видео, текст), но при этом кол-во пользователей конференции ограниченно числом 25.

Протокол **Skype** является закрытым, и исходный код официального клиента не публикуется.

### Google Talk

Программа мгновенного обмена сообщениями, разработанная компанией **Google**.

**Google Talk** позволяет общаться с помощью голосового чата и текстовых сообщений. Особенностью является тесная интеграция с почтовой службой **Gmail** (например, по **Google Talk** приходят уведомления о новых сообщениях). Для использования **Google** **Talk** обязательно наличие учётной записи **Gmail**.

**Google Talk** основан на открытом протоколе **XMPP**, но исходный код официального клиента не публикуется.

### IRC

**IRC** *(Internet Relay Chat — ретранслируемый интернет-чат)* — одна из старейших систем многопользовательских текстовых конференций (появилась в 1988 г.)

**IRC** предоставляет возможность как группового, так и личного общения. Для группового чата в **IRC** предназначены каналы, на которых пользователи могут собираться и вести общение.

Система **IRC** очень популярна, но ориентирована исключительно на текстовое общение.

Протокол **IRC** является открытым и описан в **RFC 1459**. Сам протокол содержит множество ограничений: длина сообщений ограничена 510 символами; текст передается в 8-битной кодировке (стандарт не определяет способ передачи текста в многобайтной кодировке); отсутствует поддержка передачи файлов и т.п. Подобная отсутствующая функциональность реализуется клиентскими приложениями поверх протокола.

IRC, в отличие от перечисленных выше систем общения, не требует регистрации пользователя в системе; при подключении к серверу достаточно ввести псевдоним (ник).

### Веб-чаты

Описанные выше протоколы многопользовательских конференций реализуются в клиентских приложениях, устанавливаемых на компьютер пользователя. Существуют также так называемые веб-чаты, для участия в которых необходимо лишь наличие веб-браузера. Взаимодействие с сервером в таких чатах основано на протоколе HTTP.

Первоначально веб-чаты представляли собой страницу с разговором, содержащую форму, посредством которой введённый текст отсылался на сервер. Сервер добавлял новые сообщения в текстовую область, удалял старые и обновлял файл; сама страница регулярно перезагружалась. Такой чат осуществлялся с заметной задержкой в десятки секунд, из-за того, что сервер не мог сам сообщить клиенту об изменениях — клиент мог только запрашивать данные сам с некоторой периодичностью, обновляя страницу целиком или - позднее - отправляя асинхронные HTTP-запросы.  Другие системы чатов не имели такого недостатка, поскольку клиентские приложения могли устанавливать дуплексное соединение с сервером.

Веб-чаты завоевали популярность, отчасти благодаря отсутствию необходимости установки специального клиентского приложения на компьютер пользователя. Стоит отметить, что различные веб-чаты не совместимы между собой, так как не существует общего стандарта на реализацию протокола веб-чата поверх HTTP.

Большинство веб-чатов не требуют регистрации пользователя в системе.

К веб-чатам можно отнести продукт **Google Wave**, работающий в веб-браузере. **Wave** - экспериментальное средство общения, призваное соединить в себе функции электронной почты, мгновенной передачи сообщений, чата, веб-форума, вики, совместного создания и редактирования документов с системой управления версиями и социальной организацией сети. Компания **Google** позиционировала свой продукт как "убийцу электронной почты", чем несколько дезориентировала публику; в результате **Wave** так и не получил широкого применения.

## Подведение итогов обзорного анализа

Проведенный обзорный анализ позволяет составить следующий перечень функций, которые могут присутствовать в реализациях многопользовательских конференций:

* передача текстовых сообщений;
* передача личных сообщений в дополнение к групповому общению;
* передача файлов;
* передача изображений, нарисованных пользователем в программе-клиенте;
* голосовая и видео-связь в реальном времени;
* многопользовательские игры.

Клиентские приложения могут быть реализованы как:

* полновесные приложения, устанавливаемые на компьютер пользователя;
* веб-страницы, не требующие установки.

В различных реализациях присутствуют следующие требования или ограничения, накладываемые на участников общения:

* необходимость регистрации в системе для создания аккаунта, хранящего личную информацию о пользователе;
* необходимость использования только официального клиента (в случае, если протокол является закрытым);
* необходимость использования исключительно специального клиентского приложения или же исключительно браузера для доступа к системе;
* ограничение на число участников конференции (например, в Skype - 25).

## Вывод

Для реализации в рамках данного курсового проекта решено создать продукт со следующими параметрами:

* передача текстовых сообщений;
* передача личных сообщений в дополнение к групповому общению;
* передача изображений, нарисованных пользователем в программе-клиенте
  + Из всех перечисленных выше вариантов приложений только **Windows Live Messenger** обеспечивает эту возможность; в то же время она незаменима при передаче формул или схем;
* использование в качестве клиентского приложения как веб-страницы, так и устанавливаемой на компьютер пользователя программы;
* отсутствие жестких ограничений на число участников конференции
  + **Skype** устанавливает ограничение в 25 участников, не всегда оправданное ограничениями технологий связи. Столкновение с такими ограничениями, как правило, приводит к разочарованию пользователей в системе, поэтому предполагается, что количество пользователей будет ограниченно только ресурсами сервера.

Отличительной чертой разрабатываемого продукта будет возможность одновременного редактирования несколькими пользователями одного изображения, отсутствующая в перечисленных выше популярных продуктах.

В целях обеспечения высокой скорости отклика веб-клиента разрабатываемый протокол будет основан на технологии **WebSocket [2]**, позволяющей веб-странице установить полнодуплексное соединение с сервером поверх **TCP**. Использование этой технологии позволит ликвидировать такой недостаток веб-чатов, как относительно длительные задержки обновления. **WebSocket** также легко реализуется в обычных приложениях; это, в конечном счете, позволит пользователю выбирать, какой тип клиентской программы ему удобнее – веб-клиент или обычное приложение.

Таким образом, в данном курсовом проекте будут разработаны:

* протокол многопользовательской конференции с поддержкой передачи изображений;
* серверное приложение, работающее по данному протоколу;
* клиентские приложения, работающие по данному протоколу – веб-клиент и обычное настольное приложение.

# Конструкторская часть

## Терминология

Перечислим термины, использующиеся при описании структуры и функционирования разрабатываемой системы (синонимы и взаимозаменяемые слова перечисляются через запятую):

* Онлайн-конференция, конференция, чат (chat) - способ сетевого общения, позволяющий нескольким участникам обмениваться сообщениями в реальном времени
* Участник конференции, участник чата - пользователь, подключившийся к конференции при помощи клиентской программы и участвующий в общении.
* Серверная программа, сервер - центральный компонент разрабатываемой системы, реализованный в виде отдельного приложения и обеспечивающий взаимодействие клиентских программ, подключающихся к нему.
* Клиентская программа, клиент - приложение, выполняющееся на компьютере пользователя, подключающееся к серверной программе и предоставляющее пользователю графический интерфейс для общения в чате.
* Веб-приложение - программа, как правило, написанная с использованием HTML и JavaScript, и выполняющаяся в среде веб-браузера, обмениваясь в процессе работы данными со своей серверной частью.
* Текстовый чат - элемент конференции, позволяющий участникам обмениваться текстовыми сообщениями.
* Лог чата - элемент графического интерфейса текстового чата, отображающий тектстовые сообщения и системные уведомления в порядке их поступления. Не путать с логом работы сервера.
* Лог работы сервера - файл, в который сервер записывает все происходящие события и поступающие от клиентов датаграммы. Не путать с логом чата.
* Доска рисования - элемент конференции, представляющий собой поверхность для рисования и набор инструментов рисования. Изменения, сделанные на доске одним участником, моментально становятся видны другим участникам конференции.
* Ник, имя участника – уникальное имя, под которым участник конференции зарегистрировался в системе.

## Общая структура системы

Разрабатываемая система реализует классическую централизованную клиент-серверную архитектуру. Как показано на рисунке 2.1 , в системе имеется один сервер и произвольное число клиентов.

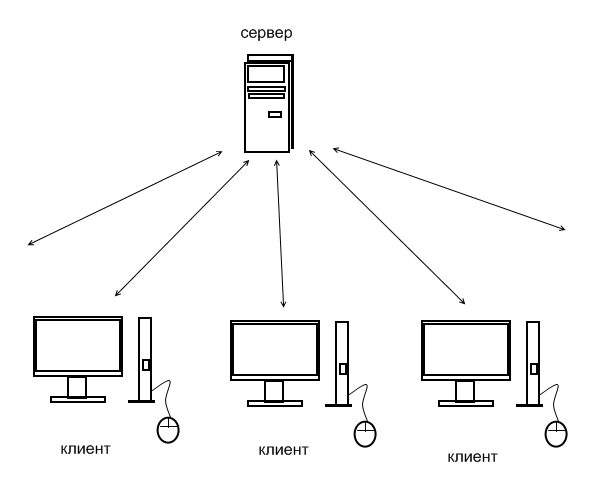


Рисунок 2.1 Классическая клиент-серверная архитектура

При этом сервер и клиенты могут выполняться как на разных компьютерах, так и на одном.

## Обзор протокола ENCP

Experimental Network Chat Protocol (ENCP) - протокол, используемый в системе для обеспечения функционирования онлайн-конференции.

Протокол ENCP описывает сообщения (датаграммы ENCP), которыми обмениваются серверная и клиентские программы в процессе работы, и порядок обработки этих сообщений, задавая таким образом функционирование системы во времени.

Для передачи датаграмм между приложениями используется стандартный протокол WebSocket**[2]**, поддерживаемый современными веб-браузерами. WebSocket является относительно новой технологией, позволяющей установить дуплексное TCP-соединение между веб-приложением и сервером; наличие такого соединения устраняет необходимость регулярного опроса сервера веб-приложением, что снижает нагрузку на сеть и одновременно увеличивает скорость реакции клиентской программы на серверные события. Датаграммы ENCP передаются по каналу WebSocket в сериализованном виде в нотации JSON **[]**. Схема передачи датаграмм представлена на рисунке 2.2.

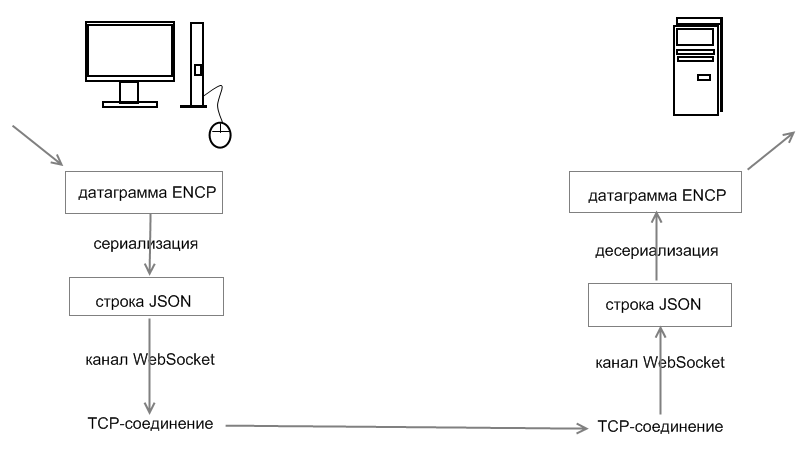


Рисунок 2.2 Схема передачи датаграмм

С точки зрения ENCP, клиент может находиться в четырех различных состояниях: "отключен", "подключен к серверу", "аутентификация" и "участник конференции" (см. рисунок 2.3).

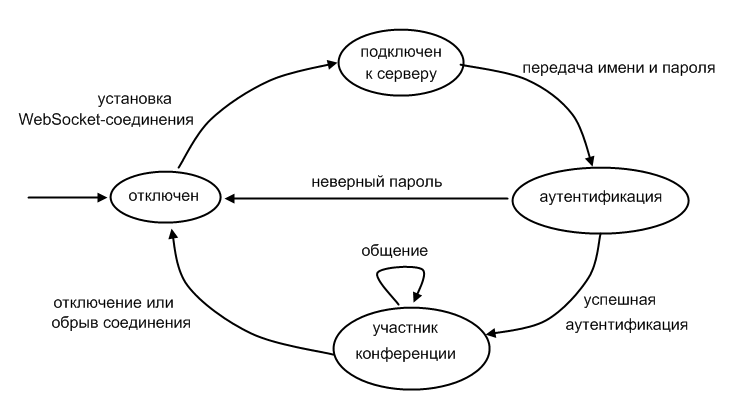


Рисунок 2.3 Диаграмма состояний клиента с точки зрения протокола ENCP

Состояние "отключен" является начальным. При входе в конференцию клиент проходит через состояния "подключен к серверу" и "аутентификация", и, при успешном прохождении аутентификации попадает в состояние "участник чата". В этом состоянии клиент находится во время общения. При выходе из конференции, инициированном пользователем или же произошедшем при обрыве связи, клиент возвращается в начальное состояние.

Каждая датаграмма ENCP, описанная в протоколе, имеет свой код, состоящий из буквы "М" и числа. Более подробное описание датаграмм будет приведено позже.

## Структура клиентской программы

Разбиение клиентской программы на модули и их взаимодействие представлено на рисунке 2.4.

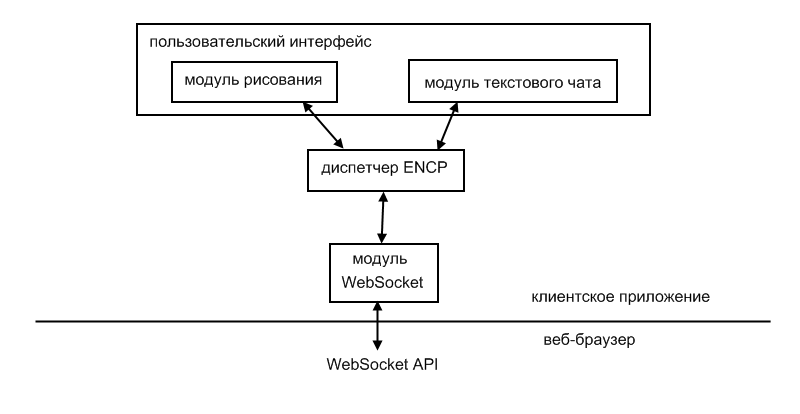


Рисунок 2.4 Структура клиентской программы

Основные модули клиентской программы:

* Модуль WebSocket отвечает за управление сокетом, используемым для подключения к серверу. Модуль использует WebSocket API, предоставляемый веб-браузером. Взаимодействие модуля с модулями более высокого уровня заключается в передаче и приеме датаграмм ENCP; при этом данный модуль берет на себя работу по сериализации и десериализации датаграмм в нотацию JSON.
* Диспетчер ENCP выполняет первичную обработку датаграмм, поступающих от сервера. В задачу диспетчера входит определение типа датаграммы и вызов соответствующей процедуры-обработчика из модулей верхнего уровня.
* Модуль текстового чата отвечает за отображение текстовых сообщений, написанных участниками чата, и уведомлений сервера в логе чата, а также содержит интерфейс для ввода и отправки пользователем текстовых сообщений.
* Модуль рисования отвечает за графическую часть конференции, а именно поддержку общей доски рисования, реализацию различных инструментов рисования, составление и обработку датаграмм ENCP для синхронизации доски рисования.

## Структура серверной программы

Разбиение серверной программы на модули и их взаимодействие представлено на рисунке 2.5.

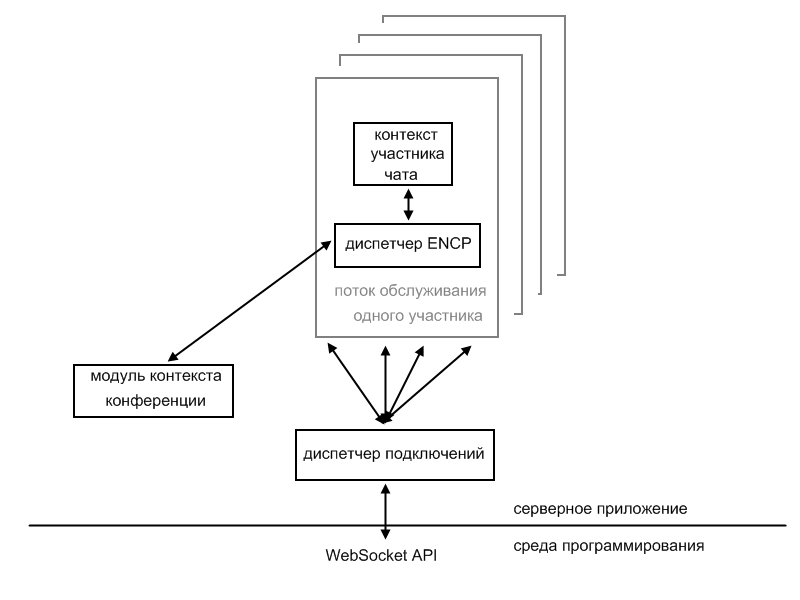


Рисунок 2.5 Структура серверной программы

Серверное приложение является многопоточной программой со следующими потоками выполнения:

* Главный поток, в котором выполняется диспетчер подключений;
* Набор обслуживающих потоков (по одному для каждого подключенного участника чата), выполняющих код диспетчера ENCP;
* Отладочный поток, выполняющий код интерактивной отладочной консоли.

Основные модули серверной программы:

* Диспетчер подключений управляет серверным сокетом, принимает входящие подключения и создает новые потоки для обслуживания клиентов.
* Модуль контекста конференции содержит информацию, необходимую для работы конференции в целом, в том числе:
  + Список зарегистрированных участников, содержащий регистрационные имена, пароли, права и дополнительную информацию по каждому участнику чата;
  + Список участников, подключенных к серверу в настоящий момент;
  + Историю последних сообщений - фиксированное число последних текстовых сообщений чата;
  + Полную историю изменения доски рисования с момента последней ее очистки.
* Диспетчер ENCP, выполняющийся в каждом потоке обслуживания. Каждый диспетчер работает со своим клиентским сокетом, и принимает и обрабатывает датаграммы ENCP.

С каждым потоком обслуживания, и, соответственно, с каждым диспетчером ENCP связан свой контекст участника чата.

## Спецификация протокола ENCP

В протоколе ENCP объекты передаются в виде сообщений, одни из которых идут только от клиента к серверу или только от сервера к клиенту, а другие ходят в обоих направлениях, но имеют различный набор полей в зависимости от направления.

Общая схема для большинства сообщений такова: клиент посылает сообщение серверу, сервер дополняет его некоторыми полями (например, именем отправителя) и рассылает всем клиентам, подключенным к серверу (в том числе и тому, от которого оно только что пришло)

Каждому типу сообщений соответствует код - буква M и порядковый номер по приведенному ниже списку[[2]](#footnote-3). Для краткости вместо фразы "формирует сообщение М1" будем использовать "формирует М1".

Ниже приведён список существующих сообщений и их краткое описание:

1. **M1 – реализует передачу текстового сообщения между участниками конференции**

* ***Внутреннее представление:***

*{ 'type': 'text', 'sender': sender, 'value': value }*

*sender: имя (ник) отправителя. Изначально отсутствует, добавляется сервером*

*value: строка-текст сообщения.*

* ***Жизненный цикл сообщения:***

1. Пользователь вводит сообщение и нажимает кнопку «Отправить»
2. Программа-клиент формирует сообщение M1 с полем *value*, содержащем введенный текст, и отправляет серверу.
3. Сервер добавляет поле *sender*, содержащее имя (ник) отправителя, опционально проводит валидацию *value* (для того, чтобы значение не включало в себя вредоносный JavaScript и т.п.) и отправляет всем клиентам.
4. Клиент, получив M1, использует его содержимое для добавления строки в лог чата.

Диаграмма канала управления, связанного с передачей сообщений, изображена на рисунке 2.8.

1. **М2 – уведомляет участников конференции о новом участнике чата**

* ***Внутреннее представление:***

*{ 'type': 'notify', 'subtype': 'user\_joined', 'user': user }*

*user: имя (ник) нового участника*

* ***Жизненный цикл сообщения:***

1. Программа-клиент нового участника производит первое (за сеанс) обращение к серверу. Выполняется процедура рукопожатия WebSocket.
2. По завершении рукопожатия сервер присваивает участнику имя (ник), формирует M2 с этим ником и отправляет всем клиентам.
3. Клиент, получив M2, добавляет в лог чата фразу "к нам пришел %user%" и добавляет нового участника в отображаемый список сидящих в чат-комнате.
4. **M3 – уведомляет об уходе из конференции одного из участников чата**

* ***Внутреннее представление:***

*{ 'type': 'notify', 'subtype': 'user\_left', 'user': user }*

*user: имя (ник) участника покинувшего чат-комнату*

* ***Жизненный цикл сообщения:***

1. Участник закрывает страницу в браузере или программу-клиент, таким образом обрывая подключение по веб-сокету.
2. Сервер фиксирует закрытие веб-сокета, формирует M3 с именем участника, сокет которого закрылся, и рассылает всем клиентам сообщение.
3. Клиент, получив M3, добавляет в лог чата фразу "нас покинул %user%" и удаляет user'а из отображаемого списка участников.
4. **M5 – запрос списка участников**

* ***Внутреннее представление:***

*{ 'type': 'roommates', 'list': roommates }*

*roommate: - массив объектов {'nick': nick, 'color': color} - ников и зарезервированных цветов участников чата*

* ***Жизненный цикл сообщения:***

1. Программа-клиент раз в 30 секунд, а так же при приходе сообщения имеющего тип – «notify», производит синхронизацию списка участников чата с серверным списком.
2. Программа-клиент формирует М5 без полей (кроме 'type') и отправляет серверу.
3. Сервер дополняет M5 полем roommates и отправляет сообщение клиенту, запросившему список (а не всем клиентам).
4. Клиент, получив М5 со списком, использует его для обновления отображаемого списка участников чата.

Кроме того, при подключении нового клиента сервер формирует и отправляет ему сообщение М5.

Диаграмма канала управления, связанного с синхронизацией клиентского списка участников, представлена на рисунке 2.9.

1. **М7 – изменение общей доски для рисования**

* ***Внутреннее представление:***

*{'type':'public\_drawing','sender':sender,'commands': [drawing\_command, ...] }*

*sender: добавляется сервером*

*drawing\_command: объект {'color': color, 'tool': tool, 'param': param}, где tool - имя инструмента ('pencil', 'line', 'rectangle', ...), а param – параметры необходимые для построения объекта инструментом tool.*

или

*drawing\_command: 'clearall' – команда очищающая доску для рисования (доступна только пользователям с администраторскими правами)*

***Интерпретация param и color в зависимости от tool:***

* + *pencil - рисуется отрезок из точки (x1,y1) в точку (x2,y2) цветом color*
  + *line - рисуется отрезок из точки (x1,y1) в точку (x2,y2) цветом color*
  + *rectangle - рисуется, но не закрашивается, прямоугольник с противоположными углами в точках (x1,y1) и (x2,y2) цветом color*
  + *fillrect - то же что и rectangle, но прямоугольник закрашивается*
  + *text – в точке (x,y) цветом color выводится текстовое сообщение*
* ***Жизненный цикл сообщения:***
  + ***Жизненный цикл A:***

1. Пользователь рисует изображение на доске рисования. В процессе рисования клиентский буфер команд (draw\_history) пополняется командами рисования.
2. По завершении операции рисования (пользователь отпустил кнопку мыши) программа-клиент формирует сообщение М7 с содержимым буфера команд (одно сообщение содержит массив, представляющий буфер целиком), отправляет М7 на сервер и очищает буфер.
3. Сервер, получив М7, дополняет его именем пользователя (sender) и рассылает всем клиентам, кроме клиента-отправителя.
4. Сервер сохраняет команды в истории команд рисования, чтобы выдавать новым участникам уже готовую картинку.
5. Серверная история команд очищается, если commands содержит 'clearall'.
6. Клиент, получив М7 от сервера, выполняет содержащиеся в нем команды рисования одна за другой, применяя их к общей доске для рисования. Дополнительно, клиент может отобразить имя пользователя, указанное в поступившем сообщении, в надписи "Автор последнего изменения".
   * ***Жизненный цикл Б:***
     1. При подключении к чату нового участника пользователь отправляет ему всю серверную историю рисования в виде сообщения М7, но без поля sender.
     2. См. п.4 цикла А.М9 – отправление только что вошедшему в чат пользователю последних N сообщений.
7. **M9 – отправление только что вошедшему в чат пользователю последних N сообщений**

* ***Внутреннее представление:***

*{'type':'notify','subtype':'last\_messages', messages':self.websocket.lastNMessages}*

*lastNMessages: массив словарей*

*{'sender': datagram['sender'], 'value': datagram['value'], 'time': datetime.datetime.now().strftime('%H:%M:%S')}*

* ***Жизненный цикл сообщения:***
  1. Сервер, узнав, что к конференции присоединился пользователь отсылает ему массив последних N сообщений.
  2. Клиент, получив сообщение М9, уведомляет пользователя о новых сообщениях и выводит их в окно чата.

1. **М10 – вход на сервер**

* ***Внутреннее представление:***

*{'type': 'login', 'nick': chat\_nick, 'password': password}*

* ***Жизненный цикл сообщения:***

1. Пользователь решает подключиться к чату. Он вводит адрес сервера, логин, пароль, и нажимает на кнопку «Войти».

2. Клиент устанавливает WebSocket-соединение с сервером. После успешного установления соединения клиент формирует и отправляет на сервер сообщение М10.

3. Сервер, получив М10, проверяет его содержимое; результатом проверки является сообщение М11.

Диаграмма канала управления, связанного с входом на сервер, представлена на рисунке 2.7.

1. **М11 – результат входа на сервер**

* ***Внутреннее представление:***

*{'type': 'login\_result', 'logged\_in': logged\_in, 'is\_super': is\_super, 'color': color, 'message': message}*

* ***Жизненный цикл сообщения:***
  1. Сервер получает М10.
  2. Выполняются следующие проверки:
     1. есть ли указанный ник в списке зарегистрированных пользователей;
     2. если выполнен первый пункт, то правильно ли указан пароль;
     3. является ли уже этот пользователь участником чат-комнаты.

3. Если (I) неверно, сервер выполняет регистрацию пользователя в своей базе, сохраняя ник и пароль. После этого считается, что условия (I) и (II) выполнены.

4. Вход на сервер считается успешным при одновременном выполнении первых двух условий и невыполнении третьего.

Алгоритм аутентификации изображен на рисунке 2.7.

***5а.*** Если вход успешен, то сервер формирует М11 с полем logged\_in = True. Поле color содержит значение цвета, ассоциируемого с пользователем. Сервер отправляет М11 клиенту; затем сервер выполняет прочие действия, связанные с входом клиента в чат, такие как отправка последних сообщений.

***5б.*** Если вход не успешен, то сервер формирует М11 с logged\_in = False. В поле message заносится причина отказа во входе (эту строку увидит пользователь). Сервер отправляет клиенту сформированное сообщение и разрывает связь.

## Каналы управления

### Подключение нового участника конференции

На рисунке 2.6 представлен канал управления для подключения нового участника конференции. Все события сопровождаются передачей ENCP датаграмм.

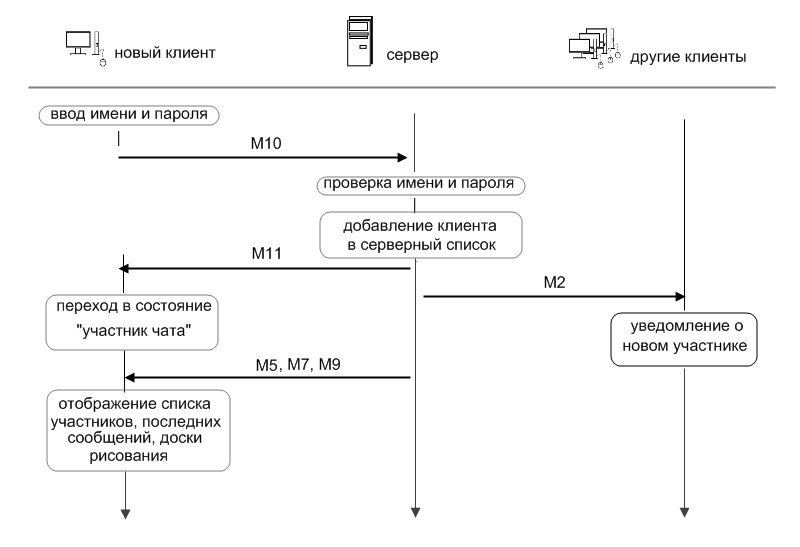


Рисунок 2.6 Канал управления подключения нового клиента

На рисунке 2.7 изображён алгоритм аутентификации нового участника конференции.

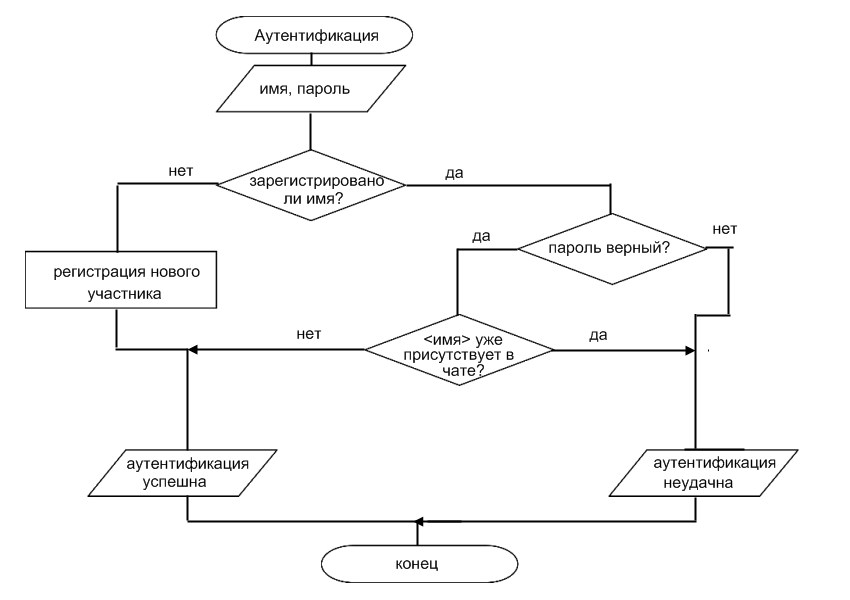


Рисунок 2.7 Алгоритм аутентификации клиента

### Обмен сообщениями

На рисунке 2.8 изображён канал управления обмена сообщениями. Сообщения могут быть как текстовые, так и связанные с рисованием общего изображения. Все сообщения, проходя через сервер, сохраняются в лог работы сервера.

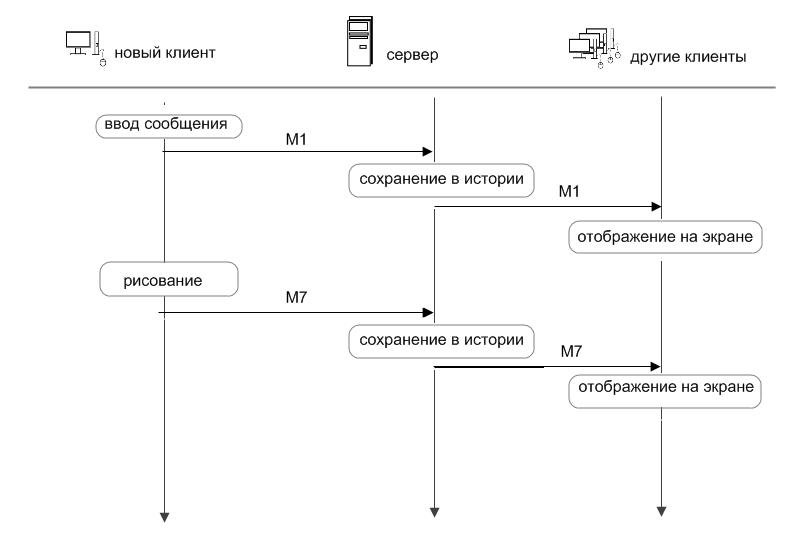


Рисунок 2.8 Канал управления обмена сообщениями

### Синхронизация клиентского списка

На рисунке 2.9 изображен канал управления синхронизации клиентского списка.

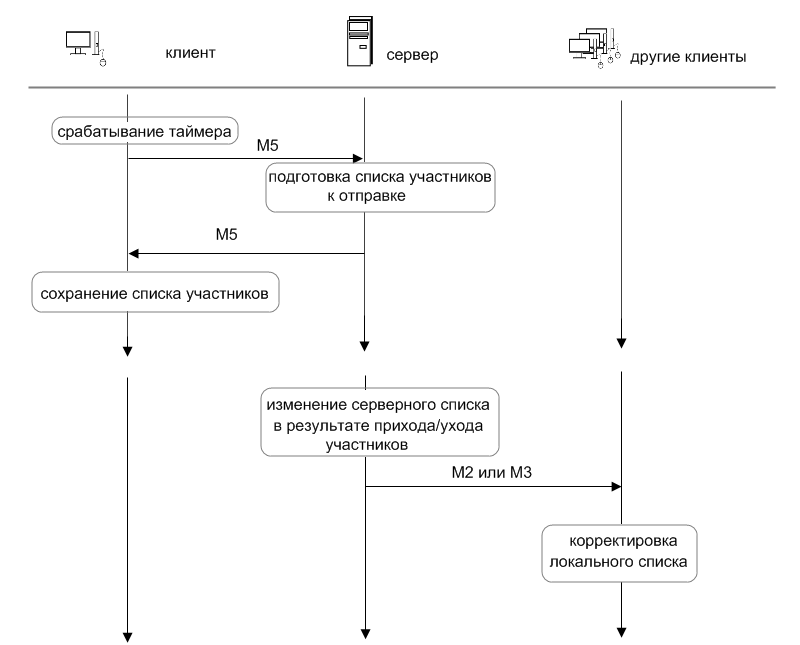


Рисунок 2.9 Канал управления клиентского списка

# Технологическая часть

## Выбор языка и среды программирования

**Клиентская часть** демонстрационной программы является веб-приложением. Веб-приложения разрабатываются с использованием языка разметки HTML и языка программирования JavaScript, поддерживаемого любым современным браузером на любой платформе, либо с помощью таких технологий, как Flash, Java или Silverlight, требующих для выполнения наличия специальных браузерных плагинов, доступных далеко не на всех платформах. Было выбрано сочетание **HTML/JavaScript** – из-за отсутствия требований к дополнительным плагинам, а также из-за встроенной поддержки стандарта WebSocket, на который опирается протокол ENCP.

Для отладки клиентской части использовался встроенный отладчик веб-браузера **Google Chrome**.

**Серверная часть** может быть написана на любом языке программирования, поддерживающем работу с TCP-сокетами и многопоточность. Выбор пал на язык **Python**. Выразительный синтаксис языка позволяет свести к минимальному объем кода, необходимый для реализации алгоритмов. Отсутствие этапа компиляции в цикле разработки программы на Python позволяет сократить длительность цикла и ускорить процесс отладки и разработки в целом. Стандартная библиотека языка, помимо модулей работы с сокетами и с потоками, содержит функции сериализации в JSON, необходимые для работы протокола ENCP.

Для отладки серверной части использовался встроенный интерактивный отладчик языка Python.

## Дополнительные инструменты

Так как курсовой проект рассчитан на работу в группе, требуются инструменты для совместной разработки. В качестве такого инструмента был выбран **GitHub** —  веб-сервис для хостинга проектов и их совместной разработки, основанный на системе контроля версий **Git**. Разработан на Ruby On Rails компанией Logical Awesome.

GitHub предоставляет: возможность создания единого репозитория для хранения исходного кода, возможность создания заданий, и их комментирования и редактирования другими пользователями, а также систему контроля версий.

## Обзор файлов исходного кода серверной части системы

Листинг кода из приведённых ниже файлов можно найти в приложении А(5).

### Модуль user.py

Содержит класс, описывающий пользователя конференции.

### Модуль utils.py

Содержит функцию для выбора случайного различимого для человеческого глаза цвета, возвращаемого в формате строки ‘#RGB’, где R, G и B – 16-ое представления соответственно красного, зелёного и голубого цвета.

### Модуль websocket.py

Содержит класс, описывающий сервер, а также методы связанные с ним: проверка подключения клиента к серверу, регистрацию нового пользователя, логирования происходящих серверных изменений, сохранения сообщения в список последних N сообщений.

### Модуль wsthread.py

Содержит класс, описывающий поток, общающийся с одним клиентом, а также методы связанные с ним: поиск пользователя, получение списка пользователей, методы отправки и получения датаграмм, метод реализующий цикл общения с клиентом, а также инициализацию работы с WebSocket-ом.

## Обзор файлов исходного кода клиентской части системы

Листинг кода из приведённых ниже файлов можно найти в приложении А().

### Файл protocol.js

Файл содержит клиентские функции для реализации общения с сервером, менеджеры обработки приходящих от сервера сообщений, а также функцию настройки сокета.

### Файл ui.js

Файл содержит клиентские функции, связанные с пользовательским интерфейсом, а также интерфейс для подключения к серверу.

### Файл utils.js

Файл содержит клиентские функции общего назначения такие как, функция получения клиентского времени и реализацию быстрого доступа к элементу по его идентификатору.

### Файл drawning.js

Файл содержит клиентские функции, связанные с рисованием на «клиентском холсте».

### 3.4.5. Файл client.html

Файл содержит «скелет» клиентского веб-приложения.

### 3.4.6. Файл chat.css

Файл содержит каскадные таблицы стилей клиентского веб приложения никак не зависящие от выбранного визуального стиля приложения.

### Файлы orange\_style.css, grey\_style.css, blue\_style.css

Файлы содержат каскадные таблицы стилей для клиентского веб приложения. В зависимости от выбранного визуального стиля приложения, подгружается тот или иной файл.

## Копии экранов системы

В данном разделе представлены снимки, соответствующие состояниям системы в момент выполнения некоторых задач.

Работа в клиентском приложении начинается с подключения к серверу. В случае, если это невозможно, выводится сообщение «Невозможно подключиться к серверу» (рисунок 3.1.)

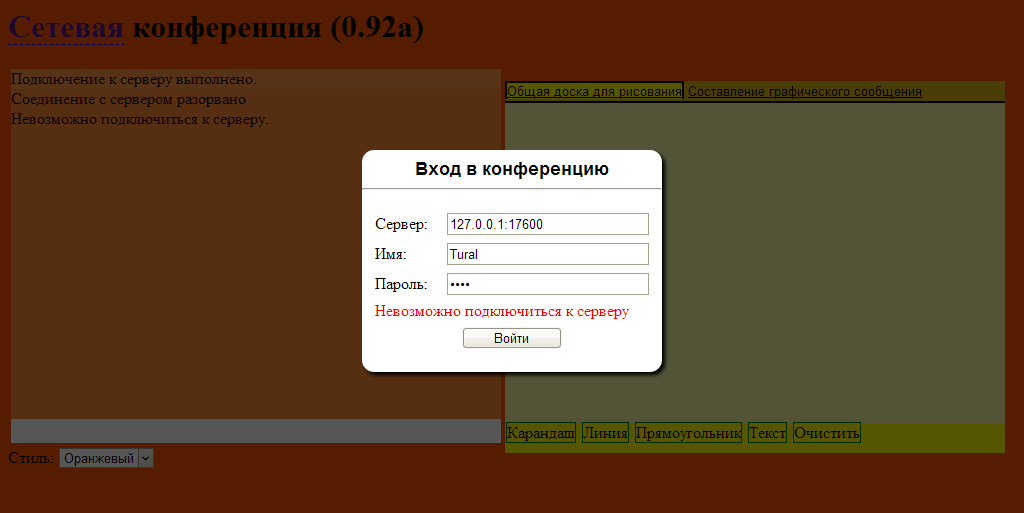


Рисунок 3.1 Невозможно подключиться к серверу

В случае если подключение к серверу возможно, но введён неверный пароль, для зарегистрированного пользователя, выводиться сообщение «Неверный пароль» (рисунок 3.2.).

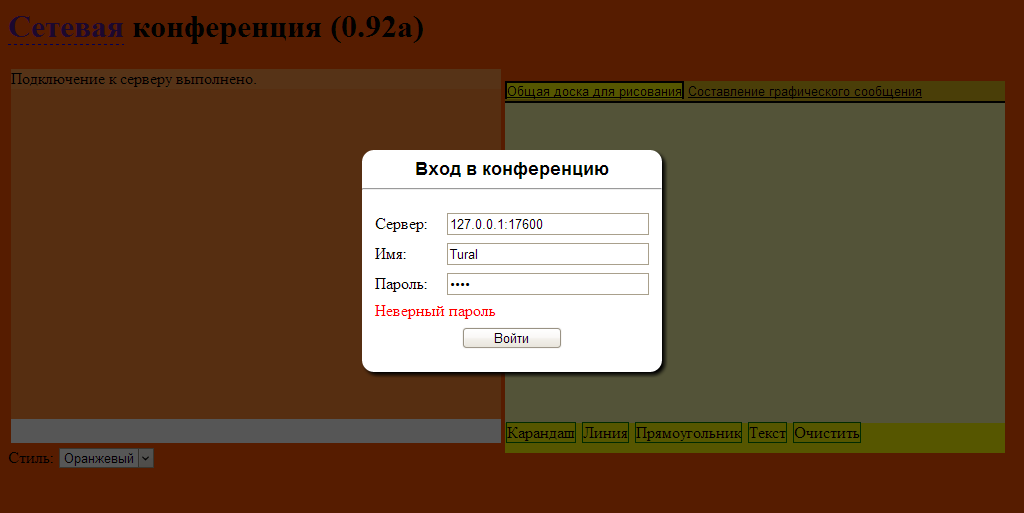


Рисунок 3.2 Неверный пароль

На рисунке 3.3 изображена рабочая сессия. В левой части окна изображена область чата, а в правой доска для рисования.

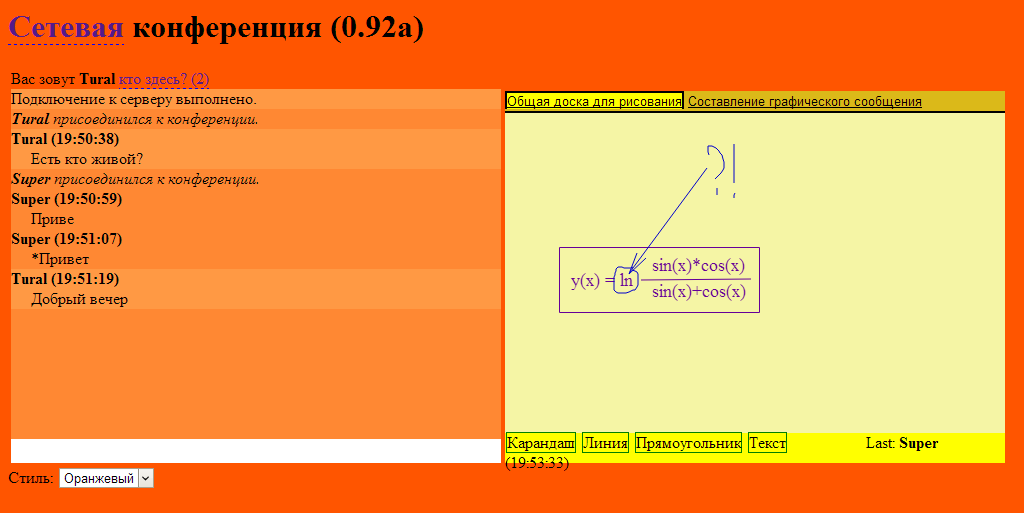


Рисунок 3.3 Рабочая сессия

В области чата, сообщения, написанные текущим пользователем, выделяются цветом отличным от цвета фона чата. Помимо текстовых сообщений, в области чата отображаются системные уведомления (рисунок 3.4).

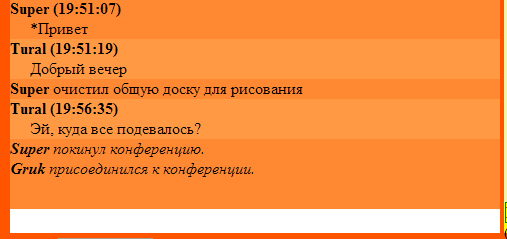


Рисунок 3.4 Системные уведомления в области чата

В списке участников конференции (рисунок 3.5), кроме имени участников, выводится цвет, которым данный участник рисует на доске для рисования.

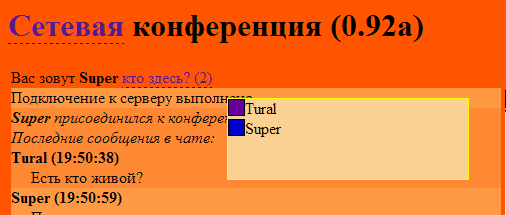


Рисунок 3.5 Список участников конференции

Кроме стандартных инструментов для рисования, таких как линия и прямоугольник (рисунок 3.6), предусмотрена возможность ввода текстовых сообщений (например, для облегчённого написания математических формул, как это показано на рисунке 3.7)

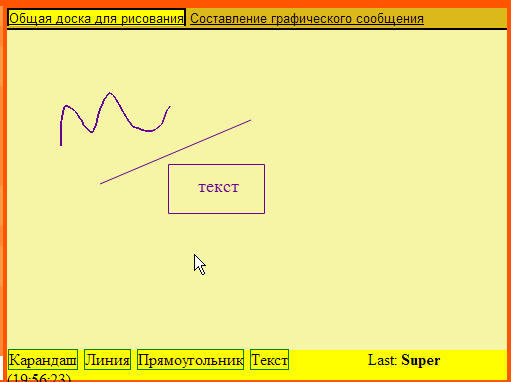


Рисунок 3.6 Инструменты рисования

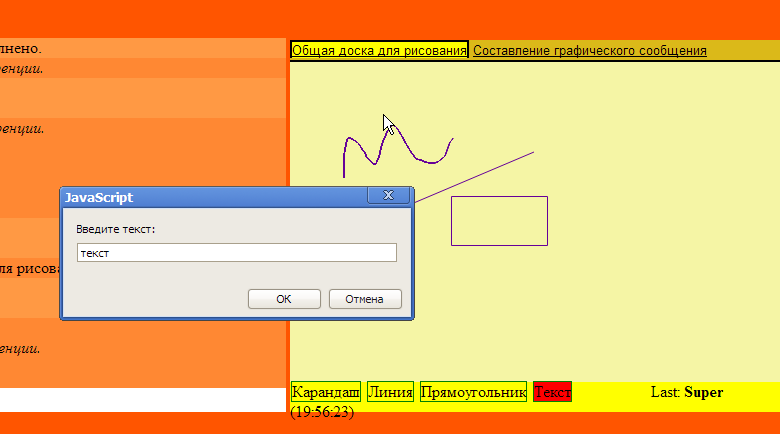


Рисунок 3.7 Инструмент "Текст"

Супер-пользователь обладает правами для очищения общей доски рисования (см. рисунок 3.8).



Рисунок 3.8 Инструменты супер-пользователя

# Заключение

В процессе работы над данным курсовым проектом были выполнены следующие задачи:

* анализ существующих систем текстовых онлайн-конференций;
* разработка собственного протокола онлайн-конференции;
* реализация разработанного протокола в конкретном программном продукте.

Анализ существующих систем позволил выявить типичную функциональность, достоинства и недостатки различных онлайн-конференций. С учетом результатов анализа были выдвинуты требования к разрабатываемому протоколу и продукту. Протокол был успешно разработан и реализован в демонстрационной программе.

Проведенные испытания показали, что демонстрационная программа, хотя и содержит ряд недостатоков: отсутствуют личные сообщения, инструмент рисования "текст" недостаточно удобен, оформление пользовательского интерфейса проигрывает существующим программам - тем не менее, является вполне пригодной для практического использования.

# Список литературы

1. Свободная энциклопедия Wikipedia. <http://en.wikipedia.org/>
2. Протокол WebSocket (черновик стандарта). <http://www.whatwg.org/specs/web-socket-protocol/>
3. Стандарт WebSocket от W3. <http://dev.w3.org/html5/websockets/>
4. Нотация JSON. <http://www.json.org/>
5. Creating an HTML 5 canvas painting application. <http://dev.opera.com/articles/view/html5-canvas-painting/>
6. Python documentation. <http://docs.python.org/>

# Приложение А

## Выборочный листинг исходного кода серверной части системы

Ниже приведены листинги некоторых файлов серверной части системы.

### Файл utils.py

**def** get\_random\_rgb\_color\_string():

""" Получение случайного RGB цвета типа WEB-colors """

rgb\_string = "#"

rgbColorsCount = 3

deltaColorExchange = 51

numberOfColorGradations = 256

indexOfZeroColor = random.randrange(rgbColorsCount)

i = 0

**while** i < rgbColorsCount:

**if** i != indexOfZeroColor:

rpb\_part = random.randrange(0, numberOfColorGradations, deltaColorExchange)

**if** rpb\_part == 0:

rgb\_string += "00"

**else**:

rgb\_string += hex(rpb\_part)[-2:]

**else**:

rgb\_string += "00"

i += 1

**return** rgb\_string

### Файл websocket.py

**class** WebSocket():

uid=0

server=0

**def** save\_last\_message(self, datagram):

""" сохранение последнего полученного сообщения в N последних """

#LOCK

self.last\_n\_messages.append({'sender': datagram['sender'], 'value': datagram['value'], 'time': datetime.datetime.now().strftime('%H:%M:%S')})

**if** len(self.last\_n\_messages) > LAST\_N\_MESSAGES\_MAX\_COUNT:

self.last\_n\_messages = self.last\_n\_messages[1:]

**def** save\_chat\_log(self, log\_message):

""" Логирование происходящих серверных изменений (кроме roommates) """

#LOCK

self.chat\_log\_file.write(log\_message.encode('utf-8'))

self.chat\_log\_file.flush()

**def** register\_new\_user(self, nick, password):

""" Регистрация нового пользователя """

color = utils.get\_random\_rgb\_color\_string()

is\_super = (nick == 'Admin')

record = {'nick': nick, 'password': password, 'color': color, 'is\_super': is\_super}

#LOCK

str\_nick = nick.encode('utf-8') # shelve requires byte-strings as keys

self.registered\_users[str\_nick] = record

self.registered\_users.sync() # flush

**def** is\_user\_on\_server(self, nick):

""" Проверка, подключен ли nick к серверу """

#LOCK

result = False

**for** user **in** self.users:

**if** user.nick == nick:

result = True

**break**

**return** result

**def** \_\_init\_\_(self, address, port, connections, server):

#последние N сообщений

self.last\_n\_messages = []

self.public\_picture\_history = []

#пользователи

self.users=[]

self.chat\_log\_file = open('chat\_log.log', 'ab')

self.registered\_users = shelve.open('users.db')

**for** reg\_user **in** self.registered\_users:

**print** self.registered\_users[reg\_user]

self.server = server

server = socket.socket ( socket.AF\_INET, socket.SOCK\_STREAM )

server.bind ( ( address, port ) )

# поток с отладчиком, которому доступен экземпляр WebSocket под именем wsk

**def** run(wsk):

**import** pdb

pdb.set\_trace()

debug\_thread = threading.Thread(target=run, args=(self,))

debug\_thread.start()

server.listen ( connections )

**print** "Yo, server here"

**while** True:

channel, details = server.accept()

**print** "Wassup?" + str(details)

self.uid = self.uid + 1

self.users.append(User(channel, self.uid))

**print** "Now starting the thread"

wsthread.WebSocketThread (channel, details, self).start()

### Модуль wsthread.py

Функция interact, реализующая диспетчер ENCP:

**def** interact**(**self**,** client**):**

""" цикл общения с клиентом

возвращает: True, если общение может продолжаться;

False, иначе.

"""

users **=** self**.**websocket**.**users

this\_user **=** self**.**this\_user

**try:**

data **=** self**.**recieve\_websocket\_string**(**client**)**

**except** ConnectionClosedError**:**

**print** "//closed: #" **+** unicode**(**this\_user**.**nick**)** **+** " has left"

self**.**broadcast**({**'type'**:** 'notify'**,** 'subtype'**:** 'user\_left'**,** 'user'**:** unicode**(**this\_user**.**nick**)})**

# TODO: broadcast (warning: will probably get err 10053 from

# this very client

**return** **False**

**except** MalformedPacketError**:**

**print** "//malformed packet from #" **+** unicode**(**this\_user**.**nick**)**

**return** **True**

#print data

**try:**

datagram **=** json**.**loads**(**data**)**

**if** datagram**[**'type'**]** **!=** 'roommates'**:**

log\_entry **=** u"%s\t%s\t%s\t%s\n" **%** **(** datetime**.**datetime**.**now**(),** self**.**details**,** this\_user**.**nick**,** datagram **)**

self**.**websocket**.**save\_chat\_log**(** log\_entry **)**

**except** Exception **as** e**:**

**import** pdb**;** pdb**.**set\_trace**();**

**if** datagram**[**'type'**]** **==** 'text'**:** #M1

datagram**[**'sender'**]** **=** this\_user**.**nick

self**.**websocket**.**save\_last\_message**(**datagram**)**

self**.**broadcast**(**datagram**)**

**elif** datagram**[**'type'**]** **==** 'login'**:** #M10

str\_nick **=** datagram**[**'nick'**].**encode**(**'utf-8'**)** # shelve requires byte-strings as keys

**if** str\_nick **in** self**.**websocket**.**registered\_users**:**

new\_user\_registered **=** **False**

user\_already\_on\_server **=** self**.**websocket**.**is\_user\_on\_server**(**datagram**[**'nick'**])**

password\_correct **=** **(**self**.**websocket**.**registered\_users**[**str\_nick**][**'password'**]** **==** datagram**[**'password'**])**

**else:**

self**.**websocket**.**register\_new\_user**(**datagram**[**'nick'**],** datagram**[**'password'**])**

new\_user\_registered **=** **True**

password\_correct **=** **True**

user\_already\_on\_server **=** **False**

successfull\_login **=** password\_correct **and** **not** user\_already\_on\_server

**if** successfull\_login**:**

# приветствуем нового участника!

this\_user**.**is\_super **=** self**.**websocket**.**registered\_users**[**str\_nick**][**'is\_super'**]**

this\_user**.**color **=** self**.**websocket**.**registered\_users**[**str\_nick**][**'color'**]**

this\_user**.**nick **=** datagram**[**'nick'**]**

self**.**send\_private**(** **{**'type'**:** 'login\_result'**,** 'logged\_in'**:** **True,** 'is\_super'**:** this\_user**.**is\_super**,**

'color'**:** this\_user**.**color**},** this\_user**)** # M11

self**.**broadcast**({**'type'**:** 'notify'**,** 'subtype'**:** 'user\_joined'**,** 'user'**:** datagram**[**'nick'**]})**

#отправка последних N сообщений новому пользователю

**if** self**.**websocket**.**last\_n\_messages **!=** **[]:**

self**.**send\_private**({**'type'**:** 'notify'**,** 'subtype'**:** 'last\_messages'**,** 'messages'**:** self**.**websocket**.**last\_n\_messages**},** this\_user**)**

# отправка истории рисования на публичной доске

self**.**send\_private**({**'type'**:** 'public\_drawing'**,** 'commands'**:** self**.**websocket**.**public\_picture\_history**},** this\_user**)**

**if** new\_user\_registered**:**

self**.**send\_private**({**'type'**:** 'text'**,** 'sender'**:** 'Сообщение от сервера'**,**

'value'**:** 'Вы были зарегистрированы как новый пользователь чата. Запомните введенный вами пароль.'**},**

this\_user**)**

**elif** **not** password\_correct**:**

self**.**send\_private**(** **{**'type'**:** 'login\_result'**,** 'logged\_in'**:** **False,** 'message'**:** 'Неверный пароль'**},** this\_user**)** # M11

**return** **False** # разрываем связь

**elif** user\_already\_on\_server**:**

self**.**send\_private**(** **{**'type'**:** 'login\_result'**,** 'logged\_in'**:** **False,** 'message'**:** 'Юзер уже есть на сервере'**},** this\_user**)** # M11

**return** **False** # разрываем связь

**else:**

self**.**send\_private**(** **{**'type'**:** 'login\_result'**,** 'logged\_in'**:** **False,** 'message'**:** 'Что-то еще пошло не так'**},** this\_user**)** # M11

**return** **False** # разрываем связь

**elif** datagram**[**'type'**]** **==** 'set-name'**:** #M4

**if** this\_user**.**nick **is** **not** **None:**

old\_nick **=** this\_user**.**nick

this\_user**.**nick **=** datagram**[**'new\_name'**]**

self**.**broadcast**(**

**{**

'type'**:** 'notify'**,**

'subtype'**:** 'user\_renamed'**,**

'new\_nick'**:** datagram**[**'new\_name'**],**

'old\_nick'**:** old\_nick

**}**

**)**

**elif** datagram**[**'type'**]** **==** 'public\_drawing'**:** #M7

datagram**[**'sender'**]** **=** this\_user**.**nick

#LOCK

**if** **(**'clearall' **in** datagram**[**'commands'**])** **and** this\_user**.**is\_super **:**

self**.**websocket**.**public\_picture\_history **=** **[]**

self**.**broadcast**(**datagram**)**

**elif** 'clearall' **not** **in** datagram**[**'commands'**]:**

self**.**websocket**.**public\_picture\_history**.**extend**(**datagram**[**'commands'**])**

self**.**broadcast**(**datagram**,** except\_user**=**this\_user**)**

**elif** datagram**[**'type'**]** **==** 'roommates'**:** #M5

datagram**[**'list'**]** **=** self**.**get\_users\_list**()**

self**.**send\_private**(**datagram**,** this\_user**)**

**return** **True**

## Листинг исходного кода клиентской части системы

Ниже приведены листинги некоторых файлов клиентской части системы.

### Файл protocol.js

// общение с сервером - уровень пакетов/датаграмм

**var** datagram\_recieved **=** **function** **(**packet**)** **{**

datagram **=** JSON**.**parse**(**packet**);**

**switch** **(**datagram**.**type**)** **{**

**case** 'login\_result'**:** //#M11

**if** **(**datagram**.**logged\_in**)** **{**

hide\_login\_window**();**

window**.**drawing\_color **=** datagram**.**color**;**

canvases**.**public\_canvas**.**set\_color**(**window**.**drawing\_color**);**

$**(**'inputbox'**).**focus**();**

$**(**'userNickInfo'**).**innerHTML **=** "Вас зовут <b>" **+** window**.**chat\_nick **+** "</b>" **+**

//" (<a href='#' onclick='renameNickname()'>сменить ник</a>)";

""**;**

**if** **(**datagram**.**is\_super**)**

$**(**'clear\_all\_btn'**).**style**.**visibility **=** 'visible'**;**

**else**

$**(**'clear\_all\_btn'**).**style**.**visibility **=** 'hidden'**;**

$**(**'whoIsHere'**).**innerHTML **=**

" <a href='#' onclick='toggle\_people\_list()'>кто здесь? (<span id='people\_count'>0</span>)</a>"**;**

//обновляем список пользоватлей и запускаем таймер,

//список так же обновляется однократно при приходе сообщения типа 'notify'

intervalId **=** window**.**setInterval**(**"updateUserList();"**,**30000**);**

**}** **else** **{**

set\_login\_status**(**datagram**.**message**,** 'red'**);**

**}**

**break;**

**case** 'text'**:**

//время на клиенте

text\_message\_recieved**(**datagram**.**value**,** datagram**.**sender**,** getLocalTime**());**

**break;**

**case** 'notify'**:**

notification\_recieved**(**datagram**);**

**break;**

**case** 'public\_drawing'**:**

canvases**.**public\_canvas**.**execute\_command\_sequence**(**datagram**.**commands**);**

**if** **(**datagram**.**sender**)** **{**

$**(**'last\_public\_draw'**).**innerHTML **=** "Last: <b>" **+** datagram**.**sender **+** "</b> (" **+** getLocalTime**()** **+** ")"**;**

**if** **(**datagram**.**commands**.**indexOf**(**'clearall'**)** **>=** 0**)** **{**

canvases**.**public\_canvas**.**clear\_canvas**();**

logg**(**"<b>" **+** datagram**.**sender **+** "</b> очистил общую доску для рисования"**);**

**}**

**}**

**break;**

**case** 'roommates'**:**

$**(**'people\_count'**).**innerHTML **=** datagram**.**list**.**length**;**

$**(**'list\_of\_people'**).**innerHTML **=** ""**;**

// заполнить список людей

**for(**i **in** datagram**.**list**){**

$**(**'list\_of\_people'**).**innerHTML **+=** "<div>" **+** datagram**.**list**[**i**].**nick **+**

"<div style='float: left; width: 15px; height: 15px; border: #000 solid 1px; background-color:" **+**

datagram**.**list**[**i**].**color **+** ";'/></div>"**;**

// TODO : отображать цвет!!!

**}**

**break;**

**}**

**};**

**var** send\_datagram **=** **function(**datagram**)** **{**

msg **=** JSON**.**stringify**(**datagram**);**

packet **=** msg**;**

window**.**ws**.**send**(**packet**);**

**};**

// общение с сервером - уровень текстовых сообщений

**var** send\_text\_message **=** **function** **(**msg\_text**)** **{**

datagram **=** **{**'type'**:** 'text'**,** 'value'**:** msg\_text**};**

send\_datagram**(**datagram**);**

**};**

// получение текстового сообщения

**var** text\_message\_recieved **=** **function** **(**msg\_text**,** sender**,** time**)** **{**

logg**(**msg\_text**,** sender**,** time**);**

**};**

// системные сообщения

**var** notification\_recieved **=** **function** **(**datagram**)** **{**

**switch** **(**datagram**.**subtype**)**

**{**

**case** 'user\_left'**:**

logg**(**"<i><b>" **+** datagram**.**user **+** "</b> покинул конференцию.</i>"**);**

**break;**

**case** 'user\_joined'**:**

logg**(**"<i><b>" **+** datagram**.**user **+** "</b> присоединился к конференции.</i>"**);**

**break;**

**case** 'user\_renamed'**:**

logg**(**"<i><b>" **+** datagram**.**old\_nick **+** "</b> теперь известен как <b>" **+** datagram**.**new\_nick **+** "</b></i>"**);**

**break;**

**case** 'last\_messages'**:**

logg**(**"<i>Последние сообщения в чате:</i>"**);**

**for(**i **in** datagram**.**messages**){**

text\_message\_recieved**(**datagram**.**messages**[**i**].**value**,** datagram**.**messages**[**i**].**sender**,** datagram**.**messages**[**i**].**time**);**

**}**

**break;**

**}**

//обновление списка пользователей

updateUserList**();**

**};**

1. http://www.digitaltrends.com/how-to/guide-to-instant-messenger-clients/ [↑](#footnote-ref-2)
2. В связи с тем, что список постоянно менялся, число, идущее после буквы «М» не является порядковым *(прим. авторов)* [↑](#footnote-ref-3)