# **5. Описание архитектуры мессенджера:**

Сам по себе мессенджер – это система, состоящая из 2-х частей: серверной и клиентской частей.

Серверная часть занимается работой по приему запросов от клиента, их анализом, изменениями и извлечениями информации из базы данных, отсылкой информации другим пользователям.

Клиентская часть, в свою очередь, занимается обработкой событий, генерируемых пользователем (таких, например, как нажатие на клавишу, выбор диалога и т.д.), генерированием запросов на сервер, выводом информации на консоль пользователя и т.д.

Для того, чтобы сервер и клиент были синхронизированы и «понимали друг друга» необходим некоторый «язык», который они оба будут знать и на котором они будут общаться. В роли этого языка выступает протокол связи, или, протокол общения клиента с сервером.

В данном проекте, я реализовал собственный протокол – Message WebSocket Protocol (MWSP).

## **5.1 Структура протокола MWSP:**

Это протокол, посредством которого происходит «общение» между сервером и клиентом.

Сервер и клиент передают друг другу данные в виде json следующего формата:

{

Code: code,

From: id,

Dialog id: dialog id,

DataType: datatype,

Data: data

}

Code – код запроса,

From - id пользователя на стороне клиента,

Dialog\_id – id диалога, который просматривает сейчас пользователь,

DataType – тип данных в data (сделано для удобства проверки и в силу технических причин языка Java),

Data – непосредственно, данные, которые присылаются.

Code – код запроса. В зависимости от того, кто отправляет запрос (клиент или сервер), коды принимают различные значения. Вот основные диапазоны значения code для клиента (в случае, если MWSP запрос генерирует клиент):

0 – 9 – события (например, нажатие на клавишу),

10 - 19 – передача данных (важных данных, требуется ответ от сервера response),

20 - 29 – запросы на получение данных,

30 - 39 – запросы на подтверждение,

40 - 49 – служебные запросы,

50 – 59 – запросы на создание (добавление, удаление, обновления).

В свою очередь, со стороны сервера:

100 - 109 – передача информации о событиях от других клиентов,

110 - 119 – передача данных,

120 – 129 – ответ сервера на запрос о получении данных,

130 – 139 – ответ на подтверждение о чем-то,

140 – удаление сессии пользователя,

141 – 149 – различные некритические ошибки,

150 – ошибка в сообщении протокола,

160 – 170 – критические ошибки на сервере (соединение немедленно закрывается).

## **5.2. Описание работы протокола MWSP:**

### **5.2.1. Установление соединения и получение первых данных:**

Пользователь, после страницы входа (или регистрации) получает сессию. Далее, он, автоматически, перенаправляется на главную страницу, страницу с диалогами. Это jsp файл, который начинает генерироваться на сервере. При этом происходит проверка существования сессии у пользователя, после чего, если сессия существует, генерируется html файл и перекидывается пользователю. Если же не существует данной сессии, то, пользователь перекидывается на главную страницу входа.

Далее, устанавливается webSocket соединение. При этом, на стороне сервера, создается объект Endpoint, при этом, срабатывает метод onOpen(). Этот метод добавляет данную сессию в контейнер сессий и отправляет ответ 130 (подтверждение добавления в контейнер сессий) пользователю.

Если добавление не произошло (в результате какой-то ошибки, или чего-то еще), то сервер отправляет ответ с кодом 161 (ошибка в базе данных, немедленный разрыв соединения).

При этом, на стороне пользователя происходят несколько иные действия. OnOpen() пользователя создает таймер, который срабатывает через какое-то количество времени, вызывая некоторую функцию. При этом, если сервер пришлет данные в метод onMessage, причем с кодом 130, то данный таймер уничтожается и поднимается флаг активации (начала работы). Если же сработает данная функция, то она проверит, поднят ли флаг активации. Если он поднят, то уничтожится таймер и все. Если же он опущен, то клиент пошлет MWSP запрос с кодом 30 на сервер и будет установлен, снова, таймер. Таким образом, клиент будет пытаться посылать данный запрос (номер 30) определенное число раз. Если, в течении всего этого времени, сервер не ответит ничем, то клиент разрывает соединение с сервером. Почему сервер не отвечает на запрос клиента – это отдельный вопрос. Ситуации бывают разные, но они не связаны с падением сервера, скорее всего, ибо в таком случае был бы вызван метод onError (или onClose) на стороне клиента, и клиент сам бы разорвал бы соединение. Отсутствие ответа от сервера может быть связано с несколькими причинами:

- На сервере неправильно обработан протокол MWSP

- На сервере возник бесконечный цикл

- На сервере отвалился модуль, который отвечает за отправку сообщений

Если же клиент получает код 130, то, как уже было сказано, он поднимает флаг активации и проверяется флаг получения диалогов. Если этот флаг опущен, то отправляется запрос 20 и включается таймер, который, действует по аналогичной схеме коду 30.

Сервер, получив запрос 20 от клиента, ищет данные и отправляет их клиенту с кодом 120. Если же, в результате поиска данных, возникнет какая-та ошибка, то будет отправлен код 141-149, в зависимости от проблемы (может быть проблема в базе данных, или, слишком большое число запросов на сервер и т.д.). Клиент, получая коды ошибок сервера, не шлет подтверждение о получении данного запроса.

Далее, если клиент получает список диалогов, он отправляет подтверждение с номером 31 и поднимает флаг получения диалогов. Таким образом, клиент переходит в режим ожидания. Теперь, клиент будет отправлять данные на сервер, только если клиент нажмет на что-то, или же, если получит какую-то информацию с сервера. Тоже самое и с сервером, получив код 31, он переходит в режим ожидания. Таким образом, произошла вступительная часть общения клиента с сервером, или, так называемое «рукопожатие».

**Запрос 30 (Запрос на подтверждение добавления в контейнер сессий):**

*Клиент после авторизации, получает страницу, которая производит веб сокетное соединение с сервером. После соединения, клиент посылает запрос 30 на подтверждение добавления в контейнер сессий.*

*Этот запрос имеет следующий вид:*

Code: 30,

From: user\_id (который сгенерировал event),

Dialog\_id: -1,

Data\_Type: DATA\_TYPE\_TEXT (эта константа = 0),

Data: “”

*Сервер проверяет, находится ли клиент в контейнере сессий, а именно его сессия, и, если он там находится, то посылает подтверждение 130.*

*Ответ имеет следующий вид:*

Code: 130,

From: -1 (SERVER.ID),

Dialog\_id: -1,

Data\_Type: DATA\_TYPE\_TEXT (сейчас, это 0),

Data: “”

***Если же возникнет ошибка в БД, или ошибка в протоколе, то будет отправлен соответствующий запрос об ошибке (либо 161 – ошибка в БД, либо 150 – ошибка в протоколе).***

**Запрос 20 (запрос на получение списка диалогов пользователя):**

*Если клиент получит подтверждение о том, что он добавлен в контейнер сессий, то после этого клиент сгенерирует запрос 20 на получение списка диалогов. Этот запрос имеет следующий вид:*

Code: 20,

From: user\_id (id пользователя),

Dialog\_id: -1 (или 0, тестирование проводилось на 0, но, по идее, можно и -1),

Data\_Type: DATA\_TYPE\_TEXT (сейчас, это 0),

Data: “”

*Сервер получает этот запрос и начинает искать в БД список диалогов. Получив список диалогов, отправляет ответ 120 в следующем виде:*

Code: 120,

From: -1 (SERVER.ID),

Dialog\_id: тот dialog\_id, который указал пользователь в запросе 20(вообще, не имеет значения какое тут будет значение),

Data\_Type: DATA\_TYPE\_DIALOGS\_LIST (сейчас, это 2),

Data: {

dialogInfList: [

{

dialog\_id: (id диалога),  
 dialog\_name: (название диалога),  
 dialog\_img: (путь к аватарке диалога),  
 last\_msg: (последнее сообщение в диалоге),  
 from\_user\_name: (имя пользователя, который отправил посл. Сообщ.),  
 last\_msg\_time: (время последнего сообщения),  
 from\_user\_id: (id пользователя, который отправил посл. Сообщ.),  
 last\_read\_time: (время последнего чтения списка сообщений пользователями, которые не отправляли последнее сообщение),  
 my\_last\_reading\_time: (время последнего чтения диалога клиентом, орому отправляется этот запрос от сервера)

},

{

…

}

]

}

### **5.2.2. Отправка данных о получении списка сообщений:**

На стороне клиента хранится информация о том, какой диалог мы сейчас просматриваем. Если пользователь выберет какой-то диалог, то на сервер будет направлен запрос 21, с id диалога, на который нажал пользователь. При этом будет поставлен таймер и флаг ожидания получения сообщений. Если таймер отработает, при этом флаг ожидания сообщений не будет поднят, то пользователю отобразится информация о невозможности загрузки. Ему будет предложено нажать на кнопку «повторить запрос». В случае, если сервер получил данные из базы данных, он отправляет их с запросом 121 об успешном получении данных из бд. Далее, получив данные, пользователь не отправляет подтверждения. Он вставляет эти данные в DOM дерево у себя на странице.

Если, в результате вытаскивания данных из базы данных, возникла ошибка, то сервер возвращает код 162 (ошибка извлечения данных из БД).

Таким образом работает запрос на получение списка сообщений.

**Запрос 21 (запрос на получение списка сообщений в диалоге):**

*При переходе на конкретный диалог (выборе диалога пользователем и клике на него), клиент посылает на сервер запрос 21 следующего вида:*

Code: 21,

From: user\_id (который сгенерировал запрос),

Dialog\_id: id диалога, в котором сгенерировался запрос (отправляет клиент),

Data\_Type: DATA\_TYPE\_TEXT (сейчас, это 0),

Data: “”

*Сервер, получив данный запрос, начинает искать в БД данные (из разных таблиц), и, получив эти данные, отсылает пользователю ответ следующего вида:*

Code: 121,

From: -1 (SERVER.ID),

Dialog\_id: id диалога, в котором сгенерировался запрос (отправляет клиент),

Data\_Type: DATA\_TYPE\_MESSAGES\_LIST(сейчас, это 3),

Data: {

lastMsgInf: - информация о последнем сообщении в диалоге  
{  
 from\_user\_id: , (id пользователя, отправившего последнее сообщение в диалоге, из которого клиент ожидает получить список сообщений)  
 last\_read\_time: , (поле last\_read\_time в таблице ReadTable)  
 last\_msg\_time: , (время последнего сообщения в диалоге)  
 my\_last\_reading\_time: (время последнего просмотра диалога пользователем)  
},  
messageInfList: [ - непосредственно, список сообщений диалога(в хрон. Поряд.)  
 {  
 from\_user\_id: , (id пользователя, отправившего сообщение)  
 from\_user\_name: , (имя пользователя, отправившего сообщение)  
 from\_user\_avatar: , (путь к его аватарке)  
 msg: , (само сообщение)  
 time: (время отправки сообщения)  
 },   
 {  
 ... (следующее сообщение)  
 } (и так далее)  
]

}

### **5.2.3. Отправка событий на сервер:**

Отправка событий на сервер чрезвычайно проста. При возникновении какого-то события (с учетом дополнительной обработки), отправляется запрос с определенным кодом от 0-9 на сервер. На этом запрос заканчивается.

**Запрос 0:**

*Клиент отправляет следующий запрос:*

Code: 0,

From: user\_id (который сгенерировал event),

Dialog\_id: id диалога, в котором сгенерировался event (отправляет клиент),

Data\_Type: DATA\_TYPE\_TEXT (сейчас, это 0),

Data: Name пользователя (генерирует и отправляет клиент, сервер не проверяет имя и не лезет в БД)

*Сервер, получив данный запрос, отправляет его ВСЕМ участникам (которые сейчас онлайн) диалога dialog\_id, в том числе данному пользователю, который отправил этот запрос. Поэтому, клиент, получив ответный запрос от сервера (Сервер перенаправляет запрос PQuery, не изменяя его, поэтому все участники диалога получат запрос в таком же виде, который был отправлен пользователем) должен его проигнорировать, если он видит, что его id совпадает с id в запросе.*

**Запрос 1 (запрос о прочтении списка сообщений):**

*Пользователь кликает на какой-то диалог. Клиент отправляет запрос на сервер на получение списка сообщений. Получив список сообщений, начинается его анализ: производится проверка того, нужно ли отправлять запрос о прочтении сообщений на сервер. Если окажется, что нужно отправить запрос на сервер, то генерируется PQuery следующего вида:*

Code: 1,

From: user\_id (который сгенерировал event),

Dialog\_id: id диалога, в котором сгенерировался event (отправляет клиент),

Data\_Type: DATA\_TYPE\_TEXT (сейчас, это 0),

Data: “”

*Сервер, получив данный запрос, сделает все необходимые действия в БД. Ответный запрос будет получен только в том случае, если есть какая та критическая ошибка на сервере. Если же все прошло успешно, то ответного запроса не будет.*

Если же возникнет ошибка в БД, или ошибка в протоколе, то будет отправлен соответствующий запрос об ошибке (либо 161 – ошибка в БД, либо 150 – ошибка в протоколе).

### **5.2.4. Отправка сообщений на сервер:**

Если пользователь нажал на кнопку отправки сообщения, то сообщение прогоняется через энкодер, и, далее, отправляется с запросом 10. При этом, включается таймер ожидания и поднимается флаг ожидания ответа на сообщение. Сервер, получив запрос 10, добавляет его в бд, далее, сразу же после этого, если добавление прошло успешно, генерирует ответ 110. После этого ответа (110), сервер отправляет сообщение всем пользователям, которые находятся в данном диалоге и онлайн.

Клиент, получив сообщение 110, удостоверяется, что сообщение успешно сохранено в базу данных.

Сервер генерирует ошибку 161 о невозможности добавить данные в базу данных, после чего разрывает соединение с клиентом. Клиент, получив данный код, сохраняет код ошибки и разрывает соединение (сервер тоже разрывает соединение). Срабатывает метод onClose(), который проверяет код ошибки и, в соответствии с этим, выводит пользователю соответствующую информацию.

Если срабатывает таймер отправки сообщения, при этом флаг не поднят, то сообщение отправляется еще раз с кодом 10 (в таком случае, может быть, сообщение добавится дважды в бд), при этом, снова включается таймер. Так делается определенное число раз, после чего, если сервер так и не ответит (т.е. флаг не будет поднят), то клиент разрывает соединение с соответствующим выводом на экран (для пользователя).

**Запрос 10 (отправка сообщений на сервер):**

*Клиент, написав сообщение и нажав на кнопку отправить, генерирует следующий PQuery запрос:*

Code: 10,

From: user\_id (который сгенерировал event),

Dialog\_id: id диалога, в котором сгенерировался event (отправляет клиент),

Data\_Type: DATA\_TYPE\_TEXT,

Data: Name msg (текст сообщения)

*Сервер, получив данный запрос, пытается сохранить его в БД. Если у него все получается, то он отправляет запрос 110 о подтверждении доставки сообщения на сервер. Этот запрос имеет следующий вид:*

Code: 110,

From: -1 (SERVER.ID),

Dialog\_id: -1,

Data\_Type: 0 (DATA\_TYPE\_TEXT),

Data: “”

*Клиент, получив данный запрос от сервера, убеждается, что его сообщение успешно доставлено на сервер. Далее, сервер отправляет сообщение всем пользователям, которые сейчас сидят в онлайне, и которые есть в диалоге dialog\_id (имеется ввиду, которые вообще находятся в этом диалоге, а не которые в конкретный момент просматривают этот диалог).*

*Этот запрос имеет некоторый видоизменный вид:*

Code: 10,

From: user\_id (который отправил сообщение),

Dialog\_id: id диалога, в котором было отправлено сообщение (отправляет клиент),

Data\_Type: 1 (DATA\_TYPE\_MESSAGE),

Data: {

Id: id пользователя, отправившего сообщение

name: имя пользователя, отправившего сообщение

avatar: название аватарки пользователя, отправившего сообщение

msg: текст сообщения

}

Если же возникнет ошибка в БД, или ошибка в протоколе, то будет отправлен соответствующий запрос об ошибке (либо 161 – ошибка в БД, либо 150 – ошибка в протоколе).

### **5.2.5. Выход из мессенджера:**

При нажатии на кнопку выхода из мессенджера, на сервер отправляется код 40, устанавливается таймер ожидания. Сервер, получив сообщение 40, уничтожает сессию из контейнера сессий, уничтожает сессию и отправляет код 140. Клиент, получив данное сообщение, уничтожает таймер и переходит на страницу входа.

Если сервер долго не отвечает, то производится уничтожение сессии в браузере и переход на страницу входа.

При этом, при уничтожении сессии, происходит разрыв веб сокетного соединения, что, в свою очередь, вызывает на сервере, автоматически, вызов функции onClose в endpoint, который, в свою очередь, автоматически удалит сессию пользователя из контейнера сессий и сделает все другие необходимые действия для очистки ресурсов сервера. Так же, будет очищено еще и так называемое хранилище HST, которое будет описано чуть позже в данной работе.

## **5.3. Сервер:**

### 5.3.1. Архитектура Сервера:

Сервер состоит из модулей, каждый из которых выполняет специфическую для него задачу.

Каждый из модулей связан с другими модулями, которые могут вызывать некоторые методы данного модуля.

Архитектура сервера представлена на рисунке:

Рис. 1. Архитектура Сервера

#### 5.3.2. Connector:

Connector – это некоторый «шлюз» к базе данных. Контейнер сервлетов TomCat имеет некоторый пул соединений к базе данных, который настроен в программном коде посредством объекта DataSource. Однако, работа с базой данных по-прежнему остается на низком уровне. Чтобы повысить API работы с БД, вводится данный класс.

Этот объект отвечает только за соединения с базой данных. Он управляет количеством соединений, отслеживает различные ошибки и другие параметры соединений, такие, например, как ограничение на количество используемых пулов соединений в данный момент времени, количество попыток захватить соединение из пула, время между такими попытками и так далее.

Например, если один из модулей, в данном случае, модуль DB Handler, захочет выполнить запрос к базе данных, он попросит у Connector выдать ему соединение. В Connector, в этот момент,  
производится проверка на то, сколько соединений уже установлено в данный момент. Если это количество превосходит максимальное количество, которое установлено в программном коде, то производится задержка на случайную величину, после чего запрос повторяется. Количество таких попыток прописано в программном коде.

Если соединение, через максимальное число попыток, не получено, возвращается null. В противном случае, происходит попытка соединиться с DataSource и получить реальное соединение. Если пул соединений TomCat не дает соединение, то производятся, снова, несколько попыток. В случае отказа на последней попытке, возвращается null. После каждой попытки, метод простаивает случайное время. Если соединение получено, то оно возвращается.

Таким образом, модуль Connector «всеми силами» пытается получить соединение с базой данных.

### 5.3.3. DB HANDLER:

Этот модуль отвечает за все запросы к базам данных на сервере. Любой запрос, который нужно будет выполнить в базе данных, выполняется этим модулем.

При выполнении каждого запроса, DB HANDLER обращается к модулю Connector для получения соединения из пула соединений. Если соединение будет получено, то начнется выполнение SQL запроса. В противном случае, если Connector не может выдать соединение, считается, что возникла критическая ошибка, и, DB Handler генерирует исключение, которое, впоследствии, будет перехвачено модулями более высоких уровней, которые уже примут решение, что делать в таком случае.

Также, DB HANDLER, при выполнении запросов, производит защиту от SQL-injection атак.

### 5.3.4. SESSIONS CONTAINER:

Это очень важный модуль, который хранит в себе список всех установленных на данный момент WebSocket сессий. Для каждого клиента, при установке соединения с сервером, создается своя websocket сессия, которая добавляется в контейнер сессий. Если же происходит разрыв соединения с сервером, то, автоматически, вызывается удаление сессии из контейнера сессий. Таким образом, контейнер сессий хранит в себе самую «свежую» информацию о соединениях с клиентами, и, именно поэтому, другие модули очень часто обращаются к данному модулю за получением данных.

Данный модуль, на самом деле, помимо внутреннего хранения сессий, так же, производит и другие действия. Например, когда пользователь добавляется в контейнер сессий, SESSIONS CONTAINER автоматически просит DB HANDLER изменить таблицы «DialogsLastSessionsChanges» и «UsersInformation» (их описание будет ниже), которые, в свою очередь, так же хранят самую «свежую» информацию о соединениях и необходимы для различного рода задач, которые будут возникать у других модулей (об этом чуть позже).

### 5.3.5. GROUP HANDLER:

Этот модуль отвечает за получение списка сессий пользователей, которые находятся в одном диалоге. Этот модуль использует кэшированные данные, для ускорения выдачи данных другим модулям. Единственный модуль, который обращается к Group Handler, это Sender.

Sender вызывает определенный метод у модуля Group Handler с запросом на выдачу списка сессий пользователей, которые находятся в определенном диалоге dialog\_id. Модуль Group Handler проверяет у себя в кэше информацию о данном диалоге. Если информация о диалоге dialog\_id есть в кэше, то модуль Group Handler производит проверку на то, что данные «свежие». Для этого, Group Handler обращается к модулю DB Handler для проверки времени в таблице «DialogsLastSessionsChanges». Если информация оказывается свежей, то на этом модуль заканчивает свою работу.

Если же окажется, что информация не свежая, или же, информация о диалоге (т.е. список сессий пользователей, находящихся в данном диалоге) не присутствует в кэше, в таком случае, Group Handler обращается к модулю SESSIONS CONTAINER, с просьбой выдать список сессий пользователей, находящихся в диалоге. После получения списка сессий, этот список сохраняется в кэш, а результат выдается модулю, запросившему этот список.

### 5.3.6. SENDER:

Этот модуль отвечает за отправку MWSP запросов клиентам. К данном модулю обращаются, непосредственно, Protocol Handler и DB Broker.

Sender определяет список сессий пользователей, которым нужно отправить запрос. Далее, Sender формирует MWSP запрос и отправляет его. На этом работа этого модуля заканчивается.

### 5.3.7. DB BROKER:

Этот модуль является промежуточным модулем взаимодействия Protocol Handler и DB Handler. Он предоставляет удобный API для модуля Protocol Handler, который «ничего не знает об устройстве сервера». Этот модуль вспомогательный, и нужен лишь для того, чтобы сделать Protocol Handler независимым от архитектуры сервера. Protocol Handler описывает лишь действия сервера с точки зрения протокола MWSP. Если измениться модуль DB Handler, то не нужно будет переписывать Protocol Handler, достаточно будет произвести изменения в DB Broker, который заточен на работу с модулем DB Handler.

Таким образом, этот модуль «налаживает» взаимодействие более «высокоуровневого» Protocol Handler и «низкоуровневого» DB Handler.

### 5.3.8. PROTOCOL HANDLER:

Этот модуль получает сообщения от клиентов, обрабатывает их и, в соответствии с запросом, производит вызовы необходимых для обработки запроса модулей на сервере.

Получив запрос от клиента, Protocol Handler проверяет корректность запроса. Если MWSP запрос построен неверно, то вызывается модуль Sender с отправкой сообщения об ошибке 150 (ошибка в запросе).

Если же запрос построен правильно, то, далее, производится проверка того, что id пользователя, отправившего данный запрос (поле from) совпадает с id в классе EndPoint. Таким образом, если они не совпадают, это означает, что запрос подделан. В таком случае, генерируется запрос 150 об ошибке.

Если же эти поля совпадают, производятся проверки других полей на корректность.

Если все прошло успешно, то в таком случае, Protocol Handler начинает выполнение действий, для обработки запроса.

## **5.4. Архитектура таблиц:**

Вся информация в мессенджере находится в одной базе данных: Messenger.

Мессенджер содержит несколько таблиц:

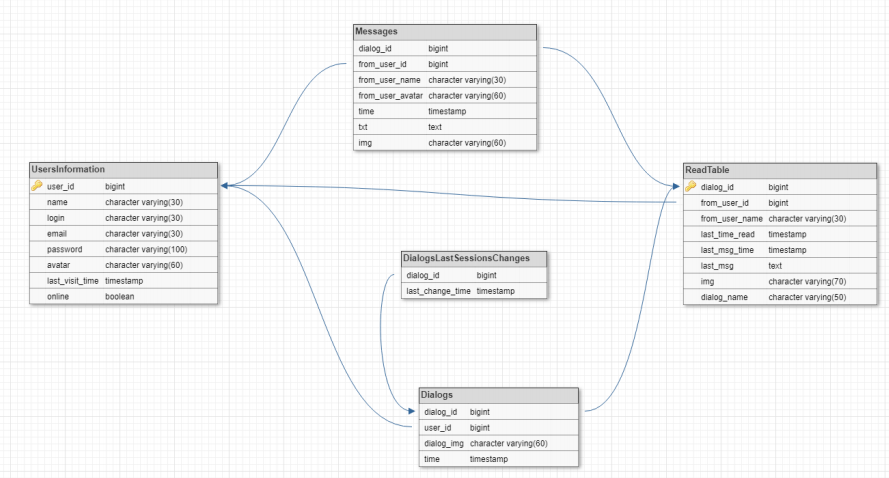


Рис 2. Архитектура таблиц

### 5.4.1. UsersInformation:

(**user\_id, name, login, email, password, avatar, last\_visit\_time, online**);

Параметры **last\_visit\_time** и **online** изменяются динамически контейнером сессий, при входе и выходе из мессенджера.

Эта таблица – статический источник данных о пользователе. При авторизации, при проверке последнего посещения мессенджера, при проверке онлайн или нет, все это можно делать через эту таблицу.

Реал тайм этой таблицы поддерживается контейнером сессий, который автоматически, при входе или выходе из мессенджера, изменяет параметры **last\_visit\_time** и **online.**

### 5.4.2. Messages:

**(dialog\_id, from\_user\_id, from\_user\_name, from\_user\_avatar, time, txt, img)**

В этой таблице хранится информация об отправленных сообщениях.

### 5.4.3. DialogsLastSessionsChanges:

**(dialog\_id, last\_change\_time)**

В этой таблице хранится информация о том, когда в последний раз происходило изменение (вход или выход из мессенджера) списка сессий пользователей, находящихся в диалоге dialog\_id.

Параметр last\_change\_time показывает, когда в последний раз происходило изменение в диалоге dialog\_id. Т.е., например, если какой-то пользователь вошел в сеть (или вышел из нее), то все диалоги, в которых он сидит, изменят время last\_change\_time. Это необходимо для того, чтобы корректно работал кэш в Group Handler, который хранит у себя список сессий из какого либо диалога, и при каждом обращении к нему, он сверяет время последнего изменения с last\_change\_time, и если оно меньше времени, записанного у него, то можно спокойно использовать кэш, в противном случае производится refresh кэша.

### 5.4.4. ReadTable:

(dialog\_id, from\_user\_id, from\_user\_name, last\_time\_read, last\_msg\_time, last\_msg, img, dialog\_name)

Эта таблица хранит dialog\_id в единичном экземпляре, причем, если диалог был создан, то он обязательно находится в ReadTable.

В базе данных Messenger установлен триггер на данную таблицу: при добавлении диалога в данную таблицу, автоматически, добавляется строка с этим диалогом в таблице “DialogsLastSessionsChanges”.

Last\_time\_read – это время последнего прочтения сообщения пользователями, которые не отправляли последнее сообщение.

### 5.4.5. Dialogs:

(dialog\_id, user\_id, dialog\_img, time)

Эта таблица хранит в себе информацию о диалогах с конкретными пользователями. Т.е., например, по этой таблице мы можем вытащить изображение диалога с точки зрения конкретного пользователя. Параметр time определяет время последнего прочтения данного диалога данным пользователем.

## **5.5. Client:**

Клиент, так же, как и сервер, имеет свою внутреннюю архитектуру, состоящую из различных модулей.

Внутреннее взаимодействие модулей обеспечивает синхронную работу обмена данными с сервером по протоколу MWSP.

В отличии от сервера, клиентская сторона производит обработку приходящих данных не только по каналу клиент-сервер (т.е. запросы, передающиеся по веб сокетному соединению), однако, так же, и обработку действий пользователя, который производит различные действия на странице.

Рис 3. Архитектура клиента.

### 5.5.1. Sender:

Этот модуль абсолютно аналогичен модулю Sender в сервере. Он формирует необходимые MWSP запросы и отправляет их на сервер.

При формировании запроса, необходимые данные, такие как, например, id пользователя, или его имя, берутся из модуля SST.

### 5.5.2. Painter:

Этот модуль отвечает за изменение DOM дерева, т.е., фактически, прорисовку html страницы.

### 5.5.3. Timer:

Этот модуль хранит таймеры, которые создаются при отправке различных запросов, и отвечает за работу с ними: создание, удаление, подсчет количества и так далее.

### 5.5.4. SST:

Этот модуль служит, своего рода, хранилищем состояний клиента. Модуль хранит информацию о том, какие запросы были отправлены на сервер, какие запросы ожидаются от сервера, информация о клиенте (его id, name, login и т.д.), информация о том, в какой диалог выведен на экран в данный момент времени, информацию о возникших ошибках и так далее.

Многие модули обращаются к SST чтобы получить различную информацию, начиная с данных пользователя, заканчивая информацией о возникших проблемах на сервере и пришедших запросах.

### 5.5.5. User Controller (UC):

Этот модуль обрабатывает различные действия пользователя на странице, такие, например, как нажатие на кнопку, переход к диалогу, выход из мессенджера и так далее.

Этот модуль обрабатывает действия пользователя, и, в соответствии с ними, производит вызовы различных внутренних модулей.

### 5.5.6. Protocol Handler (PH):

Аналог модуля Protocol Handler на сервере: обрабатывает приходящие MWSP запросы от сервера.

## **5.6. Описание работы модулей на стороне клиента:**

***Запрос 30:***

*При подключении (Веб сокетном) к серверу, вызывается метод UC.start(). Этот метод инициализирует (приводит в начальное состояние) SST, инициализирует Timer, после чего просит sender отправить запрос 30 на сервер. После этого, просит SST зафиксировать отправление запроса 30, после просит Timer создать таймер на отправку запроса 30.*

*После всего этого, возможно несколько вариантов:*

*Сервер присылает запрос 130 достаточно быстро (до того, как срабатывает таймер). PH проверяет у SST, что, действительно, ожидается запрос 130 (т.е. был выполнен запрос 30), и, если это так, то просит SST зафиксировать прием запроса 130. Далее, после этого, PH просит Timer уничтожить таймер на запрос 30, который был создан ранее. После этого, PH вызывает UC.req\_20();*

*Если же SST не ожидает запроса 130, то данный ответ от сервера игнорируется и завершается.*

*Сервер присылает запрос 161 о критическом завершении работы. PH просит SST объявить об критической ошибке. Далее, просит Timer уничтожить все таймеры. Далее, вызывается UC.err\_161()*

*Сервер присылает запрос 150 об ошибке протокола. Таким образом, выполняются все те же действия, только вызывается не UC.err\_161, а UC.err\_150();*

*Таймер срабатывает раньше, нежели сервер пришлет ответ. Таймер, в первую очередь, проверяет SST на наличие ошибок. Если обнаружилось, что есть ошибки, тогда таймер автоматически обнуляется и игнорирует дальнейшие действия.*

*Если же ошибок нет, то таймер, далее, проверяет, ожидает ли SST запрос 130. Если да, то таймер увеличивает у себя счетчик посылки запросов на 1 и вызывает, снова, UC.start();*

*Если нет, то таймер обнуляется и игнорируется.*

*Если кол-во попыток превысит определенное число, тогда, вызывается UC.err\_timer\_30();*

***Запрос 20:***

*Вызывается метод UC.req\_20(). UC проверяет у SST, что пришел запрос 30 (приход запросов 20 и 30 специально сохраняется). Если это не так, то ничего не делается.*

*Если же запрос приходил, то вызывается sender с отправкой запроса 20 на сервер. После этого, SST фиксирует отправку запроса 20, далее, вызывается Timer с просьбой создать таймер на отправку запроса 20.*

*После этого, возможны следующие события:*

*Сервер присылает запрос 120. Проверяется у SST, действительно ли ожидается запрос 120. Если нет, то игнорируется.*

*Если да, то SST фиксирует факт прихода 120, далее, уничтожается таймер на 20 запрос, после чего, вызывается Painter, с просьбой, вставить в DOM дерево список диалогов. После этого, просится SST объявить о произошедшем рукопожатии.*

***ПРО ЗАПРОСЫ, ПРИХОДЯЩИЕ ОТ СЕРВЕРА, В МОМЕНТ УСТАНОВЛЕНИЯ «РУКОПОЖАТИЯ»:***

***ЛЮБОЙ ТАКОЙ ЗАПРОС ИГНОРИРУЕТСЯ!***

*После того, как произошло рукопожатие, перед пользователем имеется список его диалогов.*

***Пользователь кликает на диалог:***

*Вызывается метод UC, в который передается id диалога. Далее, UC вызывает Painter с просьбой скрыть интерфейс диалогов (хотя он по-прежнему сохранен в DOM дереве) и отрисовать интерфейс диалога (кстати говоря, у Painter’а есть некоторый буфер, который хранит не отправленные сообщения в диалоге, поэтому, при отрисовке интерфейса диалога, в textarea вставляется это последнее, не отправленное, сообщение из буфера, если оно там есть). Далее, после этого, UC просит sender отправить запрос 21 на сервер, и SST фиксирует это, после чего Timer создает таймер на запрос 21.*

*Далее, возможно несколько вариантов событий:*

*Сервер, достаточно быстро, присылает ответ 121 со списком сообщений. PH просит SST зафиксировать это, после чего удаляет таймер в Timer, после чего вызывает отрисовку списка сообщений. Так же, PH, после этого, производит проверку данных на то, нужно ли посылать на сервер запрос 1, о прочтении сообщений. Информации в ответе 121 от сервера достаточно для того, чтобы определить это. PH делает это, и, если не нужно ничего делать, то ничего не делается, а если нужно, то вызывается метод UC, отвечающий за отправку запроса 1. Он вызывает sender, и все.*

*Сервер получает запрос об ошибке. В таком случае, действия все те же, что и выше (вообще, при любом получении ошибки, действия будут одинаковыми, поэтому дальше этот случай не оговаривается).*

*Срабатывает таймер. Он делает все то же, что и описывалось выше, поэтому этот случай тоже больше не будет рассматриваться (за исключением некоторых случаев).*

***Пользователь, во время загрузки списка сообщений, нажал на кнопку перехода, обратно, к списку диалогов:***

*Срабатывает метод UC, после чего, UC смотрит на SST и видит, что SST ожидает ответа 121 (это и описывает ситуацию, записанную в заголовке). После того, как он удостоверился, что пользователь ждет список сообщений от сервера, SST информируется о том, что мы переходим к списку диалогов (тем самым, SST больше не ожидает запроса 121). Далее, уничтожается таймер запроса 21, после чего, просит Painter скрыть интерфейс диалога и разблокировать интерфейс списка диалогов.*

*Таким образом, после того, как PH получит ответ от сервера 121, PH проверит SST и убедится, что мы не ждем данный запрос и проигнорирует ответ.*

***Пользователь начинает печатать сообщение в диалоге:***

*Срабатывает метод UC, который дергает метод Timer’а. На этом все заканчивается.*

*Этот метод таймера устроен достаточно умно: он смотрит, был ли установлен таймер до этого. Если он был установлен, то игнорируется вызов функции. Если нет, то устанавливается таймер.*

*При срабатывании данного таймера, вызывается sender для отправки запроса 0 на сервер.*

*Таким образом, на компьютеры пользователей запрос 0 приходит с той же частотой, с которой он генерируется на стороне клиента.*

***Пользователь нажимает на кнопку отправки сообщения:***

*Вызывается метод UC, который просит Painter блокировать возможность написания или отправки сообщений пользователем. Далее, после этого, просит sender отправить запрос 10, после чего, просит SST зафиксировать отправку запроса 10, после чего, устанавливается таймер.*

*Далее, возможны следующие ситуации:*

*Приходит запрос 110 об успешном добавлении сообщений (доставке). Тогда, PH проверяет у SST, что ожидается запрос 110. Если не ожидается, игнорируется ответ 110 и ничего не делается.*

*Если же ожидается, то, PH информирует SST о том, что запрос пришел, далее, уничтожает таймер на 10 запрос, после чего, просит Painter нарисовать сообщение (Painter, при этом вызове функции, снимает блокировку, и отрисовывает новое сообщение).*

*Приходит запрос о том, что сообщение не добавилось. Проверяется SST об ожидании запроса 110. Если не ожидается, то игнорируется.*

*Если ожидается, то SST информируется о том, что запрос пришел, уничтожается таймер и, после этого, PH информирует Painter’а об этом (что сообщение не доставлено).*

***Пользователь, во время отправки сообщения (когда происходит ожидание ответа 110), нажал на кнопку перехода к списку диалогов:***

*Срабатывает метод UC. После этого, UC проверяет SST и удостоверяется, что SST ожидает ответа 110 от сервера. Тогда, UC информирует SST о том, что переходит к списку диалогов. Далее, уничтожается таймер на запрос 10. После этого, говорится Painter’у о переходе на страницу списка диалогов.*

*При этом сам Painter, перед тем как завершить работу вызванного метода (который отвечает за отрисовку перехода от страницы диалога к странице списка диалогов), сохраняет в свой буффер сообщение, которое было в textarea.*

***Подведем итог к нажатию на кнопку перехода к списку диалогов:***

*Вызывается метод UC, который производит проверку у SST об ожидании каких-либо запросов. В случае, если они удовлетворяют описанным выше, то мы действуем так, как описывалось выше. В противном случае (при данном количестве протоколов), ничего из имеющихся запросов не ожидается, и тогда, SST информируется об переходе к странице списка диалогов. Далее, вызывается метод Painter’а на переход на страницу списка диалогов. При этом, Painter, как уже говорилось выше, сохраняет сообщение, которое в textarea (не отправленное) в буфер.*

***Приход сообщения (запрос 10):***

*Имеется 2 запроса, которые не могут ожидаться SST – это запрос 10 (получение сообщения) и запрос 0 (нажатие клавиши каким-то пользователем).*

*Из-за того, что эти запросы могут прийти в любое время, достаточно сложно описывать. Их описание будет разделено на различные случаи, которые будут описаны.*

***Клиент ничего не ожидает (не ожидает никакого запроса):***

*После того, как PH зафиксирует запрос 10 (т.е. вызовет handl\_10), PH спросит у SST о том, ожидается ли какой-то запрос. Если SST сообщит, что нет, то PH спросит, какой диалог сейчас просматривает пользователь. В связи с этим, возможны следующие ситуации:*

*Пользователь находится в списке диалогов. В таком случае, PH вызовет метод Painter’а, попросив его добавить сообщение в конкретный диалог.*

*Пользователь находится в диалоге, который не совпадает с тем, с которого пришло сообщение. В таком случае, PH, также, как и выше, попросит Painter’а добавить сообщение в список диалогов (который сейчас скрыт. Добавить сообщение в список диалогов означает, что, просто, добавить его в DOM дереве).*

*Пользователь находится в том же диалоге, в который и приходит сообщение. В таком случае, PH попросит Painter’а вставить сообщение в конкретный диалог (т.е. список сообщений диалога), после этого, попросит вставить сообщение в список диалогов.*

***Ожидается запрос 121 (список сообщений):***

*Т.е. это тот момент, когда пользователь перешел в диалог, и происходит прогрузка диалога.*

*В таком случае, PH просит Painter’а добавить сообщение в список диалогов, и все.*

***Ожидается запрос 110 (подтверждение доставки сообщения):***

*В таком случае, PH проверяет у SST, в каком диалоге мы находимся. Если это диалог, в который и пришло сообщение, то производим добавление сообщения в диалог, после чего, производим добавление в список диалогов.*

*Если же это другой диалог, то просто добавляем в список диалогов.*

***Приход события нажатия клавиши (запрос 0):***

***Клиент ничего не ожидает (не ожидает никакого запроса):***

*PH получает запрос 0, далее, просит Timer создать таймер на это событие для данного диалога.*

*Далее, PH спросит у SST о том, какой диалог просматривает пользователь. В зависимости от варианта, действуем следующим образом:*

*Если пользователь находится на странице списка диалогов, то PH просит Painter’а, добавить информацию о том, кто пишет сообщение в нужный диалог в списке диалогов.*

*Если пользователь находится в другом диалоге, нежели тот, в который пришло событие. То просто добавляется запись в список диалогов, как и описывалось выше.*

*Пользователь просматривает тот диалог, в который и пришло событие. В таком случае, помимо списка диалогов, надпись о том, кто пишет сообщение, добавится и в список сообщений диалога.*

*После срабатывания таймера, вызовется функция, которая проверит, какой диалог мы просматриваем. Обязательно будет удалена запись из списка диалогов. Если мы просматриваем диалог, в котором и была прорисована запись (эта информация содержится в Timer’е), то удалится и в списке сообщений этого диалога. И все.*

***Ожидается запрос 121 (список сообщений):***

*В таком случае, PH попросит Timer создать таймер на это событие, после этого, попросит Painter’а добавить информацию о написании в список диалогов. И все.*

***Ожидается запрос 110 (подтверждение доставки сообщения):***

*В таком случае, снова, как и описывалось выше, создается таймер. Далее, PH попросит Painter’а добавить эту информацию в список диалогов. Далее, т.к. 110 – подтверждение информации о доставке сообщения, то, значит, пользователь находится в диалоге. PH спросит у SST, в каком диалоге мы находимся.*

*Если это тот диалог, в который и пришло событие, то эта информация добавится в список сообщений диалога (это все делает Painter).*

*Если это другой диалог, то ничего, помимо добавления информации в список диалогов, не нужно делать*