Санкт-Петербургский государственный университет

Кафедра системного программирования

Танков Владислав Дмитриевич

Инструменты коллаборации в Web Modeling Project

Курсовая работа

Научный руководитель:
к. т. н., доцент Брыксин Т. А.

Санкт-Петербург
2017

Оглавление

Введение

Web Modeling Project (WMP) — это проект по созданию полнофункциональной платформы предметно-ориентированного моделирования (Domain-Specific Modeling, DSM платформы) в веб-среде. Под DSM платформой подразумевается инструмент, позволяющий разрабатывать и поддерживать предметно-ориентированные языки. В данном случае речь идёт о визуальных предметно-ориентированных языках.

На кафедре системного программирования уже несколько лет занимаются исследованиями в области разработки и поддержки визуальных предметно-ориентированных языков. В частности, с 2007 года существует проект QReal[1] по созданию DSM платформы для настольных компьютеров. В 2014 году было решено перенести наработки QReal в веб-среду и создать веб DSM платформу. Такая платформа позволила бы существенно снизить требования к мощности используемого при работе персонального компьютера, а в перспективе могла бы дать доступ и мобильным пользователям. Основные работы по проекту были начаты летом 2016 года.

Была реализована базовая функциональность создания и изменения графовых диаграмм, их хранения и авторизованного доступа к ним одного пользователя. Безусловно, это крайне скупой набор функциональности, не отвечающий всем современным требованиям к веб-приложениям и, в частности, к веб-редакторам.

Если обратить внимание на современные веб-редакторы (такие как Google Docs[2], Microsoft Office 365[3]), то важной частью предоставляемых ими функций являются инструменты коллаборации — инструменты для совместной работы над документами. Как правило, это совместное владение документами и их совместное редактирование.

Учитывая практическую важность таких инструментов, было решено реализовать их в Web Modeling Project.

[1]Репозиторий QReal на Github — `https://github.com/qreal/qreal`

[2]Основной сайт Google Docs — `https://docs.google.com`

[3]Основной сайт Microsoft Office 365 — `https://products.office.com/`

Постановка задачи

Целью данной работы является разработка инструментов коллаборации для WMP.

Были поставлены следующие задачи:

1. реализация возможности совместного владения ресурсами (папками);

2. реализация возможности редактирования ресурсов (диаграмм) в режиме online.

1. Обзор

Прежде чем говорить о реализации инструментов коллаборации, требуется обзорно обсудить архитектуру проекта.

Обзор архитектуры проекта

Основным архитектурным подходом в Web Modeling Project является микросервисный подход [5]. Вся функциональность проекта разделена на сервисы, работающие отдельно друг от друга и связанные между собой с помощью протокола удалённого вызова процедур (Remote Procedure Call, RPC протокол).

Для логической группировки сервисов используется абстракция слоёв [4].

В частности, используется классификация сервисов по характеру предоставляемых услуг, соответствующая трёхзвенной архитектуре.

1. Слой клиента — это код, исполняемый в браузере клиента, и предоставляющий графическую оболочку пользователя. Часть запросов, требующих дополнительных вычислительных мощностей или доступа к хранящимся в слое данных сущностям, он передаёт сервисам первой линии.

2. Сервисы первой линии — это сервисы, обрабатывающие запросы в соответствии со внутренней бизнес логикой. В случае необходимости данные сервисы могут обращаться ко слою данных.

3. Слой данных — это группа сервисов, фактически представляющая собой распределенную базу данных, где каждый сервис является некоторой сущностью или сущностями схемы данных, а связи между ними разрешаются с помощью RPC вызовов. Такой подход позволяет добиться чрезвычайной гибкости в выборе технологий хранения и обработки данных [1].

Группировка сервисов в слои позволяет рассуждать о свойствах не

одного сервиса, а группы. Это, в свою очередь, упрощает задачи обеспечения безопасности сервисов, обеспечения взаимодействия между ними.

Наконец, перечислим присутствующие на данный момент в проекте сервисы:

1. Editor service — сервис редактора диаграмм, сервис первой линии;
2. Dashboard service — сервис панели управления редакторами, сервис первой линии;
3. Authentication service — сервис OAuth аутентификации, сервис первой линии, данные изолированы от других сервисов;
4. Diagram DB Service — сервис хранения диаграмм, сервис данных;
5. User DB Service — сервис хранения пользователей, сервис данных.

Используемые на данный момент в проекте технологии не позволяли осуществлять коммуникацию сервер-клиент, что требовалось для реализации инструментов коллаборации, поэтому был проведён обзор существующих технологий такой коммуникации и выбрана подходящая.

Обзор существующих реализаций коммуникации сервер-клиент

Для передачи данных от сервера к клиенту существуют различные технологии. Мы рассмотрим лишь основные с учётом основного требования, накладываемого проектом: выбранная технология должна обеспечивать малую задержку передачи данных и низкую нагрузку на клиент.

Polling

Поллинг (polling) — постоянный опрос сервера на предмет наличия сообщений для передачи. Фактически вместо передачи данных от сервера к клиенту по событию мы заставляем клиент постоянно опрашивать

сервер, узнавая, не произошло ли событие. При этом, чтобы достигнуть низкой задержки, нам потребуется достаточно часто выполнять запросы (к примеру, задержка в 100 миллисекунд потребует опрашивать сервер 10 раз в секунду). Это решение популярно при использовании протоколов без поддержки постоянного соединения. Оно достаточно требовательно к ресурсам, а задержка передачи данных оставляет желать лучшего.

Long Polling

Лонгполлинг (long polling) — опрос сервера на предмет наличия данных для передачи с поддержанием соединения открытым некоторое время. В отличие от поллинга, при лонгполлинге сервер не отвечает, что данных нет, а поддерживает соединение открытым до тех пор, пока данные не появятся или соединение не будет сброшено по инициативе клиента или в соответствии с протоколом соединения. В случае, если на сервере происходит событие и данные появляются, сервер отправляет данные клиенту и закрывает соединение. Клиент должен обработать данные и открыть новое соединение. Данная технология позволяет существенно сократить количество запросов к серверу со стороны клиента, однако она может быть излишне требовательна к ресурсам при передаче большого числа мелких пакетов данных.

Push-технологии

Push-технологии — это группа технологий, позволяющих напрямую передать сообщение клиенту без запросов с его стороны. Polling и Long Polling — это, фактически, эмуляция push-технологий с помощью pull-технологий (технологий, в которых запрос поступает от клиента). Запрос инициирует клиент и некоторое сообщение от сервера является лишь ответом на этот запрос. Это неэффективно.

Как правило, push-технологии реализуют модель издатель-подписчик. Клиент своим запросом подписывается на некоторые события сервера, и когда данные события происходят, клиент получает сигнал об этом.

Простейшими примерами push-технологий являются различные чаты, мессенджеры и современные push-центры мобильных операционных систем (такие как Google Cloud Messaging).

Одной из наиболее популярных реализаций push-уведомлений для бразуеров является STOMP over WebSocket[4]. STOMP — простой текстовый протокол сообщений [3]. Он реализует модель издатель-подписчик и позволяет использовать в качестве транспорта любое полнодуплексное надёжное соединение.

В реализации STOMP over WebSocket транспортом для STOMP служит WebSocket [2] — полнодуплексный протокол передачи данных, использующий в свою очередь в качестве транспорта TCP. WebSocket поддерживается во всех современных бразуерах, а свободно распространяемая библиотека SockJs[5] позволяет обеспечить его поддержку и в устаревших браузерах.

WebSocket не потребляет много ресурсов, а соединение устанавливается единожды и на всё время работы клиента. Большим плюсом данного протокола является полнодуплексное соединение, которое позволяет реализовать передачу инкрементальных обновлений, оставив на RPC лишь передачу больших пакетов данных от клиента к серверу.

Также важно отметить, что WebSocket изначально разрабатывался как протокол для обмена сообщениями в реальном времени. Соответственно, задержка передачи данных сведена к минимуму, что чрезвычайно важно для реализации инструментов коллаборации.

Итог

Была выбрана технология STOMP over WebSocket. Она менее требовательна к ресурсам, чем остальные, обладает куда меньшей задержкой передачи данных, и при этом полнодуплексное соединение WebSocket позволяет отказаться от использования RPC протокола для передачи инкрементальных обновлений диаграмм.

[4]Репозиторий STOMP over WebSocket (stomp.js) на Github — `https://github.com/jmesnil/stomp-websocket`

[5]Репозиторий SockJs на Github — `https://github.com/sockjs`

2. Реализация

Прежде чем приступать к реализации, требовалось провести некоторую подготовительную работу. Это было связано как с накопившимся в проекте техническим долгом, так и с необходимостью обеспечить безопасность вносимых изменений.

2.1. Подготовка слоя данных

На этой стадии проводилось документирование слоя данных, написание пользовательской документации (wiki-страницы) и составление общей схемы данных.

Также на этом этапе создавались модульные регрессионные тесты для сервисов слоя данных. Архитектура слоя данных довольно сложна, любые изменения в коде сервисов могут непредсказуемым образом повлиять на поведение всей системы. Регрессионные тесты позволяют удостовериться, что изменения не содержат ошибок и интеграция их в основной код с большой вероятностью пройдёт нормально.

Наконец, был проведён обширный рефакторинг кода слоя данных. Код был приведён в соответствие со стандартами проекта. Была добавлена поддержка исключительных ситуаций в работе сервисов, в частности, появилась обработка нарушений схемы и обрывов соединения.

Всё это позволило проводить в коде слоя данных изменения безопасно.

2.2. Подготовка клиента

В рамках подготовительной работы на клиенте весь код был переведён на AMD модули. Модуль — это специальный конструкт в JavaScript, позволяющий скрыть переменные, методы и классы (в ECMAScript 6) из глобального поля видимости и придать им некоторый идентификатор, конструируемый по пути до данного файла с исходным кодом. AMD модули — это один из стандартных шаблонов модулей[6].

[6]Спецификация AMD модулей — `https://github.com/amdjs/amdjs-api/blob/master/AMD.md`

В клиентском коде Web Modeling Project используется TypeScript[7] — строго типизируемый язык, компилируемый в JavaScript. В зависимости от параметров компиляции директива “module” из TypeScript будет скомпилирована в JavaScript модуль нужного типа (AMD, ES6 и т.д.). AMD модули были выбраны из-за наиболее полной поддержки во всех версиях браузеров.

В отличие от обычных файлов JavaScript модули нельзя загрузить прямой вставкой в html-страницу. Для этого требуется клиентская библиотека. В Web Modeling Project используется RequireJs[8] как один из наиболее распространённых и хорошо поддерживаемых загрузчиков модулей.

Разбиение на модули позволило обновить используемую версию JavaScript до ECMAScript 6 и исключить возможные проблемы с конфликтами имён, а использование загрузчика модулей позволило загружать исключительно нужные файлы с кодом, а не весь код клиента.

2.3. Изменение схемы слоя данных

Далее было необходимо внести изменения в схему данных, чтобы поддержать возможность совместного владения ресурсами на уровне слоя данных. Изначальная схема данных отражена на Рис. 1.

[7]Сайт TypeScript — `http://www.typescriptlang.org`
[8]Сайт RequireJs — `http://requirejs.org`

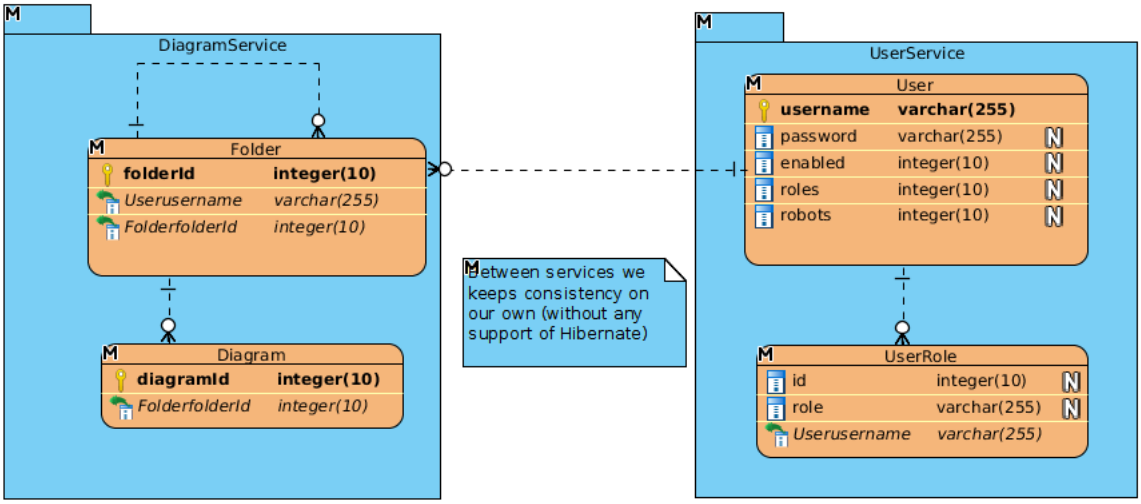

Рис. 1: Изначальная схема слоя данных

Необходимо было заменить отношение “один-к-одному” (пользователь владелец — папка) на “один-ко-многим” (пользователи владельцы — папка) и отношение “один-ко-многим” (родитель папка — дети папки) на “многие-ко-многим” (родители папки — дети папки), так как теперь одна папка в совместном владении может лежать в нескольких папках (по папке родителю на каждого владельца).

Изменённая схема слоя данных показана на Рис. 2.

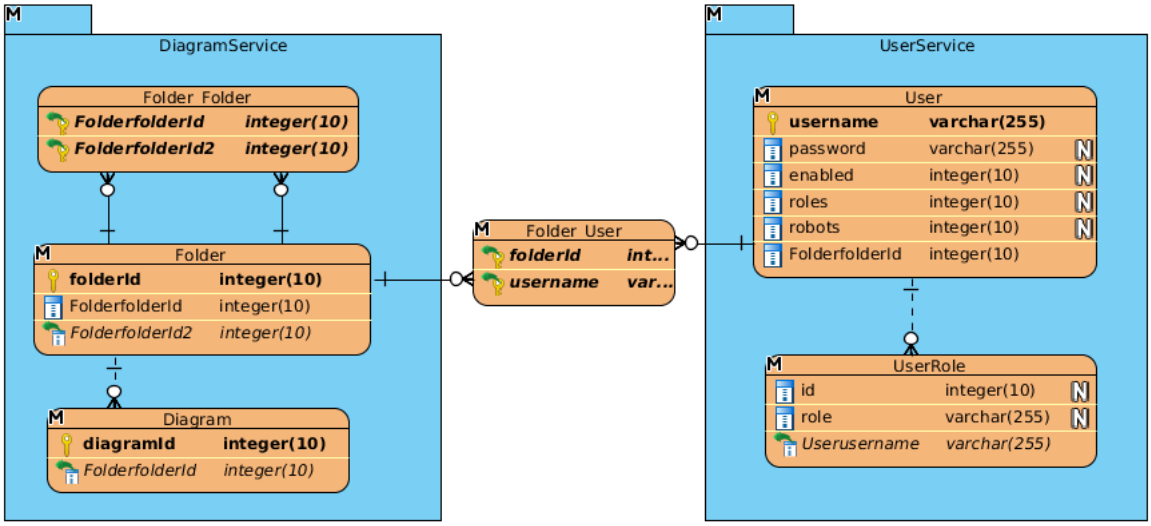

Рис. 2: Новая схема слоя данных

В результате был изменён формат передаваемых между сервисами данных. Если ранее ресурс Folder содержал одного родителя, гарантированно принадлежащего текущему пользователю, то теперь ресурс Folder содержит несколько родителей, все из которых, кроме одного, текущему пользователю не принадлежат.

Это изменение концептуально повлияло на целый ряд аспектов системы.

- Появилась уязвимость в коде хранения ресурсов. Пользователь может изменить множество родителей папки и таким образом повлиять на схему слоя данных. В частности, таким образом пользователь может запретить доступ к папке части её владельцев.

- Передача ресурсов между сервисами усложнилась. Если ранее у каждого ресурса Folder всегда был единственный родитель, передаваемый как идентификатор, то теперь у каждого такого ресурса множество родителей, из которых при отображении необходимо выбрать принадлежащего текущему пользователю. Если передавать только идентификаторы родителей папок, то в другие слои придётся вынести достаточно сложную логику нахождения нужной папки родителя среди всех переданных. Если же передавать сами папки, то использование стандартных механизмов сериализации приведёт к бесконечной рекурсии. К тому же это лишняя нагрузка на сеть, так как мы будем передавать лишние данные, которые всё равно не будут отображены пользователю.

Чтобы разрешить данные затруднения, было введено деление сервисов на слой сессии пользователя и слой инфраструктуры:

- сервисы слоя сессии пользователя имеют доступ к данным текущего пользователя;

- сервисы слоя инфраструктуры имеют доступ ко всем хранимым данным.

Разделением всех сервисов на два таких слоя и реализацией интерфейса, передающего запросы между ними, была решена проблема передачи в пользовательскую сессию данных, не относящихся к данной сессии. При переходе от одного слоя к другому лишние данные отфильтровываются, при переходе обратно — добавляются.

Таким образом была реализована принципиальная возможность совместного владения ресурсами.

2.4. Создание схемы сигналов

Далее было необходимо разработать общую последовательность сигналов, оповещающую всех клиентов о том, что диаграмма была изменена. Приведём диаграмму последовательностей:

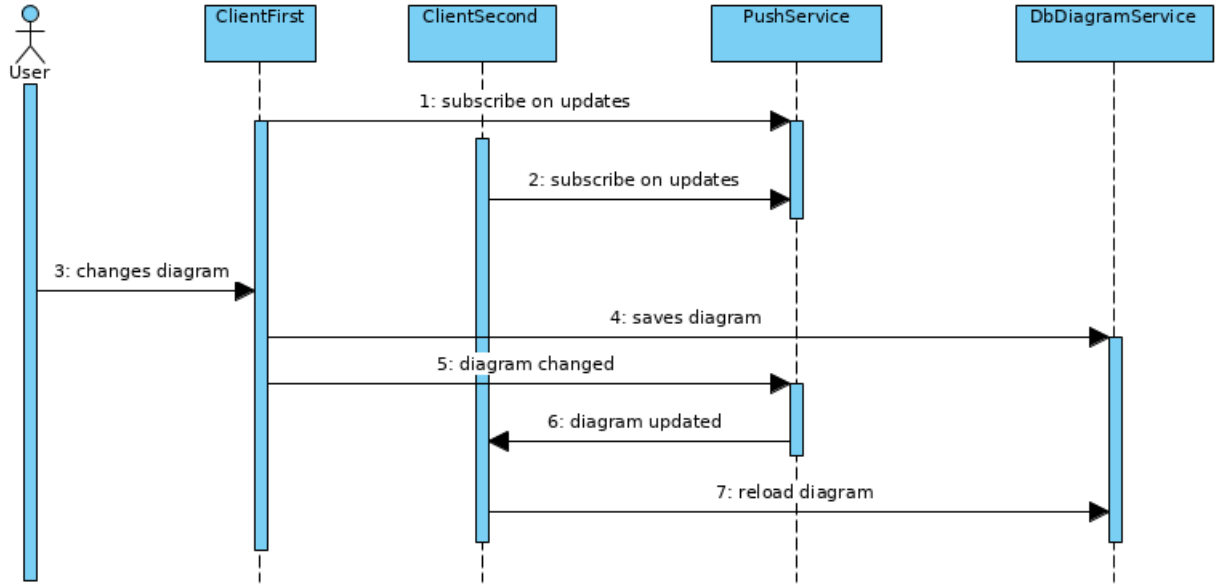

Рис. 3: Схема сигналов

Каждый клиент подписывается на очередь сообщений открытой в данный момент диаграммы (1, 2). После этого, если пользователь изменяет диаграмму (3), она автоматически сохраняется на сервисе хранения диаграмм (4). После этого клиент уведомляет Push сервис о том, что данная диаграмма была изменена (5). Push сервис добавляет в очередь сообщений для данной диаграммы сообщение о том, что она была

изменена и сообщение рассылается всем подписанным клиентам (6). Клиенты обновляют диаграмму (7).

Это простейший пример шаблона проектирования — издатель-подписчик. Издателем является клиент, совершивший некоторое изменение диаграммы. Он создаёт сообщение с указанием темы (идентификатора обновлённой диаграммы) и отправляет его брокеру сообщений — Push сервису. Тот сортирует сообщения по темам и отправляет их подписчикам — другим клиентам.

Рассмотрим некоторые аспекты реализации этой схемы.

2.5. Реализация Push сервиса

STOMP over WebSocket интерфейс Push сервиса позволяет клиентам подписаться на очередь сообщений интересующей их диаграммы. Все приходящие в данную очередь сообщения рассылаются всем подписчикам в соответствии со STOMP протоколом. При этом нужно отметить, что для каждого клиента постоянно поддерживается WebSocket соединение. Это позволяет снизить нагрузку на клиент, но предъявляет повышенные требования к серверу. Впрочем, это решается с помощью стандартных методов балансировки нагрузки.

STOMP over WebSocket интерфейс также предоставляет воможность получать сообщения от клиентов. Это используется для получения сигналов об изменении диаграммы и может использоваться для инкрементального обновления диаграммы, о котором речь пойдёт далее.

2.6. Реализация событийной схемы в клиенте

Заметим, что в серверной части по существу есть лишь одно событие — обновление диаграммы. Сообщение о любом обновлении должно вызывать перезагрузку диаграммы клиентами, и потому нет никакой необходимости разделять обновления на разные подтипы. Однако, в клиентской части разные обновления диаграммы генерируют совершенно различные события. Соответственно, необходимо соотнести различные

обновления диаграмм с единственным результатом — отправкой сообщения об обновлении Push сервису. Сделать это оказалось несколько сложнее, чем могло бы показаться.

В текущей реализации клиента часть изменений диаграммы происходит через отменяемые команды — класс Command, а часть напрямую — через взаимодействие с классом сцены DiagramScene. Это не позволяет найти единую точку генерации события изменения диаграммы. К тому же, хотелось бы дифференцировать события, чтобы в будущем иметь возможность изменять поведение в зависимости от типа произошедшего события. Были добавлены следующие классы — AddElementEvent, DeleteElementEvent, ChangePropertiesElementEvent и др.

Каждый Event класс — это статический класс со списком функций (callback), которые необходимо вызвать при соответствующем событии. Все заинтересованные в некотором событии объекты могут подписать на него некоторую функцию и таким образом получить возможность исполнять некоторый код по сигналу о событии. Фактически, это довольно простая реализации концепции событий для TypeScript. Создавать её понадобилось, так как TypeScript не имеет встроенной поддержки событий, а предоставляемые библиотеки пока плохо совместимы с ECMAScript 6.

Во всех требуемых местах кода клиента были расположены вызовы signalEvent у соответствующих классов событий. При создании же основного класса клиента, DiagramController, на необходимые события подписывается функция сохранения диаграммы, а на сигнал об обновлении диаграммы от Push сервиса (он предоставляется библиотекой STOMP over WebSocket) подписывается функция обновления диаграммы.

Таким образом, при любом изменении диаграммы она будет сохранена, после чего сервис хранения диаграмм оповестит Push сервис об изменении этой диаграммы, а он в свою очередь оповестит всех заинтересованных клиентов, и те обновят диаграмму.

2.7. Инкрементальные обновления диаграммы

Естественно, сохранять и обновлять диаграммы можно разными способами. При нормальной работе пользователь совершает не более десятка событий в секунду и, учитывая то, что WMP на данный момент не поддерживает очень большие диаграммы, вполне приемлем вариант с полной загрузкой и полным же обновлением диаграммы на каждый сигнал. Однако, есть возможность дополнительно снизить нагрузку — применять инкрементальное сохранение и загрузку.

Как мы уже упоминали WebSocket — это полнодуплексный транспорт. Протокол STOMP также поддерживает полнодуплексную передачу. На основе данных протоколов можно реализовать инкрементальное обновление. Приведём обновлённую диаграмму последовательностей (см. Рис. 2.7).

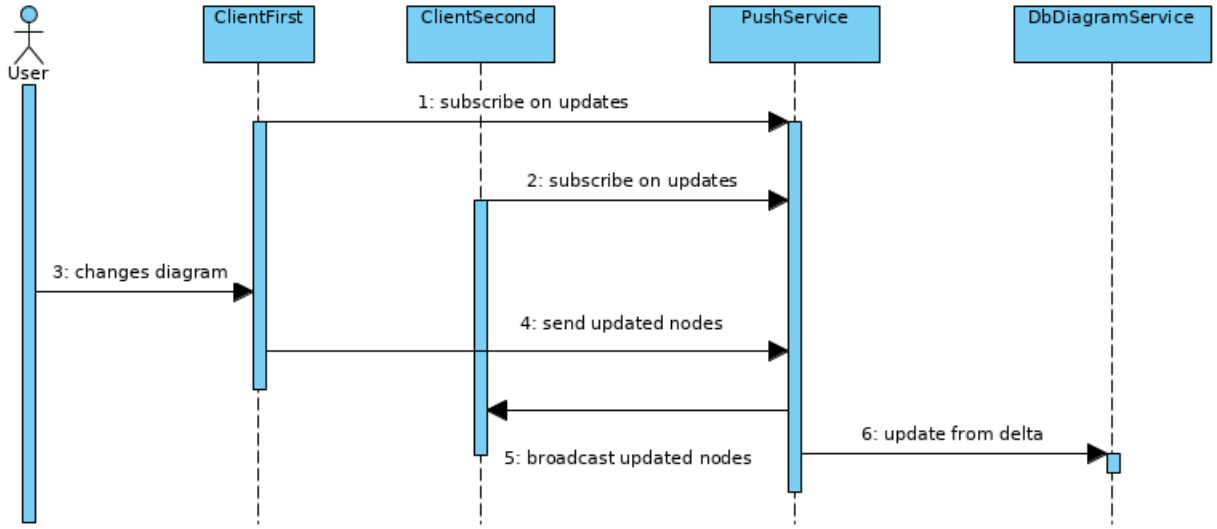

Рис. 4: Схема вызовов

Шаги (1) и (2) полностью соответствуют предыдущей диаграмме. Однако после шага (3) клиент отправляет не просто сигнал об обновлении диаграммы, а всю дельту изменений Push сервису (4). При этом клиенту уже не нужно отправлять отдельно сообщение об обновлении

диаграммы. Push сервис, получив дельту, передаёт её всем заинтересованным клиентам (5), которые применяют её ко своим диаграммам. Теперь и получателям не нужно перезагружать всю диаграмму. Так же Push сервис передаёт дельту сервису хранения диаграмм (6), таким образом обновляя диаграмму в базе данных.

Несмотря на кажущуюся техническую сложность данной схемы, реализация её довольно проста.

Все генерируемые на клиенте события могут на месте генерации получить данные об элементе, который был изменён. Таким образом решается проблема с выделением дельты изменения.

Сериализация и десериализация объектов диаграммы уже реализована для формата Thrift[9] и аналогично может быть реализована для любого другого формат (например, JSON[10] или Protocol Buffers[11]).

Применение дельт к диаграммам при обновление может производиться уже существующими функциями DiagramScene, ну а применение дельт на стороне сервиса хранения диаграмм производится пообъектно с помощью Hibernate[12].

Инкрементальное обновление ресурсов позволило снизить сетевую нагрузку как на клиенты, так и на сервер. Кроме того, в случае больших диаграмм инкрементальное обновление необходимо для нормальной работы клиента.

[9]Сайт Thrift — `https://thrift.apache.org`
[10]Сайт JSON — `http://json.org`
[11]Сайт Protocol Buffers — `https://developers.google.com/protocol-buffers/`
[12]Сайт Hibernate — `http://hibernate.org`

Заключение

В рамках данной работы был проведён обзор текущей архитектуры проекта, существующих технологий коммуникации сервер-клиент.

Были проведены работы по устранению накопившегося технического долга в слое данных, а именно — документирование кода, составление модульных регрессионных тестов и рефакторинг кода.

Было реализовано совместное владение папками. В процессе была изменена архитектура слоя данных. Соответственно была изменена документация, а новая функциональность была покрыта тестами.

Была разработана схема сигналов и предложено её улучшение для поддержки инкрементального обновления диаграмм.

Были проведены работы по устранению накопившегося технического долга в слое данных, а именно внедрение AMD модулей, добавление загрузчика модулей и обновление версии JavaScript до ECMAScript 6.

Было реализовано online редактирование диаграмм и предложено улучшение текущей реализации, позволяющее снизить сетевую нагрузку.

Результаты работы представлены в pull request “longpoll” и ветке “master” в репозитории проекта[13].

[13]Репозиторий проекта на Github — `https://github.com/qreal/wmp/`. Имя пользователя, вносившего изменения — TanVD

Список литературы

[1] Fowler Martin. Polyglot Persistence. — 2011. — URL: http://martinfowler.com/bliki/PolyglotPersistence.html (online; accessed: 17.12.2016).

[2] (IETF) Internet Engineering Task Force. RFC 6455 — The WebSocket Protocol. — URL: https://tools.ietf.org/html/rfc6455.

[3] STOMP Protocol Specification, Version 1.2. — 2012. — URL: https://stomp.github.io/stomp-specification-1.2.html (online; accessed: 20.03.2017).

[4] Михаил Ксензов. Рефакторинг архитектуры программного обеспечения: выделение слоев. — 2004. — URL: http://citforum.ru/SE/project/refactor/ (online; accessed: 17.12.2016).

[5] Ньюмен Сэм. Создание микросервисов. — Питер, 2016. — ISBN: 978-5-496-02011-4.