Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Лаборатория открытых информационных технологий

Дипломная работа по теме

Разработка веб-интерфейса для сервера уведомлений.

Выполнил студент ВВО:

Левин Юрий Владимирович

Научный руководитель:

к.ф.-м.н. Намиот Д. Е.

Лаборатория ОИТ факультета

ВМиК МГУ

Москва 2013 г

СОДЕРЖАНИЕ

1 АННОТАЦИЯ

В работе поставлена задача проектирования и разработки веб-интерфейса для сервера уведомлений, который обеспечивает удобный доступ к функциям управления рассылкой уведомлений и групп уведомлений, а также обеспечивает набор функций для управления подпиской на группы уведомлений.

Предложена архитектура и реализация такого веб-интерфейса, использующего в своей основе асинхронное взаимодействие клиента и сервера (AJAX).

2 ВВЕДЕНИЕ

В последние годы всемирная сеть Интернет играет все более важную роль в жизни человека. Одной из главных тенденций развития Интернета сегодня является организация массового доступа в сеть с переносных устройств (телефоны, планшетные компьютеры и др.) с помощью беспроводных технологий. Возможность постоянно быть подключенным к сети Интернет, позволяет пользователям следить за интересующими их событиями, например, обновлять новостные ленты. Но в силу высокого энергопотребления и ограниченного времени автономной работы переносных устройств, классический подход клиент-серверного взаимодействия, когда клиент всегда запрашивает нужную ему информацию у сервера, является затратным. Технология Push использует другой подход – она сама оповещает клиентов об обновлении содержимого. Самым распространенным применением технологии Push – является рассылка сообщений по подписке. Для каждой мобильной платформы существуют свои службы Push-уведомлений. Например, для iOS - Apple Push Notification Service (APNS), для Android - Google Cloud Messaging for Android (GCM). На базе этих служб создаются системы рассылок со сложной организацией взаимодействия между клиентом и сервером.

Для обеспечения удобного удаленного доступа к функциям некоторой системы посредством сети Интернет применяются веб-интерфейсы. Веб-интерфейс – это совокупность средств, при помощи которых пользователь взаимодействует с веб-сайтом или любым другим приложением с помощью браузера [1]. Удобство данного способа взаимодействия заключается в возможности пользоваться веб-интерфейсом с помощью любого компьютера, подключенного к сети Интернет. Кроме этого, важным

достоинством является постоянная доступность веб-интерфейса для конечного пользователя. В контексте системы рассылок веб-интерфейс служит для организации взаимодействия заказчика рассылок и сервера уведомлений. Заказчик получает удобное, интуитивно понятное средство для организации новых, ведению текущих рассылок и получению статистики по ним.

В данной работе рассмотрена разработка компонентов веб-интерфейса для сервера уведомлений. Для хранения сообщений рассылки используется многоуровневая структура вложенных подразделов (групп уведомлений). Кроме этого, рассмотрен раздел подписки пользователей на группы уведомлений и отображение статистики по рассылкам.

В процессе разработки веб-интерфейса проведен анализ предметной области и имеющихся решений по сервисам push-уведомлений, определены требования к функционалу веб-интерфейса, разработана схема базы данных для хранения информации о рассылках и статистики пользователей, разработан прототип веб-интерфейса.

3 ПОСТАНОВКА ЗАДАЧИ

Разрабатываемый веб-интерфейс можно разделить на 2 части: интерфейс заказчика – предоставляет доступ к функциям создания, редактирования, удаления многоуровневых рассылок, а также доступ к статистике; интерфейс клиента – личный кабинет подписчиков (разрабатывается для мобильных платформ и служит для управления подпиской).

Основная цель работы заключается в проектировании и разработке таких компонентов для сервера уведомлений, с помощью которых легко построить масштабируемую систему веб-интерфейса, которая реализует функционал, предоставляющий возможность взаимодействия с многоуровневыми группами рассылок, их модификацию. Также необходимо предусмотреть возможности подписки на рассылаемые сообщения.

Для достижения данной цели необходимо решить ряд задач:

- Сформулировать требования взаимодействия сервера веб-интерфейса с клиентом (браузером). Определить основные технологии взаимодействия для создания быстрого веб-интерфейса.
- Исследовать предметную область и имеющиеся решения по веб-интерфейсам серверов уведомлений. Выделить, что в существующих решениях не устраивает пользователей систем, что необходимо заменить.
- Проанализировать возможные средства реализации и выбрать наилучшие. Одним из основных параметров, влияющим на выбор инструментария, должно являться быстродействие получаемой системы.

- Спроектировать функционал веб-интерфейса, отвечающий сформулированным требованиям и предусматривающий удобную работу с ним пользователей.
- Разработать схему базы данных для хранения информации о рассылках.

4 ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Рассмотрим существующие в настоящее время реализации веб-интерфейсов для серверов уведомлений. Проанализируем насколько они соответствуют требованиям пользователей.

4.1 AirBop

AirBop – это первая система организации информационных рассылок для мобильной платформы Android реализованная по модели Platform as a Service (PaaS, «платформа как услуга»). Основной идеей данного подхода является предоставление доступа к использованию информационно-технологических платформ размещенных у облачного провайдера [3]. Данная система предоставляет полный функционал для внедрения Push-уведомлений в приложения на мобильной платформе Android. Для реализации отправки Push-уведомлений используется Google Cloud Messaging for Android (GCM).

Работа с платформой начинается с регистрации пользователя в системе. На этом этапе пользователь заполняет регистрационные данные, в том числе пару логин/пароль. После завершения данного этапа требуется пройти аутентификацию. Для этого необходимо ввести пару логин/пароль, указанную при регистрации. Далее платформой проверяется правильность введенных данных и, в случае успеха операции, пользователь получает доступ к функционалу веб-интерфейса сервера уведомлений.

Данная система позволяет быстро интегрировать поддержку Push-уведомлений в любое приложение для мобильной платформы Android. В код проекта интегрируется набор классов, написанный на языке Java (JAR архив).

Данный набор классов, кроме внедрения технологии Push-уведомлений, позволяет собирать статистику использования приложения и отправлять ее платформе AirBop. Полученное после интеграции приложение регистрируется в системе AirBop и генерируется ключ приложения в описываемой системе. AirBop предоставляет веб-интерфейс для регистрации приложения в системе, рассылки сообщений и получения статистики по ней. Веб-интерфейс системы позволяет установить локализацию рассылки сообщений, используя геолокацию, то есть рассылка сообщений производится, основываясь на месте положения переносного устройства на котором установлено приложение. Airbop позволяет задать точный адрес и радиус в километрах от указанного адреса, находясь внутри которого, пользователь получит сообщение. Также в веб-интерфейсе реализован функционал графика рассылки, который позволяет составить график рассылки с точностью до минут.

На рис. 1 представлена форма добавления сообщения. Интерес представляет поле Mode – тип сообщения. Может быть текстовый или JSON.

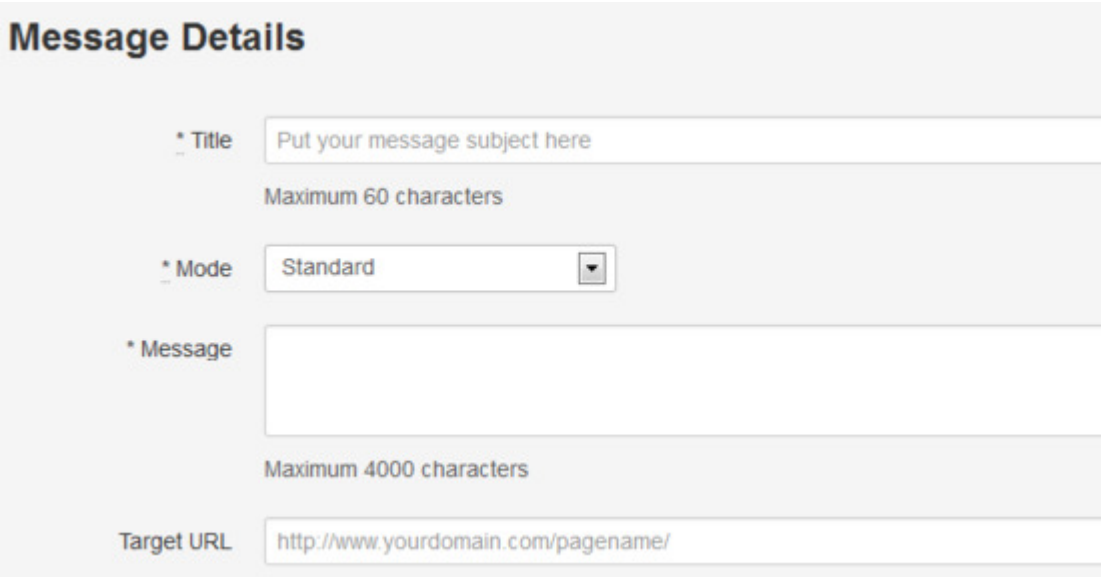

Рис. 1. Форма добавления сообщения.

На рисунке 2 представлен список добавленных в рассылку сообщений. Столбец Status отражает текущее состояние сообщения: draft –

запланированная отправка (не задана дата отправки), queued – запланирована рассылка, sent – сообщение отправлено.

Message Title	Status
Upcomming Events	Draft
Winter Sale!	Queued
Forced Error	Error 401: GCM authentication error
Special Offer	Sent

Рис 2. Список сообщений.

Одним из недостатков данной платформы является то, что реализован только график рассылки, но не график получений сообщений пользователем. Иногда необходимо, чтобы пользователь, использующий приложение, мог установить график получения сообщений. Например, пользователю необходимо получать свежие сообщения из рассылки только по выходным (с 12:00 субботы по 19:00 воскресенья).

Для одного приложения создается единая тематика рассылки. Поэтому в интерфейсе рассылки сообщений отсутствует возможность разделения всех сообщений на подразделы по темам. Подписка на эту тему происходит с помощью установки приложения в мобильную платформу, а отписаться можно удалением приложения. Подписываться на отдельные подгруппы рассылки пользователь не может. То есть, если необходимо реализовать рассылку по двум разным тематикам, придется делать два различных приложения.

4.2 Urban Airship Push

В отличие от AirBop, Urban Airship Push – это платформа для разработки сервиса Push-уведомлений для большинства известных мобильных платформ. На момент написания дипломной работы,

поддерживаются следующие платформы: iOS, Android, Windows, Blackberry и Phonegap. Как и платформа, рассмотренная выше, продукт от Urban Airship представляет собой полный функционал для регистрации пользователей в системе, их авторизации, внедрения Push-уведомлений, организации рассылки, а так же для сбора, анализа и представления статистических данных. Внедрение Push-уведомлений происходит аналогично AirBop [3].

Особый интерес представляет раздел статистики в веб-интерфейсе платформы. Собираемая в приложении статистика отправляется на сервер Urban Airship, затем он обрабатывает полученную информацию, формируя, за выбранный пользователем период, несколько графиков и диаграмм. Платформа позволяет строить следующую статистику:

- количество отправленных сообщений и количество открытий приложения на мобильной платформе,
- среднее время открытия приложения,
- количество подписчиков и их разделение по мобильным платформам.

Веб-интерфейс реализующий добавление сообщения и отображение списка сообщений в рассылке по функционалу идентичен платформе AirBop.

Одним из достоинств Urban Airship Push является наличие API (Application Programming Interface) [4], с помощью которого возможно построение внешних программных продуктов сторонними разработчиками. Набор функций из API полностью дублирует функционал веб-интерфейса, позволяя, при необходимости, разработать дополнительный веб-интерфейс для платформы. Например, веб-интерфейс для мобильных платформ. Кроме этого, API позволяет автоматизировать процесс добавления сообщений в рассылку.

Недостатками данной системы являются отсутствие разделения уведомлений на группы (темы) и возможности подписки на интересующие темы внутри мобильного приложения. Кроме этого, не реализован функционал графика доставки сообщений рассылки, рассмотренный в предыдущем пункте.

4.3 Требования к организации веб-интерфейса для сервера уведомлений

Сформулируем требования к организации веб-интерфейса сервера уведомлений со стороны пользователей:

1) веб-интерфейс должен быть постоянно доступен для пользователей;
2) веб-интерфейс должен обеспечивать механизм аутентификации пользователей;
3) организация удобного доступа к управлению данными сервера уведомлений через веб-интерфейс;
4) веб-интерфейс должен предоставлять механизм подписки.

На основе обзора нескольких платформ, рассмотрим организацию веб-интерфейса сервера уведомлений, удовлетворяющую сформулированным критериям.

1) Веб-интерфейс должен быть постоянно доступен для пользователей.

Основной особенностью веб-интерфейса является его постоянная доступность для пользователей.

2) Веб-интерфейс должен обеспечивать механизм аутентификации пользователей.

Механизмом аутентификации веб-интерфейса является

общепринятый алгоритм аутентификации - аутентификации по паре логин/пароль, используемый, как в AirBop, так и в Urban Airship Push.

3) Организация удобного доступа к управлению данными сервера уведомлений через веб-интерфейс.
Для реализации удобного доступа к данным используется многоуровневая система групп для уведомлений, участвующих в рассылке. Схема, используемая в AirBop и Urban Airship Push – показ всех рассылаемых уведомлений одним списком – при большом количестве уведомлений усложняет восприятие информации пользователем, что делает доступ к данным менее удобным.

4) Веб-интерфейс должен предоставлять механизм подписки.
Подписка пользователя на интересующую его группу (тему) уведомлений происходит в приложении на мобильной платформе. Для этого генерируется страница со списком тем рассылок. Реализовать данный функционал возможно с помощью создания API веб-интерфейса, подобно тому, что реализован в платформе Urban Airship.

Итак, выбранная концепция веб-интерфейса для сервера уведомлений обладает всеми свойствами, необходимыми для того, чтобы организовать веб-интерфейс, удовлетворяющий вышеописанным критериям.

•

5 ИССЛЕДОВАНИЕ И ПОСТРОЕНИЕ РЕШЕНИЯ ЗАДАЧИ

Веб-интерфейс разрабатывается для обеспечения нужд заказчика по созданию, редактированию и удалению, как групп рассылок, так и отдельных сообщений (уведомлений). Заказчик может разделять все сообщения на группы по темам, а клиент (пользователь приложения на мобильной платформе) может подписываться на группы рассылок. Кроме этого, с помощью приложения на мобильной платформе собирается статистика, которая затем обрабатывается сервером уведомлений. Заказчик видит построенное веб-интерфейсом графическое представление обработанной статистики (графики, диаграммы).

5.1 Общая схема

При проектировании веб-интерфейсов в качестве архитектурного решения часто применяются шаблоны проектирования. Это обусловлено следующими факторами:

- позволяет не решать подзадачи с нуля, а использовать решения, оказавшиеся удачными в прошлом [7],
- за счет шаблонов происходит унификация терминологии, названий модулей и элементов проекта.

Разделим веб-интерфейс на три части:

- модель - это объект приложения (база данных),
- вид - экранное представление,
- контроллер описывает, как интерфейс реагирует на управляющие воздействия пользователя.

Данный подход получил название модель/вид/контроллер (Model/View/Controller, MVC). До появления схемы MVC эти объекты в пользовательских интерфейсах смешивались. MVC отделяет их друг от друга, за счет чего повышается гибкость и улучшаются возможности повторного использования [7]. Важной особенностью данного подхода, является то, что модель и вид не имеют прямой связи – они взаимодействуют через контроллер. Взаимосвязь контроллера и вида осуществляется с помощью механизма асинхронных запросов, описанного в следующих разделах.

5.1 Аутентификация

Одной из первых задач, требующих решения при разработке любого веб-интерфейса является аутентификация. Аутентификация – это процедура проверки подлинности пользователя. Рассмотрим основные типы аутентификации.

5.1.1 Аутентификация, основанная на IP-адресе

Простейший алгоритм аутентификации. Сервер хранит таблицу разрешенных IP адресов и/или имен хостов пользователей. Данная аутентификация достаточно проста в реализации, но может быть громоздка для веб-интерфейсов с потенциально большим числом пользователей. Кроме этого, является самым слабым по безопасности способом аутентификации.

5.1.2 Basic-аутентификация

Более надежный алгоритм. Пользователь при попытке попасть в веб-интерфейс должен ввести логин (идентификацию) и пароль доступа, тем самым подтверждая свои права доступа. Данный вид аутентификации поддерживается современными веб-браузерами, что упрощает реализацию данного алгоритма.

5.1.3 Digest-аутентификация

Начиная с версии протокола HTTP 1.1 был введен новый тип аутентификации. Он использует для опознания пользователя механизм запрос/ответа, когда посылается случайное значение (nonce), которое предназначено для использования с логином и паролем. В данном случае информация, вводимая пользователем, соединяется вместе с переданным nonce и запрошенным URL, и вычисляется криптографический хэш, который затем посылается в качестве ответа [5]. Является более сильной с точки зрения безопасности аутентификацией, поскольку пароль не передается в открытом виде.

5.1.4 TLS/SSL

Протоколы TLS и SSL обеспечивают аутентификацию клиента и сервера и шифрование соединения. Поскольку протокол прикладного уровня HTTP выполняется поверх протокола TCP, а TLS/SLL расположен между протоколом TCP и протоколами прикладного уровня, это позволяет использовать TLS/SSL совместно с HTTP [5].

В данной работе основным вариантом аутентификации является Basic-аутентификация. При успешном прохождении аутентификации пользователю присваивается уникальный номер (ID сессии), который участвует в дальнейшем при выполнении асинхронных запросов. Время сессии ограничивается определенным интервалом, после которого необходимо повторить аутентификацию. В целях обеспечения безопасности передаваемой информации возможно использование протокола TLS/SSL.

5.2 Авторизация

Авторизация – процесс подтверждения прав пользователя при попытке выполнения определенных действий [6]. Авторизация в дипломном проекте используется для подтверждения прав доступа при выполнении

асинхронного запроса. В тексте запроса передается сгенерированный при аутентификации ключ, который проверяется сервером на пригодность и, в случае успеха, действие разрешается.

5.3 Шаблонизатор

При построении сложных html страниц, как правило, возникает задача разделения данных и исполняемого кода. Такого рода разделение, позволяет распределить между программистом и дизайнером-верстальщиком работу [10]. Для этих целей применяются специальные html шаблоны. Программист сосредотачивается на программном коде, а дизайнер-верстальщик на html шаблонах. Язык html шаблонов, как правило, достаточно прост и быстр в изучении, что не требует специальных навыков при работе с html шаблонами. Шаблонизатором называется процессор шаблонов, строящий по заданному html шаблону и входным данным (уникальная информация, размещаемая на странице) html страницу.

Поскольку, основным языком, на котором разрабатываются компоненты веб-интерфейса, является Java, ниже рассмотрены популярные шаблонизаторы для данного языка, также оценены плюсы и минусы. Показатели, по которым оцениваются исследуемые шаблонизаторы:

1) сложность интеграции процессора шаблонов и дальнейшей разработки в проекте;
2) разделение представления данных от программного кода;
3) наличие дополнительных программных средств для ускорения разработки с помощью данного шаблонизатора;
4) наличие хорошей документации.

5.3.1 Apache Velocity

Apache Velocity является проектом с открытым кодом, разрабатываемый Apache Software Foundation. Данный процессор шаблонов в качестве входных параметров получает Java объекты и преобразует их в формат, указанный в применяемом шаблоне, находящемся в отдельном файле. Поэтому при подготовке начальных данных достаточно реализовать загрузку данных в классы, которые затем будут передаваться в шаблонизатор. Velocity обладает простым языком [10]. Шаблоны и исполняемый код разделены, поэтому разработка шаблонов для веб-интерфейса не требует глубоких знаний программирования и может быть выполнена дизайнером-верстальщиком. Кроме этого, простой язык процессора шаблонов ускоряет процесс отладки получаемых с помощью Apache Velocity html страниц.

Apache Velocity предоставляет IDE (Integrated Development Environment, Интегрированная среда разработки), позволяющую ускорить разработку шаблонов. Например, Veloeclipse для среды разработки Eclipse или встроенная поддержка в IntelliJ IDE [10].

Интеграция проста: необходимо включить в проект библиотеку данного шаблонизатора и реализовать классы, формирующие начальные данные для html страниц.

Проект обладает хорошо документированным функционалом, что упрощает работу с данным процессором шаблонов.

5.3.2 XSLT

XSLT (eXtensible Stylesheet Language Transformations) — язык преобразования XML-документов. При применении таблицы стилей XSLT, состоящей из набора шаблонов, к XML-документу (исходное дерево)

образуется конечное дерево, которое может быть сериализовано в виде XML-документа. Данный шаблонизатор является универсальным обработчиком XML документов. В контексте веб-интерфейса шаблонизатор XSLT позволяет по сгенерированному дереву XML построить любую html страницу веб-интерфейса. Шаблонизатор в качестве входных параметров получает xslt шаблон (набор стилей для обработки XML дерева) и XML документ, содержащий данные, которые необходимо разместить на html странице. Шаблонизатор просматривает XML дерево и сравнивает элементы (тэги) документа XML с указанными в XSLT стилями. Если подходящий стиль найден, к данному элементу XML применяется преобразование, указанное в стиле XSLT, после чего результат записывается в поток вывода.

Реализации процессора шаблонов XSLT существуют на большинстве языков программирования, что обеспечивает независимость написанных XSLT шаблонов от выбранной среды разработки. Поскольку XSLT является стандартом W3C, шаблонизатор хорошо документирован.

Так как XSLT в качестве входных параметров использует XML, то проект, использующий данный шаблонизатор, должен генерировать XML дерево при каждом запросе. Возможны случаи (например, XML дерево мало), когда доля разметки XML документа в его общем объеме будет достаточно высокой, что ставит под вопрос использование XML.

XSLT использует более сложный язык по сравнению с Apache Velocity. Это увеличивает время, необходимое на обучение работе с XSLT.

Поддержка XSLT встроена в стандартную библиотеку тегов JSP (JSTL), поэтому интеграция в Java проект достаточно проста. Основной задачей при использовании шаблонизатора является задача генерации XML, что при больших объемах выборок замедляет интеграцию.

Так как XSLT является XML документом, средства для разработки XSLT встроены в большинство современных сред разработки.

5.4 Сценарии на стороне клиента

Сценарии на стороне клиента (client-side scripting) – это класс компьютерных программ, предназначенный для исполнения на стороне клиента (браузера) [12]. С точки зрения проектирования интерфейсов сценарии являются неотъемлемой их частью, обеспечивая удобство пользования интерфейсом и скорость его работы. Они также позволяют разгрузить html страницы от неиспользуемых в данный момент элементов. Основным языком, используемым сегодня на стороне клиента, является JavaScript. Для решения часто встречающихся задач на стороне клиента обычно используют готовые реализации функций, объединенные в библиотеки. Сегодня для языка JavaScript написано множество библиотек, решающие целый спектр задач на стороне клиента. В данной работе, интересно рассмотреть библиотеки, реализующие асинхронное взаимодействие с сервером, предоставляющие возможности по построению пользовательского интерфейса и используемые для построения графиков и диаграмм.

Основными показателями, по которым оцениваются рассматриваемые библиотеки, являются:

1) простота интеграции и дальнейшего использования в проекте;
2) наличие документации по внедрению библиотеки;
3) является ли данная библиотека массово используемой;
4) поддержка библиотеки разными браузерами.

5.4.1 Асинхронное взаимодействие

Подход асинхронного взаимодействия клиента и сервера основывается на обновлении только той части интерфейса, которая требует обновления [9].

Данный подход получил название AJAX - Asynchronous Javascript and XML — асинхронный JavaScript и XML. В основе данного подхода для взаимодействия клиента и сервера использовался формат обмена данными XML. Но в дальнейшем от него начали отказываться в силу больших издержек связанных с этим форматом. Основным форматом обмена данных, используемым сегодня, является JSON.

JSON (JavaScript Object Notation) - это текстовый формат данных, позволяющий описать любую структуру данных. На рис. 3 представлен пример записи объекта в формате JSON.

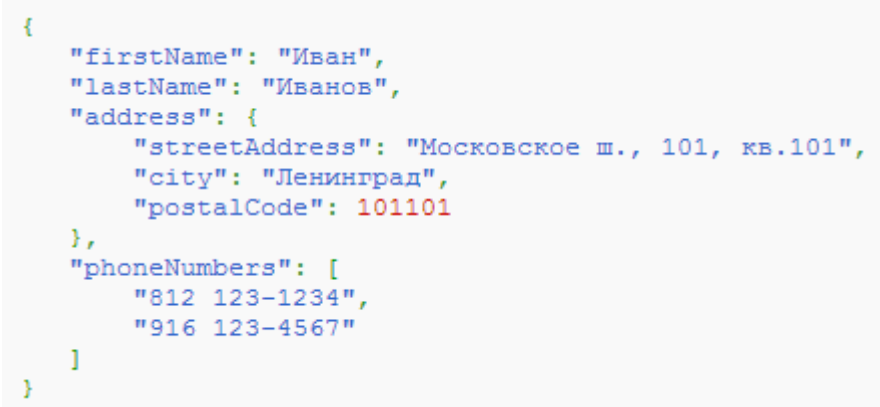

рис 3. Формат JSON

Основной идеей обработки формата JSON является его преобразование в объект JavaScript. Это делает функция eval. Функция принимает в качестве единственного параметра строку. Например, *eval(“{"result":true,"count":1}”);* создает 2 объекта: result со значением true и count равное 1. Потенциальной опасностью данного метода является то, что функция никак не контролирует строку, преобразуемую в объект.

Реализация асинхронных запросов включена в базовый функционал JavaScript с помощью метода XMLHttpRequest. Однако, рассмотренная ниже библиотека дает ряд преимуществ.

JQuery – широко используемая библиотека, фокусирующаяся на взаимодействии JavaScript и HTML. Для реализации асинхронных запросов в

библиотеку добавлена соответствующая функция ajax, имеющая большое количество параметров для настройки асинхронного взаимодействия. Отличием в реализации от базового метода XMLHttpRequest является то, что функция ajax позволяет быстро настроить реакцию сценария на происходящие события во время отправки асинхронного запроса. Например, пока выполняется асинхронный запрос можно показывать специальный элемент HTML страницы с текстом «Пожалуйста, подождите». Интеграция библиотеки проста: достаточно включить файл библиотеки в HTML, и вызвать для определенного элемента на HTML странице функцию JQuery. Существует множество плагинов для библиотеки. Плагины сами являются библиотеками, использующими JQuery в качестве основы и дополняющие его функционал. Например, существует плагин JQuery UI реализующий всевозможные элементы интерфейса на странице [14]. Данные библиотеки имеют достаточно полную и подробную документацию. Из-за того, что они очень популярны сегодня в сети Интернет существует множество литературы по данной тематике. С точки зрения кроссбраузерности – библиотеки работают во всех современных браузерах.

5.4.2 Графические библиотеки

Для отображения статистической информации необходимо строить динамические графики и диаграммы. Сегодня существует целый спектр библиотек, специализирующихся на построении графиков и диаграмм. Выбранная библиотека, дополнительно к требованиям в начале раздела, должна иметь максимально большой функционал по построению графиков и диаграмм. Рассмотрим некоторые из них.

1) Flot [15]. Графическая библиотека с открытым кодом, использующая за основу JQuery. Библиотека имеет полностью документированный функционал с примерами интеграции. Сама

интеграция достаточно проста. Необходимо задать данные для построения графика или диаграммы в формате JSON, указать элемент HTML страницы, где необходимо отрисовать графический элемент библиотеки, а затем выполнить определенную функцию.

2) Highcharts. Библиотека для построения графиков и диаграмм, написанная на HTML5/JavaScript, поддерживает множество различных типов графиков и диаграмм. За основу берется JQuery. Интеграция аналогична библиотеке Flot. По сравнению с Flot предоставляет больший функционал по построению графиков и диаграмм. Данная библиотека хорошо документирована. Также существует печатная книга [20], описывающая возможности данной библиотеки. Вместе с библиотекой поставляются примеры построения основных графиков и диаграмм данной библиотеки.

5.5 Основная задача и ее декомпозиция на подзадачи

Таким образом, после того, как определены требования к веб-интерфейсу сервера уведомлений и названы компоненты, которые будут использованы при разработке, задача дипломной работы формулируется следующим образом: разработка веб-интерфейса для сервера уведомлений, который бы обеспечивал выполнение трех требований:

1) веб-интерфейс должен обеспечивать механизм аутентификации пользователей;
2) организация удобного доступа к управлению данными сервера уведомлений через веб-интерфейс;
3) веб-интерфейс должен предоставлять механизм подписки.

Рассмотрим какими именно методами разрабатываемый веб-интерфейс обеспечит выполнение поставленной цели:

1) *Веб-интерфейс должен обеспечивать механизм аутентификации пользователей.*
Авторизация и аутентификация пользователей происходит с помощью компонента аутентификации. Основным вариантом аутентификации является Basic-аутентификация. При успешном прохождении аутентификации пользователю присваивается уникальный номер (ID сессии), который участвует в дальнейшем при выполнении асинхронных запросов. Время сессии ограничивается определенным интервалом, после которого необходимо повторить аутентификацию. В целях обеспечения безопасности передаваемой информации возможно использование протокола TLS/SSL.

2) *Организация удобного доступа к управлению данными сервера уведомлений через веб-интерфейс.*
Под удобным доступом понимается представление информации в удобном виде, организация быстрого взаимодействия между клиентом и сервером, а также организация информативной статистики. Концепция асинхронных запросов разрабатывалась для увеличения скорости взаимодействия между клиентом и сервером. Представление данных в удобном виде возлагается на компонент шаблонизатор. В данном проекте основным шаблонизатором является Apache Velocity. Для реализации информативной статистики выбрана библиотека Highcharts, как обладающая наиболее большим функционалом из рассматриваемых библиотек.

3) *Веб-интерфейс должен предоставлять механизм подписки.*
Реализация механизма подписки возлагается на API веб-интерфейса. Кроме того, для проверки прав доступа используется компонент авторизации и аутентификации.

Практическая реализация поставленной задачи разбивается на ряд подзадач:

- выбор средств разработки (платформа, СУБД),
- разработка компонентов веб-интерфейса,
- реализация методов взаимодействия между компонентами,
- разработка графического интерфейса сервера уведомлений.

6 ОПИСАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ

6.1 Обоснование выбранного инструментария

Для выполнения требования постоянной доступности веб-интерфейса в качестве основного языка программирования выбран язык Java и веб-сервер Glassfish. Язык Java предоставляет удобные интерфейсы для программирования функций веб-интерфейса (HttpServlet, Filter и др.).

6.2 Используемые технологии

Веб-интерфейс разработан на платформе Java EE 6. Для организации клиент-серверного взаимодействия используются интерфейсы HttpServlet. В свою очередь для препроцессинга запросов используется интерфейс фильтров (Filter). Основной моделью взаимодействия клиент-сервер является асинхронное взаимодействие AJAX.

Для хранения пользовательской информации в проекте используется объектно-реляционная система управления базами данных PostgreSQL. Данный выбор обусловлен надежностью данной СУБД. По результатам автоматизированного исследования в исходном коде СУБД PostgreSQL выявлено 20 проблемных мест на 775000 строк (примерно одна ошибка на 39000 строк кода). Для взаимодействия контроллера с моделью используется промышленный стандарт взаимодействия Java приложений с СУБД, называемый JDBC (Java Data Base Connectivity — соединение с базами данных на Java) [9]. PostgreSQL является бесплатной и свободно распространяемой СУБД.

Для построения веб-интерфейса используется язык разметки HTML в связке с каскадными таблицами стилей CSS. Внедрение динамического функционала в веб-интерфейс реализуется с помощью JavaScript и сторонних библиотек. Для асинхронных запросов и взаимодействия с HTML применяется JQuery, для построения интерфейса пользователя - JQuery UI и Highcharts – для отображения статистической информации.

6.3 Общая архитектура и схема работы веб-интерфейса для сервера уведомлений

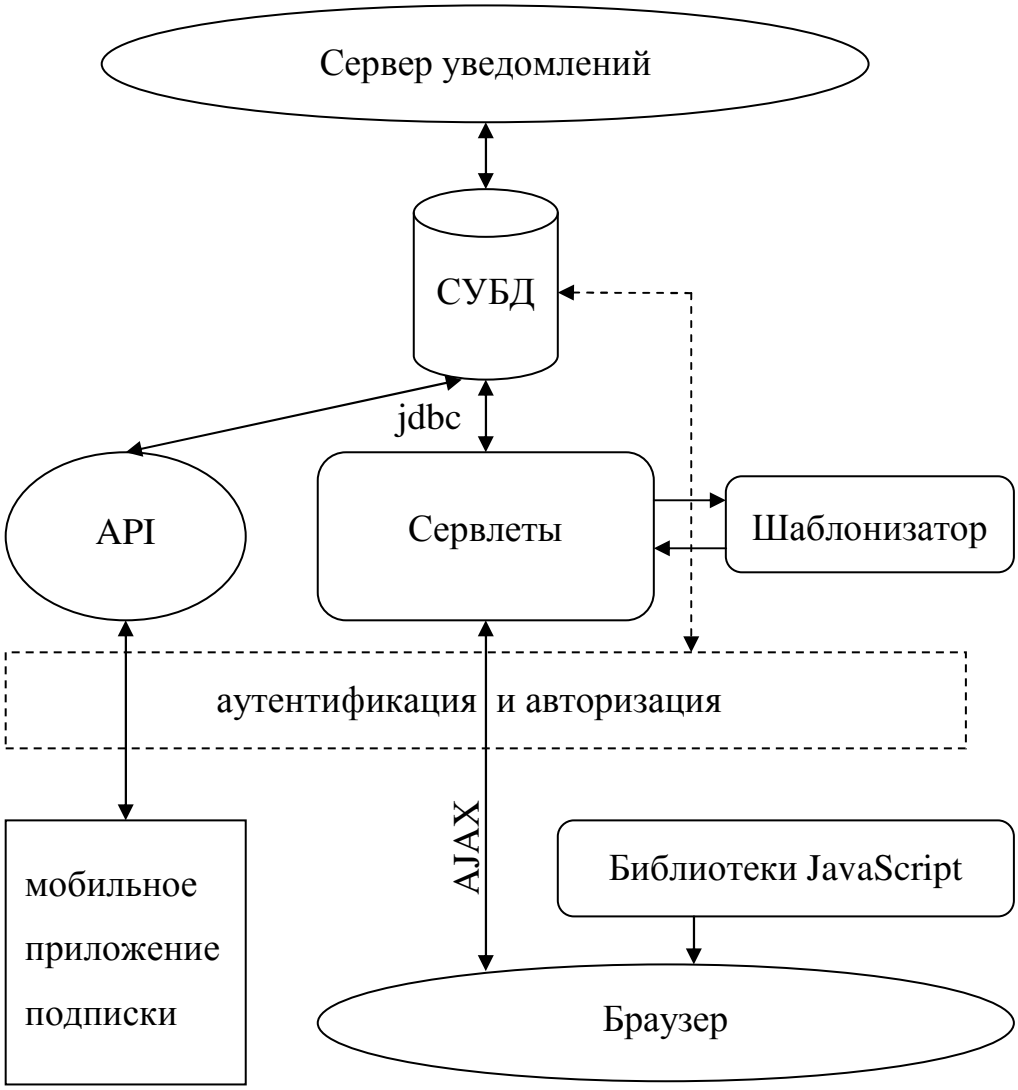

Схема 1. Общая архитектура веб-интерфейса

Вся информация, необходимая для функционирования проекта, хранится в СУБД. Для получения необходимой информации клиент должен пройти аутентификацию веб-интерфейса. Затем пользователь попадает в раздел рассылок. Часть данной страницы загружается синхронно, оставшаяся часть загружается с помощью AJAX (после события onload страницы). API предоставляет функционал для построения сторонних приложений, работающих с веб-интерфейсом для сервера уведомлений. JavaScript библиотеки подгружаются в полученную браузером страницу. Основным протоколом передачи информации в данной схеме является HTTP. Авторизация и аутентификация реализована с помощью интерфейса Filter.

6.3.1 Начальная загрузка html страницы

HTML код, показанный на рис. 4, строится средствами шаблонизатора и сервлета, используемого для отображения входной html страницы. В разделе body созданы: логотип заказчика (img id=”user-logo”), название заказчика и дополнительная информация (блок client-name), блок контента (div id=”tabs”).

```
<html>
<head>
<!-- ... -->
<link type="text/css" rel="stylesheet" href="style.css">
    <!-- javascript -->
</head>
<body>
<img id="user-logo" class="logo" alt=""
    src="/img/no-photo.png"/>
<div class="client-name">
</div>
<div class="clear"></div>
<div id="tabs">
</div>
</body>
</html>
```

Рис. 4 Код каркаса страницы

Дальнейшее построение страницы возлагается на AJAX. После загрузки страницы в браузере срабатывает JavaScript код, блокирующий доступ пользователя к странице веб-интерфейса и реализующий AJAX запросы к контроллеру на предоставление данных о пользователе (блоки “user-logo”, “client-name”) и данных о его компаниях и темах (блок “tabs”). Контроллер возвращает запрошенную информацию адресанту. Затем информация, полученная с помощью AJAX, обрабатывается и записывается в соответствующие блоки на странице веб-интерфейса (информация о компаниях записывается в блок “tabs” и т. д.). После этого веб-интерфейс становится заполненным актуальной информацией, а пользователь получает доступ к управляющим элементам.

Основными управляющими элементами на начальной странице являются ссылки: «Добавить тему», «Настроить компанию», «Статистика по компании» и их аналоги для тем. Редактирование сообщений производится двойным нажатием на сообщение.

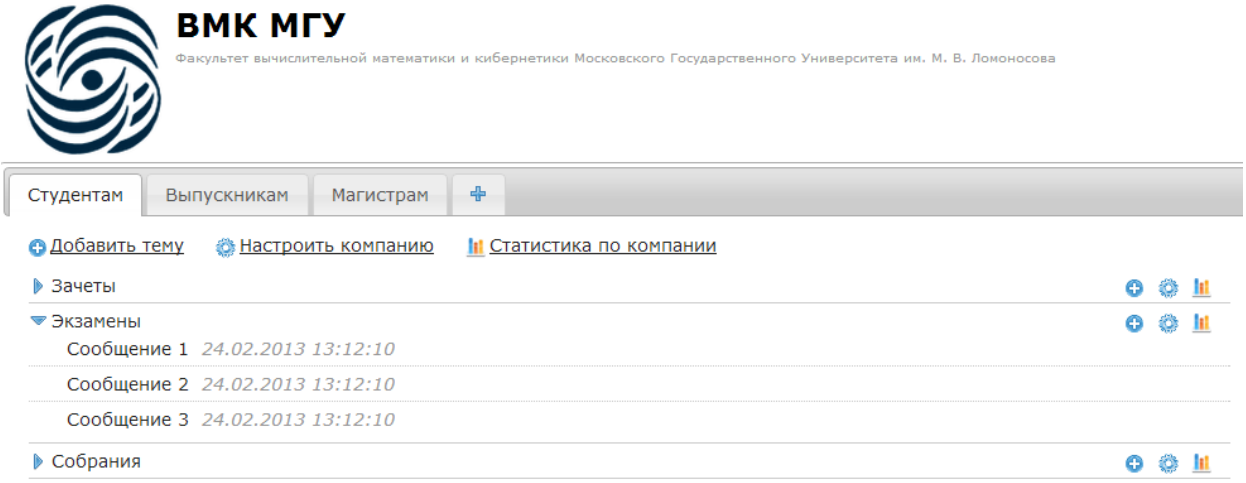

Рис. 5 Основный вид веб-интерфейса

При первом попадании клиента на страницу веб-интерфейса система показывает вид аутентификации. Пользователь заполняет HTML форму (логин/пароль) и отправляет ее. Система, проверив логин/пароль, либо разрешает доступ к веб-интерфейсу, либо запрещает доступ, перенаправляя на вид аутентификации клиента. В случае разрешения доступа между

клиентом (браузером) и сервером создается сессия, действующая фиксированное время.

В качестве примера (рис. 5) заказчиком выбран факультет ВМК МГУ, который ведет различные рассылки по следующим компаниям: студенты, выпускники и магистры. У каждой компании есть темы (здесь показаны темы компании «Студентам»): зачеты, экзамены и собрания. В каждой теме содержатся сообщения рассылки.

6.3.2 Асинхронная загрузка данных

Основным способом клиент-серверного взаимодействия в данной работе является AJAX. Асинхронный запрос начинает выполняться либо при начальной загрузке страницы, либо при нажатии на управляющий элемент. Ниже рассмотрен алгоритм работы асинхронного запроса:

1. Функция формирует запрос к серверу. Более подробно форматы запроса описаны в Приложении 1.
2. Посылается AJAX запрос на определенный URI (Uniform Resource Identifier – унифицированный идентификатор ресурса) и ожидается ответ. Выбор URI зависит от цели запроса.
3. Сервер, получив запрос, производит проверку параметров, и генерирует ответ. Ответ генерируется в следующем формате: *{“success”: 1; “html”:”html code”}*, где success – либо 1, если выполнение запроса успешно, либо код ошибки, html – содержит в себе html блок для вставки в тело html страницы.
4. Сервер, сгенерировав ответ, посылает его адресанту.
5. Клиент получает ответ, преобразует его в объект с помощью функции eval и проверяет код выполнения запроса (поле success).
6. В случае, если код равен 1, JavaScript функция производит вставку html кода в тело html страницы. Если код не равен 1, то

пользователю показывается сообщение об ошибке. Текст ошибки может быть представлен в поле html ответа сервера.

7. Если необходимо, производится запуск функции одной из JavaScript библиотек. Например, при построении вкладок компании, после вставки html кода, необходимо запустить функцию *$("#tabs").tabs();* для визуализации вкладок.

После начальной загрузки страницы, при срабатывании события onload на построенной браузером странице, по алгоритму, показанному выше, выполняется JavaScript код, производящий загрузку компаний (на рис. 5 это пункты «Студентам», «Выпускникам» и «Магистрам»). Затем для первой компании загружается список тем. На рис. 6 представлен один из вариантов поля «html» ответа сервера. Для удобства восприятия html текст представлен в формате дерева.

```
<ul class="themes">
    <li id="1">
    <div class="theme" onclick="roll_mess(this);">Зачеты</div>
    <div class="icons">
    <a href="#" onclick="addMessage(true);return false;" class="btn-add"></a>
    <a href="#" class="btn-conf"></a><a href="#" class="btn-stat"></a>
    </div>
    <div class="clear"></div>
    </li>
    <li id="2">
    <div class="theme" onclick="roll_mess(this);">Экзамены</div>
    <div class="icons">
    <a href="#" onclick="addMessage(true);return false;" class="btn-add"></a>
    <a href="#" class="btn-conf"></a><a href="#" class="btn-stat"></a>
    </div>
    <div class="clear"></div>
    </li id="3">
    <li>
    <div class="theme" onclick="roll_mess(this);">Собрания</div>
    <div class="icons">
    <a href="#" onclick="addMessage(true);return false;" class="btn-add"></a>
    <a href="#" class="btn-conf"></a><a href="#" class="btn-stat"></a>
    </div>
    <div class="clear"></div>
    </li>
</ul>
```

Р

ис 6. Поле html ответа сервера

Отметим, что асинхронный запрос не блокирует дальнейшее выполнение JavaScript кода. В свою очередь, при асинхронной загрузке есть участки, которые должны выполняться строго друг за другом (например, построение списка тем рассылок, возможно только после построения вкладок компаний). Поэтому необходим алгоритм, позволяющий выполнять последовательное выполнение асинхронных запросов. Для синхронизации вызовов функций их вызовы ставятся в очередь с помощью *window.setTimeout(someFunction, 100);*, что гарантирует, что функция someFunction будет выполнена после обработки текущего события. Таким образом добивается последовательное выполнение функций асинхронной загрузки [17].

После окончания построения всех элементов html страницы асинхронные запросы инициирует пользователь, взаимодействуя с управляющими элементами.

6.3.3 Реализация интерфейсов класса HttpServlet

Веб-интерфейс в качестве функций, реализующих построение страницы (в том числе и для асинхронного взаимодействия), использует набор реализаций интерфейса HttpServlet для работы с базой данных. Каждый класс определяется уникальным URI. В качестве входных параметров класс получает данные по HTTP протоколу. Данные передаются с помощью метода GET или POST.

Все необходимые для правильной работы класса параметры проверяются (проходят валидацию). Например, проверяется наличие всех обязательных параметров и правильность их заполнения. При отсутствии ошибок класс, основываясь на параметрах, формирует ответ.

Описание классов приведено в Приложении 1.

6.3.4 База данных

Для использования в дипломном проекте была разработана схема базы данных, представленная на рис. 7 [19].

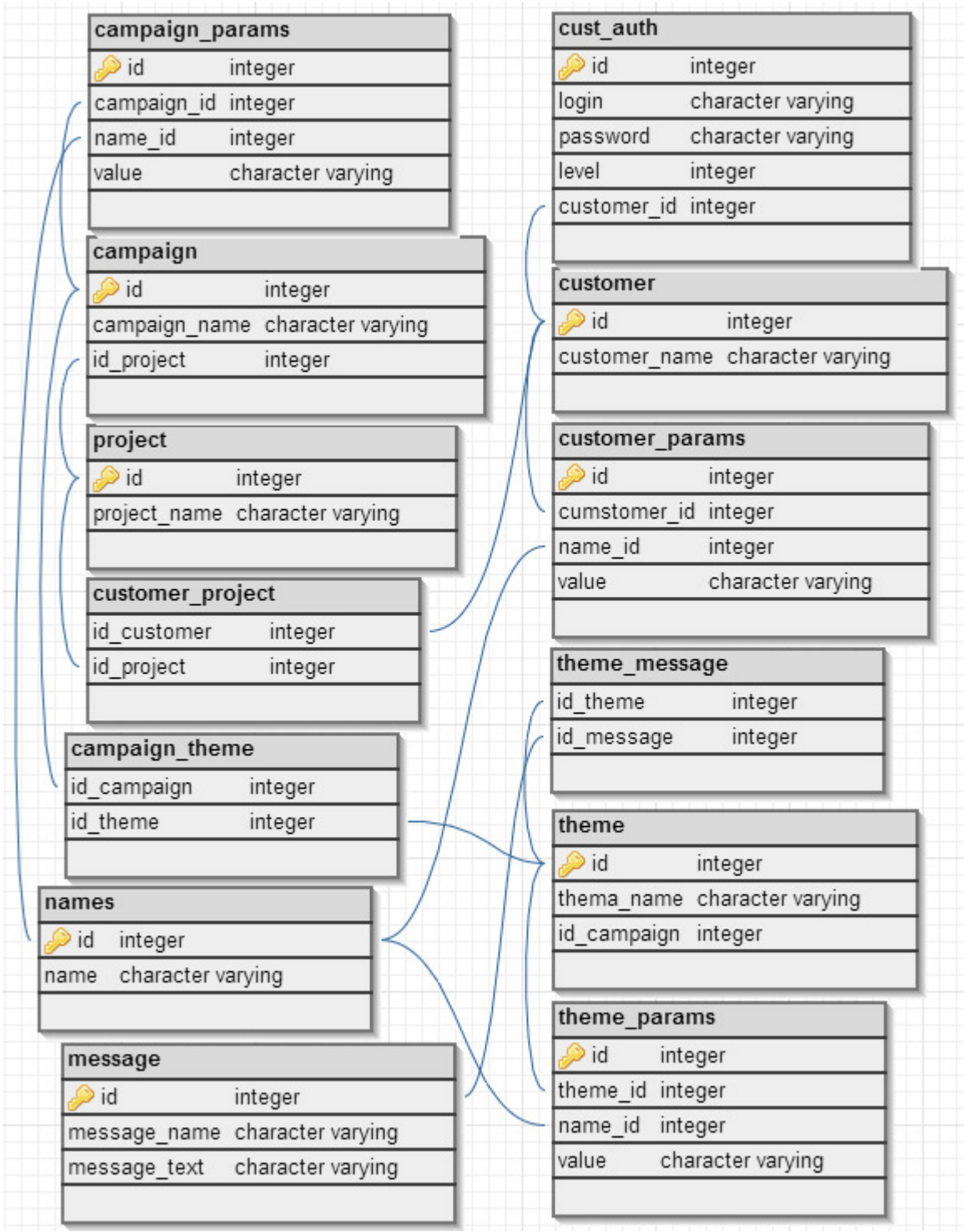

Рис. 7 Схема СУБД веб-интерфейса

Данные таблицы предоставляют трехуровневую иерархию для уведомлений. Эти уровни представлены в таблицах campaign, theme, message. Для первых двух из этих таблиц существуют таблицы параметров. Соответственно, campaign_params и theme_params. Название каждого свойства (name_id) берется из отдельной таблицы свойств – names. Также описаны отношения компания – тема и тема - сообщение. Это сделано в таблицах campaign_theme и theme_message. Для хранения пользователей существует таблица customers. Для хранения их параметров – таблица customers_params. Логин, пароль и уровень доступа для всех пользователей хранятся в таблице cust_auth.

6.3.5 API

Для реализации возможностей подписки на темы сообщений разрабатывается набор функций, доступный любому разработчику посредством сети Интернет. Каждая функция имеет определенный URI и набор входных параметров. В качестве ответа клиенту каждая функция генерирует JSON массив.

При аутентификации пользователя возможно два случая: API используется для организации доступа с переносного устройства, зарегистрированного в системе сервера уведомлений или API используется для дублирования функций основного интерфейса. В первом случае аутентификация может производиться с помощью зарегистрированного в системе id устройства. Иными словами, сразу после установки приложения для мобильной платформы, оно регистрируется в системе сервера уведомлений, получая id устройства. Этот id, может использоваться для дальнейшей авторизации. Во втором случае, пользователь должен получить id сессии, пройдя Basic-аутентификацию. Авторизация обеспечивается указанием уникального номера (id) в теле каждого запроса к серверу.

Полный список функций представлен в Приложении 1.

7 ЗАКЛЮЧЕНИЕ

В работе поставлена задача проектирования и разработки веб-интерфейса для сервера уведомлений, который обеспечивает удобный доступ к функциям управления рассылкой уведомлений и групп уведомлений, а также обеспечивает набор функций для управления подпиской на группы уведомлений.

Для достижения поставленной цели была исследована предметная область и рассмотрены имеющиеся веб-интерфейсы для сервера уведомлений: платформа AirBop и UrbanAirship Push. Выявлены их недостатки и сформулированы требования к разрабатываемому веб-интерфейсу: постоянная доступность для пользователей, обеспечение механизма аутентификации, удобный доступ к управлению данными и реализация механизма подписки. В качестве средств реализации были выбраны язык программирования Java, СУБД PostgreSQL и библиотеки JavaScript.

Разработанный веб-интерфейс превосходит существующие аналоги следующими ключевыми особенностями:

- При разработке веб-интерфейса использовалась многоуровневая иерархия уведомлений (все уведомления разбиваются на группы и подгруппы), что упрощает управление интерфейсом при большом количестве уведомлений.
- Для веб-интерфейса выбрано асинхронное взаимодействие клиента и сервера, что сделало веб-интерфейс более быстрым и интуитивно понятным.

8 СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81
2. http://airbop.com/features
3. http://urbanairship.com/products/push-messaging
4. http://docs.urbanairship.com/reference/api/index.html
5. О.Р. Лапонина Межсетевое экранирование: Учебное пособие. – М: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2007. – 343 с.: ил., табл. (Серия «Основы информационных технологий»).
6. http://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F
7. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2001. — 368 с.: ил. (Серия «Библиотека программиста»)
8. http://ru.wikipedia.org/wiki/Шаблон_проектирования
9. Левин Ю. Разработка веб-интерфейса для сервера уведомлений //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 2. – С. 12-16.
10. http://wiki.apache.org/velocity/
11. http://www.w3.org/TR/xslt
12. http://en.wikipedia.org/wiki/Client-side_scripting
13. http://www.w3.org/TR/WD-script-970314
14. http://jquery.com/
15. https://code.google.com/p/flot/
16. http://www.highcharts.com/
17. http://javascript.ru/tutorial/events/timing

18. Б. Скотт, Т. Нейл Проектирование веб-интерфейсов, - Пер. С англ. – Спб.: Символ-Плюс, 2010 г.352с., ил.
19. D.Namiot "Local Area Messaging for Smartphones." International Journal of Open Information Technologies 1.2 (2013): 8-11.
20. http://dbdsgnr.appspot.com/app
21. http://www.packtpub.com/learning-highcharts-for-javascript-data-visualization/book
22. https://code.google.com/p/google-gson/
23. D.Namiot and M. Sneps-Sneppe. "Mobile Services and Network Proximity." arXiv preprint arXiv:1305.4348 (2013).

9 ПРИЛОЖЕНИЕ 1

9.1 Формат запросов для реализаций интерфейса HttpServlet

Так как класс однозначно определяется своим URI, будем обозначать класс его адресом. Ниже представлены основные классы реализующие интерфейс HttpServlet с описанием параметров (символом * помечены обязательные для заполнения параметры). Перед названием стоит метод, указываемый в HTTP пакете.

GET company – получение компаний конкретного пользователя. Принимает параметры:

- user_id* – уникальный номер пользователя, выданный при успешной аутентификации.

POST company – добавление, редактирование компании. Принимает параметры:

- user_id* – уникальный номер пользователя, выданный при успешной аутентификации;
- company_id – уникальный номер компании в базе;
- company_name* – имя компании;
- company_descr – описание компании;
- company_image – логотип компании;
- company_person – лицо, отвечающее за данную компанию (служебная информация);
- company_phone – контактный телефон.

В зависимости от установленного метода протокола HTTP (GET или POST) формируется различный ответ. Если установлен метод GET,

формируется HTML код для компаний данного пользователя. Если POST, то, в случае успешного выполнения запроса, в переменную html запишется сообщение «Компания успешно добавлена». Если заполнен параметр company_id, происходит редактирование компании. Однако, если компании с таким значением company_id не существует, то формируется сообщение об ошибке.

GET theme – получение тем для данной компании. Принимает параметры:

- user_id* – уникальный номер пользователя, выданный при успешной аутентификации;
- theme_id* – уникальный номер темы в базе.

POST theme – добавление, редактирование тем. Принимает параметры:

- user_id* – уникальный номер пользователя, выданный при успешной аутентификации;
- theme_id* – уникальный номер темы в базе;
- theme_name* – имя темы.

GET message – получение сообщений для данной темы. Принимает параметры:

- user_id* – уникальный номер пользователя, выданный при успешной аутентификации;
- theme_id – уникальный номер темы в базе данных, которой принадлежит сообщение.

POST message – добавление и редактирование сообщения. Принимает параметры:

- user_id* – уникальный номер пользователя, выданный при успешной аутентификации;

- message_id* – уникальный номер сообщения в базе данных, которой принадлежит сообщение;
- message_name* – имя компании;
- message_text* – текст сообщения.

Функция действует по тому же принципу, что и функция company.

GET stats/company – получение статистики по компании. Принимает параметры:

- user_id* – уникальный номер пользователя, выданный при успешной авторизации;
- company_id* – уникальный номер компании в базе.

GET stats/theme – получение статистики по теме. Принимает параметры:

- user_id* – уникальный номер пользователя, выданный при успешной аутентификации;
- theme_id* – уникальный номер темы в базе.

Классы company и message используют метод в HTTP пакете как один из параметров запроса. В остальных функциях метод HTTP запроса не используется и может быть как POST, так и GET.

9.2 Функции API

Клиент и сервер обмениваются информацией с помощью формата обмена данными JSON. Для преобразования объектов Java в JSON и наоборот применяется библиотека google_gson[21]. Нижеописанные функции реализованы с помощью интерфейса HttpServlet.

При каждом запросе клиента сервер должен провести авторизацию клиента. Это делается с помощью секретного ключа master_secret. Ключ

master_secret может быть получен либо при регистрации приложения на мобильной платформе, либо при прохождении аутентификации с помощью функции API.

GET /api/auth – функция для получения master_secret. В качестве параметров указывается следующий JSON объект: *{“login”:”..”, “pass”:”..”};*, где указываются соответственно логин и пароль. В качестве ответа, сервер возвращает *{“master_* secret*”:”..”};*, содержащий master_secret, если аутентификация прошла успешно, master_secret становится идентификатором пользователя на какой-то фиксированный промежуток времени (до окончания сессии). Иначе выдается ключ со значением 0. В дальнейшем master_secret указывается в каждом запросе к API сервера.

GET /api/themes/ - получить список тем. Посылает входные данные: *{“master_secret”:”..”};* В ответ, в случае, если такой master_secret есть в системе, посылает список тем в следующем формате: *{"result": 1, "themes": [{“name": "1","id": 1} ,…, { "name": "2", "id": 2}]};* Если результат выполнения не успешен, то в значении поля result пишется 0, а массив themes оставляется пустым. Если произошла ошибка авторизации, то result = -1.

GET /api/subscribe/ - подписаться/отписаться на тему. Входные данные: *{“master_secret”:”..”, “theme_id”: “..”, “value”: 1};*, где theme_id – идентификатор темы, value – значение либо 0, если нужно отписаться от темы, либо 1, если необходимо подписаться на нее. Ответ сервера: *{"result": 1};* Если результат выполнения успешен, то result = 1. Если ошибка авторизации, то result = -1. В остальных случаях result = 0.