электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 04, апрель 2016

УДК 004.416.6

Проект Knap (Bids). Серверное программное обеспечение

Мазырин Р. А. *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Удалов А. А. *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю. Е., доцент, к.т.н.
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»
gapyu@bmstu.ru

1. Идея проекта

Проект «Bids» представляет собой мультиплатформенную систему, основное предназначение которой предоставить пользователям возможность потренироваться на тренажере по экономике.

Основные особенности:

- игра на малые ставки;
- мультиплатформенность;
- простая система выбора исхода события;
- мощная математическая модель, позволяющая динамически рассчитывать коэффициенты.

1.1. Игра на малые ставки

Так как основное взаимодействие пользователей и системы происходит с мобильных устройств, то нами была предусмотрена система малых ставок, которые было бы удобно делать буквально одним нажатием на экран смартфона. Для этого вводится ограничение на максимальный размер ставки которую может сделать пользователь. За счет этого достигается баланс во всей системе. Пользователь может сделать множество мелких ставок на различные события и выиграть в некоторых из них. В свою очередь в

других он может проиграть, но так как ставки были небольшими, то размеры проигрыша от одних событий компенсируются размерами выигрыша от других событий.

1.2. Мультиплатформенность

Вся система написана для трех основных систем: *Android*, *IOS*, *Web*. В дальнейшем планируется так же написать клиент-приложение для платформы *WinPhone*. Естественно, такая обширная система не может существовать без серверной части. Таким образом, достигается максимальный охват потенциальной аудитории.

1.3. Простая система выбора исхода события

На каждое событие существует только два исхода, среди которых пользователь может выбрать на какое поставить. Это значительно облегчает задачу сделать ставку в пользу того или иного исхода и не позволяет пользователю распыляться среди бесчисленного множества возможных вариантов на что поставить.

1.4. Отличия от подобных систем

Данная система имеет ряд отличий не имеющих аналогов. К достоинствам системы относятся перечисленные выше “1.3 Простая система выбора исхода события”, и “1.1 Игра на малые ставки”.

2. Используемые технологии

В качестве языка программирования для написания серверной части системы был выбран язык *Java*. Аргументы в пользу этого языка были такие, что он довольно быстрый, легко поддерживаемый, популярный, очень хорошо предназначен для написания сервера в силу его синтаксиса.

Так же в качестве основного фреймворка был выбран *Spring 4*, позволяющий довольно быстро и гибко написать *REST API* для взаимодействия с клиентской частью. Так же *Spring* имеет большое сообщество, обширную документацию, множество модулей для написания серверных компонентов и гибкую архитектуру.

Из-за того что система на этапе проектирования обещала высокие нагрузки и большое количество данных, то очень остро встал вопрос о выборе базы данных, а точнее между выбором *SQL* и *NoSQL*. В конце концов, мы пришли к выводу, что нам подойдет обычная *SQL* СУБД, и выбрали *MySQL 5.5*.

Для сборки сервера проекта и подключения в него необходимых библиотек и зависимостей мы воспользовались системой сборки проектов *Maven*. Она позволяет

быстрым действием подключать необходимые компоненты для проектов написанных на *Java*, распространяется под открытой лицензией и является довольно популярным инструментом в комплекте разработчика при написании больших проектов.

В процессе разработки использовались так же такие технологии и парадигмы как аспектно-ориентированное программирование, применение распространенных паттернов проектирования, аннотированные методы, шаблоны DAO, но описание данных технологий выходит за рамки данной статьи.

3. Серверная часть

3.1. *RESTful API*

Рассмотрим, программную часть сервера. Сервер представляет собой приложение на *Spring Framework 4*, использующее *Apache Tomcat* контейнер сервлетов. При запуске, сервер занимает определенный порт и слушает входящие запросы на нем. Так как одна из особенностей нашей системы является то, что она мультиплатформенная, то было решено реализовать *RESTful API*. Что это из себя представляет:

При запуске, сервер начинает слушать на указанном порту входящие запросы по определенным адресам. Эти адреса имеют вид <название раздела>/<название метода>/<параметры>. На каждый запрос, он отвечает определенным сообщением, со статусом ошибки или статусом ОК, а так же с телом ответа. В качестве формата ответа был выбран формат *JSON*, в силу своей простоты и распространенности. Получая ответ от сервера, каждый клиент может по своему обработать его, что в свою очередь позволяет иметь безграничное число видов клиентов и отсутствие необходимости писать огромное количество серверного кода под каждого клиента. Подробнее о *RESTful API* можно ознакомиться в интернете.

Spring Framework очень удобен тем, что позволяет легко создавать новые методы *RESTful API* и расширять существующие. Это достигается при помощи аннотаций, которыми помечаются как методы, так и целые классы. Давайте рассмотрим небольшой кусок кода, который демонстрирует работу одного из методов API.

3.2. Пример кода с пояснениями

```
=========================================================
@RestController
@RequestMapping(path = "api/v1/user/", produces =
MediaType.APPLICATION_JSON_VALUE)
public class User {
```

```
  @Autowired
  UserDAO userDAO;

  @RequestMapping(path = "/profile", method = RequestMethod.GET)
  @AuthorizedOnly
  public String profile(HttpServletRequest request,
                        HttpServletResponse response,
                        @RequestParam(name = "access_token", required = false) String token) {

    Bids.main.Models.DataAccessModels.User user = userDAO.findByAccessToken(token);
    if (user != null) {
      ApiResponse res = new ApiResponse(StatusList.OK, new
JSONObject(user.getParamsAsHashMap()));
      return res.getAsJson();
    } else {
      return new ApiResponse().getAsJson();
    }
  }
}
==================================================================
```

Разберем более детально приведенный фрагмент кода: в основном мы будем обращать внимание на аннотации, которые как раз избавляют нас от большого количества ненужного кода, делая за нас большую работу как на стадии компиляции, так и во время выполнения.

@RestController - данная аннотация показывает, что класс User является классом, который может обрабатывать *RESTful API* запросы в своих методах. Использование этой аннотации избавляет нас от необходимости делать вручную обработку входящих запросов и выявление необходимого класса для их обработки.

@RequestMapping - с помощью этой аннотации мы указываем, какой именно запрос будет обрабатывать данный метод и какой тип ответа от него ожидать. Если эта аннотация применяется к классу, то ее параметры применяются ко всем методам-обработчикам и расширяются другими аннотациями @RequestMapping.

@Autowired - избавляет нас от необходимости писать код для инициализации компонентов. Написав эту аннотацию, Spring Framework сделает за нас всю грязную работу по рутинной инициализации. Чтобы компонент был верно инициализирован, его класс должен быть помечен как @Component.

@AuthorizedOnly - наша собственная аннотация. Используется для *RESTful API* методов, к которым может получить доступ только авторизованный пользователь. Проверка авторизации происходит в *аспектах*, которые будут рассмотрены далее. Модель авторизации так же кратко будет рассмотрена в этой статье.

3.3. Аспекты. Краткая теория

Во время написания данного проекта, мы воспользовались такой технологией, как аспектно-ориентированное программирование (АОП), или программирование на аспектах. Что это такое и для чего это нужно? Если кратко, то аспекты - это функции, содержащие в себе весь тот функционал, который не относится напрямую к какому-то методу. Возьмем для примера метод получения профиля пользователя. Основная задача данного метода это получение всех данных о пользователе из базы данных и отдача их клиенту. Но помимо этого, требуется сделать очень большое количество действий до того, как достать пользователя из базы: проверить права доступа, залогировать что-то, проверить наличие пользователя вообще и еще много того, что не относится напрямую к методу получения профиля. Для этого функционала создается отдельный метод, или даже много отдельных методов, для каждого функционала свой. Затем, указывается, в каких случаях вызывать этот сторонний функционал над основным методом. При компиляции, *Spring* создаст прокси-классы, содержащие весь функционал, и основной или сторонний в одном методе, но это уже будет скрыто от глаз разработчика. На этапе же разработки и проектирования, аспектно-ориентированное программирование очень удобно, так как позволяет сосредоточиться на необходимых вещах. Для использования АОП необходимо подключить модуль *Spring* называемый spring-aop.

3.4. Модель аутентификации

Кратко затронем тему авторизации. Так как вся система работает через *RESTful API*, то стандартная аутентификационная модель через cookie как в браузере нам не подошла. Мы использовали аутентификацию по access_token’у. Весь процесс аутентификации работает по следующему алгоритму: сначала пользователь отправляет заявку на получение кода. Для этого он делает запрос на метод auth/code, передавая в качестве параметра свой телефон. На сервере генерируется случайный 4-значный

цифровой код, а так же хеш из 128 символов. Хеш является уникальным ключом пользователя, который показывает что именно этот пользователь отправлял заявку на получение кода, и именно этот пользователь получил данный код. После генерации кода и хеша, код отправляется клиенту по СМС, а хеш отдается сервером в ответе. После того как клиент получил хеш и код, он должен отправить запрос на метод auth/authorize в который должен передать параметры: хеш, код и тип платформы. Тип платформы - это одно из 4 значений - android, ios, web, winphone, которое показывает, с какой платформы был осуществлен вход. Если все параметры переданы правильно, то пользователю выдается access_token, который он должен передавать с каждым запросом который требует авторизации, для подтверждения своей личности.

4. Сложности с которыми мы столкнулись

Во время таких крупных проектов не обходится и без проблем, которые приходится решать разными путями и способами. Мы не исключение, и поэтому опишем некоторые из проблем, которые встретились нам во время разработки, и с которыми нам удалось справиться.

Одна из самых главных проблем была оптимизация нагрузки. Так как сервис предполагает большое количество вычислений и работу с большим количеством пользователей, то скорость отдачи ответа от сервера должна быть на соответствующем уровне. Данную проблему нужно решать несколькими путями, один из которых правильное распределение потоков в приложении. Так как выделение потока это довольно дорогостоящая с точки зрения ресурсов компьютера операция, то мы использовали пул потоков, который помог увеличить производительность в несколько раз, но недостаточно для того, чтобы мы остались довольны. Следующим шагом была оптимизация базы данных. База данных являлась довольно узким местом при работе сервера, и поэтому приходилось оптимизировать её всеми силами. Мы построили индексы для часто выполняемых запросов, оптимизировали конфигурацию, пересмотрели структуру таблиц, и в конце концов добились желаемого результата.

Следующей проблемой была нехватка времени для реализации всего необходимого функционала. К сожалению, данная проблема не относится к тому типу проблем, которые решаются по мере их поступления. Пожалуй, это самая острая проблема.

Так же, к большому удивлению, оказались проблемой слишком большие возможности *Spring Framework*. Каждый раз при написании нового функционала приходилось тратить время на изучение всех возможностей *Spring*, которые могли помочь при написании. И почти во всех случаях вариантов оказывалось больше чем два, и

приходилось выбирать, что соответственно отнимало время и подводило нас к рассмотренной выше проблеме.

5. Дальнейшие перспективы

В дальнейшем планируется поддержка платформы WinPhone. Так же требуется

написать большое количество модулей для работы с администраторской частью, написание системы учета предпочтений участников.

Несомненно, требуется большое количество событий, на которые можно сделать ставку. В следующих версиях планируется реализовать автоматическую систему добавления событий, которые будут браться из новостных источников.

Список литературы

[1] Spring Developer Support. Available at: http://spring.io, accessed 01.10.2015

[2] «Хабрахабр» - крупнейший в Европе ресурс для IT-специалистов. Режим доступа: http://habrahabr.ru (дата обращения 01.10.2015) .

[3] Фримен Э., Фримен Э., Сьерра К., Бейтс Б. Паттеры проектирования. СПб.: Питер, 2015. 656 с.: ил. [Freeman E., Robson E., Bates B., Sierra K. Head First Design Patterns. O'Reilly Media, 2004. 694 p.]

[4] Уоллс К. Spring в действии. СПб.: ДМК Пресс, 2014. 752 с. [Walls C. Spring in Action. 4th ed. Manning, 2014. 624 p.].

[5] Java documentation. Available at: https://docs.oracle.com/javase/tutorial/, accessed 01.10.2015.

[6] Статья про ajax. Режим доступа: https://ru.wikipedia.org/wiki/AJAX (дата обращения 15.10.2015).

[7] Форум вопросов и ответов для программистов. Режим доступа: http://stackoverflow.com/ (дата обращения 01.10.2015).