

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет информатики, математики и компьютерных наук

Программа подготовки бакалавров по направлению

09.03.04 Программная инженерия

Бакаев Никита Александрович

КУРСОВАЯ РАБОТА

Интернет-магазин (серверная часть, инфраструктура)

Научный руководитель
Старший преподаватель
Сорокоумов Андрей Викторович

Нижний Новгород, 2016

Содержание

1. Введение	2
2. Архитектура приложения	3
3. REST API	4
4. Swagger	6
5. MVC	7
6. Data Layer. Repository	9
7. Основные модели	11
8. MongoDB и sharding	13
9. Docker	15
10. Параметры для запуска	16
11. Запуск	18
12. Загрузка файлов. AWS S3	19
13. Тестирование	21
14. Continuous Integration	23
15. Доменная область	25
16. Защита информации	27
17. Дальнейшее развитие	28
18. Заключение	30
Список литературы	31
Приложения	32

Введение

В современное время развития IT технологий многие аспекты человеческой жизни перетекают в Интернет. Одним из таких проявлений может служить коммерция: продажа товаров и услуг. До развития Интернета, продажа осуществлялась оффлайн. Однако с появлением сети Интернет, как связующего звена множества людей, появился еще один канал коммуникации людей. В последние годы, серьёзное развитие получило направление **e-commerce**, электронная коммерция, продажа товаров через Интернет.

Площадками для таких продаж служат множественные интернет-магазины, как российские, так и зарубежные, глобальные. О разработки серверной части интернет-магазина и пойдет речь в данной работе. Разработка интернет-магазина включает множество составляющих, однако в данной работе вниманию будет уделен лишь технический аспект данного вопроса.

Основная часть

Архитектура приложения

Приложение написано на **Java** и использует Spring как Dependency Injection Container. Активно используется паттерн **MVC** для REST. **DI** через конструкторы, где это возможно и через сеттеры в остальных местах.

Для запуска приложения используется **Spring Boot**. Это современное решение, созданное для создания микросервисов и включает в себя embedded сервер приложений, автоконфигурация некоторых популярных решений, включая логирование и многое другое. В данном случае в качестве встроенного сервера используется embedded tomcat. Это позволяет запускать приложений без дополнительного уровня - сервер приложений и deploy на него. При этом возможен полный контроль над встроенным сервером.

Приложение рассчитано на работу по REST и не использует состояний на сервере, как сессии и прочее. Это позволит просто использовать горизонтальное масштабирование в будущем. Все изменений сбрасываются в persistence storage (MongoDB).

В качестве базы данных используется **MongoDB**. Именно эта NoSQL база данных выбрана в качестве основной из-за высокой скорости работы, возможности горизонтального масштабирования из коробки и отсутствия явной схемы на стороне базы данных.

REST API

Основой архитектуры служит клиент-серверная архитектура. Клиент (браузер, JavaScript) обращаться к серверу через публичный API (средствами асинхронных запросов). API представляет из себя REST сервис. Все endpoints версионированы по URL. Базовый URL для всех запросов на данный момент *протокол:адрес_сервера:порт/api/v1/*.

Возможна работа только по HTTPS. Например, <https://s2.nbakaev.ru/api/v1/>. Все запросы будут иметь ввиду именно этот адрес как относительный.

Для передачи данных по умолчанию используется распространённый формат сериализации JSON. Однако, возможно использование XML, в случае использования при запросе HTTP заголовка Accept: application/xml.

Пользователи:

- Получение всех пользователей (GET /users). Только админ
- Регистрация (Добавление нового пользователя). Требуется email и пароль для нового пользователя (POST /users). Добавляет пользователя и отправляет на указанный email приветственное письмо об успешной регистрации
- Обновление данных о пользователе (PUT /users). Только админ
- Удалить пользователя (DELETE /users/{id} где {id} - ID пользователя). Только админ

Общее:

GET / - Health-checking

Покупки:

- Получить список всех покупок (GET / purchase). Только админ
- Совершить покупку (POST / purchase).
- Редактировать данные о покупках (PUT / purchase). Только админ
- Удалить покупку (DELETE /users/{id} где {id} - ID покупки).
Только админ

Товары:

- Получить список всех товаров (GET /good).
- Добавить товар (POST /good). Только админ
- Редактировать данные о товаре (PUT /good). Только админ
- Удалить товар(DELETE /good/{id} где {id} - ID товара). Только админ
- Получить список товаров, которые нужно показать на главной странице (GET /good/category/index).
- Получить список товаров по фильтру (POST /good/filter). Можно использовать регулярные выражения по одному или более полям (например поиск по имени или описанию), фильтры по цене, дате создания, категориям и многое другое. Поддерживается постраничный вывод (pagination)
- выгрузка товаров в excel (POST /good/excel/download) для выгрузки по определенному критерию (критерии аналогичны API POST /good/filter), либо (GET /good/excel/download.xlsx) для выгрузки всех товаров.

Хранение данных:

- POST /storage/upload. Загрузка любого файла в облачное хранилище AWS S3. Подробнее в секции AWS S3. Возвращается публичная ссылка на загруженный файл.

Swagger

В идеи REST есть одна серьёзная проблема. В отличии от протоколов, где есть валидация данных и жесткая структура, в REST фактически нет никаких соглашений те клиент API фактически не знает, что ему вернет сервер в ответе. Остаётся лишь надеяться, что документация API актуальна. Кроме того, невозможно сформировать клиента(или SDK) под любой язык из-за проблем выше. Как способ решения этой проблемы есть swagger.

Swagger определяет структуру обмена сообщений, включая HTTP URL, объекты, что передаются и прочее. По сути, это JSON файл в котором определен актуальный API сервера.

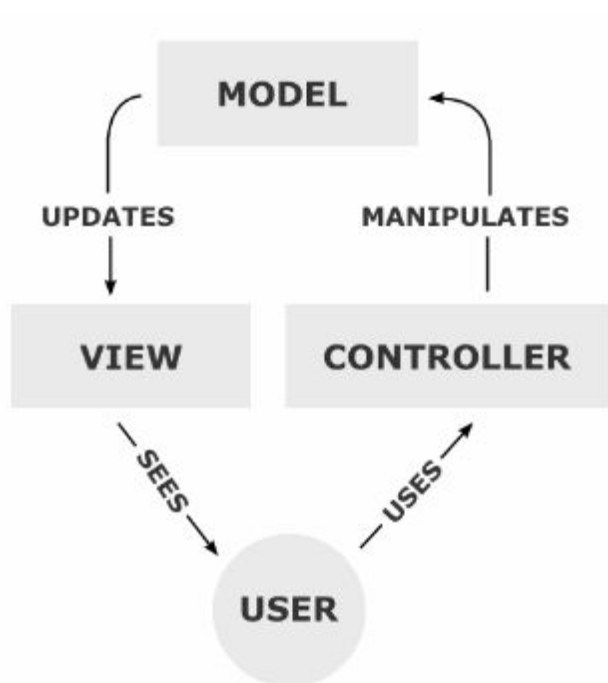
Кроме того, этот JSON имеет определенную структуру и благодаря этому возможно автоматически сгенерировать клиенты¹(SDK) под более чем 40² языков программирования и платформ.

¹ проект Swagger Code Generator <https://github.com/swagger-api/swagger-codegen>

² <https://github.com/swagger-api/swagger-codegen#customizing-the-generator>

MVC

Model-view-controller (MVC, «модель-представление-контроллер», «модель-вид-контроллер») — шаблон проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.



Реализация паттерна MVC и пример HTTP контроллера для работы с объектом товар.

```
@Controller
@RequestMapping("/api/v1/good")
@Api("Goods")
public class GoodController {

    private final GoodRepository goodRepository;
    private final GoodExcelExportService goodExcelExportService;

    @Autowired
    public GoodController(final GoodRepository goodRepository, final GoodExcelExportService goodExcelExportService) {
        this.goodRepository = goodRepository;
        this.goodExcelExportService = goodExcelExportService;
    }

    @Secured({UserAccountRoles.ROLE_ADMIN})
    @RequestMapping(value = "", method = RequestMethod.POST)
    public
    @ResponseBody
    Good addGood(@RequestBody Good good, HttpServletRequest request) {
        goodRepository.createGood(good);
        return good;
    }

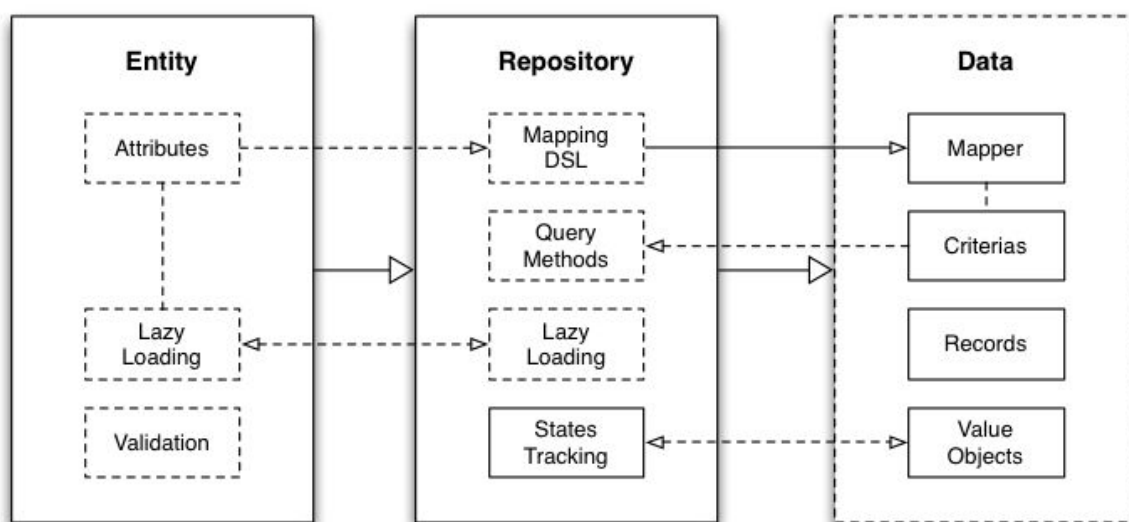
    @RequestMapping(value = "", method = RequestMethod.GET)
    public
    @ResponseBody
    List<Good> getAllGoods() {
        return goodRepository.getAllGoods();
    }

    @Secured({UserAccountRoles.ROLE_ADMIN})
    @RequestMapping(value = "", method = RequestMethod.PUT)
    public
    @ResponseBody
    Good updateGood(@RequestBody Good good, HttpServletRequest request) {
        goodRepository.updateGood(good);
        return good;
    }

    @Secured({UserAccountRoles.ROLE_ADMIN})
    @RequestMapping(value = "{id}", method = RequestMethod.DELETE)
    public void deleteGoodById(@PathVariable("id") String id) {
        goodRepository.deleteGoodById(id);
    }
}
```

Data Layer. Repository

Репозиторий - это паттерн, представляющая собой идею хранения коллекции для сущностей определенного типа. Другими словами, для доступа к объектам определенного типа используется сервис посредник, который выполняет работу с базой данных и знает как выполнять различные операции, например, CRUD в базе данных и этим объектом.



Этот паттерн позволяет абстрагироваться от конкретной реализации системы хранения в базе данных и зачастую реализует Object-relational mapping (ORM) для создания объектов в конкретном языке программирования из данных на основе данных из иного хранилища.

Пример сервиса по работе с объектом товар. CRUD операции.

```
@Service
public class GoodRepositoryImpl implements GoodRepository {

    private final MongoOperations mongoTemplate;
    private final EntityFilterService entityFilterService;

    @Autowired
    public GoodRepositoryImpl(final MongoOperations mongoTemplate, final EntityFilterService entityFilterService) {
        this.mongoTemplate = mongoTemplate;
        this.entityFilterService = entityFilterService;
    }

    @Override
    public List<Good> createGoods(List<Good> goods) {
        mongoTemplate.insert(goods, Good.class);
        return goods;
    }

    @Override
    public Good createGood(Good good) {
        mongoTemplate.insert(good);
        return good;
    }

    @Override
    public List<Good> getAllGoods() {
        Criteria criteria = new Criteria();
        Query query = new Query(criteria);

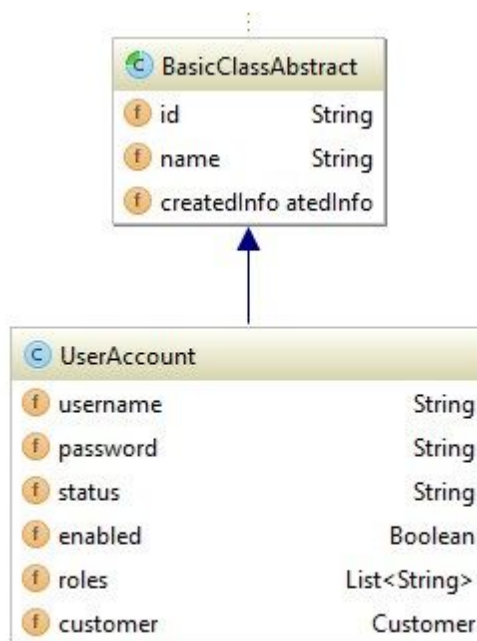
        return mongoTemplate.find(query, Good.class);
    }

    @Override
    public Good updateGood(Good good) {
        mongoTemplate.save(good);
        return good;
    }

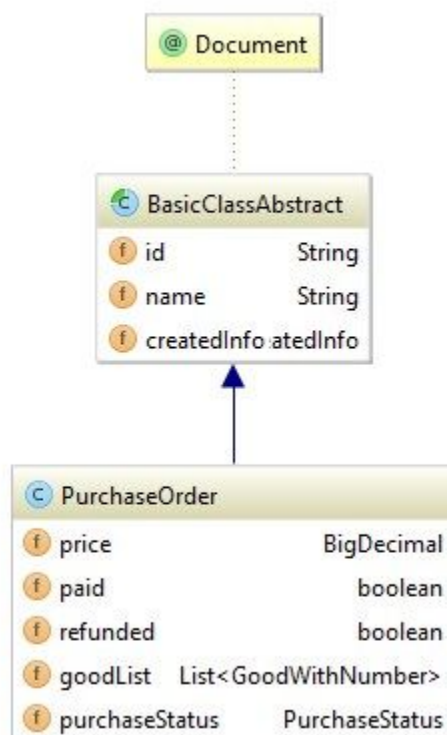
    @Override
    public void deleteGoodById(String id) {
        Good good = new Good();
        good.setId(id);
        mongoTemplate.remove(good);
    }
}
```

Основные модели

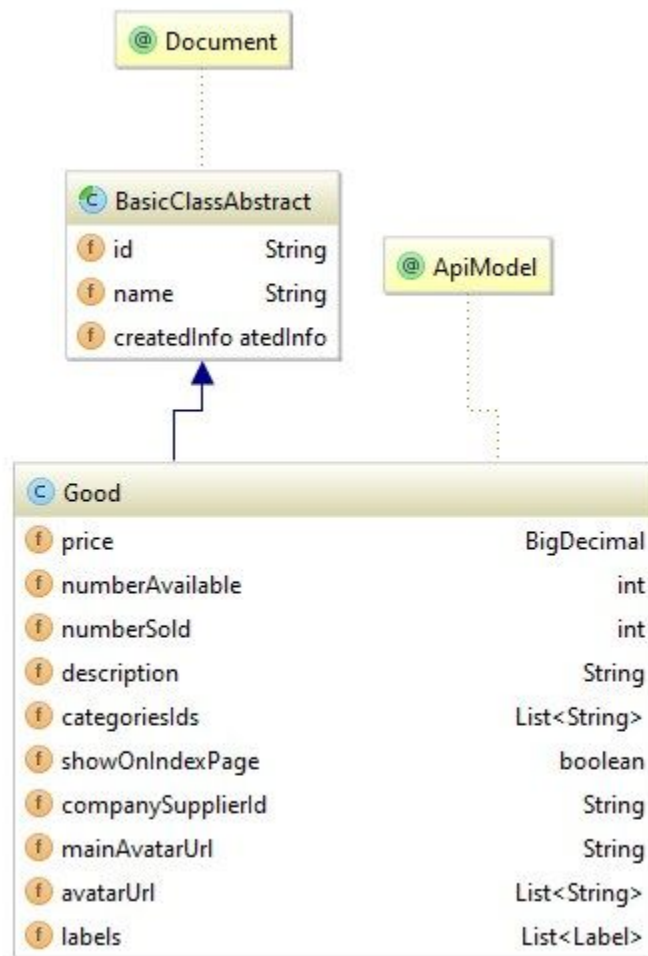
Пользователь



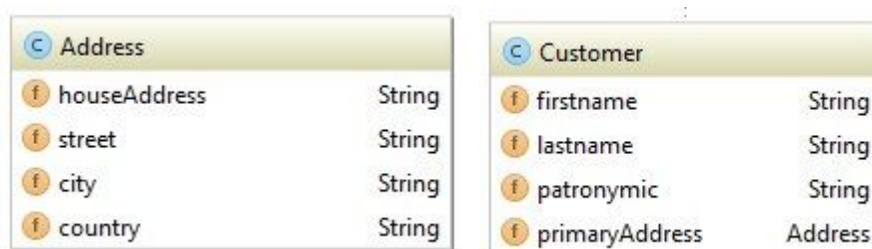
Заказ



Товар



Клиент и адрес клиента



MongoDB и sharding

MongoDB — NoSQL документо-ориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Написана на языке C++.

```
mongod --shardsvr --port 31337 --dbpath /Users/cystbear/mongo/data/r1
mongod --shardsvr --port 31338 --dbpath /Users/cystbear/mongo/data/r2
mongod --shardsvr --port 31339 --dbpath /Users/cystbear/mongo/data/r3

mongod --configsvr --port 30000 --dbpath /Users/cystbear/mongo/data/conf
mongos --configdb 127.0.0.1:30000 --port 30001

mongo --host 127.0.0.1 --port 30001

sh.addShard("127.0.0.1:31337")
sh.addShard("127.0.0.1:31338")
sh.addShard("127.0.0.1:31339")

sh.enableSharding("mgt")

sh.shardCollection("mgt.cards", { "set": 1})
```

Для проекта используется база данных MongoDB. Основными преимуществами которой является schemaless и шардирование из коробки.

Шардирование (горизонтальное партиционирование) — это принцип проектирования базы данных, при котором логически независимые строки таблицы базы данных хранятся отдельно, заранее сгруппированные в секции, которые, в свою очередь, размещаются на разных, физических и логически независимых серверах базы данных, при этом один физический узел кластера может содержать несколько серверов баз данных.³

В примере выше запускается 3 ноды(инстанса mongod) базы данных на разных портах с разными папками для хранения данных. Создается configsvr - конфиг сервер. А затем на mongos - mongo shard master добавляются в шардинг 3 ноды с ip адресом и портом. И следующей

³ <http://oleg.zorin.ru/doku.php?id=development:database:sharding>

строчкой включается шардирование между всеми участниками шарда на коллекции 'mgt'.

Docker

Одной из важных задач, которая была решена и необходима для работы программы: простота запуска и установки (installation)

В работе используется механизм контейнеризации на основе Docker. Это позволяет:

- Независимо запускать несколько экземпляров приложения одновременно
- Не зависит от окружения (dependency hell). Приложения не требуют ни установленной непосредственно Java на компьютере, ни базы данных. Кроме того, если они установлены, хоть и другой версии, например, то не будет никаких конфликтов.
- Переносимость. Образ(контейнер) легко использовать несколькими людьми, не переживая о различиях в конфигурации операционной системы.

Веб-приложение может запускать отдельно через командную строку: `java -jar target/ru.nbakaev.hishop.jar`

Затем запустить базу данных. Например через команду: `mongod --dbpath "/data/db" --storageEngine wiredTiger --config "/etc/mongod.conf"`

Сам docker файл для построения выглядит следующим образом

```
8 lines (6 sloc) | 218 Bytes
1 FROM java:8
2 VOLUME /tmp
3 ADD release.jar app.jar
4
5 EXPOSE 8080
6 RUN bash -c 'touch /app.jar'
7 ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar", "--spring.profiles.active=production"]
```


Параметры для запуска

Для запуска приложения необходимо передать программе конфигурационные параметры. Среди которых:

- MongoDB база данных
- Mailgun (сервис для отправки email сообщений с DKIM и спам фильтром)
- AWS credentials (API ключ для загрузки файлов на AWS S3)

Эти параметры могут быть переданы несколькими способами:

- Через переменное окружение

```
#!/usr/bin/env bash
export spring_data_mongodb_uri="mongodb://H7sXKfNiJ0oBPByF:X5hQ8Kx9KQZ9165.mongolab.com:39165/hishop"
export mailgun_api_key="key-b305ee369ffb69e8b559"
export mailgun_url="https://api.mailgun.net/v3/sandbox21473e1b5f33c2e779eb7a46.mailgun.org/messages"
export spring.output.ansi.enabled=ALWAYS

java -jar target/ru.nbakaev.hishop.jar
```

- Через аргументы командной строки для программы

```
java -jar target/ru.nbakaev.hishop.jar \
--spring_data_mongodb_uri="mongodb://H7sXKfNiJ0oBPByF:X5hQ8Kx9KQZ9165.mongolab.com:39165/hishop" \
--mailgun_api_key="key-b305ee369ffb69e8b559" \
--mailgun_url="https://api.mailgun.net/v3/sandbox21473e1b5f33c2e779eb7a46.mailgun.org/messages" \
--spring.output.ansi.enabled=ALWAYS
```

- В случае запуска через docker через переменные окружения

```
docker run -d -p 5555:5555 \
-e "spring_data_mongodb_uri=mongodb://H7sXKfNiJ0oBPByF:X5hQ8Kx9KQZ9165.mongolab.com:39165/hishop" \
-e "mailgun_api_key=key-b305ee369ffb69e8b559" \
-e "mailgun_url=https://api.mailgun.net/v3/sandbox21473e1b5f3344c2950ac2e779eb7a46.mailgun.org/messages" \
-e "spring.output.ansi.enabled=ALWAYS" \
--name hishop1 nbakaev/hishop-api-backend:0.2.2-RELEASE_build.127763392
```

- В конфигурационных файлах application.properties, application-development.properties или в

`application-production.properties` в зависимости от активного профиля
в `spring`, конфигурируемого через `spring.profiles.active`

Запуск

Запуск поддерживается на всех операционных системах, на которых возможен запуск Java приложений, включая Windows, Linux, Mac OS.

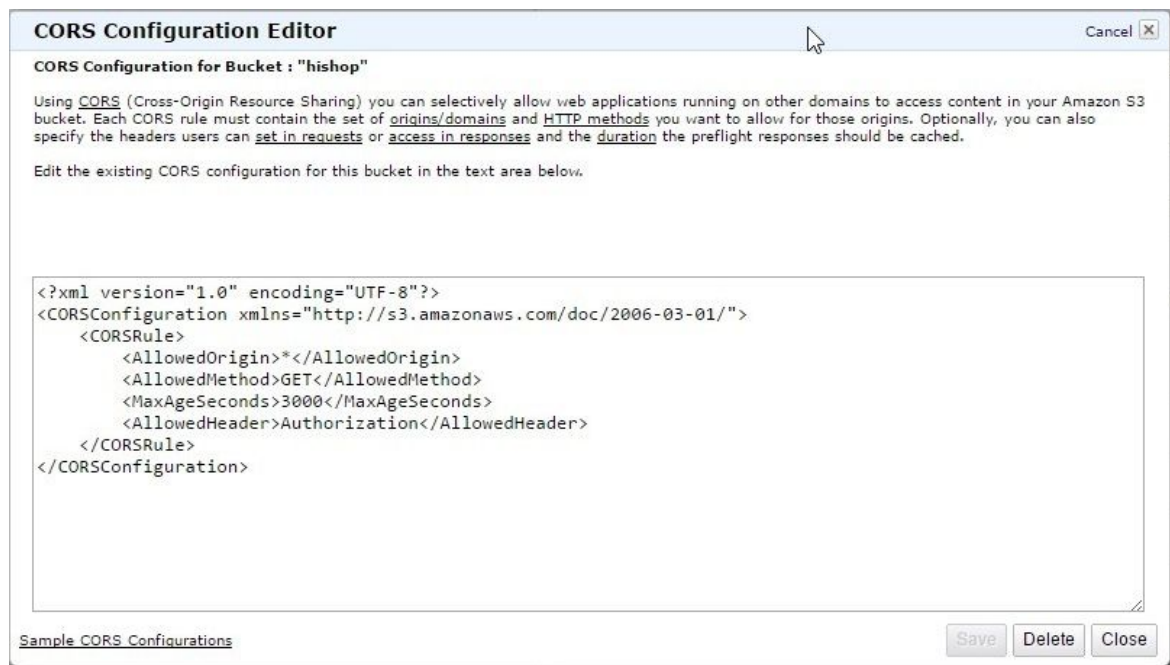
Платформозависимый код отсутствует. Возможно много вариантов запуска непосредственно самого приложения:

- запуск как обычный **jar файл** в операционной системе с jvm.
Например `java -jar hishop.jar`
- Запуск **docker контейнера**. Например `docker run -d nbakaev/hishop-api-backend`
- Запуск в PaaS **Heroku**. В корне git репозитория есть файл Procfile для запуска.
- Запуск в облаке Amazon: **AWS Beanstalk**. Этим сервисом поддерживается как запуск jar файлов, так и docker контейнеров, как описано выше
- в git репозитории присутствуют файлы для запуска в системе оркестрации docker контейнерами kubernetes (pod и service)

Загрузка файлов. AWS S3

Для загрузки файлов используется публичный AWS bucket, названный hishop. Он поддерживает CORS и доступен публично. Загрузка происходит через backend API.

Конфигурация выглядит следующим образом:



CORS Configuration Editor Cancel X

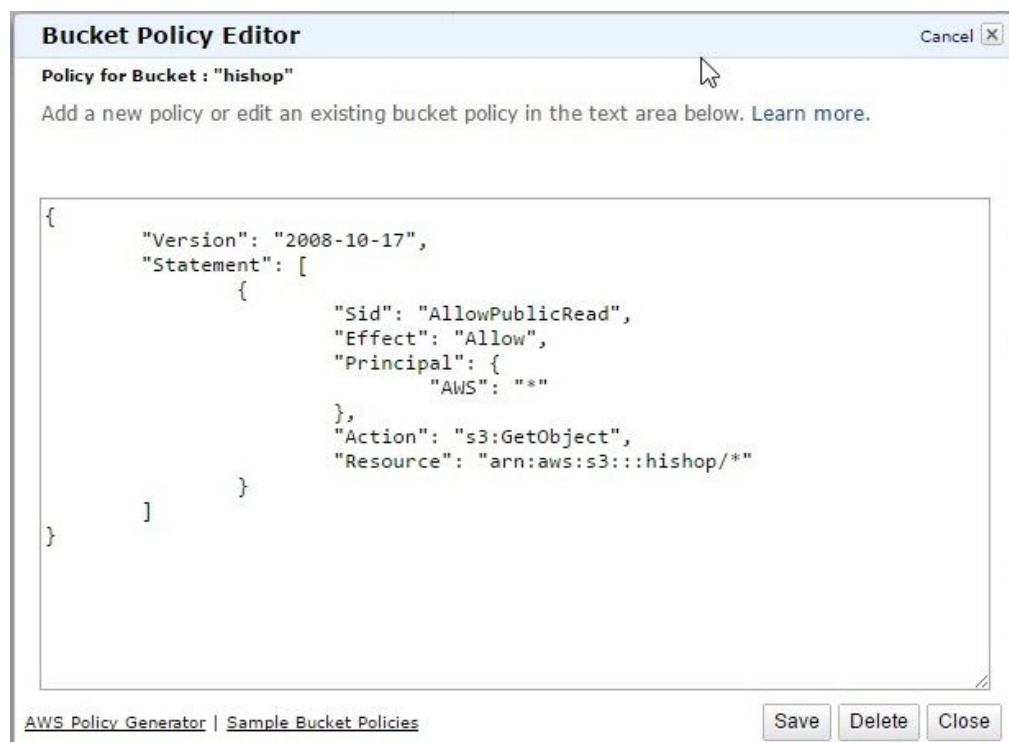
CORS Configuration for Bucket : "hishop"

Using [CORS](#) (Cross-Origin Resource Sharing) you can selectively allow web applications running on other domains to access content in your Amazon S3 bucket. Each CORS rule must contain the set of [origins/domains](#) and [HTTP methods](#) you want to allow for those origins. Optionally, you can also specify the headers users can [set in requests](#) or [access in responses](#) and the [duration](#) the preflight responses should be cached.

Edit the existing CORS configuration for this bucket in the text area below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <AllowedHeader>Authorization</AllowedHeader>
  </CORSRule>
</CORSConfiguration>
```

[Sample CORS Configurations](#) Save Delete Close



Bucket Policy Editor Cancel X

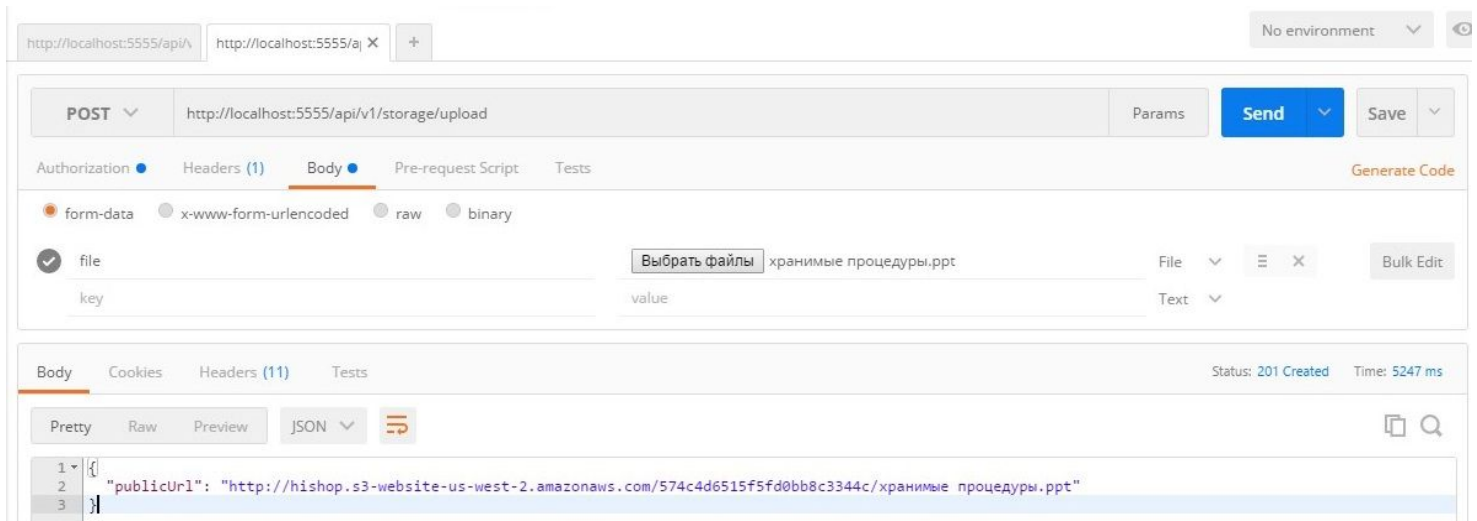
Policy for Bucket : "hishop"

Add a new policy or edit an existing bucket policy in the text area below. [Learn more.](#)

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "AllowPublicRead",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::hishop/*"
    }
  ]
}
```

[AWS Policy Generator](#) | [Sample Bucket Policies](#) Save Delete Close

Пример запроса и ответа от сервера. Сервер возвращает публичную ссылку на загруженный файл. Загрузка разрешена пользователем справками администратора.



Каждый файл загружается в отдельную “папку” (в архитектуре AWS S3 - это не папка а лишь путь). Список всех загруженных файлов.



И вот только что загруженный файл.



Тестирование

Для поддержания высокого качества кода и избежании регрессии в производительности и функционале используется тестирование кода.

Установлен уровень **покрытия кода тестами 90%**. Используется следующие методы тестирования:

- **Unit тесты.** Тестируются отдельные функции, Spring компоненты. В случае необходимости, используется мокирование методов объектов с помощью фреймворка **Mockito**.
- **Интеграционные тесты.** Используется **TestNG** и **spring-tests** фреймворки для тестов. Используется для тестирования корректности связанности компонент системы. В данном случае поднимается полностью Spring container со всеми бинами и DI. И тестируются как выполнение HTTP REST контроллеров, так и отдельных модулей.
- Создание процесса базы данных **MongoDB** полностью чистой (без каких либо коллекций или документов) при каждом запуске тестов. Это полноценная база данных, запускаемая автоматически. Это используется для полной проверки корректности работы с реальной базой данных. При этом - это просто отдельная тестовая база данных и она совершенно никак не связана на production.

Пример теста на загрузку файла в AWS S3. В этом примере мы делаем интеграционный тест. Считываем данные с локальной файловой системы в `byte[]`. Затем делаем запрос к нашему API контроллеру, как администратор. Получаем ссылку на загруженный в s3 файл. Затем делаем HTTP запрос и получаем этот файл и сравниваем загруженный файл с тем, что у нас на файловой системе.

```
@Test
public void testUploadToAmazonS3() throws Exception {

    // file name of real file in file system in resource folder
    String uploadFileName = "test_upload.jpg";
    byte[] uploadFilenameBytes = IOUtils.toByteArray(new ClassPathResource(uploadFileName).getInputStream());

    MockMultipartFile file = new MockMultipartFile("file", uploadFileName, null, new ClassPathResource(uploadFileName).getInputStream());

    // make request to our server
    byte[] result = mockMvc.perform(fileUpload("/api/v1/storage/upload")
        .file(file).with(user("test2@nbakaev.ru").roles("USER", "ADMIN")))
        .andExpect(status().is(201)).andReturn().getResponse().getContentAsByteArray();

    StorageDocumentMeta storageDocumentMeta = objectMapper.readValue(result, StorageDocumentMeta.class);
    Assert.assertNotNull(storageDocumentMeta);
    Assert.assertNotNull(storageDocumentMeta.getPublicUrl());

    // get real request to aws s3 and assert to content of file in our FS
    Assert.assertEquals(IOUtils.toByteArray(new URL(storageDocumentMeta.getPublicUrl())), uploadFilenameBytes);
}
```

Continuous Integration

Важный аспект при разработки ПО это постоянный мониторинг корректности работы актуального кода, включая контроль регрессии производительности и некорректность работы программы. С этой целью используется метод из технологий гибких методологий разработки ПО, а именно постоянный контроль работоспособности текущего программного кода.

С этой целью, каждый коммит в публичный git репозитория на github вызывает webhook на тестирование билда на CI сервере. В данном случае, в качестве CI сервера выбран бесплатный для open source проектов сервис **travis ci**. При этом, каждый билд вызывает следующие действия:

- сборка проекта с помощью **maven** (команда `mvn package`). В результате создается полностью исполняемый jar файл.
- выполнение всех java тестов. Это выполняется просто с помощью команды `mvn test`
- в случае успешного прохождения теста - загрузка jar артефакта в публичный maven binary repository (**bintray**)
- построение docker контейнера и загрузка в публичный репозиторий **docker hub**.
- отправка уведомления о результате работы в slack: время сборки, успешно прошли тесты(или сборка билда в целом) или нет.

Для описанной выше работы разработан bash скрипт, доступный в репозитории.




Так как некоторые действия, как загрузка в maven репозиторий или загрузка в docker hub требует данных для аутентификации, а репозиторий

публичен, то эти данные не могут безопасно храниться вместе с кодом и с этой целью, данные передаются через переменные окружения.

Ниже представлен интерфейс Travis CI

Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.

spring_data_mongodb_uri		
mailgun_api_key		
mailgun_url		
aws_accessKey		
aws_secretKey		

OFF

Display value in build log

Add

Доменная область

Google Analytics

Google Analytics: Measurement Protocol

Measurement Protocol - Протокол передачи статистических данных Google Analytics позволяет отправлять необработанные данные напрямую на серверы Google Analytics посредством HTTP-запросов практически в любой среде.

В данной работе протокол используется для передачи данных о транзакциях в систему веб-аналитики. Это полезно для дальнейшего построения отчетов, графиков, KPIs в различных временных и других разрезах.

Пример построения запроса, включая список параметров изображен ниже.

Запрос Measurement Protocol

<http://www.google-analytics.com/collect?v=1&tid=UA-61665202-1&cid=176149120.1418739219&t=pageview>

v 1	Версия протокола (всегда 1)
tid UA-61665202-1	Идентификатор Google Analytics
cid 176149120.1418739219	Идентификатор клиента
t pageview	Тип хита

Описание всех доступных параметров <https://goo.gl/uwOvQR>

Ниже демонстрируется пример отправки транзакции - данных о продаже товара в систему веб-аналитики. Информация включает общую цену, цену каждого товара, число товаров и некоторую другую служебную информацию.

Запрос Measurement Protocol

v 1	Версия протокола (всегда 1)
tid UA-12345678-1	Идентификатор Google Analytics
cid 176149120.1418739219	Идентификатор клиента
t transaction	Тип хита
ti 5674	Идентификатор транзакции
ta testshop.ru	Название магазина
tr 53000.00	Общая сумма транзакции
in iphone	Наименование товара
ip 53000.00	Стоимость товара
iq 1	Количество

Защита информации

Для хранения паролей используется хеш функция bcrypt 2. Она превосходит md5 и другие в плане надежности.

Для всего API используется SSL. Оценка конфигурации SSL - высшая (A+) известным сервисом. В случае обращения по HTTP будет автоматическая переадресация на защищенный протокол HTTPS. Это сделано с целью, безопасной передачи паролей и иной чувствительной информации по сети Интернет.

Аутентификация пользователей происходит по Basic access authentication.



[Home](#) [Projects](#) [Qualys.com](#) [Contact](#)

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > s2.nbakaev.ru

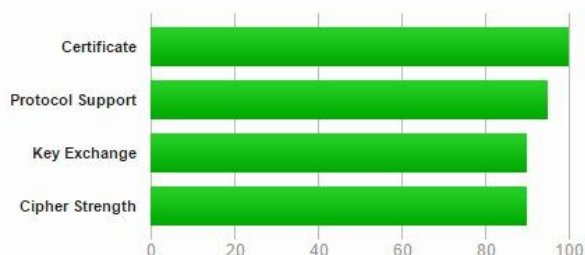
SSL Report: s2.nbakaev.ru (128.199.50.77)

Assessed on: Thu, 28 Apr 2016 11:21:10 UTC | [Hide](#) | [Clear cache](#)

[Scan Another »](#)

Summary

Overall Rating



Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. [MORE INFO »](#)

Дальнейшее развитие

Уже в данной работе продемонстрировано полностью рабочее приложение интернет-магазина.

При этом, есть основные векторы, которые будут реализованы и продемонстрированы в дальнейшем:

- **Шаблоны документов.** Пользователь полностью определяет шаблоны документов в HTML с возможностью использовать JavaScript для динамического создания документа, а затем выгружает их в PDF, Microsoft Word или иные форматы
- **Дополнительные поля (**custom fields**)** для объектов, включая товары. Отсутствие явной схемы в базе данных MongoDB позволит хранить любые поля. При этом каждая категория товаров может иметь свой уникальный список полей, доступный для выборок и полнотекстового поиска
- Создание заказа и некоторые другие ресурсоёмкие компоненты будут вынесены в отдельные **микросервисы**, таким образом будет микросервисная архитектура.
- Интернационализация и **локализация**: перевод на разные языки
- Товарные **рекомендации**
- Пример логирования, сбор метрик, **APM**
- горизонтальное масштабирование кластера **kubernetes**
- пример **шардирования** и replica set mongodb
- хранение конфигурации приложений и иных компонент во внешнем хранилище с strong consistency (**consul**) с возможностью динамического обновления конфигурации уже на запущенных приложениях

- интеграция с одной из существующей облачной **CRM, ERP** для демонстрации работы интернет-магазины
- расширенные аналитические отчеты, дашборды

Заключение

В данной работе удалось реализовать полноценный интернет магазин с работой с товарами, выгрузкой, работой с пользовательской базой и многое другое. Система полностью построена на свободно распространяемых и кроссплатформенных технологиях. При разработки использовались современные методологии разработки и проверки качества разрабатываемого ПО для достижения высокого качества кода.

Все поставленные задачи - успешно выполнены.

Кроме того, поставлен ряд задач на улучшение и на разработку нового функционала и улучшения текущего в следующей итерации.

Список литературы

Интернет-ресурсы:

- <https://spring.io/docs/reference> - Документация по Spring Framework & Spring Boot
- Документация по Docker <https://docs.docker.com/engine/reference/builder/>
- Resource Guide: Building an eCommerce Website
<https://www.drupal.org/resource-guides/building-ecommerce>

Книги:

- Spring in Action Fourth Edition Edition:
http://www.amazon.com/Spring-Action-Craig-Walls/dp/161729120X/ref=sr_1_1?s=books&ie=UTF8&qid=1454910859&sr=1-1&keywords=spring
- Spring Boot in Action:
http://www.amazon.com/Spring-Boot-Action-Craig-Walls/dp/1617292540/ref=sr_1_3?s=books&ie=UTF8&qid=1454910859&sr=1-3&keywords=spring
- Spring Recipes: A Problem-Solution Approach:
http://www.amazon.com/Spring-Recipes-Problem-Solution-Daniel-Rubio/dp/1430259086/ref=sr_1_4?s=books&ie=UTF8&qid=1454910859&sr=1-4&keywords=spring
- Building eCommerce Applications -
<http://shop.oreilly.com/product/0636920023098.do>

Приложения

Ссылки на материалы:

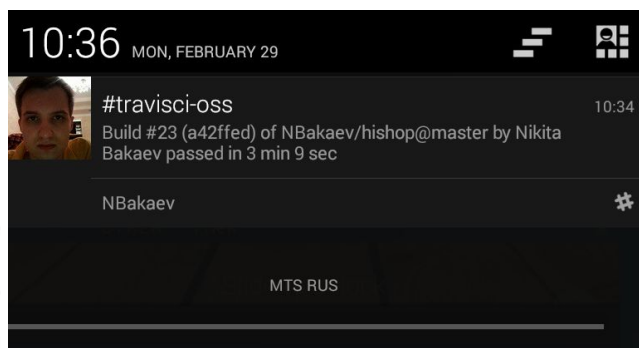
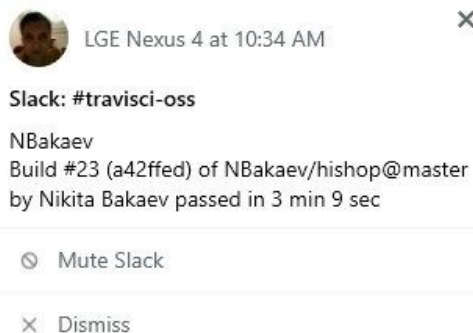
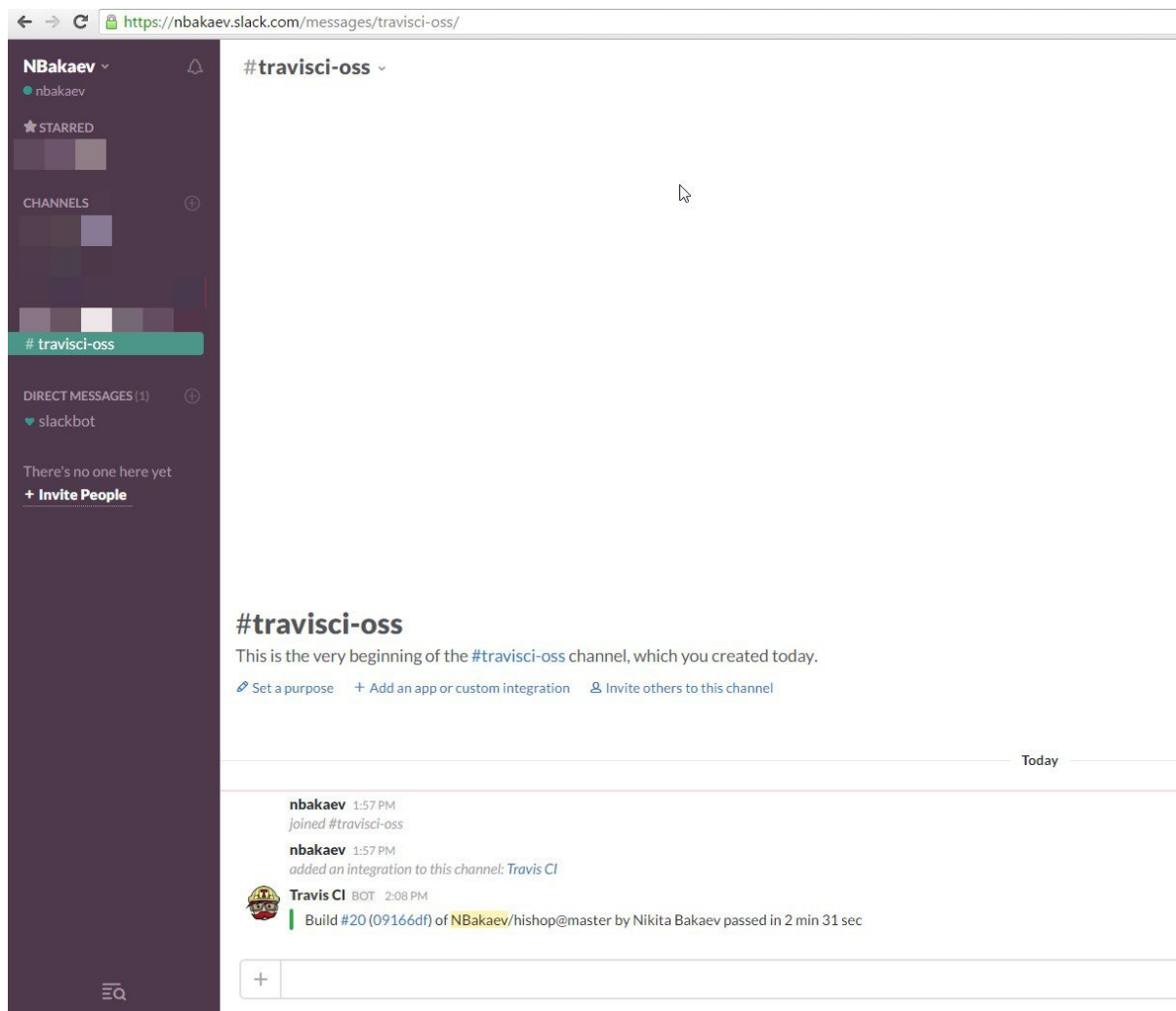
- <https://docs.google.com/document/d/1QhCjvqiGACP9OQohqe5BMcsHonedugAchMLozXtCoW64> - Текст курсовой работы в Google Docs
- <https://docs.google.com/presentation/d/1UFx-xWNX8DwdD1uIw4yO7Gc8Snupf74qfb5MSyjcJU> - Презентация курсовой работы в Google Docs

Что использовалось для работы:

- IDE: IntelliJ Idea 2016.1.2
- Ubuntu 16.04 server
- Charles - web proxy, sniffer. Для отладки запросов. (аналог Wireshark)
- Postman - тесты API и создание/формирование запросов
- MongoChef - GUI для MongoDB
- Docker 1.11
- MongoDB 3.2 - NoSQL database
- библиотеки и фреймворки, описанные выше

Приложения

Slack, уведомления



Search all repositories

My Repositories +

NBakeav/hishop

20

Duration: 1 min 20 sec
Finished: -

NBakeav / hishop

Current Branches Build History Pull Requests

master add notifications

Commit 09166df
Compare c2a03c..09166df

Nikita Bakeav authored and committed

#20 started

Elapsed time 1 min 20 sec

Remove log

Raw log

```
1 Using worker: worker-linux-docker-1082719f-prod-travis-ci.org:travis-linux-6
2
3 Build system information
4
5 export DEBIAN_FRONTEND=noninteractive
6
7 git clone --depth=50 --branch=master https://github.com/NBakeav/hishop.git NBakeav/hishop
8
9 This job is running on container-based infrastructure, which does not allow use of 'sudo', 'stella' and 'setuid' executables.
10 If you require 'sudo', add 'sudo: required' to your .travis.yml
11 See https://docs.travis-ci.com/user/workers/container-based-infrastructure/ for details.
12 $ jdk_switcher use oraclejdk8
13 Switching to Oracle JDK8 (java-8-oracle), JNA_HOME will be set to /usr/lib/jvm/java-8-oracle
14 $ java -Xmx32m -version
15 java version "1.8.0_31"
16 Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
17 Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
18 $ java -Xmx32m -version
19 java 1.8.0_31
20
21 $ pip install --user codecov
22
23 $ mvn install -DskipTests=true -Dmaven.javadoc.skip=true -B -V
24
25 Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=192m; support was removed in 8.0
26 Apache Maven 3.2.5 (12603ac0947671f900af49994c51426b0cc1; 2014-12-11T17:29:23+08:00)
27
28 Maven home: /usr/local/maven
29
30 Java version: 1.8.0_31, vendor: Oracle Corporation
31
32 Java home: /usr/lib/jvm/java-8-oracle/jre
33
34 Default locale: en_US, platform encoding: UTF-8
35
36 OS name: "linux", version: "3.13-0-40-generic", arch: "amd64", family: "unix"
37
38 [INFO] Scanning for projects...
39
40 [INFO] Downloading: http://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-parent/1.3.1.RELEASE/spring-boot-starter-parent-1.3.1.RELEASE.pom (7 KB at 48.3 KB/sec)
41
42 [INFO] Downloaded: http://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-parent/1.3.1.RELEASE/spring-boot-starter-parent-1.3.1.RELEASE.pom (7 KB at 48.3 KB/sec)
```

```
2466 2016-02-23 11:08:28.448 INFO 2639 --- [
    for suffixes {-context.xml, Context.groovy}.
2470 2016-02-23 11:08:28.450 INFO 2639 --- [
    [org.springframework.test.context.web.ServletTestExecutionListener, org.springframework.test.context.support.DefaultTestExecutionListener,
    org.springframework.test.context.support.DependencyInjectionTestExecutionListener, org.springframework.test.context.support.DirtiesContextTestExecutionListener,
    org.springframework.test.context.transaction.TransactionalTestExecutionListener, org.springframework.test.context.jdbc.SqlScriptsTestExecutionListener]
2472 2016-02-23 11:08:28.453 INFO 2639 --- [
    main] o.s.t.c.web.WebTestContextBootstrapper : Using TestExecutionListeners: [org.springframework.test.context.web.ServletTestExecutionListener@15d02022,
    org.springframework.test.context.support.DirtiesContextTestExecutionListener@2937a800, org.springframework.test.context.transaction.TransactionalTestExecutionListener@3daa0f3f,
    org.springframework.test.context.jdbc.SqlScriptsTestExecutionListener@2c68c5f1]
2472 Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.017 sec - in tk.hishopapp.StartApplicationTest
2473 2016-02-23T11:08:28.591+0000 I NETWORK [initAndListen] connection accepted from 127.0.0.1:52218 #3 (3 connections now open)
2474 [mongod output] 2016-02-23T11:08:28.592+0000 I COMMAND [conn3] terminating, shutdown command received
2475 [mongod output] 2016-02-23T11:08:28.593+0000 I FTDC [conn3] Shutting down full-time diagnostic data capture
2476 [mongod output] 2016-02-23T11:08:28.595+0000 I CONTROL [conn3] now exiting
2477 [mongod output] 2016-02-23 11:08:28.596 INFO 2639 --- [
    Thread-8] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web.context.support.GenericWebApplicationContext@55e21973: startup date
    [Tue Feb 23 11:08:09 UTC 2016]; root of context hierarchy
2478
2478 2016-02-23T11:08:28.596+0000 I NETWORK [conn3] shutdown: going to close listening sockets...
2479 [mongod output] 2016-02-23T11:08:28.597+0000 I NETWORK [conn3] closing listening socket: 5
2480 [mongod output] 2016-02-23T11:08:28.597+0000 I NETWORK [conn3] closing listening socket: 6
2481 [mongod output] 2016-02-23T11:08:28.597+0000 I NETWORK [conn3] removing socket file: /tmp/mongod-27017.sock
2482 [mongod output] 2016-02-23T11:08:28.598+0000 I NETWORK [conn3] shutdown: going to flush diaglog...
2483 [mongod output] 2016-02-23T11:08:28.598+0000 I NETWORK [conn3] shutdown: going to close sockets...
2484 [mongod output] 2016-02-23T11:08:28.598+0000 I STORAGE [conn3] WiredTigerKVEngine shutting down
2485 [mongod output] 2016-02-23T11:08:28.609+0000 I STORAGE [conn3] shutdown: removing fs lock...
2486 [mongod output] 2016-02-23T11:08:28.610+0000 I CONTROL [conn3] dbexit: rc: 0
2487 [mongod output] 2016-02-23 11:08:28.608 INFO 2639 --- [
    Thread-2] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web.context.support.GenericWebApplicationContext@0c8f96: startup date
    [Tue Feb 23 11:07:43 UTC 2016]; root of context hierarchy
2488
2488 Results :
2490
2491 Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
2492
2493 [INFO]
2494 [INFO] --- jacoco-maven-plugin:0.7.5.201505241946:report (report) @ web ---
2495 [INFO] Analyzed bundle 'web' with 24 classes
2496 [INFO]
2497 [INFO] BUILD SUCCESS
2498 [INFO]
2499 [INFO] Total time: 01:08 min
2500 [INFO] Finished at: 2016-02-23T11:08:34+00:00
2501 [INFO] Final Memory: 22M/339M
2502 [INFO]
2503 [INFO]
2504
2505
2506 The command "mvn test -B" exited with 0.
2507 $ codecov
2533
2534 Done. Your build exited with 0.
```



```
1540 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/eclipse/aether/aether-util/1.0.2.v20150114/aether-util-1.0.2.v20150114.jar (144 KB at 187.5 KB/sec)
1541 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.8.1/plexus-archiver-2.8.1.jar
1542 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.7/maven-shared-utils-0.7.jar (167 KB at 189.6 KB/sec)
1543 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.3.2/plexus-io-2.3.2.jar
1544 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/eclipse/sisu/org.eclipse.sisu.inject/0.0.0.M5/org.eclipse.sisu.inject-0.0.0.M5.jar (285 KB at 315.8 KB/sec)
1545 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/commons/commons-compress/1.9/commons-compress-1.9.jar
1546 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.3.2/plexus-io-2.3.2.jar (73 KB at 72.3 KB/sec)
1547 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.20/plexus-utils-3.0.20.jar
1548 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/commons/commons-compress/1.9/commons-compress-1.9.jar (370 KB at 343.9 KB/sec)
1549 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-shade-plugin/2.2/maven-shade-plugin-2.2.jar
1550 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-shade-plugin/2.2/maven-shade-plugin-2.2.jar (98 KB at 86.8 KB/sec)
1551 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/sonatype/sisu/sisu-bean/1.4.2/sisu-bean-1.4.2.jar
1552 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.8.1/plexus-archiver-2.8.1.jar (140 KB at 122.0 KB/sec)
1553 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/sonatype/sisu/sisu-guice/2.1.7/sisu-guice-2.1.7-noaop.jar
1554 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/sonatype/sisu/sisu-inject-bean/1.4.2/sisu-inject-bean-1.4.2.jar (150 KB at 124.4 KB/sec)
1555 [INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm/3.3.1/asm-3.3.1.jar
1556 [INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm/3.3.1/asm-3.3.1.jar (43 KB at 34.6 KB/sec)
1557 [INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-commons/3.3.1/asm-commons-3.3.1.jar
1558 [INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-commons/3.3.1/asm-commons-3.3.1.jar (38 KB at 29.8 KB/sec)
1559 [INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-tree/3.3.1/asm-tree-3.3.1.jar
1560 [INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-tree/3.3.1/asm-tree-3.3.1.jar (22 KB at 16.5 KB/sec)
1561 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/jdom/jdom/1.1/jdom-1.1.jar
1562 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/jdom/jdom/1.1/jdom-1.1.jar (150 KB at 110.8 KB/sec)
1563 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-dependency-tree/2.1/maven-dependency-tree-2.1.jar
1564 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/sonatype/sisu/sisu-guice/3.1.0/sisu-guice-3.1.0-no_aop.jar (350 KB at 252.9 KB/sec)
1565 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/vafer/jdependency/0.7/jdependency-0.7.jar
1566 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-dependency-tree/2.1/maven-dependency-tree-2.1.jar (59 KB at 42.3 KB/sec)
1567 [INFO] Downloading: https://repo.maven.apache.org/maven2/commons-io/commons-io/1.3.2/commons-io-1.3.2.jar
1568 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/vafer/jdependency/0.7/jdependency-0.7.jar (12 KB at 8.1 KB/sec)
1569 [INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-analysis/3.2/asm-analysis-3.2.jar
1570 [INFO] Downloaded: https://repo.maven.apache.org/maven2/commons-io/commons-io/1.3.2/commons-io-1.3.2.jar (86 KB at 59.0 KB/sec)
1571 [INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-util/3.2/asm-util-3.2.jar
1572 [INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-analysis/3.2/asm-analysis-3.2.jar (18 KB at 11.6 KB/sec)
1573 [INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-util/3.2/asm-util-3.2.jar (36 KB at 23.3 KB/sec)
1574 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.20/plexus-utils-3.0.20.jar (238 KB at 154.4 KB/sec)
1575 [INFO] Downloaded: https://repo.maven.apache.org/maven2/log4j/log4j/1.2.17/log4j-1.2.17.jar (479 KB at 297.7 KB/sec)
1576 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/sonatype/sisu/sisu-guice/2.1.7/sisu-guice-2.1.7-noaop.jar (461 KB at 269.7 KB/sec)
1577 [INFO]
1578 --- maven-install-plugin:2.5.2:install (default-install) @ web ---
1579 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.4/maven-shared-utils-0.4.pom (4 KB at 116.2 KB/sec)
1580 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.4/maven-shared-utils-0.4.pom (4 KB at 116.2 KB/sec)
1581 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.4/maven-shared-utils-0.4.jar
1582 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.4/maven-shared-utils-0.4.jar (152 KB at 1054.2 KB/sec)
1583 [INFO] Installing /home/travis/build/NBakeev/hishop/target/web-0.1.0-SNAPSHOT.jar to /home/travis/.m2/repository/rub/hishop/web/0.1.0-SNAPSHOT.jar
1584 [INFO] Installing /home/travis/build/NBakeev/hishop/pom.xml to /home/travis/.m2/repository/rub/hishop/web/0.1.0-SNAPSHOT.pom
1585 [INFO]
1586 --- BUILD SUCCESS ---
1587 [INFO]
1588 [INFO] Total time: 32.665 s
1589 [INFO] Finished at: 2016-02-23T11:07:11+00:00
1590 [INFO] Final Memory: 37M/311M
1591 [INFO] -----
```