TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Tarkvaratehnika õppetool

Veebiteenuste tehnoloogiate analüüs ja realisatsioon

bakalaureusetöö

Üliõpilane: Aleksander Šumilov

Üliõpilaskood: 104096IAPB

Juhendaja: lektor Kaarel Allik

Tallinn
2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

……. ……

(kuupäev) (allkiri)

Veebiteenuste tehnoloogiate analüüs ja realisatsioon

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on realiseerida veebirakendus, mis täidab samu nõudeid ja funktsionaalsust, kui Splento startapi veebirakendus. Splento – uus startap, mis pakub kvaliteetset professionaalset fotograafiat, mis on töödeldud, toimetatud ja saadetakse kliendile 48 tunni jooksul.

Eesmärgi saavutamiseks tuleb uurida, mis tehnoloogiat kasutavad enda veebirakenduses Splento ja teised kuulsad projektid ja startapid, nagu Slack, TransferWise ja Uber.

Selles töös uurime ka erinevaid tehnoloogiaid, rakenduse arhitektuure ja lahendusi, mis võivad aidata meie eesmärgi täitmiseks.

Analüüsi põhjal realiseerime veebirakendus, mis peab täitma nõudmised.

Lõputöö on kirjutatud vene keeles ning sisaldab teksti 48 lk, 3 peatükki, 8 joonist.

Web service analysis of the technologies and implementation

Annotation

The aim of this thesis is web service implementation that performs the same requirements and functionalities as Splento startup. Splento is a new project, that provides high quality professional photography which is processed, edited and delivered to you in 48 hours.

For achieving this goal, we need to make some research and investigate technologies used by Splento and other famous projects and startups, like Slack, Uber, TransferWise.

In this work we analyze the different technologies and service architectures that can help us to achieve our goal.

As the result of the work there will be realized web service architecture that must comply with the requirements.

The thesis is written in Russian and includes 48 pages of text, 3 chapters, 8 figures.

Анализ технологий для веб-сервиса и его реализация

Аннотация

Целью данной работы является реализация сервиса, который выполняет схожий функционал нового стартапа Splento. Splento – это новый проект, который предлагает качественные фотографии, обработанные и доставленные клиентам в течение 48 часов.

Для достижения поставленной цели необходимо анализировать используемые технологии в сервисах известных продуктов и стартапов, в том числе и Splento. Также изучим технологии и архитектуры, которые могут помочь в реализации нашего сервиса.

Исходя из проведенного анализа, реализуем сервис, которые будет выполнять необходимые требования и функционал.

Данная работа написана на русском языке и включает 48 страниц, 3 главы, 8 изображений.

Список сокращений и терминов

Startap	**startup, стартап** Компания, только начавшее свое дело и пытающаяся выдвинуть свой продукт или идею в массы
Backend	В основном это серверные компоненты, такие как база данных, скрипты обработчики, API, ядро и платформа сайта, то чего клиент не видит визуально
Frontend	Интерфейс взаимодействия между пользователем и backend
GPS	**Global Positioning System** Система глобального позиционирования, позволяющая определять нахождение объекта
WiFi	**Wireless Fidelity** Беспроводное интернет соединение для передачи цифровых потоков данных по радиоканалам
3G/4G	**Third/Fourth generation of mobile telecommunications technology** Тоже самое что и WiFi, только точкой доступа является вышка мобильного оператора
Tweets	Короткие сообщения, размером в 140 символов, в социальной сети Twitter
API	**Application programming interface** Программный интерфейс, обладающий неким функционалом и выполняющий необходимые операции.
Push-notifications	Посредством специальных серверов, на мобильные устройства возможно передавать уведомления(push-notifications)
Framework	**фреймворк** Каркас, состоящий из множества библиотек и функционала, облегчающий разработку программного продукта

Стек технологий Набор технологий, в совокупности образующий определённый компонент

JVM **Java virtual machine**

Виртуальная машина Java, для исполнения Java приложений

B2B **Business to business**

Вид взаимодействия субъектов. Продукт предлагает свои услуги не для конечного потребителя, а для другого бизнеса

Десктопное приложение

Приложение, разработанное для настольных персональных компьютеров

LAMP **Linux, Apache, MySQL, and PHP**

Набор программного обеспечения с открытым исходным кодом, которой обычно устанавливается на сервер для отображения динамических веб-сайтов и веб-приложений. Эта аббревиатура обозначает операционную систему Linux с установленным веб-сервером Apache, MySQL базой данных и PHP, как основным языком программирования

P2P **Peer to peer**

Одно-ранговая сеть, где отсутствует центральный элемент, и каждый узел является не только клиентом, но и выполняет функции центрального элемента

User-experience Пользовательский опыт использования интерфейса

MVP **Минимально жизнеспособный продукт**

Стадия разработки продукта, имеющий минимальный функционал для того, что бы понять реакцию потребителя и стоит ли продолжать разработку.

Список иллюстраций

Содержание

Введение

На сегодняшний день, интернет является неотъемлемой частью жизни. Интернет стал мощным инструментом, который владеет не только информацией. Ежегодно создается множество различных сервисов, которые предлагают свои услуги для упрощения жизни. Он стал лучшим инструментом бизнеса для современных организаций. Последние новости со всего мира, интернет магазины и разные услуги, всё это у вас под рукой.

На сегодняшний день Эстония является самой информационно подкованной страной в мире[40]. Кроме проектов, таких как eKool, ID Card, e-Residency, известность Эстонии дали такие продукты как, Toggl, Transferwise, Skype, которые начинали разрабатываться в Эстонии.

Естественно при построении таких сервисов необходимо тщательно анализировать не только рынок и поведение потребителей, но и технологии и необходимые ресурсы, чтобы сервис мог работать на полную мощность и выполнять все поставленные цели.

Не многие проекты, стартапы и компании раскрывают информацию о используемых технологиях в их продуктах. Но все же есть довольно крупные проекты, которые делятся этими данными[41][42][43][44].

В любом случае не для всех создаваемых новых проектов может подойти стек технологий, используемый другими проектами. В итоге приходится изучать рынок предложений. Ежегодно создается, обновляется и выпускается большое количество все возможных фреймворков, библиотек, движков и скриптов. Каждый из них хорош по своему.

В этой работе я хотел бы изучить технологии некоторых больших проектов, о которых можно сегодня услышать, а также продемонстрировать пример хорошего сервиса.

В ходе работы рассмотрим веб-сервисы таких проектов, как Uber, TransferWise, Slack для того, чтобы понять, как устроены сегодняшние веб-сервисы, какие технологии и решения они используют. Также изучим Splento продукт и выявим требования, которые необходимо выполнять подобному сервису. На основе проанализированных данных, попытаем реализовать сервис, который бы выполнял требования Splento.

1. Анализ больших сервисов

1.1. Uber

Uber Technologies Inc.(Uber) американская международная компания, созданная в марте 2009 году основателями Travis Kalanick и Garrett Camp[1]. Изначально создавалась под названием “UberCab”. Штаб квартира находится в Сан-Франциско, в штате Калифорнии.

Предлагает услуги личного водителя практически по всему миру. На октябрь 2015-ого года Uber был распространен в 334 городах и 60 странах, в том числе и в Таллинне[2].

1.1.1. Функциональность

С помощью одноименного приложения Uber клиент резервирует машину с водителем и отслеживает её перемещение к указанной точке. Водитель подбирает заказчика и довозит его до пункта назначения. По окончанию поездки, клиенту нет необходимости доставать кошелек для совершения оплаты услуги. Сервис автоматически снимет деньги со счета клиента на основании банковских данных, которые клиент ввел при регистрации учетной записи.

Водители с помощью приложения Uber Partner отслеживают заказы и при возможности принимают их. Получив местоположение клиента, водители сразу направляются к нему. В большинстве случаев водители используют свои собственные автомобили, а также машины таксопарков или партнеров.

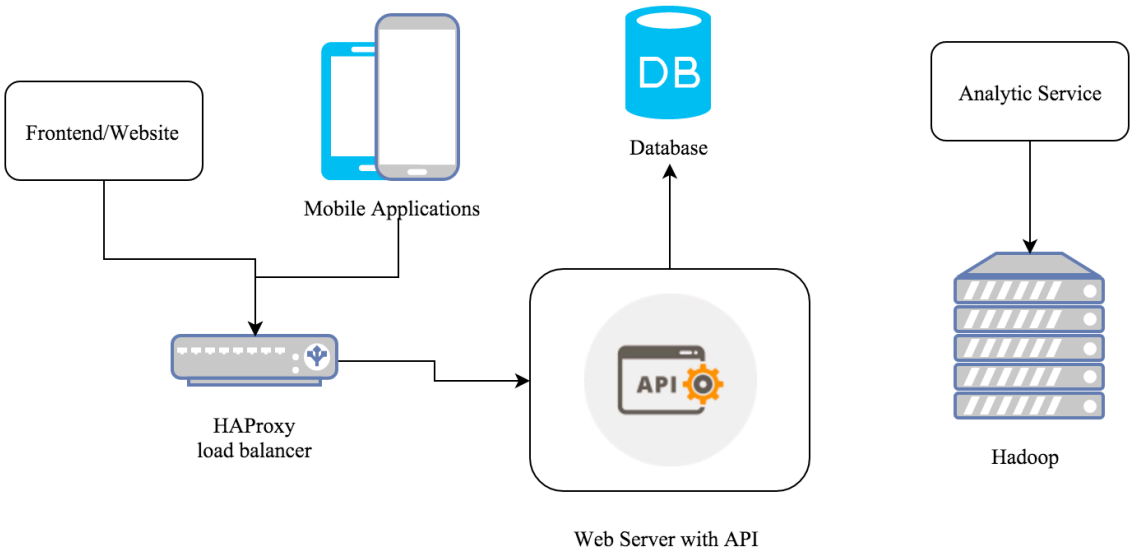
В большинстве стран, где Uber внедрил свой сервис, 80% оплаты переходят водителю, 20% перечисляются компании.

1.1.2. Архитектура сервиса Uber

Сервис представляет из себя два мобильных приложения(Uber и Uber Partner) и веб-сервис(далее API) для обработки получаемых данных.

Помимо этого у них есть сервис аналитики, диспетчерская, где они могут в реальном времени отслеживать действия водителей и клиентов, все поездки, маршруты и так далее.

Простая схема необходимых компонентов выглядит следующим образом:



Изображение 1. Простая архитектура сервиса Uber

В своей презентации 4-го Августа 2011 года, Uber Engineering Manager Curtis Chambers, говорил о том, с чего они начинали свой продукт[10]. На первом этапе их архитектура состояла только из двух компонентов. PHP сервер и база MySQL. Но для исключений ситуаций, когда водители могли одновременно взять одного клиента, необходимо было внедрить диспетчера. Роль диспетчера взял на себя сервис, написанный на Node.js, который начал обрабатывать и распределять эти задачи.

Мобильные приложения Uber и Uber Partner

Uber приложение предназначается заказчикам, то есть клиентам. Является открытым приложением и может быть установлен на четырех, лидирующих на рынке, операционых системах: Windows Phone, iOS, Android, BlackBerry 7 [3][4][5][9].

Uber Partner разработан для водителей автомобилей, заключивших договор с компанией Uber. Приложение распространяется только на две операционные системы: iOS и Android.

Оба приложения используют технологии GPS, WiFi/3g технологии.

Backend

На данный момент, основными языками, на котором написаны сервисы на backend являются Javascript, Python и Go.

Database

Для хранения данных используется Mysql и для аналитик-сервиса используется Hadoop. Который не является, так таковой, базой данных, но позволяет работать с огромными аналитическими данными, которые находятся на разных серверах или node.

Frontend

Для сайта используется, написанный на JavaScript фреймворк, React.js.

Сервисы в Uber написаны на разных языках и для коммуникации между ними данные сериализируются с помощью Apache Thrift.
Thrift представляет из себя кросс-платформенный фреймворк позволяющий обрабатывать сериализованные данные на разных языках, таких как C++, Java, Python, PHP, Ruby и т.д.
Сериализация данных представляет из себя представление объекта или структуры данных в последовательность битов.

Если мы хотим передать с одного сервиса на другой какой-то обьект, и сервисы имеют отличный друг от друга язык программирования, то Apache Thrift сериализирует и десериализирует этот объект.

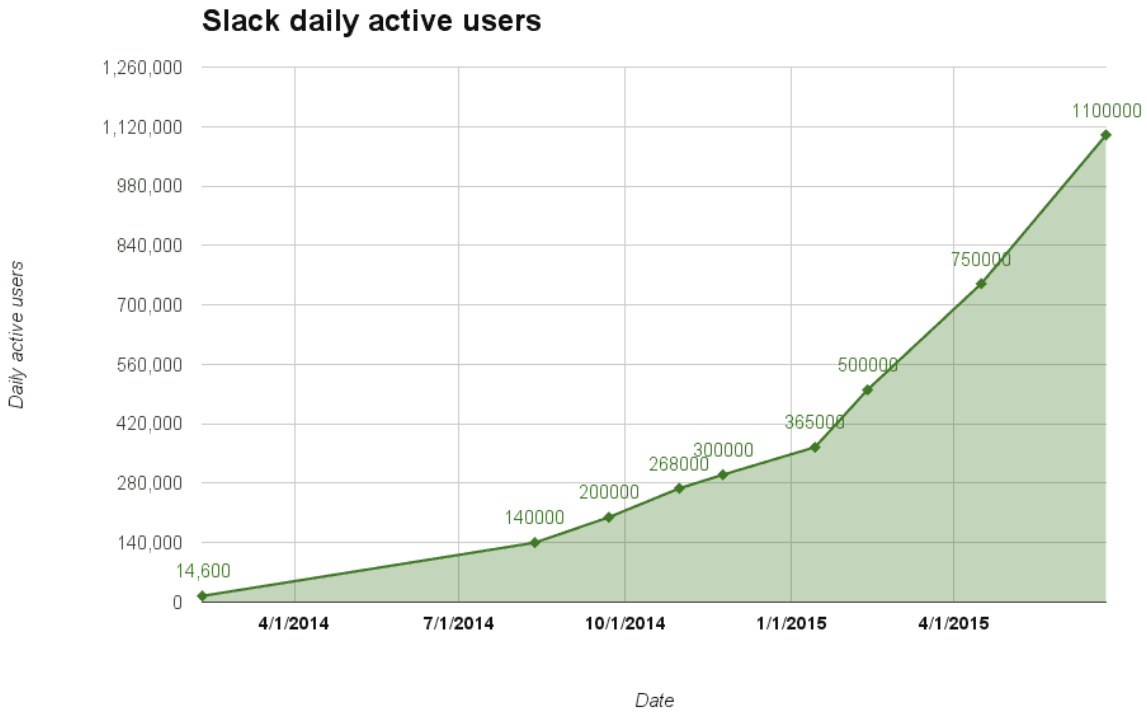
Для уменьшения нагрузки на сервера от входящего траффика, Uber использует распределитель нагрузки(load balancer) HAProxy между идентичными серверами.

1.2. Slack

Slack – это инструмент, приложение для корпоративного общения между сотрудниками, разработанный Stewart Butterfield(одним из со основателей фото хостинга Flickr), Eric Costello, Cal Henderson, Serguei Mourachov и запущенный на рынок в тестовом режиме в Августе 2013 года[11]. Публичный релиз состоялся в феврале 2014 года.

Изначально продукт разрабатывался для общения между командами в компании Tiny Speck(ныне Slack Technologies) при разработке онлайн игры Glitch. Этот мессенджер и стал основой для будущего проекта.

Проект является самым быстро растущим B2B-приложением в истории и используется 1.7 миллионом пользователей ежедневно.



Изображение 2. Статистика ежедневного использования Slack

1.2.1. Функциональность

Приложение представляет из себя интерфейс мессенджера с различными приватными и открытыми, личными группами и каналами. Позволяет общаться как лично с участником команды, так и образовывать открытые или приватные каналы для общения нескольких участников в одном окне. Приложение позволяет также обмениваться все возможными форматами файлов, размером до 1GB.

Кроме того, Slack поддерживает интеграцию более сотни сторонних сервисов. Начиная с Twitter, Google Hangouts и заканчивая Dropbox, Jira, Trello.

Если взять к примеру Twitter, то Slack позволяет отправлять и получать tweets в одном окне.

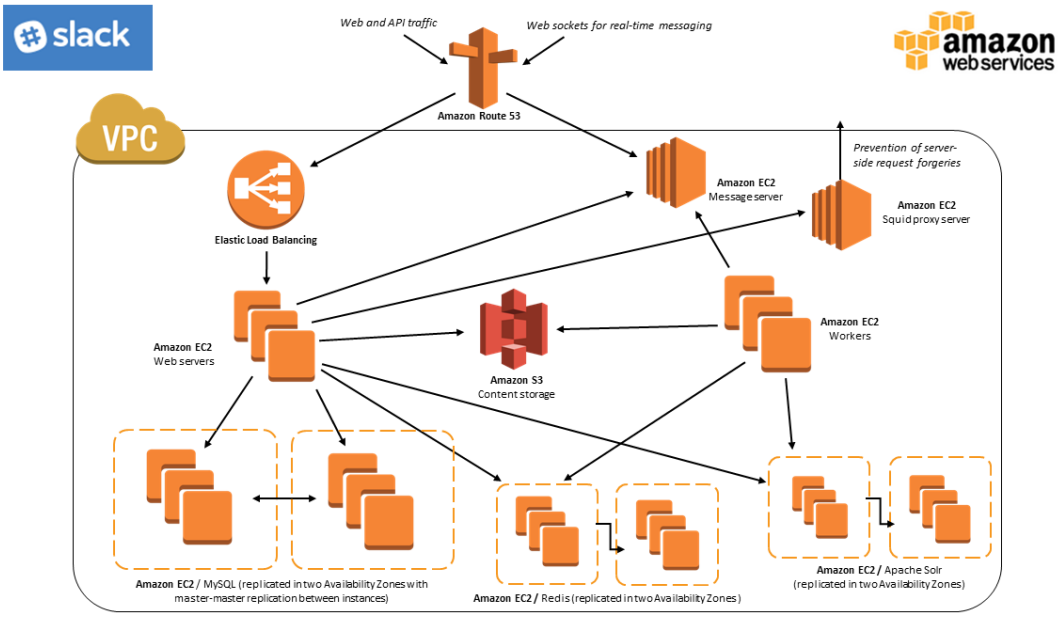
Приложение поддерживается для таких операционных систем, как iOS, Android, Windows Phone (beta), Mac OS X, Windows, Linux (beta). А также Slack имееи веб-интерфейс и может использоваться через браузер клиента[13].

1.2.2. Архитектура сервиса Slack

Сервис представляет из себя веб-интерфейс, десктопные и мобильные приложения. Slack воспользовался готовыми решениями и использует услуги Amazon Web Service(AWS). Конкретнее используются сервера, хранилище для документов и файлов, распределитель нагрузок и другие компоненты.

Richard Crowley, Director of Operations сообщил, что Slack решил воспользоваться продуктами AWS из-за их опыта и имени, которое уверяет клиентов в безопасности и надежности Slack продукта.

Схема архитектуры Slack отображена в следующей схеме, которое предоставил AWS:



Изображение 3. AWS и Slack архитектура

Backend

Slack является платформой для передачи сообщений, которые необходимо быстро доставлять между переписчиками. Обмен сообщениями по средством HTTP запросов не подойдет для такого случая. Технология Sockets(сокеты) позволяют наладить связь между сервером и клиентом и передавать сообщения напрямую. Серверный сокет прослушивает определенный порт в то время, как клиентский подключается к серверу по этому порту. После успешной установки соединения, сервер и клиент начинают обмениваться данными.

Slack использует отдельный JAVA сервер для того, чтобы прослушивать сокеты. Для API, используется сборка LAMP, которая базируется на следующих компонентах:

1. Linux, как операционная система
2. Apache HTTP Server, кроссплатформенный веб-сервер для обработки HTTP запросов
3. MySQL, как база данных
4. PHP, как основной язык написания кода

Database

Базы данных MySQL реплицированы и расположены на разных серверах. То есть на всех серверах расположены одинаковые копии базы данных и они между собой синхронизированы. Репликация базы используется, по большей части, для резервирования баз данных. То есть, если база данных на одном сервере откажет, то остальные копии будут поддерживать работоспособность сервиса.

Frontend

Приложение Slack поддерживается на многих платформах. Список включается в себя десктопные операционные системы Windows, OSX, Chrome OS и ведется разработка для Linux. Кроме того поддерживается приложение на мобильные операционные системы такие, как iOS, Android, и в разработке для Windows Phone. Также приложение имеет веб-интерфейс.

Приложение написано с помощью технологий JavaScript, HTML, CSS на Node.js. Благодаря Electron и MacGap, приложение адоптировано под десктопные и мобильные операционные системы.

Electron от GitHub позволяет создавать приложения на Mac, Windows, and Linux с использованием веб-технологий HTML, CSS, JavaScript, Chromium и Node.js. На данный моментом технологией Electron пользуется довольно много больших компаний, среди которых есть Microsoft, Facebook, и Docker[18].

MacGap выполняет ту же самую функцию, что и Electron, но специализирован только на создание родных приложений для OSX.

Для мобильных приложений используются родные языки программирования. Objective-C для iOS и Java для Android.

Хранилище

Для хранения файлов и документов, которыми обмениваются клиенты, используется готовое решение от Amazon. Amazon S3 предлагает неограниченный объём свободного пространства и высокую скорость передачи данных. Amazon также взимает плату только за реальное использование пространства и трафика.

1.3. TransferWise

TransferWise – p2p сервис для денежных переводов между клиентами. Запущен в январе 2011 года основателями Kristo Käärmann и Taavet Hinrikus. Проект начинался разрабатываться в Эстонии и со временем переехал в Англию. Штаб квартира находится в Лондоне, но имеет также офисы в Эстонии и Нью-Йорке[19].

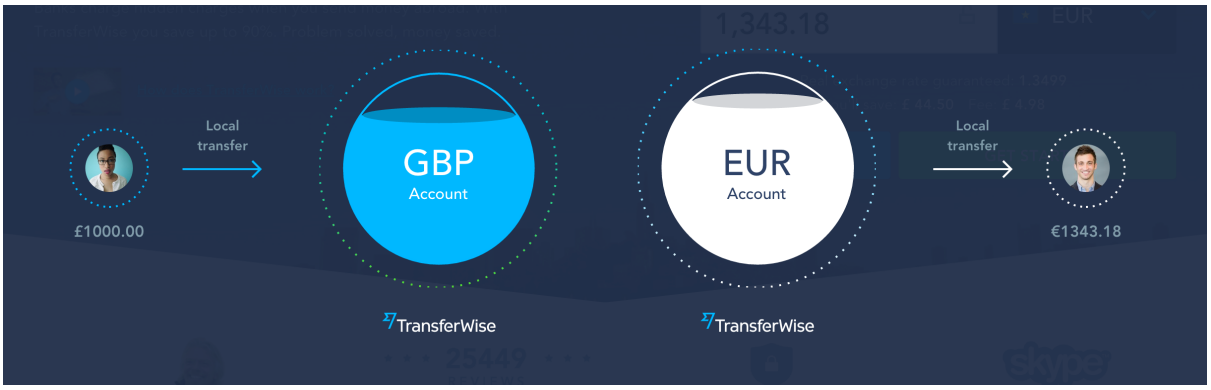
По словам со основателя Taavet Hinrikus, одного из первый работников Skype, он был разочарован в нынешней банковской системе переводов и конвертации денежных средств в иностранные банки - "I was losing five per cent of the money each time I moved it. At the same time my co-founder Kristo Käärmann (also from Estonia) was starting to get paid in the UK and was losing a lot of money transferring cash back home to pay for a mortgage there"[19].

1.3.1. Принцип работы

Принцип системы переводов в Transferwise был сравнен с принципом работы системы «Hawala», используемая преимущественно на Среднем Востоке, в Африке и Азии[20].

Система Hawala основана на переводе денежных средств путём однократных уведомлений по электронной почте, факсу или телефонному звонку. Материальные ценности в виде денег, золота и драгоценных камней не передаются между отправителем и получателем. Отправитель передает деньги местному брокеру, который в свою очередь отправляет сообщение получателю, что он может получить свои деньги у местного брокера. Расчёты между брокерами в дальнейшем проводятся по клиринговой(безналичные расчёты между странами, компаниями, предприятиями за поставленные, проданные друг другу товары, ценные бумаги и оказанные услуги, осуществляемые путём взаимного зачёта, исходя из условий баланса платежей) схеме[21].

Система переводов в Transferwise работает практически по этому же принципу.



Изображение 4. Принцип системы переводов денежных средств в TransferWise

Сравним перевод денежных средств в TransferWise и в местном банке.

Отправитель А из Эстонии хочет передать 1000 американских долларов получателю В в США.
В первом случае используем местный банк(AS SEB Pank) для перевода денежных средств электронным способом. Во втором случае используем сервис Transferwise.

Посмотрим прайс-лист AS SEB Bank для перевода заграницу денег и увидим, что для простого, не срочного перевода из Эстонии в США, используя интернет банк, необходимо будет заплатить пошлину в 24.92 евро[23]. Необходимо также учитывать и конвертацию валют. Банк будет конвертировать валюту по цене, которую они сами установили. Курс банка немного отличается от курса банка Европы и естественно в пользу местного банка.[24]

Если воспользоваться услугами Transferwise, то они возьмут за свои услуги 1 евро или 0.5% от посылаемой суммы. На 10 декабря 2015 года, при переводе 1000 долларов, можно было бы с экономить приблизительно 28.77 евро.

Изображение 5. Сравнение перевода денежных средств, TransferWise/AS SEB Bank

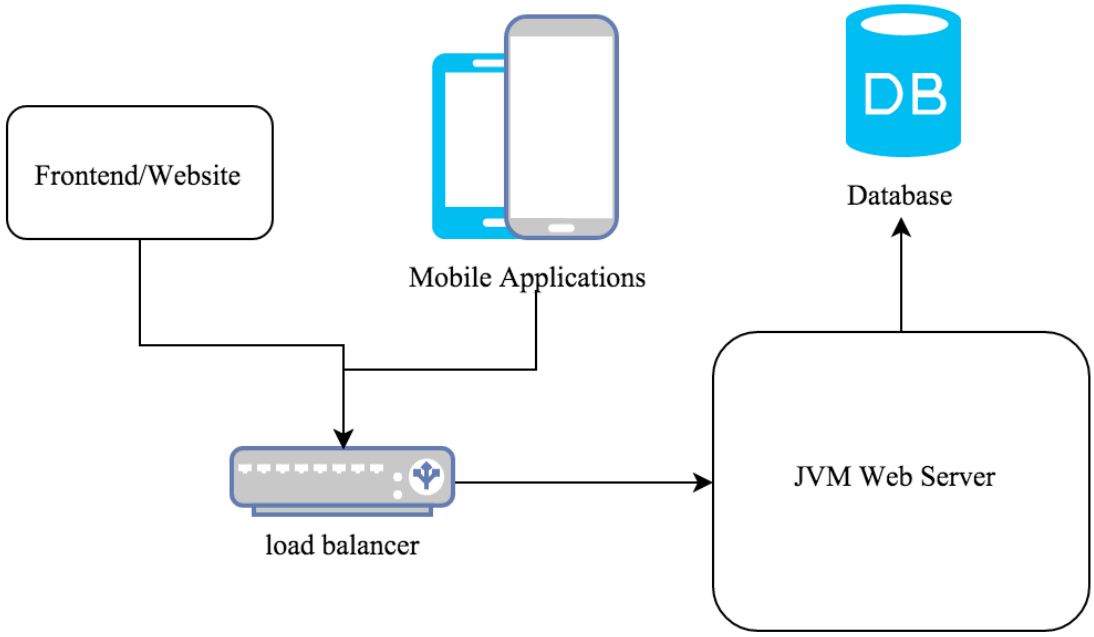
1.3.2. Архитектура сервиса TransferWise

Transferwise имеет и поддерживает мобильное приложение «TransferWise Money Transfer» для мобильных операционных систем iOS и Android. Мобильное приложение позволяет посылать денежные средства, смотреть историю, отслеживать статусы транзакций, а также изменять профильные данные учетной записи клиента.

Для тех, кто предпочтет воспользоваться услугами Transferwise через браузер, предлагается веб-интерфейс с такими же возможностями.

Transferwise не раскрывают информацию о своих серверах и о том где они расположены. Они также умалчивают и про архитектуру строения сервиса.

Можно предположить, что схема архитектуры могла бы выглядеть следующим образом:



Изображение 6. Предполагаемая структура архитектуры TransferWise

Backend

По словами Alvar Lumberg, ведущий инженер в компании Transferwise, сервис состоит из нескольких веб-приложений и небольших микро-сервисов. Точнее выразиться он не смог.

Сервера, на которых используется Apache Tomcat и Apache HTTP Server, установлен Grails framework[25], который является мощным инструментом для создания веб-приложений. Он написан на языке Groovy, который в свою очередь основан на Java.

Kristo Käärmann, со основатель компании, знал, что банки используют Java для своих сервисов и искал лучшее решение, используя этот язык. Grails оказался лучшим решением основанным на Java/JVM.

Для хранения информации используются как MySQL, так и PostgreSQL базы данных.

Для уменьшения нагрузки на сервера используется Load Balancer.

Frontend

Для сайта используется AngularJS фреймфорк, написанный на JavaScript.
AngularJS фреймворк предназначен для разработки одностраничных приложений, на которых контент меняется динамически, запрашивая данные у API путем простых запросов.

2. Анализ сервиса Splento

Для реализации веб-сервиса возьмем за основу новый стартап Splento, который начинал свою разработку в Эстонии около года назад[45]. Этот продукт представляет из себя предоставление услуг фотографов как для разного уровня мероприятий, так и для частных фотосессий. Этот проект уникален тем, что за счет автоматизированной системы проведения фотосессий, клиент или участники мероприятия получают свои фотографии в течение 48 часов.

Мобильное приложение и веб-интерфейс для фотографов позволяют незамедлительно загрузить фотографии на сервер, где подключаются в работу обработчики фотографий и модераторы.

Последним крупным проектом, в котором Splento удалось показать себя на высшем уровне, было мероприятие Web Summit, прошедшее 3-ого ноября 2015 года в Дублине[26].

Было проведено около 1750 фотосессий с участием выступавших стартапов[27].

Далее в этой работе проанализируем, что необходимо для автоматизации всех процессов, изучим составляющие компоненты и рассмотрим необходимые ресурсы и технологии для реализации похожего сервиса.

2.1. Принцип работы

Рассмотрим пример, как фото-студии и частные фотографы проводят для своих клиентов фотосессии[28].

Для проведения семейной фотосессии, клиентам необходимо найти фото-студию, в которой они могли бы сделать несколько групповых фотографий. Также есть вариант воспользоваться частными фотографами.

Обработка этих фотографий, в обоих случаях, может занимать от нескольких дней до месяцев. Зависит это, конечно, от занятости фотографов и сложности фотосессии.

Фото-студии, как правило, имеют немного фотографов и обработчиков, а также они особо не заинтересованы в масштабировании своего бизнеса. В таком случае, производительность творческого процесса будет равна производительности имеющихся фотографов и обработчиков.

Splento предлагает оптимизацию всех процессов для уменьшения времени ожидания результатов клиентами. Компания заинтересована в масштабировании сервиса и дать всем возможность воспользоваться услугами на профессиональном уровне.

Для соблюдения принципа «предоставить результат в течении 48 часов», Splento думает не только о поддержании необходимого количества профессиональных

фотографов и обработчиков, но и о поддерживания сервиса, для максимальной оптимизации действий.

Основные задачами, которые должен предоставлять веб-сервис Splento для клиентов являются:

1. Бронирование и оплата фотосессий на веб-сайте
2. Отслеживание статуса фотосессии
3. Предложение несколько разных пакетов услуг
4. Просмотр своих фото галерей
5. Просмотр открытых галерей(мероприятия или частные фотосессии)
6. Скачивание фотографий
7. Возможность купить фотографии, если не была оплачена заранее
8. Использование промо кода, для получения скидок
9. Онлайн консультации
10. Предоставление информации о продукте и услугах

Веб-сервис Splento предназначен не только для клиентов. Помимо веб-сайта, сервис имеет веб-приложение, панель управления для фотографов, обработчиков, модераторов и других лиц. Возможности этого интерфейса очень обширны и основными его задачами являются:

1. Создание и управление фотосессиями
2. Управление статусами фотосессий
3. Создание и управление информационными страницами
4. Предоставление аналитических данных по заказам, оплатам, посещениям, скачиваниям фотосессий
5. Управление контрактами между фотографами, обработчиками и других лиц
6. Управление учетными записями клиентов
7. Загрузка и управление обработанными и не обработанными фотографиями

Мобильные приложения для клиентов и фотографов решают также несколько задач, которые необходимы для оптимизации процесса.

Клиентское приложение должно позволять клиентам:

1. Заказывать и оплачивать фотоссесии и фотографии
2. Следить за статусом фотосессий
3. Смотреть собственные галереи

Приложение для фотографов должна выполнять следующие задачи:

1. Регистрация фотосессий
2. Поиск фотосессий без фотографа
3. Назначение фотографа для фотосессии
4. Передача гео-локационных данных фотографа
5. Управление статусами фотосессий
6. Отображение фотосессий с участием фотографа

2.2. Анализ необходимых ресурсов

Рассмотрев требования и функциональность, которые необходимо выполнять Splento, анализируем необходимые ресурсы для стабильной работы сервиса и выполнения всех поставленных задач.

Мобильные приложения, веб-сайт, панель управления для фотографов, обработчиков, модераторов и других лиц, являются необходимыми компонентами для работоспособности сервиса.

Естественно техническая сторона очень важна в этом случае. Как бы идеально не были продуманы эти компоненты по отдельности, их производительность полностью зависит от общения между собой и серверами.

Для того что бы мобильные приложения функционировали, необходимо реализовать, помимо всего остального, API. Который, путем запросов и ответов, будет получать и отдавать данные фотографам.

Сервера должны выполнять довольно большие нагрузки даже на первых этапах развития продукта. Операции, связанные с обработкой изображений, используют довольно тяжелые вычислительные процессы. Под обработкой изображений имеется ввиду, загрузка изображений, сжатие и их оптимизация.

Необходимо учитывать и довольно высокую нагрузку со стороны клиентов, а именно посещаемость сайта. Во время больших мероприятий, сервера подвержены наибольшим нагрузкам.

Сервис должен обладать надежным хранилищем для фотографий и архивов. Скорость загрузки и скачивания должны быть максимально высокими, на сколько позволяют технологии. Работоспособность хранилища должна быть без перебойной и в случае возникших проблем с файлами, необходимо иметь резервные копии файлов.

Для реализации похожего сервиса, выполняющего требования Splento, рассмотрим подробнее требования и создадим схему архитектуры сервиса.

2.2.1. Сайт

Для удовлетворения потребностей клиента, описанных в разделе 2.1, необходим сайт, который размещен на отдельном сервере, имеет публичный ip-address и привязан к доменному имени второго уровня. К примеру: example.com.

Сайт предназначен не только для управления фотосессиями, но он также является и визитной карточкой. Страницы сайта содержат много информации о сервисе, об предоставляемых услугах, а также контактную информацию. Что бы не терять клиентов, сайт должен всегда оставаться в рабочем состоянии и не зависеть от работоспособности других компонентов сервиса. Поэтому его необходимо иметь на отдельном сервере. Если сервис располагается на одной машине, то важно выделить для сайта отдельную виртуальную машину.

Клиентам необходимо предоставить возможность оплачивать забронированные фотосессии и фотографии через интернет. Для гибкости, на сайте должна быть форма для оплаты, позволяющая совершить покупку разными платежными системами. На эстонском рынке актуально было бы привязать к форме возможность оплаты через банки Seb, Swedbank, Nordea. Помимо этих методов, можно использовать PayPal, sagePay, которые позволяют совершать обычные междугородние платежи, имея кредитную или дебетовую карточку Visa, Mastercard и другие.
Есть более универсальные платежные системы. Braintree включает в себя сразу несколько возможностей оплат: PayPal, Apple Pay, Android Pay, Bitcoin, Credit/Debit Cards[29].

Разумеется сайт должен иметь связь с базой данных, для получения и хранения разной информации. Начиная с информации о фотосессиях, фотографиях, информации о клиентах и заканчивая деталями платежей.

Дизайн и интерфейс сайта должен быть простым и интуитивно понятным для клиентов. Важно иметь представление о user-experience, разрабатывая дизайн страниц для максимального привлечения аудитории.

При возникновении вопросов или проблем, также необходимо добавить возможность общения с группой поддержки. Это можно реализовать простой контактной формой или инструментами общения в реальном времени. На рынке существует множество готовых решений. LiveChat, Zendesk и их отдельный продукт Zopim, Freshdesk и многие другие, интеграция которых, не занимает много сил и времени.

2.2.2. Панель управления

Панель управления для фотографов, обработчиков, модераторов и других лиц можно представить в виде программы, написанную на Java или C языках, и путем общения с API посредством HTTP запросов или сокетов, отправлять запросы на сервер, а также получать данные.
Но этот вариант не очень гибок. Необходимо предоставить пользователям панели управления возможность использовать инструмент с любого компьютера или мобильного устройства без установки дополнительного программного обеспечения.

Создание веб-приложения на основе HTML, CSS, JS имеет преимущество из-за отсутствия необходимости установки дополнительных программ, так как популярные операционные системы поставляются с уже установленным браузером.

По сути, панель управления является тем же сайтом, только с ограниченным доступом. Так как клиентский сайт должен быть отделен от всех компонентов, это веб-приложение должно находиться отдельно от сайта, на сервере с выделенным ip-адресом и, для удобства использования, должен быть привязан к доменному имени третьего уровня. К примеру: control.example.com.

2.2.3. Мобильные приложения

Мобильные приложения для клиентов и фотографов могут быть написаны для разных мобильных операционных систем, как для iOS, Android, Windows Phone.

Единственными условиями для приложения фотографа являются:

1. Подключение к интернету Wifi и 3G.

   Приложение для фотографов должно быть постоянно подключено к интернету для регистрации фотосессии и для отправления статусов процесса фотосессии и гео-локационных данных на сервер.

2. Камера

   Во время регистрации фотосессии в «боевых условиях» на мероприятии через мобильное приложение, фотографу необходимо внести данные о клиенте. На сервере создается фотосессия и в некоторых случаях на сервер необходимо присылать опознавательную фотографию, что бы обработчики и модераторы смогли определить, к какой фотосессии относится определенный клиент.

Клиентскому приложению достаточно иметь доступ к интернету для получения информации о фотосессиях, просматривать галереи, заказывать фотосессии и покупать фотографии.

2.2.4. API

Для общения сервера с мобильными приложениями необходим API, основанный на простых запросах к серверу. Это по большей части, легкий интерфейс, который может передавать данные, в большинстве случаях, по HTTP или другому протоколу.
Он позволяет сохранять информацию получаемую от приложений в базу данных, а также выдавать ее обратно отправившему запрос.

2.2.5. CRON

Сервису необходимо много работать с изображения, что является довольно тяжелыми задачами для процессоров серверов. CRON – является под-сервисом, который должен работать на отдельном сервере и использовать максимум мощности этого сервера. Имя CRON позаимствовано от одноименной программы-планировщика задач в UNIX-подобных операционных системах, использующийся для периодического выполнения заданий в определённое время.

CRON под-сервис является обработчиком очереди. База данных содержит данные о процессах, которые необходимо выполнить этому под-сервису. Это могут быть, как оптимизация изображений, так и рассылка почты клиентом.

2.2.6. Базы данных

Сервис обладает большим количеством данных. Ниже представлен список большинства:

1. Учетные записи клиентов, фотографов, обработчиков, модераторов и других ролей
2. Аналитические данные выше перечисленных лиц
3. История покупок фотосессий клиента
4. Детали фотосессий, галерей, и фотографий
5. Данные о мобильных телефонах клиентов и фотографов, таких как гео-локация и уникальный ключ телефона, для push-уведомлений
6. Записи о промо кодах

Для хранения данных достаточно MySQL или PostgreSQL. Но среди этих данных есть те, которые постоянно подвергаются изменениям. К примеру гео-локационные данные клиентов и фотографов. Для таких данных не подойдут обычные реляционные базы данных типа MySQL или PostgreSQL. Необходимо дополнительно использовать более легкую базу данных справляющаяся с быстрыми и постоянным обновлениями базы данных, к примеру на основе NoSQL, MongoDB.

2.2.7. Хранилище

Для хранения фотографий и архивов необходимо очень много места. Одна семейная фотосессия может содержать до 50 фотографий. Понятное дело, что в итоге модераторы отберут лучшие фотографии, но в любом случае их необходимо где-то хранить. Помимо оригинальных фотографий, генерируются оптимизированные изображения разных размеров, для их отображения на сайте или в панели управления.

Хранилище должно быть расширяемым. То есть в любой момент должна быть возможность добавить пространства для увеличения свободного места.

Также стоит учитывать, что хранилище должно быть настроено отдельно от серверов, чтобы нагрузка процессоров и памяти от работы с файлами не влияло на работу сервиса и его компонентов.

Есть множество готовых решений, зависящих от потребностей продукта. Самый простой вариант – это использовать файловые хранилища типа Dropbox. Dropbox имеет хороший API для работы с контентом хранилища, но в нашем случае он не подойдет по следующим причинам:

1. Dropbox не предназначен для быстрой загрузки и отдачи файлов

   В нашем случае очень важно максимально быстро загружать файлы в хранилище и еще быстрее выдавать их клиентам. Фотографии в галереях на сайте должны загружаться прямо из хранилища, чтобы не нагружать сервер с сайтом лишний раз. Поэтому очень важна быстрая скорость скачивания фотографий для быстрого отображения содержимого страниц с фотосессиями.

2. Неразумные затраты на содержание хранилища.

   Ценовая политика Dropbox выглядит следующим образом. При неограниченном свободном пространстве необходимо использовать Business пакет услуг стоимостью 648€ в год.

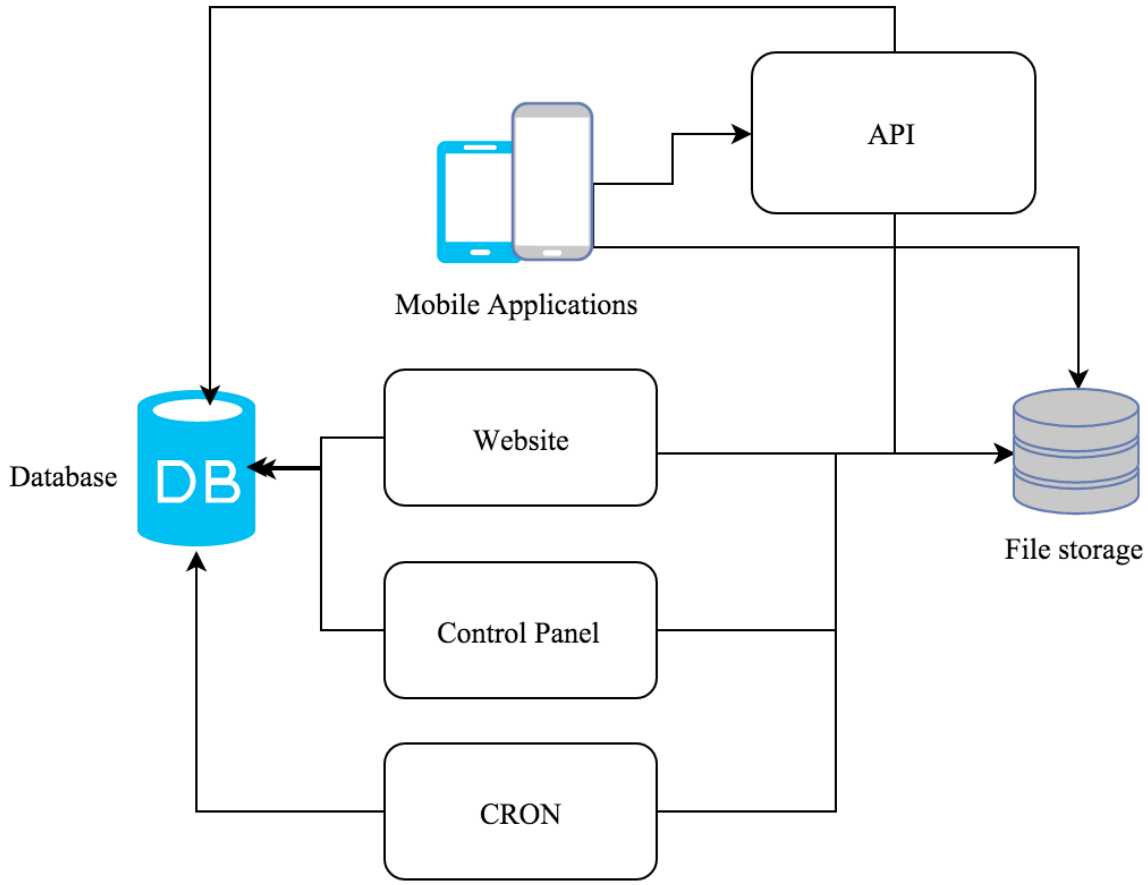
Для наших требований необходимо что-то более эластичное, с неограниченным пространством, с быстрой отдачей и загрузкой файлов и с разумной ценовой политикой.

Amazon Web Service Storage S3, Google Cloud Storage, Windows Azure Storage и похожие сервисы удовлетворяют заданным требованиям[30]. Присутствие API, множество дата-центров в разных регионах, оплата за реальное использование пространства и трафика, высокая скорость передачи данных, все это удовлетворяет нашим потребностям.

3. Реализация сервиса

Изучив данные требования и поставленные цели, можно уже представить, что сервис будет выглядеть громоздко. В совокупности сервис будет состоять из двух мобильных приложений, клиентского веб-сайта, веб-интерфейса для панели управления, API, CRON, базы данных.

Схему архитектуры сервиса можно представить следующим образом:



Изображение 7. Простая архитектура сервиса

3.1. Анализ платформ и языков

На сегодняшний день существует довольно большое количество платформ для реализации компонентов веб-сервиса. В первой главе этой работы мы изучили технологии, которыми пользуются некоторые большие продукты. Каждый из них использует, практически, отличный друг от друга, стек технологий.

Неотъемлемой частью реализации сервиса стоит выбор платформы и языка программирования. Для веб-сервиса, Transferwise анализировал рынок и определил, что большинство банков используют платформу, написанную на Java/JVM[24]. Найдя хорошую платформу на Java, Transferwise решил воспользоваться данной платформой.

Uber начал разработать свой продукт с php и mysql. В скором времени решили использовать более современное решение, и использовать Node.js из-за своей скорости работы и простоты.

По началу большинство стартапов и продуктов пытаются создать минимальный продукт или MVP, для того чтобы проверить реакцию пользователей, есть ли необходимость в этом продукте и стоит ли продолжать ли его разработку. Если мы говорим про язык или платформу, то в большинстве случаях выбирается то, что выходит дешевле и ближе разработчикам. Разработчики смотрят на популярность языков и платформ, а также на поддержку этих элементов. Все стремятся использовать более современное решение потому, что, как правило, они более совершенны.

В нашем случае, не смотря даже на популярность, PHP и JavaScript языки является ближе[34]. В дальнейшем будем отталкиваться от них.

3.2. Сайт

Клиентам и посетителям необходимо пользоваться услугами и получать информацию о нашем продукте. Веб сайт проекта является не самым важным, но играющим большую роль для сервиса.

Множество готовых инструментов, движков, фреймворков позволяют развернуть сайт в считаные минуты, но большая часть из них были разработаны для определенного типа контента, обладают функциональностью, в которой нет необходимости, а также старые и не поддерживаются разработчиками.

Wordpress, Drupal, Joomla php-движки являются готовыми решениями для создания сайта, с минимальными знаниями в программировании или о том, как устроен движок. Они обладают многими плюсами. Такими, как панель управления для менеджмент контента, визуальный и текстовый редакторы для удобного создания страниц и их содержимого, подключение новых модулей. Имеют легкий и интуитивный интерфейс.

Возможно для кого-то этого будет достаточно, но в нашем случае мы не может использовать данное решение. Эти движки являются через чур тяжелыми, обладают функциональностью, нужды в которых у нас нету, и тяжело модифицировать без глубокого изучения работы движков. Изучение функционала движков занимает много времени, а MVP версию продукта необходимо разработать, как можно быстрее.

Альтернативой этим движкам, есть легкие фреймворки типа Laravel(php), Yii(php), Rails (Ruby), Django (Python). Они обладают необходимым минимальным функционалом и набором компонентов для создания веб-приложений.

Laravel, как и другие фреймворки, разработаны так, чтобы легко можно привязаться любую библиотеку, модуль или функционал[35].
В Laravel можно отметить следующие удобства:

1. Dotenv - .env файл содержит разные глобальные параметры, пароли и другие данные для подключения к базам данных и сторонним API.
2. PSR-4 - позволяет подключать все библиотеки, необходимые для работы модулей фреймворка.
3. Валидация параметров – при подачи правил и значений параметров, модуль валидации проверяет и обрабатывает эти значения.

   К примеру для валидации эмаила проведем следующую операцию:

   ```
   $input = [
      'email' => 'email@gmail.com'
   ];
   $rules = [
      'email' => 'required|email'
   ];

   $validation = Validator::make($input, $rules);
   ```

4. Eloquent ORM - реализация щаблона ActiveRecord, связывание сложных объектов приложения с таблицами баз данных.
5. Flysystem – модуль для управления файловыми системами. Основываясь на параметрах, указанных в конфигурационных файлах, легко можно взаимодействовать с локальными файлами, awsS3, dropbox, rackspace и другими файловыми системами в облаке.
6. Cache, Artisan и многие другие удобства.

Если отойти от PHP и воспользоваться JavaScript, то существует возможность использовать технологии AngularJS или Backbone.js. Плюсом этой возможности является то, что сервер перестает затрачивать ресурсы на генерацию веб-страниц. Браузеры клиентов выполняют эти действия сами, а динамический контент для страниц получают от запросов к API.
Из представленных возможностей, использование AngularJS или Backbone.js, самый перспективный. Но в API необходимо будет добавить дополнительный функционал.

К примеру для регистрации нового пользователя на стороне клиента, стоит добавить функцию для обращения на сервер к API:

```
function Create(user) {
   return $http.post('/api/users', user).then(handleSuccess, handleError('Error creating user'));
}
```

API в свою очередь обработает этот запрос, и выдаст соответственный ответ.

Для того, чтобы расположить сайт необходим сервер, а также база данных, но об этом поговорим чуть дальше.

3.3. Мобильные приложения

Для создания мобильных приложений можно использовать родной язык аппарата. Тоесть для iPhone 6S можно написать приложение на Objective C или Swift, а для Samsung Alpha – на языке Java.

Если нет возможности разработать приложение на родном языке, то есть альтернативы в виде PhoneGap. PhoneGap позволяет создавать мобильные приложения с использованием HTML, JS и CSS.

Естественно мобильные приложения должны общаться с API для обмена информацией.

К примеру для получения списка фотосессий, в которых участвовал клиент, на сервер необходимо отправить соответствующий запрос, написанный на Objective C:

```
- (void)getUserSessionsWithAccessToken:(NSString*)accessToken onSuccess:(void (^)(NSArray* sessions))su
ccess onFail:(void (^)(NSError* error))failure
{
   [[_requestOperationManager requestSerializer] setValue:accessToken forHTTPHeaderField:kAccessTokenH
eaderName];

   NSLog(@"Obtaining session with accessToken: %@...", accessToken);

   [self.requestOperationManager GET:@"sessions" parameters:nil success:^(AFHTTPRequestOperation *oper
ation, id responseObject) {
      success(responseObject);
   } failure:^(AFHTTPRequestOperation *operation, NSError *error) {
      NSLog(@"Failure obtaining sessions: %@", error);
      if (failure) {
         failure(error);
      }
   }];
}
```

Используя AFHTTPRequestOperation модуль, отправляем на сервер запрос /sessions с HTTP Method параметром GET и HTTP Header accessToken – ключ доступа к серверу, выдаваемый сервером клиенту.

3.4. Панель управления

Необходимо реализовать панель управления для модераторов, фотографов, обработчиков и других лиц, в которой они могли бы выполнять ряд действия, которые были описаны во второй главе.

Создание веб-интерфейса позволит гибко использовать функционал, нежели программа написанная на Java или C языках, и привязанная к определенной операционной системе.

Как и веб-сайт, панель управления можно реализовать двумя способами. Использовать php-фреймворк, типа Laravel, или JavaScript-фреймворк AngularJs или Backbone.js.

3.5. API

Api является важным элементов среди всех компонентов сервиса. Он является посредником между данными и клиентами[33].
Чтобы мобильные приложения, и в некоторых случаях сайт и панель управления, могли получать данные из базы данных, а также создавать и изменять записи, необходим API.

То есть, посредством простых HTTP запросов к серверу, API примет их и выполнит необходимые действия. В случае, если клиент ожидает ответ, API его предоставит.

Приведем пример на простом запросе от мобильного приложения фотографа, на получение списка фотографий в определённой фотосессии, если мы знаем, что идентификационный номер этой фотосессии является 575.

Выполним HTTP запрос к серверу со следующими полями:

```
Request:

        HTTP Method: GET
        HTTP Url: api.example.com/sessions/575/photos
        HTTP Headers:
                accessToken: some1Long2Genreated3Key4By5SERVER6Cheers7
```

Как видно из примера, мы выполняем GET запрос по соответствующей ссылке, при этом в запросе в Header мы добавляем параметр accessToken – ключ доступа.

Ключ доступа является уникальным, сгенерированным сервером, длинным текстом содержащий как буквы, так и числа. В нашем случае он выдается зарегистрированным фотографам и клиентам. Этот ключ позволяет защитить сервис от действий выполняемых неизвестными лицами.

API, проверив ключ на корректность, выдаст в ответе необходимую информацию.

Response:

HTTP Status: 200
HTTP Body:


```
{
   "photos": {
      "url": "http://files.example.com/575/ksdfADf34SDdapljntGH4.jpg",
      "url": "http://files.example.com/575/34FDcwasdljntGH4.jpg",
      "url": "http://files.example.com/575/kf34ASDdas234ntGH4.jpg",
      "url": "http://files.example.com/575/asd3459asdGH7ysdf5.jpg"
   }
}
```


От сервера мы получили ответ со статусом 200 и содержанием в json формате.

В зависимости от корректности запроса, сервер может выдать соответствующий статус в ответе. Если придерживаться стандартам, разрабатывая API, то статус 200 означает, что все прошло успешно[36]. Сервер может выдать статус 404, если ресурс не был найдет или 500, если произошел сбой сервера.

Некоторые ответы содержат в себе исключительно статус запроса. То есть, если приложению нет необходимости в получении каких-то данных, то достаточно будет статуса, что все прошло успешно или наоборот. При попытке аутентифицировать себя в приложении, сервер может ответить только статусом 401, что значит неверные данные для аутентификации(ключ доступа или пароль).

Другие запросы требуют ответа от API. В таком случае в ответе будет параметр body, в котором заключены какие-то данные.

В примере ответа выше, мы получили объект, содержащий ссылки на фотографии фотосессии. Этот объект закодирован в формате json. Содержимое ответа можно представить разными форматами; json, xml, простым текстом и другими способами.

JSON является легким текстовым представлением объектов или строковых данных. Запросы и ответы получаются легкими и компактными.

Если мы хотим представить массив с ссылками в текстовом формате, то в php языке это можно показать следующим способом.

```
$array = [
        "photos" => [
                "url" => "http://files.example.com/575/ksdfADf34SDdapljntGH4.jpg",
                "url" => "http://files.example.com/575/34FDcwasdljntGH4.jpg",
                "url" => "http://files.example.com/575/kf34ASDdas234ntGH4.jpg",
                "url" => "http://files.example.com/575/asd3459asdGH7ysdf5.jpg"
        ]
];

$result = json_encode($array); - функция закодирует массив в формат json.
```

Для реализации API использование фреймворка Laravel неразумно, хотя и возможно. Для таких вещей используются фреймворки по легче. Slim фреймворк или Lumen от Laravel являются легкими решениями в данной ситуации.

Slim микро-фреймворк обладает самым минимальным и необходимым функционалом для того, что бы реализовать API на PHP[37].

HTTP Router позволяет с легкостью настроить маршрутизатор. В случае запроса к api.example.com/sessions/575/photos, маршрутизатор выполнит действия указанные в коде.

```
$app->get('/v1/sessions/(:photosetid)/photos', function ($photosetid) use($app) {
        …
}
```

Поддержка Middleware – позволяет внедрить в фреймворк компонент для обработки запросов.

```
$app->add(new \Core\System\HttpBasicAuth()); - класс внедреный для
аутентификации пользователей и запросов
```

Container позволяет хранить инициализированные классы и переменные внутри фреймворка.

```
$myService = $this->get('myService');
```

3.6. CRON

CRON - самый ресурсоёмкий компонент. Именно он представляет из себя очередь действий. Доставая информацию из базы данных, cron запускает скрипты, которые могут рассылать электронные письма, обрабатывать фотографии, проверять сервер на работоспособность и так далее. Его необходимо содержать на отдельном сервере или на отдельной виртуальной машине.

Название CRON исходит от одноименной программы в Linux OS и работает по тому же принципу.

Его можно реализовать в нескольких разных вариантах.

Один способ – это использовать Linux OS программу crontab для запуска скриптов в необходимое время.
Отредактировав файл очереди командой crontab –e, можно управлять очередью событий.

```
*/10 * * * * /usr/bin/somedirectory/somescript - этот скрипт будет запускаться каждые 10 минут
```

Второй способ, использовать Laravel, который имея модуль Console, позволяет запускать скрипты на php в определенное время.

К примеру чтобы разослать уведомления клиентам, необходимо выполнить некий процедурный код, описанный в php классе SendPhotosessionNotices.

```
$schedule->command('send-photosession-notices')->hourly();
```

Каждый час этот скрипт будет проверять, есть ли клиенты, которым необходимо разослать уведомление, и если есть, то уведомит их.

3.7. База данных

Для хранения нашей информации нам необходимо использовать два типа базы данных. Реляционная база данных, типа MySQL, необходима для хранения все информации, которая у нас существует, от учетных записей клиентов до деталей платежей. Она не будет хранить в себе только информацию, которая постоянно меняется. Такую, как гео-локационные данные фотографов и клиентов. Для такого типа данных необходима более легкая версия базы данных. Для этого подойдет MongoDB.

MongoDB объектно-документарная база данных. Представлена, по сути, в виде key-value хранилища. С одной стороны, она позволяет делать очень быстрые операции над объектом, зная его идентификатор, а с другой стороны, предоставляет мощнейший инструмент для сложных взаимодействий.

В нашем случае, в ней существует очень полезный инструмент работающий с гео-пространственными данными. Одной из удобных вещей, является поиск объектов на определенном расстоянии от заданной точки.

Представим себе, что мы имеем объект с параметром координат(loc) типа Point.

```
db.userslocations.insert({
        user_id:'testuser@splento.dev',
        loc:{ type : "Point", coordinates : [  -122.262168,  37.531595 ] },
        last_update: '2015-04-02', user_type:'pro'
        })
db.userslocations.ensureIndex({
        "loc" : "2dsphere"
        })
```

Следующим запросом в базу данных, мы можем получить все объекты, которые находятся на каком-то расстоянии.

```
$query = [
        'loc' => [
            '$near' => [
                '$geometry' => [
                    'type' => 'Point',
                    'coordinates' => [-122.262168,  37.531595]
                ],
                '$maxDistance' => 7000
            ]
        ]
];

$collection->find($query);
```

3.8. Хранилище

Хранилищем для фотографий будет служить AWS S3 из-за его репутации, надежности и ценовой политики. Они также имеют официальные библиотеки на разных языках, в том числе и php, для упрощенной работы с хранилищем.

К примеру, для получения всех изображений в фотосессии 567, реализуем следующим кодом:

```
$this->client = S3Client::factory([
        'credentials' => [
                'key'    => 'S3_ACCESS_KEY',
                'secret' => 'S3_SECRET_ACCESS_KEY',
        ],
        'region' => 'S3_REGION',
]);

$photoList = $this->client->listObjects(array(
        'Bucket' =>  'BUCKET_NAME',
        'Prefix' => '567/original/',
))->getAll();
```

3.9. Реализация серверов

Прежде, чем реализовывать архитектуру сервиса и распределять компоненты между серверами, необходимо проанализировать предполагаемую нагрузку, определить самые подверженные нагрузкам, компоненты, и по возможности отделить их.

Как описывалось ранее, многие компоненты сервиса стоит отделить друг от друга для уменьшения риска нестабильной работы. То есть, чтобы при нагрузке API, база данных продолжала работать на полную мощность.
В идеальном случае, стоило бы все компоненты отделить друг от друга, но на начальном этапе хватит и отделения базы данных, веб-сайта и CRON от других компонентов. API и панель управления могут находиться в одном окружении, на одном сервере.

Определимся сразу с операционной системой, которую будем ставить на сервера.
Нет определенной операционной системы, которая была бы хороша в нашем деле, но основанные на Linux больше подойдут, чем остальные[38][39]. Возьмем Ubuntu 14.04.

База данных

Для начала реализуем базу данных. Мы уже определились, что выведем ее на отдельный сервер. Это позволит увеличить ее производительность и негативное влияние на остальные компоненты. Поставив операционную систему на сервер, установим mysql.

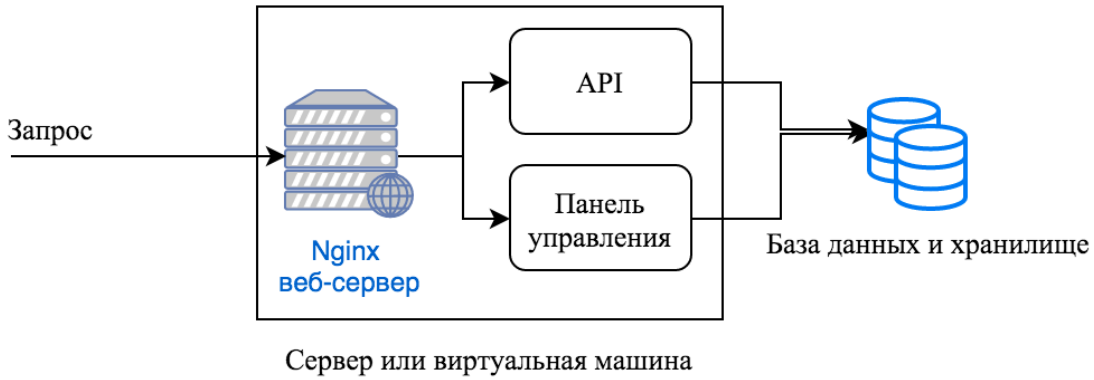
```
sudo apt-get install mysql-server
```

Необходимо также удостовериться, что к базе данных можно подключиться не только локально, но и из вне. То есть, при запросе mysql_connect('server_ip_address', 'user', 'pwd') на сервере с API, доступ к базе должен быть открыт.

Панель управления и API

Рассмотрим вариант реализации сервера с панелью управления и API. Далее будем называть их приложениями.

Простая схема может выглядеть следующим образом:



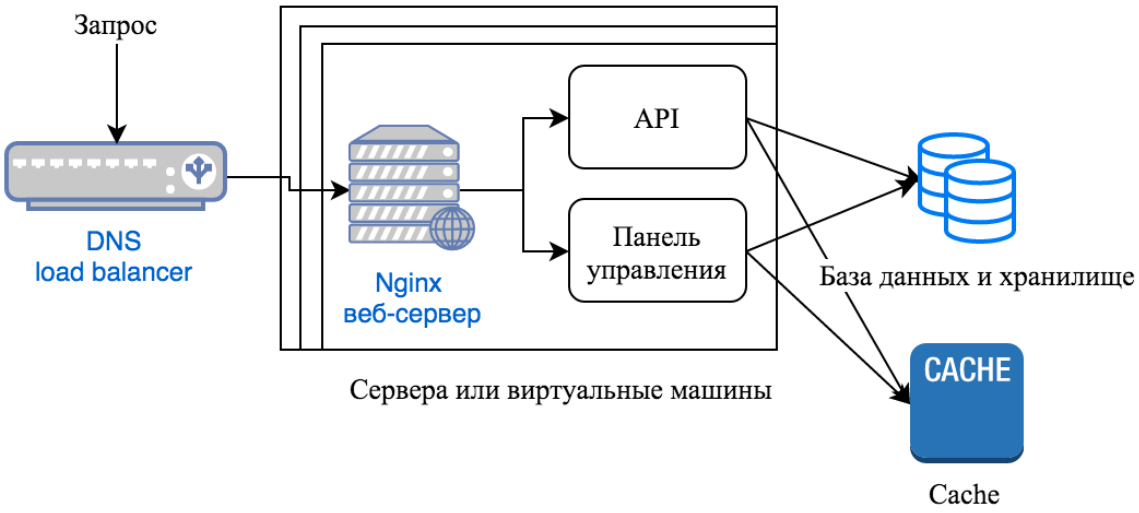
Изображение 8. Архитектура веб-сервера

В этом случае база данных будет находится на отдельном сервере, а веб-сервер nginx и приложения на одном. Каждый из компонентов, API и панель управления будут находиться в разных директориях. Nginx веб-сервер настраиваем так, чтобы при запросе к компоненту, веб-сервер направил к нужной директории приложения.

```
server {
        listen 80;
        server_name api.example.com;

        root /var/www/api;
        index index.php;

        location ~* \.(php)$ {
              fastcgi_pass 127.0.0.1:9000;
              fastcgi_index index.php;
              include fastcgi_params;
              fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
         }

        …
}

server {
        listen 80;
        server_name control.example.com;

        root /var/www/control;
        index index.php;

        location ~* \.(php)$ {
              fastcgi_pass 127.0.0.1:9000;
              fastcgi_index index.php;
              include fastcgi_params;
              fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
         }

        …
}
```

Из примера конфигурационного файла nginx, видим что про обращении к API по ссылке api.example.com/sessions, веб-сервер nginx направит в директорию /var/www/api. В этой директории API-приложение, использующее Slim framework, подхватит запрос и обработает.

Конечно при масштабировании проекта, необходимо оптимизировать архитектуру сервиса. В данном случае, можно создать несколько серверов с копиями веб-сервиса и приложений и с помощью dns-балансировки, рассредоточивать нагрузку между серверами.



Изображение 8. Улучшеная архитектура веб-сервера

Не стоит забывать и про кеширование данных. Memcache позволит кешировать часто требуемые данные в памяти, что сократит количество запросов к базе данных.

```
$m = new Memcache;
$m->get('user1');
```

Веб-сайт

Веб-сайт реализуется по тому-же принципу, который был описан выше.

CRON

Для CRON необходим мощный сервер и установленные на нем необходимые программы для обработки изображений. В нашем случае, GD Graphics Library. Веб-сервер для CRON не надо настраивать потому, что к нему обращаться из вне нет необходимости. CRON работает только с базой данных, хранилищем и другими, типа почтового сервиса, для рассылки электронных сообщений.

Разумеется необходимо настроить домен. У панели управления, где зарегистрирован домен example.com, необходимо указать адреса веб-серверов или dns-балансировщика. При нашей архитектуре api.example.com и control.example.com должны вести к одному веб-серверу, а example.com к тому, где находится веб-сайт.

Заключение

Целью данной работы было изучить новый стартап Splento и попытаться реализовать сервис, который бы выполнял похожий функционал.

В этой работе мы рассмотрели несколько крупных продуктов, которые имеют довольно таки большую популярность по всему миру.

Uber, TransferWise, Slack довольно быстро развились и их масштабность удивляет. Splento верит в свою идею и стремится достичь успеха.

Мне удалось пообщаться со всеми этими компаниями и разузнать, как устроен их сервис, какие технологии они используют, как масштабируют свой продукт. В интернете также достаточно информации, чтобы представить себе, как должен выглядеть сервис, который стремится к успеху. В этой работе мне удалось отобразить только верхушку всего, и естественно при глубоком изучении этой темы можно узнать много интересного и полезного.

Хотелось бы подчеркнуть несколько вещей, которые я пытался донести в работе:

1. Сложные сервисы, как те что мы изучили, состоят из многих компонентов и некоторые из них выполняют очень сложные операции. Очень важно отделять такие компоненты и выделять для них отдельные ресурсы. В противном случае, они будут отбирать вычислительные ресурсы у других компонентов.
2. Сайт является лицом компании и по возможности его стоит отделять от других компонентов, что бы он работал без перебоев.
3. При масштабировании продукта лучше дублировать сервера и виртуальные машины и с помощью балансировщика(load balancer), распределять нагрузку между серверами.
4. Для базы данных стоит выделять отдельный сервер и не забыть про ее реплицирование, а также использовать кеширование, чтобы уменьшить количество запросов к базе данных.
5. В большинстве случаев используйте готовые решения и не изобретайте велосипед. Это с экономит время на разработку продукта.

Kokkuvõte

Antud bakalaureusetöö põhieesmärgiks oli uurida uue startapi, Splento ja proovida rakendada teenust, mis täidab sarnast funktsionaalsust.

Töö käigus, me vaatasime läbi mitmeid kuulsaid projekte, ja tehnoloogiaid, mida nad kasutavad.

Uber, TransferWise, Slack on projektid on üsna kiiresti arenenud ja nüüd on nad suuremad tooted. Splento usub enda ideesse, ja püüab saavutada edu.

Mul oli võimalus suhelda nendega ja saada teada, kuidas need veebi teenused töötavad ja milliseid tehnoloogiaid nad kasutavad. Internetis on ka piisavalt informatsiooni, et ette kujutada, kuidas peaks välja nägema veebirakendus, mis püüdleb edu. Selles töös, mul oli võimalus näidata ainult mõned asjad, ja muidugi sügavalt selle teema uurimist saab õppida palju huvitavat ja kasulikku.

Tahaksin rõhutada, et:

1. Rasked veebirakendused, mis koosnevad paljudest komponentidest, täidavad väga keerukaid operatsioone. On oluline, et eraldada need komponendid ja anda nende individuaalseid ressursse.
2. Veebi leht on firma nägu ja see on oluline eraldada seda muudest elementidest, et töötada ilma probleemideta.
3. Dubleerige serverid ja virtuaal masinad, kui toode on tagi. Samuti kasutage load-balancer, et vähendada serveri ülekoormus.
4. Andmebaas peaks asuma oma serveris. Ära unusta andmebaasi replikatsioonist, ja kasutage cache tehnoloogiat, et vähendada andmebaasi kasutamine.
5. Enamikul juhtudel kasutage valmis lahendused ja ei ole vaja hakata jalgratast leiutama. See on aja säästmise tootearendus.

Conclusion

The main goal of this bachelor's thesis was to explore new startup Splento and try to implement the same web-service, that would perform similar functionality.
In this paper, we reviewed several famous products and startups, and technologies that they use.

Uber, TransferWise, Slack projects have been quite quickly developed and now they have big products. Splento believe in own idea and seeks to achieve success.

I was able to communicate with all of these companies and find out what's inside their service, what technologies they use and that they use for scaling product. A lot of information can be found in web to imagine how should look a service that strives for success.
In this paper, I was able to show only the main things, and of course with a deep researching can find a lot of interesting and useful.

I would like to emphasize a few things:

1. Big services, like the ones that we have explored, contain many components, and some of them perform very complex operations. It is important to separate these components and to provide for their individual resources.
2. The site is the face of the company and try to separate it from the other components.
3. Duplicate servers and virtual machines, if the product is scaling. Also use the load balancer, to decrease the servers overloading.
4. The database should be located on own server. Do not forget about database replication, and use caching to reduce the number of queries to the database.
5. In most cases, use exist solutions and do not reinvent the wheel. This saves your time for product development.

Использованная литература

1. Uber company – Wikipedia [WWW]
   https://en.wikipedia.org/wiki/Uber_(company) (28.12.2015)
2. Where is Uber Currently Available? [WWW]
   https://www.uber.com/cities (28.12.2015)
3. Uber iOS application – Apple App Store [WWW]
   https://itunes.apple.com/us/app/uber/id368677368?mt=8 (28.12.2015)
4. Uber Android application – Google Play [WWW]
   https://play.google.com/store/apps/details?id=com.ubercab (28.12.2015)
5. Uber Windows Phone application – Microsoft Store [WWW]
   https://www.windowsphone.com/en-us/store/app/uber/b905a877-bd55-4ce7-a7aa-467cdc3a21f4 (28.12.2015)
6. Introducing the Uber API – Uber Newsroom [WWW]
   http://newsroom.uber.com/introducing-the-uber-api/ (28.12.2015)
7. What is the technology stack behind Uber? – Quora [WWW]
   https://www.quora.com/What-is-the-technology-stack-behind-Uber/answer/Ryan-McKillen (28.12.2015)
8. Uber's Android app is not 'literally malware', despite what you may have read – TheNextWeb [WWW]
   http://thenextweb.com/apps/2014/11/27/ubers-app-malware-despite-may-read/ (28.12.2015)
9. Uber is now available for BlackBerry 7! – Uber Newsroom [WWW]
   http://newsroom.uber.com/2013/09/uber-is-now-available-for-blackberry-7/ (28.12.2015)
10. Node.js Meetup: Distributed Web Architectures - Curtis Chambers, Uber – Joyent [WWW]
    https://www.joyent.com/developers/videos/node-js-office-hours-curtis-chambers-uber (28.12.2015)
11. Slack (software) – Wikipedia [WWW]
    https://en.wikipedia.org/wiki/Slack_(software) (28.12.2015)
12. Come work with us – Slack [WWW]
    https://slack.com/jobs (28.12.2015)
13. Slack apps for computers, phones & tablets – Slack Help Center [WWW]
    https://get.slack.help/hc/en-us/articles/201746897-Slack-apps-for-computers-phones-tablets (28.12.2015)
14. Uploading and sharing files – Slack Help Center [WWW]
    https://get.slack.help/hc/en-us/articles/201330736-Uploading-and-sharing-files (28.12.2015)
15. AWS Case Study: Slack – AWS Case Study [WWW]
    https://aws.amazon.com/solutions/case-studies/slack/ (28.12.2015)
16. What kind of technology does Slack use? – Quora [WWW]
    https://www.quora.com/What-kind-of-technology-does-Slack-use (28.12.2015)
17. Desktop Application Engineer– Come work with us, Slack [WWW]
    https://slack.com/jobs/69902/desktop-application-engineer (28.12.2015)
18. Electron [WWW]
    http://electron.atom.io/ (28.12.2015)
19. TransferWise - Wikipedia [WWW]

https://en.wikipedia.org/wiki/TransferWise (28.12.2015)
20. Hawala - Wikipedia [WWW]
https://en.wikipedia.org/wiki/Hawala (28.12.2015)
21. Clearing (finance) - Wikipedia [WWW]
https://en.wikipedia.org/wiki/Clearing_(finance) (28.12.2015)
22. A first look inside TransferWise engineering – TransferWise Blog [WWW]
https://transferwise.com/blog/2014-06/a-first-look-inside-transferwise-engineering/ (28.12.2015)
23. Cross-border payments, price list – AS SEB Bank [WWW]
http://www.seb.ee/eng/everyday-banking/accounts-and-transactions/cross-border-payments#price-list (28.12.2015)
24. Условия ведения расчетов - AS SEB Bank [PDF]
http://www.seb.ee/files/tingimused/arveldustingimused_rus_01022014.pdf (28.12.2015)
25. Grails framework [WWW]
https://grails.org/ (28.12.2015)
26. Web Summit [WWW]
https://websummit.net/ (28.12.2015)
27. Web Summit photo sessions - Splento [WWW]
http://www.splento.com/websummit2015/list (28.12.2015)
28. Hinnakiri - MXM foto ja filmistuudio [WWW]
http://stuudio.mxm.ee/hinnad/ (28.12.2015)
29. Get Started – Braintree Developer Documentation [WWW]
https://developers.braintreepayments.com/start/overview (28.12.2015)
30. Amazon S3 Pricing - AWS [WWW]
https://aws.amazon.com/s3/pricing/ (28.12.2015)
31. Cron - ArchWiki [WWW]
https://wiki.archlinux.org/index.php/Cron (28.12.2015)
32. Архитектура высоких нагрузок - [Хд] [WWW]
http://goo.gl/xU4HeK (28.12.2015)
33. 5 Golden Rules for Great Web API Design - Toptal [WWW]
http://www.toptal.com/api-developers/5-golden-rules-for-designing-a-great-web-api (28.12.2015)
34. 10 Best Programming Languages of 2015 You Should Know - Devsaran [WWW]
https://www.devsaran.com/blog/10-best-programming-languages-2015-you-should-know (28.12.2015)
35. Что такого прекрасного в Laravel? - Laravel по-русски [WWW]
https://laravel.ru/posts/177 (28.12.2015)
36. List of HTTP status codes – Wikipedia [WWW]
https://en.wikipedia.org/wiki/List_of_HTTP_status_codes (28.12.2015)
37. Slim a micro-framework [WWW]
http://www.slimframework.com/ (28.12.2015)
38. How to choose the right Linux distro – InfoWorld [WWW]
http://www.infoworld.com/article/2687088/linux/how-to-choose-a-linux-server-distribution.html?page=2 (28.12.2015)
39. Five Reasons Linux Beats Windows for Servers – PCWorld [WWW]
http://www.pcworld.com/article/204423/why_linux_beats_windows_for_servers.html (28.12.2015)

40. Estonia's technology cluster: Not only Skype – The Economist [WWW]
http://www.economist.com/blogs/schumpeter/2013/07/estonias-technology-cluster (28.12.2015)
41. Stacks – Stackshare [WWW]
http://stackshare.io/stacks (28.12.2015)
42. Email conversation with Slack [PDF]
https://dl.dropboxusercontent.com/u/2581808/email-Slack.pdf (28.12.2015)
43. Email conversation with TransferWise [PDF]
https://dl.dropboxusercontent.com/u/2581808/email-Transferwise.pdf (28.12.2015)
44. Email conversation with Uber [PDF]
https://dl.dropboxusercontent.com/u/2581808/email-Uber.pdf (28.12.2015)
45. Splento [WWW]
http://splento.com (28.12.2015)