Представление прототипа сервиса: реализация хранения корпусов документов

Выбор программной платформы

Для реализации веб-сервисов и их компонентов были рассмотрены существующие программные платформы. Были выделены следующие характеристики для сравнения платформ:

1. Поддерживаемые языки программирования.
2. Среда выполнения.
3. Доступные объектно-реляционные отображения.
4. Механизмы подключение к реляционным базам данных.
5. Каркас веб-приложений.
6. Реализация освобождения памяти.
7. Внутренние компоненты.
8. Клиентские компоненты.
9. Поддержка XML.
10. Служба сообщений для межпроцессорной коммуникации.
11. Серверы для развертывания веб-приложений.
12. Веб-сервисы.
13. Пользовательский интерфейс.
14. Реализация Remoting.
15. Модульное тестирование.
16. Асинхронные вызовы.

Платформа «.NET Framework»

Программная платформа «.NET Framework» включает в себя общеязыковую среду выполнения и библиотеки классов. Среда выполнения является агентом, который управляет кодом во время исполнения и предоставляет основные службы. Библиотека классов является комплексной объектно-ориентированной коллекцией, которая допускает повторное использование типов.

Программы, написанные в среде «.NET Framework», компилируются в два этапа. На первом этапе исходный код компилируется во время построения проекта и вместо исполняемого файла с машинными кодами получается сборка, которая содержит команды промежуточного языка. Второй этап компиляции наступает непосредственно перед фактическим выполнением. На этом этапе общеязыковая среда транслирует промежуточный код в низкоуровневый собственный машинный код, выполняемый процессором.

* + 1. Платформа «Java»

Программная платформа «Java» позволяют разрабатывать и запускать программы, написанные на языке программирования Java. Данная программная платформа не является специфической для какого-либо определенного процессора или операционной системы. Однако компилятор с набором библиотек и механизм выполнения реализованы для различного аппаратного обеспечения и различных операционных систем, чтобы программы, написанные языке Java, могли работать везде одинаково.

* + 1. Итоги сравнения программных платформ

Итоги сравнения подведены в таблице 4.1. Благодаря полученным сравнениям решено использовать платформу «.NET Framework», т.к. данная платформа имеет большее количество преимуществ.

**Таблица 4.1. Программные платформы**

| **Характеристика** | **.NET Framework** | **Java** |
| --- | --- | --- |
| Поддерживаемые языки программирования | C#, VB.NET, C++.NET, Scala, F# и другие. | Java, Scala, Groovy, и другие. |
| Среда выполнения | CLR | JVM |
| Доступные объектно-реляционные отображения | Entity Framework, NHibernate | Hibernate  |
| Механизмы подключение к реляционным базам данных | ADO.NET | JDBC |
| Каркас веб-приложений | ASP.NET MVC, Spring.NET | Spring |
| Реализация освобождения памяти | Автоматическая | Автоматическая |
| Внутренние компоненты | .NET, COM | EJBs |
| Клиентские компоненты | .NET классы | Компоненты Java (JavaBeans) |
| Поддержка XML | System.XML | JAXP |
| Служба сообщений для межпроцессорной коммуникации | MSMQ | JMS |
| Серверы для развертывания веб-приложений | IIS | От различных поставщиков |
| Веб-сервисы | WCF/Asmx, ASP.NET | JAX-WS/Axis, JSF |
| Пользовательский интерфейс  | Windows Forms, WPF | AWT/Swing |
| Remoting | SOAP, HTTP, DCOM | RMI |
| Модульное тестирование | Microsoft Unit Testing Framework, NUnit | JUnit |
| Асинхронные вызовы  | COM+QC | EJB Message Beans |

* 1. Выбор облачной платформы

Для развертывания облачного хранилища и веб-сервисов были рассмотрены существующие облачные платформы. Были выделены следующие характеристик для сравнения платформ:

1. Вычислительные мощности виртуальных машин:
	1. Память хранилища (ГБ).
	2. Память оперативная (ГБ).
	3. Процессоры (количество).
	4. Операционные системы.
2. Хранилище данных:
	1. BLOB хранилище.
	2. Табличное хранилище.
	3. Реляционное хранилище.
3. Разработка решений:
	1. Портал разработчика.
	2. Eclipse Tools.
	3. Visual Studio Tools.
	4. Java SDK.
	5. Мобильное SDK.
	6. PHP SDK.
	7. Python SDK.
	8. Ruby SDK.
	9. .NET SDK.
4. Мониторинг.
5. Ценообразование.
6. Пробное использование.
	* 1. Платформа «Amazon EC2»

Платформа «Amazon EC2» – это облачная платформа, которая предоставляет масштабируемые вычислительные ресурсы в облаке. Простой интерфейс сервиса «Amazon EC2» позволяет получить доступ к вычислительным ресурсам и настроить их с минимальными трудозатратами. Платформа предоставляет пользователям абсолютный контроль над ресурсами, которые они могут запускать на платформе. Сокращая до нескольких минут процесс настройки и запуска новых экземпляров серверов, сервис «Amazon EC2» позволяет быстро масштабировать вычислительные ресурсы с учетом изменяющихся запросов. «Amazon EC2» меняет экономическую составляющую процесса вычислений, предоставляя возможность платить только за употребляемые ресурсы.

* + 1. Платформа «Google App Engine»

Платформа «Google App Engine»– это сервис [хостинга](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B3) сайтов и [web-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) на серверах «[Google](https://ru.wikipedia.org/wiki/Google%22%20%5Co%20%22Google)». Приложения, разворачиваемые на базе «Google App Engine», должны быть написаны на программных языках: «[Python](https://ru.wikipedia.org/wiki/Python)», «[Java](https://ru.wikipedia.org/wiki/Java)», «[Go](https://ru.wikipedia.org/wiki/Go)» или «[PHP](https://ru.wikipedia.org/wiki/PHP)». Среда исполнения «[Python](https://ru.wikipedia.org/wiki/Python)» включает в себя полную реализацию [возможностей](https://ru.wikipedia.org/wiki/WSGI) самого языка «[Python](https://ru.wikipedia.org/wiki/Python)», большинство функций [стандартной библиотеки языка](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python), ограниченную версию «Django», и т.д.

* + 1. Платформа «Microsoft Azure»

Платформа «Microsoft Azure» разработана для обработки самых различных рабочих нагрузок, от небольших проектов по разработке и тестированию до запуска глобальных продуктов. В основе работы платформы «Microsoft Azure» лежит запуск виртуальной машины для каждого экземпляра приложения. Разработчик определяет необходимый объём для хранения данных и требуемые вычислительные мощности (количество виртуальных машин), после чего платформа предоставляет соответствующие ресурсы.

* + 1. Итоги сравнения облачных платформ

Итоги сравнения подведены в таблице 1. Благодаря полученным сравнениям решено использовать платформу «Microsoft Azure».

**Таблица 1. Облачные платформы**

| **Характеристика** | **Amazon EC2** | **Google App Engine** | **Microsoft Azure** |
| --- | --- | --- | --- |
| **Вычислительные мощности виртуальных машин** |
| Память хранилища (ГБ) | 4 до 48 000. | Нет ограничений. | 4 до 64 000. |
| Память оперативная (ГБ) | 1 - 244. | 3,75 - 208. | 0,768 - 448. |
| Процессоры (количество) | 1 - 40. | 1 - 32. | 1 - 32. |
| Операционные системы | Linux(CentOS, Debian, SUSE и т.д.), Windows Server. | Linux(CentOS, Debian, SUSE и т.д.) Windows Server. | Linux (CoreOS, CentOS, SUSE, ORACLE), Windows Server. |
| **Хранилище данных** |
| Реляционное хранилище. | Максимальный размер 64 ТБ. | Максимальный размер 500 ГБ. | Максимальный размер 500 ГБ. |
| BLOB. | Не могут превышать 5 Тб. Ограничений на общий объем данных и количество объектов нет. | Максимальный размер объекта составляет 2 ГБ, но каждый вызов API может обрабатывать только максимум 1 МБ. | Блочный BLOB не превышает 200 ГБ. Страничный BLOB не превышает 1 ТБ. Одна учетная запись может содержать до 500 ТБ. |
| Табличное хранилище. | Таблицы организуются в “домены”, размер которых не может превышать 10 Гб. На количество “доменов“ ограничений не существует. | Для сохранения одной сущности выделяется до 1 ГБ памяти, количество операций чтения и записи являются неограниченными. | Запись может иметь до 255 полей и не может превышать 1 Мб размере. Размер объектов в одном аккаунте не может превышать 500 Тб.  |
| **Разработка решений** |
| Портал разработчика | Да. | Да. | Да. |
| Eclipse Tools | Да. | Да. | Да. |
| Visual Studio Tools | Да. | Да, но только Python. | Да. |
| Java SDK | Да. | Да. | Да. |
| Мобильное SDK | iOS, Android, Fire OS. | iOS, Android. | iOS, Android, Windows Phone. |
| PHP SDK | Да. | Да. | Да. |
| Python SDK | Да. | Да. | Да. |
| Ruby SDK | Да. | Нет. | Да. |
| .NET SDK | Да. | Нет. | Да. |
| **Мониторинг состояния сервисов** | Amazon CloudWatch. | Диагностическое API. | Диагностическое API. |
| **Ценообразование** | Плата производиться по факту использования. | Плата производиться по факту использования. | Плата производиться по факту использования. |
| **Пробное использование** | 12 месяцев, 750 часов/месяц на виртуальную машину, BLOB хранилище на 5 Гб, 20000 запросов на получение и 2000 на отправку. База данных NoSQL размером 25 Гб и 200 миллионов запросов в месяц. | Дается 300 долларов на любые службы Google app engine в течение 60 дней. | Дается 10 000 рублей на любых служб Azure в течение 30 дней. По программе «BizSpark» можно получить 750 $ на 5 разработчиков в течение 3 лет. А также необходимо программное обеспечение от компании «Microsoft». |

При сравнении существующих программных платформ была выбрана платформа «.NET Framework». Данная платформа предоставляет большое количество преимуществ при реализации веб-сервисов. При анализе существующих облачных платформ выявлено, что на данный момент облачная платформа «Microsoft Azure» предоставляет разработчикам больше возможностей при разработке облачных решений, бесплатное программное обеспечение фирмы «Microsoft».

Реализация архитектуры приложения

Для реализации данного хранилища и работы с ним выбрана «Луковая» архитектура (Onion architecture) (см. Рисунок 1). Диаграмма компонентов архитектуры приведена в приложении В.



**Рисунок 1. Архитектура приложения**

Данная архитектура содержит следующие элементы:

1. Модель предметной области, которая включает в себя классы, перечисления и отношения между классами. Модель предметной области задает общее видение структуры хранилища.
2. Интерфейсы для передачи данных между моделью предметной области и базой данных – это прослойка, которая необходима для того, чтобы задать сигнатуру методов, которые необходимо реализовать при работе с базой данных.
3. Фабрика репозиториев, данная фабрика вызывает репозиторий, в котором хранятся данные, а также конкретную структуру базы данных. Т.е. если вы изначально использовали базу данных, которая реализована на платформе «Oracle», а потом реализовали базу данных на платформе «MS SQL Server», то вам необходимо будет только переписать методы, которые работают с базой данных, но не их сигнатуру.
4. Класс для работы с BLOB хранилищем, данный класс используется для работы с BLOB хранилищем, которое реализован специально для данной модели предметной области.
5. Веб-приложение, которое содержит веб-сервисы. Веб-приложение используется как прослойка между программистами, которые хотят использовать данное облачное хранилище.
6. Репозитории для работы с базой данных, данная часть отвечает за то, как реализованы интерфейсы, т.е. как в реальности происходит работа с базой данных.
7. Объектно-реляционное отображениебазы данных, данная модель строиться на основе реляционной базы данных. Данный компонент позволяет устранить необходимость в написания большей части кода для доступа к данным базы данных.
8. Хранилище больших двоичных объектов, которое расположено на облачной платформе «Microsoft Azure».
9. Реляционная база данных, которая расположена на облачной платформе «Microsoft Azure».
10. Фабрика юнитов, которая вызывает конкретную модель базы данных.
11. Клиенты, которые работаю с веб-приложением.
	1. Реализация объектно-ориентированной модели

На основе выявленных требований к структуре хранилища реализована следующая модель классов (см. Приложение Г). Как мы видим, класс «User» имеет ссылки на класс: «TextCorpus», «TextCorpusFile», «Annotation» и «Template» с определенными свойствами: «AccessType» и «UserType». Тип данных «AccessType» и «UserType» являются перечислениями. Перечисление «AccessType» может принимать значения: «Read», «Write», «ReadWrite». Перечисление «UserType» может принимать значения: «Guest» и «Owner».

Основываясь на работах С. Амер-Яхиа, М. Фернандез и других для создания динамически дополняемых атрибутов использовалась структура дерева, которое ссылается само на себя, на 2 представлена структура дерева, а на рисунке 5.3 пример развернутого дерева.

 Каждый элемент «TreeElement» содержит:

1. Поле «Child», которое хранит ссылку на вложенный элемент дерева.
2. Поле «Next», которое хранит ссылку на следующий элемент дерева.
3. Поле «Value» хранит значение элемента дерева.
4. Поле «Attribute», которое ссылается на класс «AttributeName».



**Рисунок 2. Структура дерева**

Рисунок 3. **Пример развернутого дерева**

«AttributeName» хранит поле «Name» и может быть использован несколько раз при формировании элементов дерева или при формировании других деревьев. Данная структура использована для динамического добавления данных в «TextCorpus», «TextCorpusFile» и «Annotation».

Класс «TextCorpus» может содержать в себе несколько ссылок на классы «TextCorpusFile», но и каждый экземпляр класса «TextCorpusFile» может находиться в нескольких классах «TextCorpus», поэтому между ними стоит связь «ассоциация». Каждый «TextCorpusFile» может иметь несколько «Annotation», однако ни один класс «Annotation» не может быть построена без использования класса «TextCorpusFile», поэтому между ними стоит связь «композиция». Каждый класс «Annotation» должен быть построен с использованием нескольких классов «Template» или ни одного, тогда мы получим не маркированный текст, также мы можем добавлять и удалять из класса «Annotation» ссылки на класс «Template».

Класс «User», «TextCorpus», «TextCorpusFile», «Annotation» и «Template» наследуются от одного общего класса «EntityModel», который имеет только поле «Id», это сделано для того, чтобы обеспечить взаимодействие с базой данных.

Каждый класс, который реализован в модели предметной области имеет поля, конструкторы и свойства. Все поля заданные в классе являются приватными и доступны только внутри класса. Как мы помним, конструкторы не поддаются сериализации, поэтому все свойства являются открытыми, т.к. они будут использоваться на клиентской стороне. Конструкторы, определенные в классе, необходимы только на сервере, поскольку только с их помощью создаются сами экземпляры классов. Внутри конструкторов находятся проверки, которые будут генерировать исключения, если клиент передал данные, которые нельзя использовать при создании того или иного экземпляра класса.

Реализация структуры базы данных

На основе реализованной модели классов была построена реляционная база данных (см. Приложение Д). Все таблицы, кроме связующих таблиц («TextCorpus\_User», «TextCorpusFile\_User», «Annotation\_User», «Template\_User», «Annotation\_Template» и «TextCorpus\_File») имеют уникальный первичный ключ «Id». В реляционной базе данных отсутствуют связи «многие ко многим» и для их моделирования пришлось реализовывать связующие таблицы. Некоторые из связующих таблиц, такие как «TextCorpus\_User», «TextCorpusFile\_User», «Annotation\_User», «Template\_User» имеют дополнительные поля: «AccessType» и «UserType»

В базе данных было решено, не создавать отдельных таблиц для перечислений «AccessType» и «UserType», которые были созданы в диаграмме классов, поэтому они хранятся как строки.

Для создания возможности динамически дополнять атрибуты использовалась структура дерева, которое ссылается само на себя, на рисунке 5.4 представлена структура дерева в базе данных, а на рисунке 5.5 пример развернутого дерева в базе данных.

 Каждый элемент «TreeElement» содержит:

1. Поле «Id», которое хранит уникальный номер идентификатора.
2. Поле «ChildId», которое хранит номер на вложенный элемент дерева.
3. Поле «NextId», которое хранит номер на следующий элемент дерева.
4. Поле «Value», которое хранит значение элемента дерева.
5. Поле «AttributeId», которое ссылается на столбец «Id» таблицы «AttributeName».



**Рисунок 3. Структура дерева в базе данных**

 **Рисунок 4. Пример развернутого дерева в базе данных**

«AttributeName» хранит поле «Name» и «Id», которое является уникальным идентификатором и может быть использован несколько раз при формировании элементов дерева или при формировании других деревьев. Данная структура была использована для динамического добавления данных в таблицах «TextCorpus», «TextCorpusFile» и «Annotation».

Реализация структуры BLOB хранилища

При выявлении требований к структуре хранилища было выявлено, что необходимо хранить массив байтов, а для этих целей прекрасно подходит BLOB хранилище. Структура реализованного хранилища представлена на рисунке **5.**.



**. Рисунок 5. Структура BLOB хранилища**

Учетная запись должна хранить три контейнера «files», «plainfiles» «annotations». Контейнер «files» хранит текстовые файлы, где имя текстового файла формируется из уникального номера, который сохранен в базе данных, расширение также берется из базы данных. Контейнер «plainfiles» хранит текст без форматирования, где имя текста без форматирования формируется из уникального номера, который сохранен в базе данных, расширение также берется из базы данных. Контейнер «annotations» хранит аннотации, где имя аннотации формируется из уникального номера, который сохранен в базе данных, расширение также берется из базы данных.

Размещение базы данных, BLOB хранилища и веб-приложения
на облачной платформе

После того как веб-сервисы были реализованы согласно диаграммам представленным в третьей главе, на облачной платформе «Microsoft Azure» были размещены: экземпляр сервера «MS SQL Server», экземпляр BLOB хранилища и веб-приложение, которое содержит все веб-сервисы, для работы с базой данных и BLOB хранилищем.

Экземпляр сервера «MS SQL Server» был создан на сайте платформы «Microsoft Azure». С использованием программного обеспечения «Microsoft SQL Server Management Studio» выполнено подключение к экземпляру сервера, который расположен на облачной платформе и написан запрос, который создает базу данных (см.7).



1. **Выполнение запроса**

Экземпляр BLOB хранилища был создан на сайте платформы «Microsoft Azure», все необходимые контейнеры были созданы также на данном сайте (см. Рисунок .9).



**Рисунок 9. Создание контейнеров**

Для программной реализации была выбрана «Луковая» архитектура, которая позволяет не зависеть от определенной реализации базы данных, и разрешает заменить базу данных на другую, если в этом возникнет необходимость, а также создана объектно-ориентированная модель, которая является ядром данной архитектуры. На основе созданной объектно-ориентированной модели была реализована реляционная база данных и BLOB хранилище. Далее реляционная база данных, BLOB хранилище и веб-приложение были размещены на облачной платформе «Microsoft Azure». Было высчитано количество строк кода реализованного решения для того чтобы иметь общие представления о проделанной работе.

Литература

1. NET Framework // Программная платформа .NET Framework. 2016. URL: https://msdn.microsoft.com/ru-ru/zw4w595w(v=vs.110).aspx (дата обращения: 20.05.2016).
2. Java // Программная платформа Java. 2016. URL: http://www.oracle.com (дата обращения: 20.05.2016).
3. Amazon EC2 // Облачная платформа Amazon EC2. 2016. URL: https://aws.amazon.com/ec2 (дата обращения: 20.05.2016).
4. Google Cloud // Облачная платформа Google App Engine. 2016. URL: https://cloud.google.com/appengine (дата обращения: 20.05.2016).
5. Microsoft Azure // Облачная платформа Microsoft Azure. 2016. URL: https://azure.microsoft.com (дата обращения: 20.05.2016).
6. Jeffrey P., [Bogard](http://it-ebooks.info/author/707/) J., [Hexter](http://it-ebooks.info/author/486/%22%20%5Co%20%22Eric%20Hexter) E. ASP.NET MVC in Action. NY: Manning, 2012. P. 253-256.
7. Azure Blob Storage // BLOB хранилище облачной платформы Microsoft Azure. 2016. URL: https://azure.microsoft.com/ru-ru/documentation/articles/storage-dotnet-how-to-use-blobs/ (дата обращения: 20.05.2016).