Санкт-Петербургский Государственный Университет

Кафедра технологии программирования

Бабаева Ирина Витальевна

Выпускная квалификационная работа бакалавра

Разработка каталога рабочих программ учебных дисциплин СПбГУ

Направление 010300

Фундаментальная информатика и информационные технологии

Научный руководитель,

ст. преп.

Севрюков С.Ю.

Санкт-Петербург

2016

Содержание

Введение

Современные Университеты представляют собой совокупность различных учебно-административных структурных подразделений, каждое из которых является самостоятельной единицей, но при этом тесно связанных между собой едиными целями и задачами обеспечения обучающего процесса и научной деятельности.

Повысить эффективность работы такого мощного комплекса и оптимизировать рабочие процессы удается за счет использования информационных технологий, которые в последнее время быстро развиваются.

Применение в рабочих процессах информационных технологий позволяет оптимально решать производственные задачи, что приводит к минимизации времени на получение необходимой информации, уменьшению объема документооборота за счет чего повышается эффективность труда и происходит экономия человеческих и временных ресурсов.

Примером успешного применения информационных технологий на практике является использование единого интерфейса. В качестве такого интерфейса может выступать веб-приложение.

Веб-приложение - это некоторый ресурс, доступ к которому пользователь получает, используя свой веб-браузер и подключение к сети Интернет.

Веб-приложения общаются с пользователями с помощью описанных разработчиком средств, позволяющих взаимодействовать с большими объемами данных и получать результат, скрывая от пользователя особенности технической реализации.

В мировой практике существуют успешные примеры реализации единого интерфейса для доступа к документам в рамках всего Университета. Примеры таких каталогов реализованы у “Lansing Community College”, “University of Wisconsin-Madison”, “University of Michigan”, “College of

Literature, Science, and the Arts", "University of Copenhagen". Однако функции поиска в этих проектах реализованы недостаточно удобно для конечного пользователя. Одним из удачных аспектов, который все чаще встречается в контексте поисковых интерфейсов, является фасетная классификация. В упомянутых выше проектах функции фасетной классификации либо реализованы частично, либо не реализованы совсем.

В настоящее время в СПбГУ в рамках некоторых факультетов реализованы интерфейсы, обеспечивающие доступ к каталогу документов, например, Факультет свободных искусств и наук имеет приложение с некоторым набором простейших функций поиска по каталогу рабочих программ учебных дисциплин (далее РПУД). Но в рамках всего Университета не существует единого веб-приложения, предоставляющего доступ к РПУД всех факультетов.

Изначально каждый элемент каталога РПУД содержит избыточное количество информации для отдельного класса пользователей. К таким классам пользователей можно отнести: студентов, преподавателей, будущих работодателей, и т.д. Например, студента интересуют темы, которые будут изучены в рамках курса, критерии оценивания по окончании курса, используемая литература; тогда как преподавателя и административный персонал, скорее всего, будут интересовать вопросы, которые касаются материально-технического обеспечения аудиторий и аудиторного оборудования, информация о количестве академических часов, отведенных на изучение курса, требования к ученой степени преподавателя и т.д.

Каждый пользователь хочет получать интересующие его данные без лишней информации - для этой цели в веб-приложениях используются функции поиска, которые фильтруют данные на основе запроса пользователя и возвращают ожидаемый человеком результат.

Кроме желания пользователя получать объем информации, не превышающий его ожидания, следует учитывать и форму выдачи результата - т. е. продумать интерфейс, который мог бы помочь пользователю быстрее

оценить и понять, насколько точно, исходя из его запроса, выдан результат. В рамках разработки веб-приложения подобного результата можно достичь, например, посредством расстановки столбцов таблицы, в которой представлен результат, на основе некоторых приоритетов. К примеру, для студента приоритеты могли бы выглядеть следующим образом: критерии оценивания по окончании курса, предлагаемые к изучению темы, примерный список вопросов к экзамену. Остальные аспекты, освещенные в документе РПУД, могут быть либо интересны малому количеству студентов (материально-техническое обеспечение аудиторий), либо не интересны совсем (разделы, содержащие административную информацию, как-то: разработчики РПУД, оценка качества РПУД и т.д.). Подобные аспекты, которые заведомо не интересны определенной группе пользователей, можно не выводить в качестве результата. Аспекты, которые могут быть интересны, но с небольшой вероятностью, можно скрывать, и ждать от пользователя действия, означающего его желание ознакомиться с дополнительными результатами поиска.

Для СПбГУ создаётся каталог дисциплин, доступ к которому будет осуществляться со стороны всех желающих (учащиеся, их родители и наставники, абитуриенты, работодатели и т.п.). Одна из ключевых возможностей - поиск. Поиск по названию, автору, параметрам (например, по продолжительности, или предметной области) и по полному тексту аннотаций.

Для пользователя поисковые функции в веб-приложениях обычно представляют собой некоторый графический интерфейс, с которым удобно работать. На основе введенных пользователем критериев данные фильтруются, и пользователь получает результат.

Когда критериев поиска много, для повышения наглядности выполнения поисковой функции можно использовать концепцию фасетного поиска. Суть фасетного поиска заключается в отсечении подмножества заведомо недопустимых элементов и изменении списка критериев,

доступных для выбора с целью осуществления дальнейшей фильтрации данных. Благодаря этому пользователь может видеть, как его единичный выбор влияет на дальнейший результат поиска.

Помимо фасетной классификации для осуществления поиска по отдельным критериям, для поиска по полному тексту аннотаций применяется полнотекстовый поиск. За счет реализации полнотекстового поиска в веб-приложении, пользователь получает возможность искать информацию по ключевым словам.

В данной работе речь пойдет о разработке прототипа веб-приложения для доступа к каталогу дисциплин в рамках всего СПбГУ с предоставлением удобного поискового механизма, включающего в себя фасетную классификацию и полнотекстовый поиск.

Постановка задачи

Цель данной работы: с помощью введения удобного пользовательского интерфейса сократить временные затраты и упростить поиск необходимой информации среди РПУД.

Для достижения поставленной цели необходимо решить задачу разработки прототипа веб-приложения, работающего с каталогом РПУД и предоставляющего гибкие средства поиска и навигации пользователям различных классов.

Для решения данной задачи необходимо:

1. провести анализ предметной области
2. определить объекты и связи, необходимые для хранения в базе данных
3. определить необходимые и достаточные критерии поиска, для предоставления конечного результата
4. определить критерии полноты информации, которую пользователь получает в ответ на свой запрос
5. выполнить тестирование приложения

Также существуют некоторые начальные требования, сформулированные сотрудниками СПбГУ, которые могут использоваться для составления технического задания.

Каталог будет содержать порядка 12000 экземпляров с последующим ростом архивной части (возможно удвоение в случае потребности в ежегодном обновлении). Каталог должен быть публичным, доступ к каталогу будет осуществляться с помощью любого браузера последних версий под управлением как настольных, так и мобильных платформ.

Каталог будет создаваться на русском и английском языках, поэтому поиск необходимо реализовать на двух языках.

Программы дисциплин будут размещаться в реляционной БД.

Под РПУД понимается электронный документ, содержащий полнотекстовые материалы и набор метаданных. Метаданные

представляются в виде числовых значений, текстовых значений из словаря и значений типа дата.

В зависимости от класса пользователя и дополнительных критериев, выбранных пользователем, формируется возвращаемый набор актуальных данных.

Классы пользователей:

- сотрудники, отвечающие за разработку РПУД
- преподаватели
- студенты и абитуриенты
- потенциальные работодатели
- родители и наставники

Объектом разработки, как было определено ранее, является веб-приложение, имеющее доступ к хранилищу данных и предоставляющее удобный интерфейс для поиска информации пользователю в зависимости от его класса. К основным требованиям, характеризующим удобность интерфейса, относятся реализация фасетного поиска и возможность полнотекстового поиска по аннотации документа.

1. Непрерывная интеграция

Создание целевого прототипа выполнялось итеративно, т.е. на протяжении всего времени выполнения работы последовательно создавались версии приложения, имеющие ограниченное количество функций. Для оперативности предоставления конечного результата каждой итерации и автоматизации большинства действий по развёртыванию, тестированию и поставки конечным пользователям, использовались инструменты в рамках концепции непрерывной интеграции (Continuous Integration).

Термин Continuous Integration введен Мартином Фаулером (Martin Fowler) и Кентом Беком (Kent Beck). Данный термин используется для обозначения практики частой сборки (интеграции) проекта.

Использование концепции непрерывной интеграции в рамках выполнения данной работы позволяет:

- в любой момент иметь рабочую версию создаваемого приложения, доступного в сети Интернет;
- хранить историю создания/изменения приложения для того, чтобы иметь возможность в случае неудачной публикации новой версии приложения, без потерь вернуться к старой версии;
- производить частые модульные тесты для новых изменений.

Процесс непрерывной интеграции состоит из нескольких этапов.

Цикл интеграции инициируется изменением файлов в системе контроля версий. После того, как сделаны изменения, необходимо провести сборку для проверки того, что последние изменения согласуются с ранее сделанными и не приводят к неправильной работе приложения. После этого наступает следующий этап.

На следующем этапе CI сервер делает обновление своей локальной копии исходного кода проекта. В процессе обновления выясняются изменения в коде (и не только) произошедшие с последней интеграции. Отслеживание изменений необходимо для того, чтобы в каждый момент

времени знать, какой из вариантов исходного кода приводит к сбою, а какой - не влияет на состояние системы.

Далее следует обязательный этап процесса интеграции - сборка проекта. Исходный код компилируется и собирается в исполняемые файлы. Данный этап позволяет разработчикам отслеживать выполнение приложения независимо от архитектуры машины, на которой собирается проект.

После сборки проекта, когда разработчик уверен в некоторой работоспособности проекта - проект необходимо «развернуть». Веб-приложение выкладывается на веб-сервер и запускается.

После того, как разработчик убедится в работоспособности кода, код необходимо сохранить. Для этого можно использовать метки в системе контроля версий. Так же необходимо сохранить бинарные файлы проекта. Они могут понадобиться, если нужно будет воспроизвести ошибку в конкретной версии и для ручного тестирования [1].

Подобный подход поможет выявлять ошибки в ходе построения/выполнения приложения и, в случае неудачи, возвращаться к удачному решению.

Еще одним преимуществом CI можно считать уверенность в том, что приложение будет успешно развернуто в конце процесса разработки.

Для организации непрерывной интеграции необходимо выбрать систему контроля версий; среду, в которой приложение будет опубликовано, и средства для настройки автоматической сборки и публикации веб-приложения.

1.1 Выбор средств для реализации CI: VCS

В качестве основного средства для управления проектом, кодовой базой и автоматической сборкой был выбран такой сервис, как Visual Studio Online. Данный сервис создан и предлагается для использования компанией Microsoft и базируется на более ранней разработке этой компании – Team Foundation Server (далее TFS).

На ряду с TFS, возможности взаимодействия с кодовой базой предоставляют GitHub и Bitbucket. Также, GitHub и Bitbucket позволяют разработчикам отслеживать изменения в репозиториях, общаться между собой, комментировать и редактировать исходный код, но ни GitHub, ни Bitbucket не позволяют создавать сборки и управлять ими. За счет чего, предпочтение в данной работе отдается TFS.

Пользуясь управлением сборками, становится доступной такая функция как определение шагов сборки: можно выбирать в каком порядке и какие шаги будут выполняться. Для организации непрерывной интеграции разработчик должен запускать соответствующие скрипты в определенном порядке - с использованием TFS подобная задача решается средствами пользовательского интерфейса в Visual Studio Online. Также после завершения каждой сборки, TFS предоставляет отчет, который может содержать ошибки сборки или сообщения об удачном завершении работы. Такие отчеты хранятся в TFS и доступны для просмотра в любой момент времени, что позволяет отслеживать прогресс процесса разработки, публиковать только рабочие версии проекта и исправлять ошибки приложения до его публикации.

Данный инструмент был освоен с помощью таких ресурсов как курсы ИНТУИТ http://www.intuit.ru/studies/courses/582/438/info и официальная документация, доступная по ссылке: https://msdn.microsoft.com/en-us/library/ms181238(v=vs.90).aspx.

В рамках выполнения данной работы после этапа разработки проекта, при совершении очередного коммита в Visual Studio, исходный код клонируется из локального репозитория в репозиторий в Visual Studio Team Services. В TFS настроен порядок сборки, который предполагает сначала настройку среды DNX и загрузку необходимых пакетов Nuget с помощью средств PowerShell, затем сборку проекта, далее публикацию приложения в Azure, также с помощью использования PowerShell.

В настоящий момент с помощью TFS выполнено около ста коммитов и примерно столько же сборок, которые завершились либо с положительным результатом, либо с ошибками, список которых доступен разработчику в интерфейсе VSO, которые впоследствии были исправлены.

Помимо работы с исходном кодом проекта и сборками, TFS предлагает гибкие средства для взаимодействия команды разработчиков и планирования их совместной работы, что приводит к использованию гибкой методологии разработки - Agile. Суть применения Agile заключается в итеративной разработке продукта в связи с динамическим формированием требований и их выполнением в результате взаимодействия членов команды разработки. Для решения поставленной задачи подобные средства были использованы для общения с научным руководителем в виде фиксации подзадач, которые должны быть выполнены, и отслеживания прогресса их выполнения. Во время работы с подзадачами разработчики могут не только создавать их или удалять, но и комментировать, обсуждать, расставлять приоритеты, назначать ответственного за отдельный блок подзадач, устанавливать временные ограничения на решение того или иного вопроса, отмечать статус выполнения задания (не начато/в процессе/закончено).

В рамках решения поставленной выше задачи, TFS, помимо средств для разработки, тестирования и публикации веб-приложения, дает разработчику возможность планирования, отслеживания статуса и приоритета подзадач, общения с научным руководителем с помощью создания и комментирования заданий в VSO.

1.2 Выбор средств для реализации CI: Microsoft Azure

Выбирая облачную платформу в качестве сервера для публикации веб-приложения, разработчик получает возможность доступа к приложению из сети Интернет и снижает временные и ресурсные затраты на установку и настройку сервера в рабочей среде.

Для выбора наиболее подходящей облачной платформы были рассмотрены такие популярные сервисы как: Amazon Web Services, Microsoft Azure и DigitalOcean. Все выбранные платформы поддерживают создание, поддержку и масштабирование веб-приложений, развертывание веб-приложений в облаке, доступ по протоколу HTTPS и многофакторную авторизацию для обеспечения безопасности соединения.

Также между приведенными выше тремя платформами есть существенные отличия. Приложения ASP .NET публикуются только на Windows-хостингах с поддержкой IIS-сервера. DigitalOcean поддерживает CoreOS - облегченную версию операционной системы Linux, за счет чего использование сервера IIS не представляется возможным, как и публикация ASP .NET приложений.

Amazon Web Services и Microsoft Azure предоставляют Windows-хостинг и бесплатный период использования, но при схожих функциональных возможностях, в долгосрочной перспективе предпочтение отдается Windows Azure за счет наличия бесплатного тарифа: в случае Amazon Web Services минимальная цена на обслуживание хостинга по истечении бесплатного периода составляет 10$.

Microsoft Azure - облачная платформа, предоставляющая возможность хранить данные, разрабатывать и запускать веб-приложения на удаленных серверах (в данном случае - на серверах компании Microsoft).

Microsoft Azure предоставляет веб-сервер для публикации веб-приложений - IIS, особенностью которого является поддержка хостинга приложений ASP .NET. Также, наряду с IIS-сервером, Microsoft Azure содержит средства для хранения и работы с базами данных, настройки приложений, позволяющие выбрать версию .NET Framework-а и источник развертывания, функции для осуществления резервного копирования данных и проверки подлинности авторизации и многое другое.

Развертывание приложений в Windows Azure можно осуществлять двумя способами: с портала Windows Azure и с помощью Service Management

API (SMAPI). SMAPI предоставляет веб-сервисы с помощью протокола Representational State Transfer (REST) и предназначен для разработчиков. Протокол работает поверх SSL.

Аутентификация SMAPI основана на создании пользователем пары публичного и приватного ключей и самоподписанного сертификата, который регистрируется на портале Windows Azure. Таким образом, все критичные действия по управлению приложениями защищены вашими собственными сертификатами. При этом сертификат не привязан к trusted root certificate authority (CA), вместо этого он является самоподписанным, что позволяет с определенной долей точности быть уверенным в том, что к защищенным таким образом сервисам и данным будет иметь доступ только определенные представители клиента.

Хранилище Windows Azure использует собственный механизм аутентификации на основе двух ключей Storage Account Key (SAK), которые ассоциированы с каждым аккаунтом и могут быть сброшены пользователем.

Таким образом, в Windows Azure реализована комплексная защита и аутентификация. [2].

Благодаря выбору такой платформы как Microsoft Azure, разработчик получает возможность хранить все данные и файлы на сторонних серверах, разворачивать приложения в сети Интернет и управлять ими, используя графический интерфейс портала Microsoft Azure.

1.3 Выбор средств для реализации CI: PowerShell

PowerShell является платформой автоматизации с языком сценариев для Windows и Windows Server, что позволяет упростить управление используемыми системами. В отличие от других текстовых оболочек, PowerShell использует преимущества .NET Framework, предоставляя множество объектов и большой набор встроенных функций для получения контроля над средой Windows [3].

Командлеты (англ. *cmdlets*) — это специализированные команды PowerShell, которые реализуют различную функциональность. Это встроенные в PowerShell команды. Командлеты именуются по правилу Глагол-Существительное, например, Get-ChildItem, благодаря чему их предназначение понятно из названия. Командлеты выводят результаты в виде объектов или их коллекций.

PowerShell включает язык сценариев с динамическими типами, на котором можно реализовывать сложные операции с использованием командлетов. Язык сценариев поддерживает переменные, функции, конструкции ветвления (if-then-else) циклы (while, do, for и foreach), структурированную обработку ошибок и множество других возможностей, включая интеграцию с .NET.

Используя средства PowerShell, в проекте могут быть настроены сценарии автоматической сборки и развёртывания веб-приложения в Azure. Описанные выше сценарии находятся в открытом доступе по ссылке: https://github.com/quakke/SyllabusPortal.

Таким образом, после очередного обновления кода в системе контроля версий происходит автоматическая сборка проекта и публикация веб-приложения в Azure.

1.4 Реализация Continuous Delivery

Непрерывное развертывание представляет собой подход, при котором осуществляется постоянная сборка и развертывание приложения в тестовой среде, за счет этого гарантируется, что приложение может быть запущено без ошибок в любое время. Подобный подход также полезен тем, что в любой момент времени есть возможность посмотреть работу веб-приложения в тестовой среде и провести ручное тестирование.

В контексте выполнения данной работы веб-приложение публикуется в Windows Azure.

Для непрерывного развертывания приложения в среде Windows Azure используются следующие средства:

- среда разработки Visual Studio 2015
- учетная запись для Microsoft Team Foundation Service - облачное приложение, которое содержит удобные инструменты для управления кодовой базой, сборками, отслеживания задач и ошибок, а также других действий
- скрипт, написанный с использованием технологии PowerShell, который запускает сборку и развертывание приложения после каждого обновления исходного кода в репозитории

Сначала создается проект в Visual Studio Team Services. Затем необходимо подключиться к Microsoft Team Foundation Service в Visual Studio, клонировать пока пустой репозиторий проекта, создать приложение ASP .NET 5 и выполнить коммит. Результатом будет публикация исходного кода приложения в созданный ранее репозиторий в Visual Studio Team Services. Далее настраивается порядок сборки проекта.

Для сборки проекта ASP .NET 5 необходимо предварительно настроить среду DNX и загрузить необходимые пакеты Nuget. Эти шаги можно описать в скрипте PowerShell и добавить его к решению в Visual Studio. В Visual Studio Team Services первым шагом необходимо назначить запуск скрипта PowerShell. Вторым шагом будет построение решения. Для этого необходимо указать ссылку на файл метаданных с расширением *.xproj.

Для публикации ASP .NET 5-проекта в Azure требуется создать пустой экземпляр сервиса Azure Web App, для непрерывной публикации приложения ASP .NET 5 в Azure необходим еще один скрипт PowerShell, который в Visual Studio Team Services будет назначен последним шагом. Стоит отметить, что в скрипте необходимо сначала остановить работу приложения в Azure, после этого обновить версию приложения, после чего снова запустить приложение. Если пропустить шаги остановки и запуска

приложения, приложение в Azure не обновится и в VSO появится соответствующая ошибка.

Вышеописанным способом можно настроить публикацию веб-приложения, использующего только пятую версию ASP .NET, т.к. при указании ссылки на проект в VSO ожидается файл с расширением *.xproj. В более ранних версиях ASP .NET файл метаданных имеет расширение *.csproj, при указании ссылки на который, возникает ошибка на этапе сборки приложения. Подобное ожидание файла *.xproj со стороны VSO обусловлено недавним обновлением версии Git в составе VSO.

Несмотря на видимую простоту и удобство настройки непрерывной интеграции для приложения пятой версии ASP .NET, пришлось отказаться от использования пятой версии в связи с возникающими проблемами на этапе разработки. Из-за того, что технология еще довольна новая, не все действия выполняются безошибочно и не все ошибки описаны на форумах и в различных учебных материалах. Так, на этапе разработки была обнаружена проблема с добавлением новых сборок в проект, вследствие чего пришлось перейти на более раннюю версию.

Для настройки сборки и публикации ASP .NET 4.5 необходимо настроить и выполнить в Visual Studio Team Services следующие шаги:

- установить пакеты NuGet: для каждой сборки отдельно указывается шаг установки пакетов. И, важным уточнением является то, что в параметре Installation type необходимо выбрать “Install” вместо значения по умолчанию “Restore” - в противном случае возникнет ошибка.
- настроить сборку приложения
- настроить шаг “Azure Deployment”, в котором необходимо указать параметры подписки и имя web app в Azure. Имя web app можно создать заранее, либо оно создастся автоматически.

После выполнения описанных выше действий и осуществления очередного коммита, веб-приложение будет автоматически собираться и публиковаться в Windows Azure.

На этапе выполнения настройки непрерывной сборки и интеграции проекта в Azure был сделан выбор версии ASP .NET в пользу 4.5, за счет отсутствия возникновения нерешаемых ошибок, возникающих в пятой версии.

1.5 Реализация Staging

Staging - концепция, при которой приложение сначала запускается в среде аналогичной рабочей, и, в случае удачной работы приложения в тестовой среде, выкладывается в рабочую.

Данная концепция может применяться для тестирования приложения в условиях, схожих или близких к реальным.

В данном случае тестовой средой будет Azure. В случае корректной работы веб-приложения в Azure, его можно будет размещать на серверах университета.

Данный шаг необходим для того, чтобы минимизировать вероятность возникновения ошибок работы приложения в рабочей среде и иметь возможность оценивать работу приложения, скрывая его от конечных пользователей.

2. Средства разработки

В качестве средств разработки выбраны Visual Studio 2015, .NET Framework, язык разработки - C#. Выбор обусловлен тем, что C# в связке с .NET Framework предоставляет удобный способ работы с данными, а Visual Studio 2015 позволяет работать с системой контроля версиями – TFS. Также для упрощения разработки и тестирования MVC приложений используется JavaScript-фреймворк AngularJS.

В качестве технологии для создания веб-приложения используется версия ASP .NET 4.5, обоснование этого выбора приведено выше. Однако в дальнейшем имеет смысл переход на версию ASP .NET 5 в связи с рядом успешных нововведений:

- Возможность видеть изменения в приложении без повторной сборки проекта.
- Хостинг в любом месте: ASP.NET 5 позволяет размещать свое приложение на IIS или в режиме self hosting. Приложение и его зависимости полностью автономны и не зависят от установки .NET в системе. Приложение может работать на любом типе устройства или хостинговой платформы.
- Повышение производительности HTTP: ASP.NET 5 вводит новый программный конвейер для HTTP-запросов, который является модульным и позволяет добавить только те компоненты, которые необходимы.
- Готовность к использованию в облаке: проект структурируется для легкого развертывания в облаке. Новая система позволяет запрашивать именованные значения из различных источников (например, JSON, XML или переменные среды).
- Весь код для ASP.NET 5 доступен на GitHub, что позволяет отслеживать новые изменения или предлагать свои. [4]

2.1 .NET Framework

.NET Framework — это платформа разработки для создания приложений для Windows, Windows Phone, Windows Server и Microsoft Azure. Она состоит из среды CLR и библиотеки классов .NET Framework, которая содержит классы, интерфейсы и типы значений, поддерживающие широкий диапазон технологий. Платформа .NET Framework предоставляет среду управляемого выполнения, возможности упрощения разработки и развертывания, а также возможности интеграции с различными языками программирования, включая Visual Basic и Visual C# [5].

На этапе создания проекта в среде разработки Visual Studio необходимо выбрать “Веб-приложение ASP .NET”, далее выбрать шаблон MVC и версию ASP .NET. В данном случае используется версия 4.5 на основании выводов, сделанных в разделе 1.4. Обоснование выбора шаблона MVC будет приведено ниже. В результате создания проекта подобным образом, создается шаблон приложения, содержащий базовый набор классов, обеспечивающий быструю разработку.

Также на этапе создания проекта можно выбрать версию .NET Framework. За счет обеспечения обратной совместимости, приложения, разработанные на основе конкретной версии платформы .NET Framework, могут выполняться без доработок и на более поздних версиях платформы, поэтому вопрос выбора версии остается без обоснования, но, следует отметить, что используемая версия в готовом решении - 4.5.1

В рамках выполнения данной работы .NET Framework позволяет использовать библиотеку ASP .NET и шаблон ASP .NET MVC для разработки веб-приложения, обеспечивает кроссплатформенность, позволяет абстрагироваться от задач выделения памяти для объектов и работы с ней.

2.2 ASP .NET MVC

Шаблон архитектуры Model-View-Controller (MVC) разделяет приложение на три основных компонента: модель, представление и

контроллер. Платформа ASP.NET MVC представляет собой альтернативу схеме веб-форм ASP.NET при создании веб-приложений. ASP.NET MVC является легковесной платформой отображения с широкими возможностями тестирования и, подобно приложениям на основе веб-форм, интегрирована с существующими функциями ASP.NET, например, с главными страницами и проверкой подлинности на основе членства. Платформа MVC определяется в сборке System.Web.Mvc [6].

В состав платформы MVC входят следующие компоненты:

- Модели. Объекты моделей являются частями приложения, реализующими логику для домена данных приложения. Объекты моделей часто получают и сохраняют состояние модели в базе данных.
- Представления. Представления служат для отображения пользовательского интерфейса приложения. Пользовательский интерфейс обычно создается на основе данных модели.
- Контроллеры. Контроллеры осуществляют взаимодействие с пользователем, работу с моделью, а также выбор представления, отображающего пользовательский интерфейс. В приложении MVC представления только отображают данные, а контроллер обрабатывает вводимые данные и отвечает на действия пользователя.

В настоящей работе в роли представления выступает файл index.cshtml. В нем, с помощью базовых средств HTML, CSS и мощного фреймворка Bootstrap, предназначенного для разработки адаптивных веб-приложений, создается пользовательский интерфейс, и, с помощью использования JavaScript-фреймворка AngularJS заполняется данными список чекбоксов и таблица результатов.

В качестве модели выступает класс RPUDRequestParameters.cs. В нем приводится описание сущности РПУД, которое состоит из следующих полей:

- NameInEng - название дициплины на английском языке
- Number - номер дисциплины
- Requirements - требования к преподавательской степени

- Assessment - итоговое испытание студента, как-то: зачет/экзамен/дифференцированный зачет.
- Resources - материально-технические трабования к аудиториям
- Description - описание дисциплины, состоящее из множества слов. По этому полю производится полнотекстовый поиск.

В рамках выполнения данной работы разрабатывается прототип приложения, а не готовое к внедрению решение. Количество полей, описывающих РПУД, к моменту внедрения данного решения будет расширено.

Роль контроллеров выполняют такие классы, как HomeController.cs, RpudController.cs и ElasticSearchContract.cs.

Клиент инициирует ajax-запрос, который выполняет действие либо HomeController, либо RpudController. Ajax-запрос содержит набор параметров, которые передаются от клиента к ElasticSearch (данный инструмент будет описан в следующей главе).

HomeController возвращает представление приложения. RpudController обращается к методам ElasticSearchContract, который в свою очередь использует .NET-клиент Nest (описание приводится в следующей главе). Nest обращается непосредственно к ElasticSearch, который возвращает результат, затем результат возвращается клиенту, проходя те же шаги в обратном порядке.

Благодаря использованию концепции MVC разработчик отделяет интерфейс от логики приложения.

Помимо отделения интерфейса от логики приложения, концепция MVC существенно уменьшает сложность больших приложений. Код получается гораздо более структурированным, и, тем самым, облегчается поддержка, тестирование и повторное использование решений.

2.3 AngularJS

AngularJS - это JavaScript-фреймворк, написанный на JavaScript. AngularJS распространяется в виде JavaScript-файла, и подключается к html-странице путем добавления на веб-страницу с помощью тега <script>.

AngularJS расширяет HTML с помощью ng-directives:

- ng-app: определяет AngularJS приложение
- ng-model: связывает значение контроллеров HTML (input, select, textarea) с данными приложения
- ng-bind: связывает данные приложения с HTML-видом

AngularJS был изучен с использованием таких ресурсов, как официальная документация: https://angularjs.org/ и другие вспомогательные ресурсы: http://metanit.com/web/angular/ и http://www.w3schools.com/angular/

В создаваемом решении AngularJS используется в файле View, index.cshtml, для заполнения данными списка чекбоксов, используемых для организации фасетного поиска и для вывода данных в таблицу результатов поиска.

3. ElasticSearch

Elasticsearch - хорошо масштабируемый движок с открытым исходным кодом для полнотекстового поиска и аналитики. Elasticsearch позволяет хранить, искать и анализировать большие объемы данных быстро и практически в режиме реального времени [7].

Есть несколько концепций, которые являются ключевыми для Elasticsearch:

1. Near Realtime (NRT): Elasticsearch является платформой поиска почти в реальном времени. Это означает, что есть небольшая задержка (обычно, одна секунда) с момента индексирования документа до момента его доступности для поиска.
2. Cluster: Кластер представляет собой набор из одного или нескольких узлов (серверов), которые вместе содержат все данные и предоставляют возможность индексации и поиска по всем узлам. Кластер идентифицируется уникальным именем, которое по умолчанию является "elasticsearch".
3. Node: Узел представляет собой единый сервер, который является частью кластера, сохраняет данные, а также участвует в индексации и поиске.
4. Index: Индекс представляет собой набор документов, которые имеют несколько схожие характеристики.
5. Type: В индексе можно определить один или несколько типов. Тип является логической категорией/разделом индекса, чья семантика полностью зависит от разработчика.
6. Document: Документ является основной единицей информации, которая может быть проиндексирована.
7. Shards & Replicas: Индекс может потенциально хранить большое количество данных, которые могут превышать аппаратные пределы одного узла. Чтобы решить эту проблему, ElasticSearch предоставляет

возможность поделить свой индекс на несколько частей, называемых shards. Таким образом, ElasticSearch позволяет распределять и осуществлять параллельные вычисления посредством shards, тем самым увеличивая производительность/пропускную способность [8].

Для ElasticSearch существует два официальных .NET клиента:

- ElasticSearch.NET: является низко-уровневым, свободно-зависимым клиентом, который не имеет представления о том, как строятся и представляются запросы и ответы. ElasticSearch.NET позволяет абстрагироваться так, чтобы все API конечных точек Elasticsearch представлялись как методы, но этого уровня абстракции недостаточно, чтобы строить пользовательские объекты JSON, запросы и ответы.
- NEST: является клиентом высокого уровня, который имеет преимущество сопоставлять все объекты запросов и ответов, поставляется с сильно типизированным языком запросов DSL и использует преимущества специфических функций .NET. NEST использует Elasticsearch.Net клиент в качестве нижнего уровня и работает над ним.

Для того чтобы использовать ElasticSearch в приложении, развернутом в Azure, можно добавить виртуальную машину и на ней установить ElasticSearch. Далее, в параметрах установить значение ip-адреса, как статический, вместо динамического - для того, чтобы использовать ip-адрес для подключения к ElasticSearch в веб-приложении. Минимальная стоимость такого решения порядка 50$ в месяц.

Альтернативным решением может быть использование локальных ресурсов для запуска ElasticSearch. Для работы с ElasticSearch на локальной машине, необходимо скачать архив с официального сайта, распаковать его, затем запустить файл elasticsearch.bat и дождаться сообщения в консоли о том, что ElasticSearch запущен.

Далее, необходимо воспользоваться REST-клиентом для создания индекса с помощью PUT-запроса: в данной работе используется расширение для браузера Google Chrome - Postman.

После создания индекса, используя аналог командной строки cURL, индексируется json-файл.

В ходе выполнения работы используется .NET-клиент NEST для обеспечения более высокого уровня абстракции и упрощения работы с объектами и запросами.

3.1 Индексирование БД

ElasticSearch содержит в себе API для создания индекса. ElasticSearch обеспечивает поддержку нескольких индексов, в том числе выполнение операций по нескольким индексам сразу. Каждый создаваемый индекс может иметь определенные параметры, связанные с ним.

API для создания индекса позволяет добавлять или обновлять документ в формате JSON определенного индекса, что делает документ доступным для поиска. В данной работе в качестве документа для индексации используется файл rpud.json.

Для осуществления индексации необходимо сначала на сервере запустить ElasticSearch, дальше с помощью запроса PUT создать индекс, в котором будут определены параметры индекса. В данной работе в качестве параметров для создания индекса используются такие статические (не изменяемые в процессе работы) параметры, как тип (string или integer) и параметр, который указывает необходимость анализа поля: для Description параметр не указывается (по умолчанию, индекс в таком поле считается текстом), для остальных полей указывается параметр "not_analyzed", что означает, что поле не будет подвергнуто действию анализаторов. После создания индекса необходимо, используя средства cURL, индексировать документ с помощью специальной команды, в которой необходимо указать путь до файла, предназначенного для индексации.

В данной работе для осуществления описанных выше действий используется запущенный на локальной машине ElasticSeacrh и средства cURL, что позволяет создавать и заполнять индекс.

3.2 Фасетный поиск и его организация

Фасетный поиск представляет собой метод для доступа к информации, организованной в соответствии с разносторонней системой классификации, что позволяет пользователям исследовать имеющуюся информацию путем применения нескольких фильтров. Фасетная классификация определяет каждый информационный элемент вдоль нескольких метрик, называемых фасетами, что позволяет классификации быть упорядоченной несколькими способами, а не в единственном, заранее определенном, таксономическом порядке.

Фасеты соответствуют свойствам информационных элементов. Они часто получены путем анализа элементов текста с помощью методов извлечения сущности или из ранее существовавших полей в базе данных.

Для описания фасетного поиска в ElasticSearch используется понятие агрегаций. По результату работы и цели использования агрегации делятся на три основных класса:

1. Bucketing: класс агрегаций, которые строят контейнеры, где каждый контейнер связан с ключом и критерием документа. Когда агрегация выполняется, все критерии контейнера оцениваются в контексте по каждому документу и когда критерий совпадает, документ попадает в соответствующий контейнер. В конце процесса агрегации мы получаем список контейнеров - каждый из них с набором документов, которые "принадлежат" ему.
2. Metric: агрегации, которые хранят информацию и вычисляют показатели по множеству документов.
3. Pipeline: агрегации, которые являются совокупностью результатов работы других агрегаций и связанных с ними показателей.

Поскольку каждый контейнер эффективно определяет набор документов (все документы, принадлежащие к контейнеру), можно связать потенциальные агрегации на уровне контейнера, и это будет выполняться в контексте контейнера. Таким образом, агрегирование может быть вложенным [9].

Для создания списка фасетов используется функция GetRPUDAggregates(). Списки фасетов строятся по трем полям: NameInEng, Requirements, Assessment. Результатом работы функции является объект, содержащий три элемента, в которых в свою очередь содержатся значение фасета и количество элементов, удовлетворяющих соответствующему фасету.

3.3 Реализация полнотекстового поиска

Для реализации полнотекстового поиска необходимо привести текст, по которому будет выполняться поиск, к лишенному несущественных деталей виду. В ElasticSearch для таких целей применяются анализаторы.

ElasticSearch содержит в себе анализаторы, которые уже готовы к использованию. Кроме того, можно комбинировать встроенные фильтры символов, токенизаторы и фильтры токенов для создания пользовательских анализаторов [10].

Каждый анализатор состоит из нескольких обработчиков:

- символьной фильтрации;
- токенизации;
- фильтрации полученных токенов.

Основная цель анализатора: из исходного текста получить список токенов, которые отражали бы его суть путем исключения из текста информации, не несущей смысловой нагрузки, как-то: знаки препинания, вводные слова, предлоги, союзы и прочее.

Для предварительной обработки текста, прежде чем он передается токенизатору, используются символьные фильтры. Символьный фильтр,

например, может использоваться для преобразования символа “&” в “И”. Символьные фильтры поддерживают возможность использования регулярных выражений.

После применения символьных фильтров выполняются следующие действия:

- текст делится на токены/слова (токенизатор standart);
- все токены переводятся в нижний регистр (фильтр токенов lowercase);
- убираются токены, которые находятся в списке стоп-слов (фильтр стоп-слов);
- производится стемминг оставшихся фильтров - нахождение основы слова, которая не всегда совпадает с его корнем (фильтр snowball).

Список стоп-слов можно оставить пустым, задать самостоятельно или использовать стандартный - ElasticSearch содержит стандартный список стоп-слов для русского, английского и других языков.

После приведения текста к набору токенов можно выполнять поиск. Для предоставления пользователю функций полнотекстового поиска в веб-приложении создано поле для ввода поискового запроса и соответствующая кнопка “Поиск”, при нажатии на которую входные данные из поля ввода обрабатываются, и пользователь получает результат.

За реализацию полнотекстового поиска отвечает функция KeywordSearch(string[] keywords).

Тестирование

В рамках выполнения данной работы было проведено нагрузочное тестирование. Для проведения нагрузочного тестирования были настроены сценарии с использованием Selenium и Visual Studio Web test.

Тестирование было проведено с помощью инструментов, встроенных в интегрируемую среду разработки Visual Studio 2015 в связке с консольным браузером Phantom.js, управляемым через JavaScript. Тестирование было направлено на замер производительности нижних слоёв приложения (логика, скорость работы источника данных).

Для успешной организации тестирования были изучены такие ресурсы, как http://www.seleniumhq.org/docs/, http://selenium2.ru/docs.html и http://phantomjs.org/documentation/ - документация, посвященная Selenium и Phantom.js.

Phantom.js - это аналог веб-браузера, лишенный графического интерфейса. Данный инструмент позволяет полностью эмулировать работу браузера, но не создавать и не нагружать систему созданием графического окна. Выбор в пользу Phantom.js вместо веб-драйверов привычных браузеров, таких как Google Chrome или Intetrnet Explorer, обусловлен проведением нагрузочного тестирования: создание графического окна для каждого подключения существенно увеличило бы нагрузку на систему и затруднило тестирование.

Selenium - это инструмент для автоматизации тестирования веб-приложений, который позволяет взаимодействовать с браузером.

Для использования Selenium и Phantom.js их необходимо подключить к проекту с помощью менеджера NuGet пакетов в Visual Studio. После этого, в создаваемом сценарии необходимо описать действия, необходимые для тестирования: создание экземпляра веб-браузера, переход по ссылке на тестируемое приложение, отправка запроса, проверка наличия и корректности результата, закрытие веб-браузера.

После описания тестового сценария, необходимо произвести нагрузочное тестирование, для чего используются средства Visual Studio.

Также, следует отметить, что для тестирования работы прототипа была произведена генерация данных для реляционной базы с помощью инструмента dbForge Studio for SQL Server, а для наполнения индекса в ElasticSearch, была использована улитита JDBC Importer: https://github.com/jprante/elasticsearch-jdbc.

Далее приведены результаты тестирования (в индексе содержится 16000 документов):

1) Нагрузка 80 тестов в секунду. Результаты показаны на Рисунке 1.

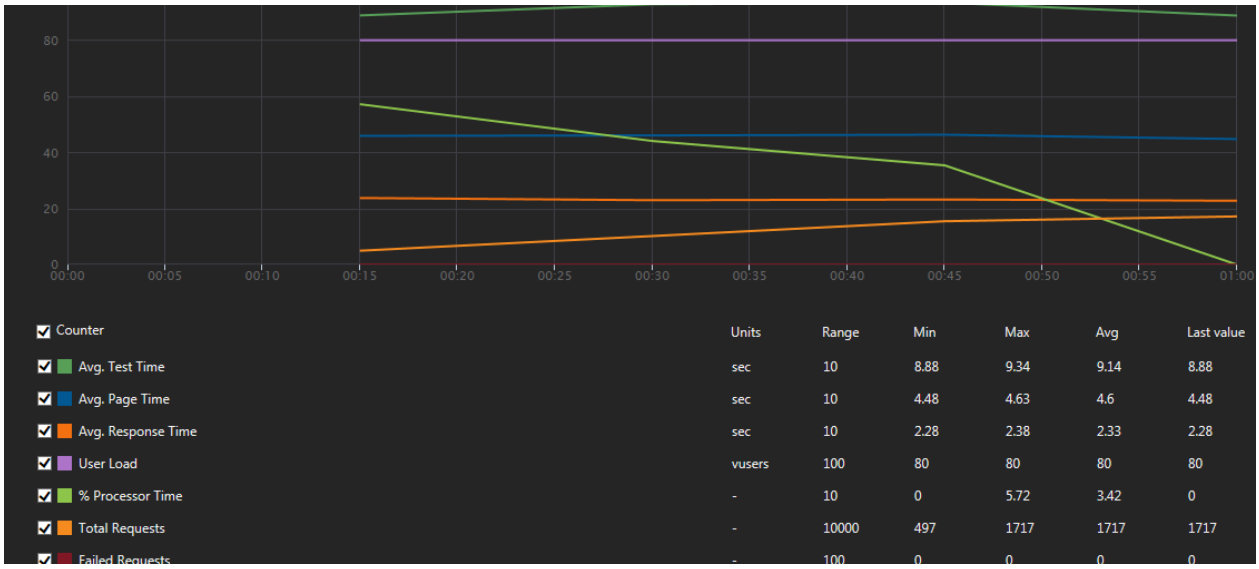

Рис. 1. Результаты тестирования

2) 5 пользователей в секунду. Результаты показаны на Рисунке 2.

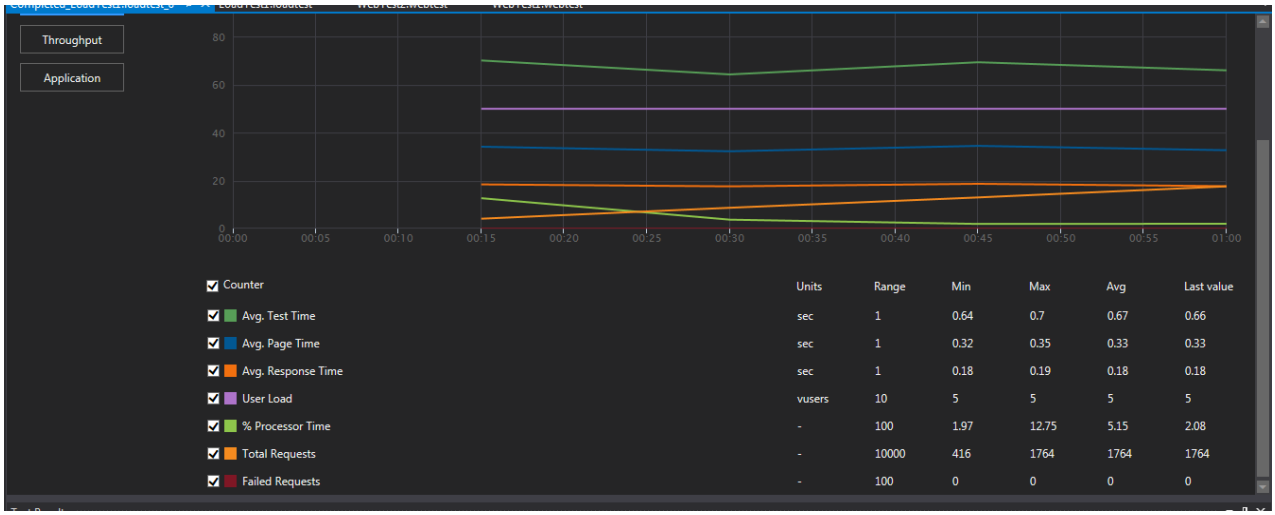

Рис. 2. Результаты тестирования

Нагрузочное тестирование разработанного прототипа показало возможность работы 370 одновременных запросов до первого отказа и среднее время отклика 0,45 сек на 100 подключений.

Вывод

В ходе выполнения данной работы был разработан прототип веб-приложения, осуществляющего доступ к каталогу рабочих программ учебных дисциплин СПбГУ, и проведено нагрузочное тестирование разработанного прототипа веб-приложения.

Решение полностью удовлетворяет поставленной задаче и прототип может быть принят к тестовой эксплуатации.

Разработанный прототип позволяет сократить временные затраты и упрощает поиск необходимой информации среди РПУД, т.к. предоставляет функции фасетного и полнотекстового поиска по документам, что на этапе постановки задачи определяло успешное выполнение поставленной цели. Прототип обладает достаточным потенциалом гибкости и масштабируемости за счет использованных средств и подходов.

К узким местам данного решения можно отнести реализацию полнотекстового поиска, которая обусловлена сложностью естественного языка и возможностью допущения неочевидных грамматических или других речевых ошибок разработчиками РПУД. Хотя ElasticSearch и предоставляет стандартные средства для работы с русским и английским языками, этого может быть недостаточно для правильного решения задачи поиска.

Для решения описанной проблемы можно обратить внимание на использование нечеткого поиска, в котором используется расстояние Дамерау — Левенштейна - количество операций вставки, удаления, замены, транспозиции для того, чтобы одна строка совпала с другой.

Заключение

В рамках проекта по внедрению интерфейса единого доступа к каталогу РПУД для различных классов пользователей был произведен анализ требований, сформулированных Университетом в виде технического задания, и поиск оптимального решения.

В ходе работы был разработан прототип интерфейса для работы с каталогом РПУД, который предоставляет пользователям различные поисковые функции. Также проведено тестирование разработанного прототипа.

Исходный код проекта опубликован на GitHub и доступен по ссылке: https://github.com/quakke/SyllabusPortal

Список источников

[1] – Continuous Integration http://martinfowler.com/articles/continuousIntegration.html

[2] – Обзор безопасности Windows Azure https://msdn.microsoft.com/ru-ru/library/dn312110.aspx

[3] – Microsoft PowerShell https://msdn.microsoft.com/en-us/powershell/mt173057.aspx

[4] – Обзор ASP.NET 5 https://habrahabr.ru/post/243667/

[5] – Начало работы с .NET Framework https://msdn.microsoft.com/ru-ru/library/hh425099(v=vs.110).aspx

[6] – Общие сведения о ASP.NET MVC https://msdn.microsoft.com/ru-ru/library/dd381412(v=vs.108).aspx

[7] – Getting Started | Elasticsearch Reference [2.3] | Elastic https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html

[8] – Basic Concepts | Elasticsearch Reference [2.3] | Elastic https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html

[9] – Aggregations | Elasticsearch Reference [2.3] | Elastic https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html

[10] – Analysis | Elasticsearch Reference [2.3] | Elastic https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis.html