Ермолаев Максим Сергеевич

Выпускная квалификационная работа бакалавра

Разработка и развитие электронного расписания СПбГУ

Направление 010300

Фундаментальная информатика и информационные технологии

Бакалаврская программа «Программирование и информационные технологии»

Научный руководитель,
ст. преподаватель
Севрюков С. Ю.

Содержание

Введение

В 2011 году была сформирована рабочая группа с целью разработки и введения в эксплуатацию информационной системы «Единое электронное расписание СПбГУ» на основе действующего аналога системы, успешно функционирующего на Юридическом факультете СПбГУ. Ключевые задачи этой системы – доведение до контингента обучающихся и профессорско-преподавательского состава актуального расписания учебных занятий и внеучебных мероприятий, расчет занятости аудиторного фонда и преподавателей. В 2012 году было произведено тестовое внедрение системы на три направления СПбГУ: Филология, Востоковедение и Искусства. С 2014 года система успешно функционирует на всех направлениях университета. В настоящий момент осуществляется техническая поддержка, разработка новых функций системы и ее модернизация.

Автор данной работы был принят на стажировку в состав упомянутой группы и перечисленные в настоящем тексте результаты затрагивают не только вклад автора, но и коллег.

Цели и задачи

Объектом работы и исследования является информационная система Электронное расписание, а также процессы, сопровождающие и обеспечивающие работу системы.

Целью данной работы является анализ и улучшение функциональных возможностей системы.

Для достижения цели ожидается решение следующих задач:

1. Проанализировать бизнес процессы и правила в такой области как планирование нагрузки и расписания.
2. Проанализировать архитектуру системы, её компоненты, используемые технологии и компоненты.
3. Выявить и перечислить основные требования к системе на основе проведённого анализа бизнес процессов и правил, реализация части которых будет настоящей работой.

Глава 1. Бизнес правила и ограничения процесса планирования нагрузки и расписания

В данном разделе будут описаны основные логические моменты, бизнес правила и ограничения, свойственные для процессов планирования нагрузки и расписания. Этот обзор поможет выявить основные требования к реализации функциональности системы и принципам её работы.

1. Если говорить о результатах планирования (нагрузки или расписания), то конечным результатом является некоторое подмножество данных, которое может меняться во времени.
2. Планирование происходит с учётом множества ограничений, накладываемых на создаваемые элементы плана (преподавательское поручение, назначение (событие, мероприятие, пара)). *Пример ограничения – допустимые дни работы преподавателя в определённом месте по заданному типу нагрузки (например, чтение лекций в Петергофе по будням дням кроме среды).*
3. Ограничения могут иметь разный уровень учёта – рекомендованные к учёту, строгие.
4. Ограничения накладываются на определённый период с возможной пролонгацией, могут действовать на определённый период времени с определённой частотой повторения. *Например, с января по май ежегодно.*
5. Элемент множества не должен вступать в конфликт с другими элементами множества по одному или более правилам в конечном счёте, однако допускается нарушение этих правил на период времени, регламентируемый здравым смыслом и под ответственность сотрудников, занимающихся планированием.

Речь идёт, например, о невозможности проведении двух и более занятий по разным дисциплинам в одном помещении или проведением двух и более занятий по разным дисциплинам в разных местах одним преподавателем. Да, ответственные за планирование должны избегать таких нарушений в итоге, но в силу разных обстоятельств могут идти на нарушение на некоторый период времени.

6. В силу допускаемых конфликтов (5), востребованы средства обнаружения подобных конфликтов.
   Условно можем это назвать валидацией заданного множества на наличие конфликтов по заданным правилам.
7. В силу допускаемых конфликтов (5), и с учётом ограничений (2-4), востребованы средства по выработке рекомендаций для устранения таких конфликтов.
8. Элементы множества планирования подлежат выборке по одному или более критериям с целью последующей обработки, визуализации или анализа (агрегации). Каждое свойство элемента может использоваться в критериях отбора.
9. Критерий отбора может быть любым из наиболее известных: сравнение, проверка на вхождение в множество, по частичному совпадению (с заданной строкой).
10. Планирование производится на периоды разной продолжительности.
11. В ряде случаев допускается пересечение периодов или наоборот, строго ограничивается и даже требуется наличие определённое количество дней между ними.
12. На этапе планирования и на этапах эксплуатации результирующего множества элементов планирования допустимы разные подходы к протоколированию изменений.
    На этапе планирование все данные обрабатываются как черновые и

допускаются множественные неточности, нарушение правил (2-5). На этапе эксплуатации количество нарушений должно минимизироваться, а любые изменения протоколироваться.

13. Два подмножества могут выступать относительно друг друга как планируемое и фактическое.
14. Планируемое и фактическое множество могут иметь идентичную или различную структуру.
    Например, множество преподавательских поручений (планируемое) и множество назначений на них (фактическое) имеют разную структуру, а множество назначений в начале периода эксплуатации и в конце имеют идентичную структуру.
15. Сравнение различий между планируемым и фактическим множеством может производится поэлементно или с использованием агрегатных числовых характеристик (суммарное количество часов с или без использования критериев отбора).
16. Основной числовой характеристикой является продолжительность в часах (как астрономических, так и академических).
17. Сравнение (15) по числовой характеристике (16) может проводиться для определения согласованности и качества данных, для определения объёма выполненной и предстоящей работы по планирования.
    Например, до начала планирования расписания, но после окончания планирования нагрузки сравнение пустого множества (расписание) с полным (нагрузка) говорит о 100% разнице и может интерпретироваться как работа с объёмом по созданию назначений на имеющееся количество поручений.
18. Результаты сравнения (15) могут использоваться как персоналом, задействованном в планировании для самоконтроля, так и руководством и администрацией с целью контроля.

19. Результаты сравнения (15) могут быть востребованы по запросу и\или в виде оповещения о выходе за пределы допустимого диапазона значений.
    Например, если планируемое множество от фактического начало различаться на 80% и меньше по сравнению с более высоким значением до момента наблюдения, на электронную почту сотрудников и руководства может быть направлено письмо, содержащее предупреждение о выходе за допустимый диапазон и при необходимости детальной информацией о причинах.
20. Средства обнаружения конфликтов (6) и результатов сравнения (15) могут работать как единый комплекс для всех участников, так и для разных групп в разное время в виде отдельных подмножеств инструментов. Т.е. востребованы средства настройки оповещений для подключения новых участников или для отключения ранее заинтересованных.
21. Для унификации учёта и работы правил (2-5, 19) и оповещений (19-20) становится востребован набор методов по кодированию правил, периодической проверки соблюдения или нарушения этих правил и заданию правил оповещения (управление подписками).
22. С учётом (12) востребована регистрация не только того, что изменилось, но и причин и заявок на изменения.
23. Причины и заявки (22) могут быть классифицированы, каждая категория может выступать в качестве критерия для отбора и последующего анализа связанных элементов планирования (9). *Например, оценка количества отмененных или перенесённых занятий с разбивкой по кафедрам может служить оценкой исполнительности преподавательского состава кафедры.*
24. С точки зрения использования аудиторного фонда существуют события, которые не связаны с учебным процессом, но препятствуют

проведению такового в отдельных аудиториях или для заданного контингента.

Примером таких событий могу стать конференции, семинары, проведение ремонтных работ. Контингент же может быть отвелчён от учебного процесса по причинам празднования наиболее значимых мероприятий в масштабе вуза или факультета, срочное отлучение на медицинское обследование, карантин. Данные об этих мероприятиях должны фиксироваться для того, чтобы эти события были учтены в правилах валидации (5).

Глава 2. Архитектура решения

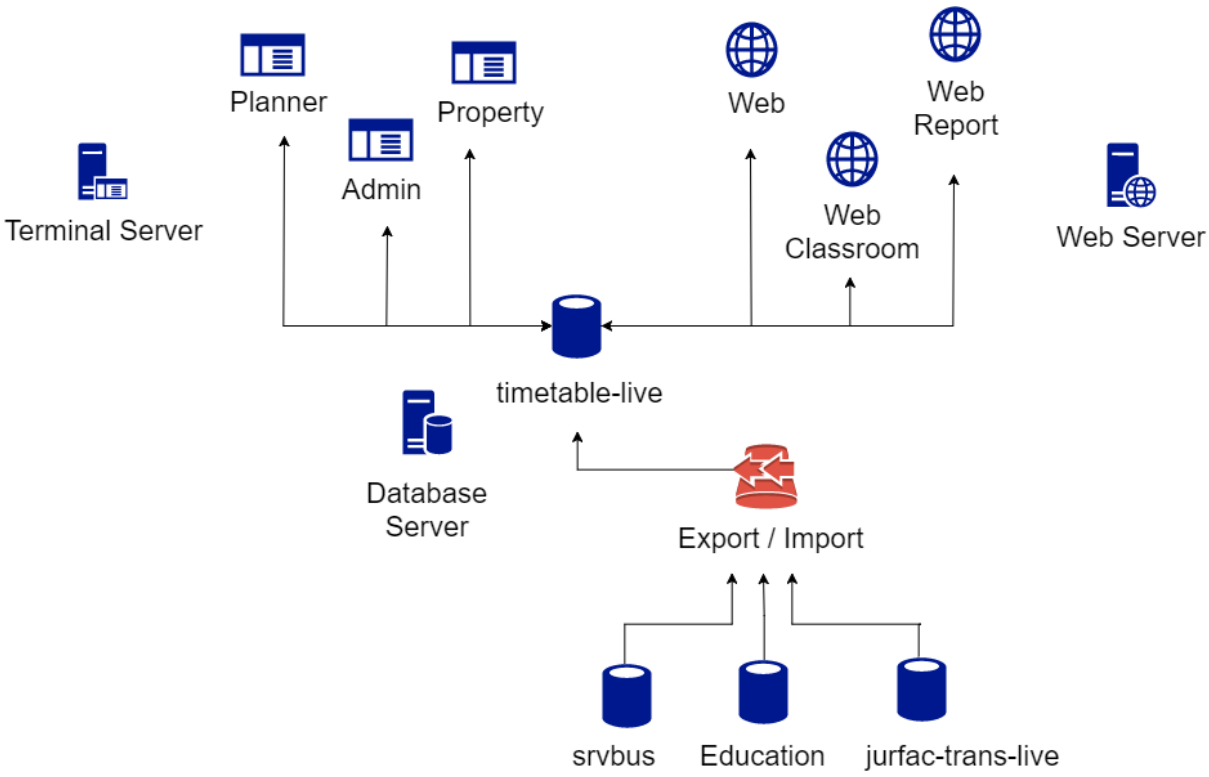

Рисунок 2.1. Схема модулей.

Данная схема, приведённая выше на *Рисунке 2.1*, отражает больше движение данных, нежели зависимость модулей (как таковые они являются независимыми друг от друга, но используют общий источник данных).

Настольные приложения-модули:

- Planner - редактор объектов планирования (расписаний, студенческих групп/потоков, учебных/внеучебных занятий и т.д.).

- Admin - административный модуль, управляющий правами пользователей программы Planner, определяющий базовые настройки системы в целом.
- Property - модуль по работе с Аудиторным Фондом СПбГУ.

Web-модули:

- Web - основной сайт для просмотра расписаний студ. групп, преподавателей и внеучебных расписаний.
- Web Report - модуль для построения и просмотра отчётов по занятости аудиторий.
- Web Classroom -- модуль для просмотра мероприятий/занятий, проводимых в аудитории. Как правило, используется для вывода на экран перед входом в аудиторию.

Модули интеграции данных:

- Exporter - экспортирует данные из внешних источников и записывает их в файлы с расширением *csv*.
- Importer - обрабатывает и импортирует записи, полученные в результате экспорта данных (результаты работы Exporter’a), в БД Электронного Расписания.

2.1. Ресурсы системы

Для развёртывания решения были выделены сервера:

1) сервер для удаленного доступа к приложениям-редакторам, 2) сервер для хостинга веб-приложений, 3) сервер БД и сервисов интеграции данных.

На серверные машины 1, 2, 3 была установлена операционная система Windows Server 2008 R2 Enterprise [1].

Основной технологией для разработки приложений была выбрана платформа .NET [2].

По этой причине, и для возможности запускать и разворачивать приложения, на серверах была установлена платформа Microsoft .NET Framework 4.5.1 [3].

В качестве СУБД был установлен программный комплекс MS SQL Server 2008 R2 Enterprise [4].

Объём БД на данный момент составляет 6 гигабайт, скорость роста - приблизительно 2 гигабайта в год, количество пользователей (уникальных, параллельно работающих за период) выполняющих операции чтения и записи одновременно (т.е. исключая пользователей Web приложений) составляет от 10 до 40 человек. Количество запросов в единицу времени с распределением по типу пользователя (редакторы, Web, сервисы) - не удалось вычислить средние значения, т.к. разброс значений существенный и сильно зависит от конкретного дня и времени суток, для трезвого оценивания необходим длительный период трасировки запросов (не меньше недели).

Web-сервер. Количество запросов за период времени с 23 апреля по 20 мая - 879 496, при этом количество уникальных запросов - 711 202 (подробный отчёт в *приложении А*), средний объём возвращаемых данных на запрос (при очистки кэша и аппаратной перезагрузке) - 850 КБ (без очистки кэша средний объем по всем страницам сайта примерно одинаков и составляет 50 КБ в среднем), соотношение уникальных запросов к повторяемым за период - примерно 4 к 5.

Был настроен планировщик задач для запуска задач резервного

копирования данных и для запуска средств интеграции данных. Время выполнения задач было выбрано таким образом, чтобы максимально сократить нагрузку на сервер БД. Как правило, это свободное от работы сотрудников время, т.е. еженощно. Продолжительность бекапирования базы - несколько минут. Продолжительность работы средств интеграции - меньше минуты. Логирование производится в текстовые файлы, создающиеся каждый день, чтобы группировать по датам. Настроены оповещения по почте в случае возникновения ошибок.

По итогам изменений исходного кода программ, время от времени является необходимым пересобирать файлы приложений на основе изменённого кода и делать обновления - замену старых файлов приложений на новые. При этом частота обновления может быть разной в зависимости от потребностей и процесса разработки. Процесс обновления заключается в запуске службы, которая препятствует запуску процессов настольных программ у всех пользователей. Пока работает служба, происходит замена файлов старой версии на файлы новой версии и обновляется БД и её версия в соответствии с новой версией приложений. Для того чтобы пользователи вновь могли запустить программу, необходимо завершить работу службы. Т.к. каждая версия программы предусматривает работу лишь с конкретной версией БД, является невозможным использование разных версий настольных программ в одно и то же время.

Для разработки настольных приложений-редакторов была выбрана технология WinForms [5], наряду с решением от DevExpress [6] - платформой для генерации форм пользовательского интерфейса - XAF [7].

Доступ к данным из (.NET) приложения обеспечен с помощью ORM [8]. ORM позволяет избежать написания большого количества кода SQL-запросов [9] в приложении. В силу увеличения скорости разработки имело

смысл рассмотреть, и, в конечном итоге сделать выбор в пользу конкретной реализации ORM от компании DevExpress, называемой XPO [10], которая используется как компонент доступа к данным в XAF.

Создание/обновление структуры БД также осуществляется средствами ORM.

Для веб-приложений - ASP.NET MVC 5 [11] с использованием движка для генерации представлений Razor [12].

Освоенные и используемые в работе библиотеки:

- jQuery [13] - набор библиотек по работе с JavaScript [14].
- Bootstrap [15] - платформа для Web-разработки на базе Html [16], CSS [17] и JavaScript.
- Swashbuckle [18] (реализация Swagger [19] для C# [20]). Библиотека помогающая разрабатывать документацию для API [21].
- NLog [22] - средства для логирования в .NET.
- NLog.Targets.Sentry [23] - пакет интеграции Sentry [24] и NLog.
- DevExpress Spreadsheet [25] - библиотека по работе с таблицами формата Excel.

Подводя итог, архитектура представляет собой совокупность различного типа модулей для отображения, редактирования, интеграции данных, а также общие библиотеки.

Глава 3. Выявленные требования и задачи развития и разработки электронного расписания СПбГУ

3.1. Разработка высокопроизводительного и удобного интерфейса для манипуляции данными нагрузки и расписания

В данном разделе будет произведён обзор и аргументация по выбору технических решений и инструментов для построения высокопроизводительного и одновременно удобного интерфейса пользователя основного модуля системы - Редактора расписания. Забегая вперёд, следует отметить, что решение, которое было проанализировано автором работы, было реализовано без его участия и нарушало предъявленные требования по производительности (удовлетворяя требованию удобства использования) и задачей стал поиск мер и решений, способных устранить этот недостаток.

Организация пользовательского интерфейса и используемые компоненты

Для того чтобы планировать электронное расписание необходим пользовательский интерфейс, который позволит создавать элементы расписания, редактировать и удалять. Также следует учесть, что в процессе создания элементов расписания учебных занятий может использоваться информация о педагогической нагрузке. Задача реализации такого интерфейса достаточно универсальна и имеет множество существующих решений, в том числе с использованием технологии WinForms. Для удобства использования был выбран визуальный компонент Scheduler Control [26] от компании DevExpress. Компонент выглядит как календарь и позволяет настраивать различные параметры по работе с ним и элементами календаря.

Календарь обладает преимуществами для пользователя:

1. Возможность просматривать неделю целиком или только выбранные дни недели.
2. Навигация по неделям/дням с помощью панели-календаря.
3. Цвета назначений [27] (labels [28]) -- может использоваться для отображения набора состояний, в которых пребывает то или иное назначение.
4. Использование drag-and-drop [29] и горячих клавиш.
5. В целом Календарь DevExpress эмулирует внешний вид и возможности календаря Outlook [30], что обеспечивает дополнительное удобство для пользователей, ранее работавших с Outlook.

Календарь обладает преимуществами для разработчика:

6. Встроенные диалоги по редактированию/работе с отдельным назначением. [31]
7. Встроенные диалоги по созданию/редактированию/настройке повторяющихся назначений.
8. Встроенная концепция “использования ресурсов для назначений” [32], позволяющая каждое назначение относить к конкретному ресурсу и соответствующим образом отображать их (для случая с *расписанием учебных занятий* в роли ресурсов выступают *студенческие группы*, а назначением является *учебное занятие*, проводящееся для конкретной студ. группы).
9. Встроенные средства по фильтрации назначений по набору ресурсов, включённых в отображение (исключённых из отображения).
10. Календарь может работать с данными, привязанными к внешнему источнику (Bound Mode), либо использовать внутреннее хранилище (Unbound Mode). [33]

11. В случае Bound Mode достаточно указать соответствие (маппинг) свойств исходного объекта (Source Object) и назначения (Appointment). [34]
12. Для реакции на изменения в отображаемых данных существует механизм подписки на события изменения/удаления/добавления назначений.
13. Возможность конфигурирования пользователем параметров сетки календаря (например, верхние и нижние границы шага, длительность события по умолчанию и т.д.).

Также календарь DevExpress предусматривает создание так называемой “серии событий” или шаблонного события [35]. Ранее было принято решение сохранять объекты, которые использует Календарь в базу данных. При этом каждое шаблонное событие хранится в источнике данных в виде одной записи с уникальным идентификатором и XML-полем [36], содержащим информацию о шаблоне. Иными словами, некоторые записи в БД соответствуют событиям календаря один к одному, в то время как другие содержат в себе совокупность связанных по шаблону событий. Чтобы интерпретировать данные о шаблонах необходимо использовать методы (API) DevExpress. Стоит отметить, что такое ограничение не позволяет работать с данными на уровне SQL-кода. В виду полученных ограничений был обнаружен ряд организационных проблем, связанных с построением отчётов в реальном времени, так как теперь, чтобы получить развёрнутую информацию о событиях требуется выполнить преобразование данных в БД. На данный момент процесс преобразования (разворачивания) данных реализован в рамках модулей, описанных в главе 2, - Exporter и Importer.

Устранение обозначенных организационных проблем, используя доступные технические средства и методы, и является одной из задач данной работы.

Устранение зависимости от API, препятствующего удовлетворению требования по производительности

Текущее решение заключается в том, чтобы по запросу от пользователя или на регулярной основе запускать процесс экспорта-импорта, который переводит события из текущего формата данных в развернутый формат, где каждому событию, в т.ч. повторяющемуся, соответствовала запись в базе данных, что позволяет работать с событиями с помощью SQL. Очевидные недостатки такого решения заключаются в том, что созданные таким образом данные быстро устаревают, и процесс приходится запускать заново. Как следствие, отсутствует возможность получать данные по этим отчетам по запросу, в режиме реального времени.

К возможным решениям можно отнести:

1. Поддерживать в актуальном состоянии 2 таблицы: существующую и таблицу с развернутыми событиями. Тут также может быть 2 варианта:
    a. Посредством отдельного синхронизационного сервиса
    b. OLTP [37]
2. Пересмотреть понятие серии событий с позиции общей сущности - пед. поручения. И реализовать ETL [38] данных таблицы событий.

В случае (1a) у нас появится возможность выполнять SQL-запросы, но не будет возможности использовать данные второй таблицы (с развёрнутыми событиями) в приложениях, в силу их неактуальности на конкретный момент времени, что может привести к серьёзным проблемам.

Сложность решения (1б) заключается, с одной стороны, в реализации компонентов по работе с двумя таблицами, с другой стороны, каждая транзакция [39] по изменению любой записи в исходной таблице приводила бы к изменению в среднем 16 записей во второй таблице, где 16 – средняя длина серии событий. Таким образом, каждая операция записи в исходную таблицу СУБД создавала бы значительные блокировки на уровне второй таблицы, что нивелирует значимость второй таблицы как источника для чтения данных.

Что касается решения (2), оно достаточно простое и не имеет явных недостатков, в отличие от других решений.

Для перехода требуется выполнить следующие шаги:

1. Реализовать скрипт [40] или модуль по миграции данных [41], который заменит все повторяющиеся и неповторяющиеся события в текущей таблице на эквивалент неповторяющихся событий. Также необходимо мигрировать все таблицы, связанные с таблицей событий.
2. Реализовать логику работы с сериями событий, которые теперь объединены на уровне реляционной модели [42] отношением одна серия – множество событий.

Упомянутый в пункте 1 модуль был разработан, а код этого модуля находится в открытом доступе:
https://github.com/lainiwakurafan/DXEventsMigrator

В процессе реализации модуля возникли проблемы со скоростью обработки данных, поскольку для этого, ввиду ограничений, являющихся первопричиной данной работы, были использованы компоненты уровня приложения. Был опробован ряд подходов, в том числе XPCursor [43] от

DevExpress, а также разбиение данных по годам – направлениям обучения. Вопрос с производительностью этого модуля по-прежнему открытый, однако, данные необходимо мигрировать только один раз.

Упомянутая в пункте 2 логика достижима за счёт использования такого общего объединяющего элемента как преподавательское поручение, ссылка на которое присутствует в каждом назначении. Такой способ связи можно использовать для оперирования выборкой назначений как единым подмножеством (в частности, повторяющихся событий).

В данный момент задача управления таким подмножеством в виде аналогичном предыдущему (настройка повторяющихся событий) находится в стадии разработки. Основная сложность, которая возникает в решении данной задачи – это разработка и тестирование значительного объема кода по работе с повторяющимися событиями. Значительно упрощает работу тот факт, что вследствие выбранного решения среди прочих альтернатив, текущие компоненты приложения уже умеют работать с событиями, которые мы получили в результате миграции.

3.2. Web API (timetable.spbu.ru/help)

Т.к. разрабатываемая система является многомодульной и распределённой, то вырастает спрос на применение различных архитектурных подходов, способствующих не только максимально переиспользовать фрагменты кода и логики, но и целых слоёв. Наиболее востребованным общим слоем является слой доступа к данным, который бы возвращал непротиворечивые наборы данных в любом модуле при запросе таковых.

API (Application Programming Interface) -- это интерфейс, набор методов для взаимодействия и интеграции приложений. API позволяет получить

результаты работы одной системы (приложения) и использовать эти результаты в другой системе (приложении).

В контексте решаемых задач основной целью является не упомянутая интеграция, а удовлетворение требования повышения доступности данных:

- **API облегчает доступ к данным**.
- **Мобильные приложения для своей работы используют API**. Мобильные технологии сейчас набирают популярность, вместе с ними растёт и потребность в мобильном приложении для Электронного Расписания.
- **Интеграция общеуниверситетских систем.**

Решение

API может быть реализован с помощью различных платформ и языков программирования с применением разного вида архитектур. В данной работе в качестве начального шага было решено ограничиться уже используемыми технологиями и создать API внутри основного Web приложения.

Технические детали решения

В качестве базового механизма управления запросами от клиентов, их маршрутизации и обработки был использован класс ApiController [44]. В ходе построения самого API были повторно использованы методы доступа к данным, используемые Web-приложением, что значительно сократило время на разработку и отбросило необходимость создания логики получения данных.

Документация

Следующая задача после получения рабочей версии API -- это создание документации того набора методов, который API предоставляет. Следует учесть, что при изменениях в структуре API (добавление нового метода, изменение параметров) необходимым является поддержание документации в актуальном состоянии. Это важный момент, поскольку разработка API может продолжаться и после введения его в эксплуатацию.

Именно по причине частых изменений в структуре (особенно на начальных этапах проектирования) были рассмотрены программные решения автоматической генерации документации на основе существующего набора методов.

В частности была опробована библиотека Swashbuckle (подробно описанная в параграфе 2.1), позволяющая описывать структуру API и читать её. Чтение структуры в свою очередь позволяет Swashbuckle построить интерактивную API документацию.

Итоги

В рамках настоящей работы для доступа к данным используются методы, которые используются в Web-приложении, возвращающем по запросу расписание студентов, преподавателей и аудиторий. Этого было достаточно для разработки прототипа, но такое решение не может использоваться в промышленной эксплуатации. Основная причина этому - низкая производительность текущей реализации методов доступа к данным и следующий раздел будет посвящен теоретическому и практическому исследованию с целью поиска наиболее оптимальной замены.

3.3. Повышение производительности доступа к данным

Решив проблему, описанную в параграфе 3.1, отпадает необходимость использования API визуального компонента Календаря для получения данных на Веб. Это даёт возможность рассмотреть альтернативные решения, повысив при этом производительность. Прежде чем рассмотреть решения, выработаем критерии сравнения.

Критерии сравнения

1. Среднее время отклика. [45]
2. Объем хранимых ресурсов на диске.
3. Поддержание исходного кода в состоянии валидности (валидность можно проверять Unit Test'ами [46], но это дополнительный объем работы [критерий 4]).
4. Объем работы при реализации.
5. Объем потребляемой оперативной памяти

Возможные решения

Для начала рассмотрим текущее решение. В текущем решении существует БД с данными и для работы с ней используется редактор. При этом используется слой доступа, реализованный посредством ORM XPO. Также этот слой используется и для доступа к данным со стороны Web.

Что можно было бы сделать (какие группы решений есть):

1. Оставить текущее решение (**XPO Base Objects [47]**). При этом подходе ORM извлекает из БД весь набор полей объектов, тем самым тянет большой трафик из базы (что прямо влияет на производительность [критерий 1]), т.е. не может делать выборку из множества полей таблицы.

2. Оптимизировать решение (1) с использованием следующих возможностей:
    - Отложенная загрузка отдельных свойств объектов (**Delayed Loading [48]**)
    - “Явная загрузка” свойств объекта (**Explicit Loading [49]**)
    - Настроить **кэширование на уровне провайдера данных [50]**. Уже опробовано и применено с целью повышения производительности [критерий 1].
    - **XPView [51]**. Используется только для отображения данных. Можно указать подмножество колонок таблицы, данные которой нужны. Из БД извлекаются только необходимые поля объектов, что позволяет сократить время запросов [критерий 1]. XPView также позволяет агрегировать данные используя строковые выражения языка SQL.
3. Заменить решение (1) на запросы к существующей БД через альтернативные слои.
4. **Индексирование данных**. Подразумевает генерацию индекса (т.е. некоторых избыточных данных) на основе данных нормализованной (существующей) БД. Данные индекса следует хранить в отдельной документ-ориентированной БД [52] (далее ДОБД) с поддержкой JSON [53] и REST [54] API. Данный подход оправдан, т.к. речь идёт о сложной структуре данных. В центре структуры данных находится объект типа Event, который имеет массу вложенных объектов. Например, Event содержит в себе коллекцию EventLocations, которые в свою очередь также имеют вложенные объекты и т.д. Это отлично вписывается в концепцию ДОБД. Кроме того, данные будут храниться в формате JSON и их не придётся преобразовывать, также не придётся писать SQL-код для получения данных. Помимо всего прочего - легкость в плане масштабируемости и использовании.

Создав дополнительное хранилище данных для веб-приложений, запросы на редактирование и на чтение данных будут распределены по разным БД и не будут блокировать друг друга.

5. **Хранимые процедуры [55] (скомпилированный [56] SQL)**. В отличие от прямых SQL-запросов, хранимые процедуры легче поддерживать и они обеспечивают валидность запроса [критерий 3].

Для решения (3) в свою очередь может быть несколько вариантов (альтернативных слоёв):

A. **ADO.NET [57]**. Общение с базой в ADO.NET происходит посредством языка SQL и требует написания большого объема кода [критерий 4]. ADO.NET является базовым (низкоуровневым) механизмом для доступа к данным и отличается высокой скоростью работы [критерий 1].

B. **Легковесные ORM, типа Dapper [58]**.

Решения (1, 2, 4) можно рассматривать независимо от остальных и друг от друга, в то время как (3) состоит в выборе ORM, либо использовании ADO.NET.

Критерии\решения	**Текущее решение (1)**	**Оптимизация текущего решения (2)**	**ADO.NET (3A)**	**Dapper (3B)**	**Индексация (4)**
Среднее время отклика	~ 5-6 с	~ 1 с	~ 0.5 с	~ 0.6 с	~ 0.5 с
Объем потребляемых ресурсов	значительное потребление оперативной памяти	малое потребление оперативной памяти	малое потребление оперативной памяти	малое потребление оперативной памяти	малое потребление оперативной памяти, нужно

					дополнительное место на диске для отдельной БД
Поддержка кода на валидность	отсутствует валидация названий свойств для Criteria, есть валидация типов и свойств при использовании LINQ to XPO	отсутствует валидация названий свойств	отсутствует валидация Sql кода, нужна поддержка соответствия между C# классами и моделью данных БД	отсутствует валидация SQL кода, нужна поддержка соответствия между C# классами и моделью данных БД	----------
Объем работы при реализации	ничего не нужно делать	C# код для XPView, настроить кеширование, настроить атрибуты для отложенной и явной загрузки	много Sql кода, C# код для маппинга объектов из БД в приложение	много Sql кода, C# код классов для Dapper (немного)	создание БД, логика синхронизации, C# код для доступа к данным

Таблица 3.1. Сравнение решений для повышения производительности

Чтобы минимизировать риск ошибок в SQL-коде для решений (1, 2, 3) имеет смысл использовать хранимые процедуры (решение 5) для выполнения SQL-запросов. Хранимые процедуры содержат в себе уже скомпилированный SQL-код, прошедший проверку на наличие ошибок. Более того выполнение именно хранимых процедур может повлечь за собой небольшой прирост производительности. Однако в сравнении с

ORM на написание хранимых процедур требуется больше работы с кодом в целом.

Вывод

Решения 3A и 3B сильно похожи, но благодаря Dapper можно избежать написания C# кода для маппинга объектов БД и приложения, при этом производительность останется на том же уровне. Если выбирать между ADO.NET (3A) и Dapper (3B), то решение в пользу Dapper (3B).

Очевидно, что решение 1 нужно развивать в решение 2, мы ничего не теряем, кроме времени.

Решение 4 масштабно в плане реализации, т.к. необходимо предусмотреть механизм синхронизации изменений между базами, создать и настроить ДОБД, обеспечить к ней доступ веб-приложениям. Такой подход к решению задачи займёт немало времени и ресурсов.

В рамках данной работы уже предприняты шаги в сторону оптимизации текущего решения (2). А именно - настроено кэширование для веб-проекта, поэтому будет логично продолжить улучшать существующее решение, наблюдать за результатами, и иметь в виду (на будущее) Dapper (3B) и вариант с индексированием данных в отдельную ДОБД (решение 4).

При всём при том, совершенно неважно, какое решение в итоге будет достигнуто (или на каком решении всё остановится), параллельно в качестве улучшения и минимизации рисков ошибок в SQL доступно использование хранимых процедур (решение 5).

3.4. Накладки или поиск конфликтов

В текущем решении ограничения, описанные в главе 1 (пункты 2, 3, 4) не предусмотрены правилами самой системы и контролируются сотрудниками.

В силу пункта 6 главы 1, было разработано решение: в момент изменения элементов множества планирования необходимо проверять эти элементы на наличие конфликтов с другими элементами множества.

При попытке сохранить изменения, приводящие к конфликтам, сотрудник системы планирования должен быть уведомлен с помощью всплывающего окна о наличии конфликтов.

Сотрудник может подтвердить внесение конфликтных изменений. В таком случае система больше никак не будет автоматически индицировать полученный конфликт.

Сотрудник в любой момент времени может проверить любой элемент множества планирования на наличие конфликтов описанных в п. 5.

Как дополнительное правило использования имеет смысл рассмотреть и реализовать проверку конфликтов в пределах конкретного расписания. Полученный сценарий использования позволит оценивать состояние и качество работы процесса планирования в любой момент времени. На случай отклонения качественного показателя больше чем на норму следует оповещать сотрудников по почте. Мерой оценки качества возьмём суммарное время всех пересечений (конфликтов). Чтобы определить группу сотрудников, связанных с конфликтами, необходимо обладать знаниями о принадлежности того или иного сотрудника к элементу расписания или расписанию в целом. В настоящей работе была использована связь сотрудника с направлением, на котором он работает и к которому допущен.

3.5. Логирование

Логирование нужно Электронному расписанию для анализа работы системы и диагностирования неполадок. Реальные случаи когда могут потребоваться логи:

- Экспорт/импорт данных. Как уже говорилось в главе 2, интеграция данных происходит каждую ночь, и один из способов проверить успешность выполнения - исследовать логи. В логах могут храниться не только записи об ошибках экспорта/импорта, но и записи с информацией о протекающих процессах (например время начала и окончания, что позволило бы оценивать среднее время выполнения).
- Выявление деталей и контекста возникновения той или иной ошибки для помощи в отладке.
- Построение статистики ошибок за промежуток времени.

Задача ведения логов была решена с использованием библиотеки NLog (библиотека упомянута в параграфе 2.1) в два этапа:

1. Установка и настройка библиотеки для логирования - NLog - таким образом, чтобы логи сохранялись в текстовые файлы и БД.
2. Улучшение решения (1) с сохранением логов в специальный WEB-сервис Sentry для последующего анализа и статистики. Sentry - это готовое решение для построения отчётов и агрегирования информации на основе логов, при этом предоставляющее пользовательский WEB-интерфейс.

Детали установки и настройки NLog

Установка заключалась в подключении файла библиотеки *NLog.dll* и создании файла настройки *NLog.config* [59] в котором можно определить места и формат для логирования (NLog targets [60]). Первоначальной целью являлась запись логов в текстовые файлы, отсортированные по дням, при этом формат сообщений включает детали контекста возникновения лога (URL, название контроллера, действия (Action), представления) [61], имя и логин пользователя, характер сообщения (Error, Info, etc.) [62], само сообщение. Следующим шагом было создание таблицы *TimetableLog* в БД и включения её в качестве альтернативного места хранения логов. БД позволяет хранить информацию логов структурированно и более удобно её использовать.

Детали установки и настройки Sentry

Был осуществлён поиск способов интеграции Sentry в проект. В частности удалось найти и реализовать решение с использованием уже предустановленного пакета NLog, что сократило затраты времени и написания кода на реализацию. Решение заключалось в установке библиотеки NLog.Targets.Sentry (подробнее в главе 2, параграф 2.1), добавлении настроек Sentry в файл *NLog.config*, а также регистрации проекта в веб-сервисе.

Заключение

Итак, была поставлена цель - исследовать и по возможности улучшить функциональное состояние информационной системы Электронное расписание. Аналитика системы не была проведена и исследована в полной мере, однако проделанная работа вывела аналитику на новый уровень за счёт а) популяризации данных посредством WEB API (параграф 3.2), б) продвинутых средств анализа логов (параграф 3.5) и в) системы оповещения и индикации конфликтов процесса планирования (параграф 3.4). Всё ещё есть проблемы с производительностью, но данная работа положила начало вопросу её улучшения, т.к. были предложены методы оптимизации и дальнейшего развития в этом направлении (параграф 3.3). Не лишним будет сказать, что проделана большая коллективная исследовательская работа в отношении того, как хранятся данные в БД, как они должны храниться и что нужно для развития. Также автором работы было предложено решение на базе ETL - мигратор, использующий API DevExpress для разворачивания данных в удобный формат и последующей перезаписи в базу (параграф 3.1). Возвращаясь к вопросу аналитики, базовые характеристики системы всё же удалось обозначить и зафиксировать в работе, что тоже несёт позитивный вклад. На основе полученных характеристик можно сделать выводы:

- Объем возвращаемых данных в ответ на запрос пользователя веб-сайта - выше нормы, причина тому - избыточный JavaScript-код.
- Число повторяющихся запросов на сайте относится к общему числу запросов как 1 к 5, тем самым можно оценить эффективность работы механизма кэширования (получается, что в среднем 20% всех запросов берётся из кэша).

Ссылки

[1] Windows Server 2008 R2 and Windows Server 2008 // https://technet.microsoft.com/en-us/library/bc831894-e291-4610-a89f-3ee0758e4a96

[2] .NET Documentation // https://docs.microsoft.com/en-us/dotnet

[3] .NET Framework 4.7, 4.6, and 4.5 // https://msdn.microsoft.com/en-us/library/w0x726c2(v=vs.110).aspx

[4] SQL Server Technical Documentation // https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation

[5] Windows Forms // https://msdn.microsoft.com/en-us/library/dd30h2yb(v=vs.110).aspx

[6] DevExpress Documentation // https://www.devexpress.com/support/documentation

[7] eXpressApp Framework // https://documentation.devexpress.com/xaf

[8] ORM // https://ru.wikipedia.org/wiki/ORM

[9] SQL // https://ru.wikipedia.org/wiki/SQL

[10] XPO Documentation // https://community.devexpress.com/blogs/xpo/archive/2006/05/23/xpo-documentation.aspx

[11] Getting Started with ASP.NET MVC 5 // https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started

[12] Razor Syntax // https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor

[13] jQuery API // https://api.jquery.com

[14] JavaScript reference - JavaScript | MDN // https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

[15] Bootstrap // http://getbootstrap.com

[16] HTML | MDN // https://developer.mozilla.org/en-US/docs/Web/HTML

[17] CSS Reference // https://www.w3schools.com/cssref

[18] Swashbuckle: Seamlessly add a swagger to WebApi projects! // https://github.com/domaindrivendev/Swashbuckle

[19] Swagger Specification // http://swagger.io/specification

[20] C# Programming Guide // https://docs.microsoft.com/en-us/dotnet/articles/csharp/programming-guide

[21] Wiki: API // https://ru.wikipedia.org/wiki/API

[22] NLog Wiki // https://github.com/nlog/nlog/wiki

[23] Sentry Target for NLog // https://github.com/bjarneriis/NLog.Targets.Sentry

[24] Sentry Documentation // https://docs.sentry.io

[25] DevExpress: Spreadsheet docs // https://documentation.devexpress.com/#WindowsForms/CustomDocument12063

[26] DevExpress: Scheduler Control docs // https://documentation.devexpress.com/#WindowsForms/CustomDocument1729

[27] DevExpress: Appointment Interface // https://documentation.devexpress.com/#CoreLibraries/clsDevExpressXtraSchedulerAppointmenttopic

[28] DevExpress: Appointment Labels and Statuses // https://documentation.devexpress.com/#WindowsForms/CustomDocument1754

[29] Wiki: Drag-and-drop // https://ru.wikipedia.org/wiki/Drag-and-drop

[30] Wiki: Outlook // https://en.wikipedia.org/wiki/Outlook.com

[31] DevExpress: Edit Appointments Dialogs // https://documentation.devexpress.com/#WindowsForms/CustomDocument2357

[32] DevExpress: Resources for Appointments // https://documentation.devexpress.com/#WindowsForms/CustomDocument1756

[33] DevExpress: Scheduler Data Binding: Bound and Unbound modes // https://documentation.devexpress.com/#WindowsForms/CustomDocument8386

[34] DevExpress: Scheduler Mappings // https://documentation.devexpress.com/#WindowsForms/CustomDocument15468

[35] DevExpress: Recurring Appointments and Exceptions // https://documentation.devexpress.com/#WindowsForms/CustomDocument1755

[36] Wiki: eXtensible Markup Language // https://ru.wikipedia.org/wiki/XML

[37] Wiki: Online Transaction Processing // https://en.wikipedia.org/wiki/Online_transaction_processing

[38] Extract, transform, load process // https://en.wikipedia.org/wiki/Extract,_transform,_load

[39] Wiki: Transaction // https://en.wikipedia.org/wiki/Transaction

[40] Wiki: Scripting Language // https://en.wikipedia.org/wiki/Scripting_language

[41] Wiki: Data Migration // https://en.wikipedia.org/wiki/Data_migration

[42] Wiki: Relational model // https://en.wikipedia.org/wiki/Relational_model

[43] DevExpress: XPO: XPCursor Class // https://documentation.devexpress.com/#CoreLibraries/clsDevExpressXpoXPCursortopic;
DevExpress: Using Pageable Collections with XPCollection and XPCursor // https://documentation.devexpress.com/#CoreLibraries/CustomDocument2075

[44] MSDN: ApiController Class // https://msdn.microsoft.com/ru-ru/library/system.web.http.apicontroller(v=vs.118).aspx

[45] Wiki: Response time // https://en.wikipedia.org/wiki/Response_time_(technology)

[46] Wiki: Unit testing // https://en.wikipedia.org/wiki/Unit_testing

[47] DevExpress: XPO: Base Persistent Classes // https://documentation.devexpress.com/#eXpressAppFramework/CustomDocument113146

[48] DevExpress: XPO: Delayed Loading // https://documentation.devexpress.com/#CoreLibraries/CustomDocument2024

[49] DevExpress: XPO: ExplicitLoadingAttribute Class // https://documentation.devexpress.com/#CoreLibraries/clsDevExpressXpoExplicitLoadingAttributetopic

[50] DevExpress: XPO: Session Management and Caching // https://community.devexpress.com/blogs/xpo/archive/2006/03/27/session-management-and-caching.aspx

[51] DevExpress: XPO: XPView Concepts // https://documentation.devexpress.com/#CoreLibraries/CustomDocument2068

[52] Wiki: Document-oriented database // https://en.wikipedia.org/wiki/Document-oriented_database

[53] Wiki: JSON // https://en.wikipedia.org/wiki/JSON

[54] Wiki: Representational state transfer // https://en.wikipedia.org/wiki/Representational_state_transfer

[55] Wiki: Stored procedure // https://en.wikipedia.org/wiki/Stored_procedure

[56] Wiki: Compiler // https://en.wikipedia.org/wiki/Compiler

[57] MSDN: ADO.NET documentation // https://msdn.microsoft.com/en-us/library/aa286484.aspx

[58] Dapper - a simple object mapper for .Net. // https://github.com/StackExchange/Dapper

[59] NLog Configuration file // https://github.com/NLog/NLog/wiki/Configuration-file

[60] NLog Targets // https://github.com/nlog/nlog/wiki/Targets

[61] NLog Context Layout Renderer // https://github.com/nlog/nlog/wiki/Event-Context-Layout-Renderer

[62] NLog: Log levels // https://github.com/nlog/nlog/wiki/Log-levels

Приложение А. Статистика данных по веб-сайту из Google Analytics.

Timetable.Web
Все данные по веб-сайту

ПЕРЕХОД К ОТЧЕТУ

Страницы

23 апр. 2017 г. - 20 мая 2017 г.

Все пользователи
Просмотры страниц: 100,00 %

Статистика

Страница	Просмотры страниц	Уникальные просмотры страниц	Средняя длительность просмотра страницы	Входы	Показатель отказов	Процент выходов	Ценность страницы
	879 496 % от общего количества: 100,00 % (879 496)	711 202 % от общего количества: 100,00 % (711 202)	00:00:44 Средний показатель для представления: 00:00:44 (0,00 %)	221 597 % от общего количества: 100,00 % (221 597)	43,45 % Средний показатель для представления: 43,45 % (0,00 %)	25,20 % Средний показатель для представления: 25,20 % (0,00 %)	0,00 $ % от общего количества: 0,00 % (0,00 $)
1. /	**71 977** (8,18 %)	58 112 (8,17 %)	00:00:17	51 292 (23,15 %)	5,42 %	7,60 %	0,00 $ (0,00 %)
2. /EducatorEvents/Index	**17 317** (1,97 %)	13 332 (1,87 %)	00:00:13	1 379 (0,62 %)	7,54 %	2,43 %	0,00 $ (0,00 %)
3. /LETT	**7 758** (0,88 %)	5 259 (0,74 %)	00:00:21	573 (0,26 %)	9,60 %	3,54 %	0,00 $ (0,00 %)
4. /EARTH	**6 828** (0,78 %)	4 931 (0,69 %)	00:00:15	536 (0,24 %)	10,45 %	3,69 %	0,00 $ (0,00 %)
5. /ECON	**6 806** (0,77 %)	5 535 (0,78 %)	00:00:13	816 (0,37 %)	11,40 %	3,76 %	0,00 $ (0,00 %)
6. /AMCP	**6 385** (0,73 %)	5 241 (0,74 %)	00:00:12	370 (0,17 %)	11,89 %	3,16 %	0,00 $ (0,00 %)
7. /JOUR	**5 767** (0,66 %)	4 741 (0,67 %)	00:00:12	264 (0,12 %)	10,61 %	2,32 %	0,00 $ (0,00 %)
8. /HIST	**4 811** (0,55 %)	3 612 (0,51 %)	00:00:15	650 (0,29 %)	15,38 %	4,47 %	0,00 $ (0,00 %)
9. /GSOM	**4 638** (0,53 %)	3 956 (0,56 %)	00:00:10	382 (0,17 %)	9,69 %	2,52 %	0,00 $ (0,00 %)
10. /MATH	**4 461** (0,51 %)	3 435 (0,48 %)	00:00:14	751 (0,34 %)	8,52 %	4,33 %	0,00 $ (0,00 %)

Строки 1–10 из 55863