САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра компьютерных технологий и систем

Иванов Кирилл Андреевич

Магистерская диссертация

Адаптация системы тестирования для работы в сети филиалов

Направление 02.04.02

Фундаментальная информатика и информационные технологии

Магистерская программа «Автоматизация научных исследований»

Научный руководитель,

ст. преподаватель,

Севрюков С. Ю.

Санкт-Петербург

2018

Содержание

Введение

С 1 января 2015 года иностранные граждане, желающие оформить разрешение на работу, патент, разрешение на временное проживание либо вид на жительство, в соответствии с Федеральным законом № 74 от 20.04.14 «О правовом положении иностранных граждан в Российской Федерации» и приказом Минобрнауки России № 1156 от 28.08.14 «Об утверждении формы, порядка проведения экзамена по русскому языку, истории России и основам законодательства РФ и требований к минимальному уровню знаний, необходимых для сдачи указанного экзамена» должны подтвердить знание этих предметов соответствующим сертификатом. Для получения сертификата необходимо успешно сдать экзамен.

2 декабря 2014 года Центр языкового тестирования Санкт-Петербургского государственного университета был включен в перечень организаций, которые могут проводить такой экзамен и выдавать сертификаты.

Важной особенностью системы тестирования СПбГУ является то, что она является многофилиальной. Под филиалом подразумевается один или более классов, где проводится тестирование. Филиалы расположены в разных городах. Класс для тестирования представляет собой помещение с компьютерами, за которыми тестируемые отвечают на вопросы.

Тестирование проводится с помощью программного комплекса **СТКПлюс**, разработанного в СПбГУ.

Программный комплекс состоит из нескольких компонентов:

1. **DataEditor (Модуль управления).** Служит для разработки тестов, регистрации тестируемых, управления процессом тестирования, проверки ответов. В филиале устанавливается на компьютеры проверяющих и на компьютер оператора.

2. **Testing (Модуль тестирования)**. С помощью модуля тестирования экзаменуемый отвечает на вопросы.
3. **Сервис управления**. Устанавливается сервер филиала - отдельный компьютер внутри филиала. Сервис является связующим звеном для передачи данных внутри филиала.
4. **База данных.** БД работает под управлением СУБД SQL Server и установлена на сервере филиала. Помимо данных, БД содержит логику в хранимых процедурах и триггерах.

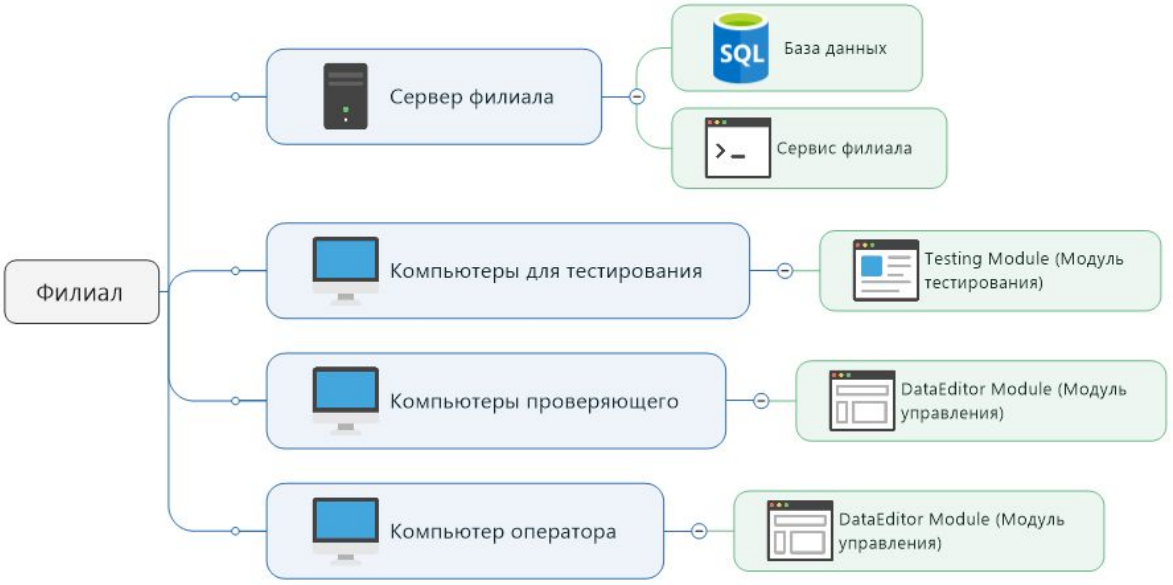

Рис. 1. Схема филиала

На Рис. 1. представлена схема филиала, состоящего из одного класса.

Вопросы, отображаемые **Модулем тестирования**, могут содержать сложное форматирование и включать в себя изображения, видео и аудио. Вопросы делятся на следующие типы:

1. Требующие выбора правильного ответа среди заданных. Отвечая на них, экзаменуемый нажимает на выбранный ответ на экране сенсорного монитора;
2. Требующие устного ответа. На них тестируемый отвечает в микрофон;

3. Требующие письменного ответа на бумаге. В этом случае на мониторе отображается только вопрос.

Вопросы 2 и 3 типа требуют проверки специалистом. С помощью модуля **DataEditor** проверяющий, находящийся в другом помещении, может прослушивать ответы и отмечать ошибки. По окончанию проверки программа рассчитывает результаты экзаменуемого.

Печать и учет сертификатов происходит в центре, в другой программе, которая была создана до **СТКПлюс** тоже сотрудниками УСИТ — **БД Сертификаты**. Сотрудник филиала регулярно выгружает результаты экзаменов с помощью **Модуля управления** в файл в формате CSV и отправляет их в Центр для печати сертификатов.

Развертывание и обновление программного комплекса в филиалах проводится вручную сотрудниками СПбГУ согласно инструкции путем подключения к удаленному рабочему столу сервера филиала.

Следует отметить, что еще до разработки **СТКПлюс**, Центр языкового тестирования СПбГУ начал осуществлять тестирование в Едином Центре Документов. Тестирование проводилось с помощью иного ПО, которое разрабатывалось и поддерживалось сторонней организацией.

Автор работы был вовлечен в разработку **СТКПлюс** в рамках производственной практики в 2016 году. В **СТКПлюс** были недостатки в архитектуре передачи аудио-ответов тестируемых на сервер филиала. Во многом этой проблеме в системе была посвящена бакалаврская работа автора. В начале 2017 года автор был привлечен к работе в роли стажера разработчика УСИТ СПбГУ.

Летом 2017 года Центр языкового тестирования решил внедрить в Единый Центр Документов **СТКПлюс**. Возникла необходимость адаптации **СТКПлюс** к работе в Едином Центре Документов.

Наряду с этим к лету 2017 года возросло количество филиалов и число тестируемых в них. Обострились следующие проблемы:

1. Передача результатов тестирования для печати сертификатов в центр происходит в ручном режиме.

   Эта проблема особенно актуальна, когда тестируемых в филиале много. Требуется следить за тем, какие тестируемые уже были выгружены, а какие нет.

2. Низкий контроль за качеством тестирования в филиалах из центра.

   В центр поступают только результаты, на основании которых можно печатать сертификаты. Контролировать тестирование можно только путем подключения к удаленному рабочему столу сервера.

3. Отсутствие какого-либо предварительного анализа аудио-ответов тестируемых.

   Ответы часто содержат тишину в самом начале — в это время тестируемый обдумывал ответ. Проверяющему приходится тратить время в ожидании голоса, либо вручную перематывать аудио.

4. Ручное развертывание и обновление филиалов.

   При увеличении числа филиалов, развертывать и обновлять их вручную, путем подключения к удаленному рабочему столу становится сложно.

Постановка задачи

Цель работы — усовершенствовать систему тестирования **СТКПлюс** так, чтобы она могла работать в большем количестве филиалов с минимальным количеством ручного труда.

Для достижения цели были поставлены следующие задачи:

1. Интегрировать **СТКПлюс** с внешней информационной системой для возможности работы в филиале ЕЦД;
2. Разработать и протестировать модули для синхронизации данных, избавив сотрудников от необходимости ручной передачи данных;
3. Применить Continuous Integration, Continuous Delivery, Continuous Deployment для разработанных модулей. Благодаря этому станет доступен более быстрый выпуск новых версий и более быстрое устранение ошибок, т. к. будет требоваться меньше времени на ручное развертывание разработанных модулей с целью проверки их работоспособности. Также исчезнет необходимость ручного развертывания разработанных модулей в филиалах.
4. Разработать и протестировать прототип компонента, устанавливающего метки начала и конца ответа в аудио-волне, отображаемой проверяющему. В дальнейшем это позволит проверяющим тратить меньше времени на оценку аудио-ответов. Также наработки в этой области позволят в будущем выявлять наличие голоса в тех ответах, где его быть не должно (с целью контроля качества тестирования).

Обзор литературы

Integration architecture: Comparing web APIs with service-oriented architecture and enterprise application integration [1]**.** Статья размещена на портале DeveloperWorks, который представляет собой сайт для разработчиков и ИТ-специалистов, существующий с 1999 года, и находящийся в ведении IBM.

Автор, Kim J. Clark, сравнивает Web API с другими средствами интеграции программного обеспечения и перечисляет основные этапы развития средств интеграции в хронологическом порядке:

1. Соединение типа точка-точка используя низкоуровневое API
2. Enterprise Application Integration (EAI)
3. SOA
4. Web API.

В статье разъяснены основные предпосылки появления Web API, среди которых — рост числа смартфонов. Для всех этапов развития приведены соответствующие визуальные схемы. Далее автор детально описывает чем отличается Web API от всех прочих технологий и как его можно применять.
При объяснении REST, Kim J. Clark, ссылается на диссертацию Roy Fielding (первым ввел термин REST и является одним из создателей HTTP) «Architectural Styles and the Design of Network-based Software Architectures».
В конце, автор рассуждает о том, какие технологии интеграции будут актуальны в будущем.

В целом статья носит обзорный характер связанных технологий, которые позволяют решать интеграционные задачи. Написанная в 2015 году, она актуальна сегодня.

RESTful Web API [2]**.** Книга, написанная Leonard Richardson и Mike Amundsen, и выпущенная в 2013 году затрагивает теоретические основы

REST API. Предшественником этой книги того же издательства является книга RESTful Web Services авторов Leonard Richardson и Sam Ruby. Книга была издана в 2007 году и, в основном, в ней REST противопоставлялся SOAP. Через несколько лет после выхода первой книги, по мнению авторов, REST «победил» SOAP, но возникла новая проблема: не все сервисы, которые называли себя REST, действительно являлись REST. Зачастую разработчики REST не понимали значение и применение Hypermedia. Это и послужило предпосылкой для Leonard Richardson к написанию книги RESTful Web API.

В начале книги авторы знакомят читателя с основными концепциями, лежащими в основе REST. Прежде всего это такие понятия, как ресурсы и представления. Особое внимание уделяется технологии Hypermedia, которая подразумевает наличие в представлении дополнительных ссылок, например на другие ресурсы. Далее авторы описывают какие форматы Hypermedia могут использовать в REST: Collection+JSON, Maze+XML, OpenSearch, Problem Detail Documents, The Atom Publishing Protocol, OData и другие. Немалую часть книги занимает изложение роли HTTP в API. Конец книги посвящен протоколу CoAP, который часто применяется для межмашинного взаимодействия. CoAP значительно отличается от HTTP, но его архитектура соответствует принципам REST.

Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions [3]**.** В самом начале книги авторов Bobby Woolf и Gregor Hohpepa рассмотрены типичные проблемы с которыми сталкивается разработчик интеграционных решений: ненадежность сети передачи данных, низкая скорость передачи данных, различия между приложениями, неизбежность изменений интегрируемых приложений. Далее рассмотрены различные подходы к интеграции, среди которых — обмен сообщениями. В основе такого стиля лежат следующие шаблоны: канал сообщений,

сообщение, каналы и фильтры, маршрутизатор сообщений, транслятор сообщений, конечная точка сообщения. Каждому из этих шаблонов посвящена отдельная глава. Важно, что авторы не только описывают достоинства обмена сообщениями, но и недостатки. Книга сопровождается практическими примерами на языке C#.

И хотя книга вышла в 2003 году, большая ее часть актуальна за счет того, чтобы авторы описывают именно фундаментальные проблемы, связанные с интеграцией приложений.

Введение в распределенные вычисления [4]**.** Пособие Косякова М. С. содержит основные аспекты распределенных вычислений. В начале книги описываются основные цели построения распределенных систем и требования к ним. Немалая часть учебника излагается математическим языком и содержит фундаментальные вопросы распределенных вычислений. Особое внимание уделено механизму Логических часов. В конце автор приводит список рекомендуемой к прочтению литературы. В этот список входит

Comparison of Voice Activity Detection Algorithms for VoIP [5]**.** Статья посвящена сравнению методов VAD (обнаружение голосовой активности). В начале авторы описывают причины появления VAD, среди которых сокращение объемов передачи голосовых данных. Затем изложена основная сложность при создании VAD — наличие постороннего шума, который нужно отделять от голоса. Далее детально сравниваются основные алгоритмы VAD. Сравнение опирается на следующие основные требования: хорошее правило определения наличия голосовой активности в участке аудио, адаптивность к меняющемуся фоновому шуму, низкая сложность вычислений. В конце статьи представлено сравнение алгоритмов на тестовых данных.

Следует заметить, что статья была выпущена в 2002 году, и с тех пор было написано несколько новых обзоров техник VAD, например Comparison of Speech Activity Detection Techniques for Speaker Recognition [6]. Однако рассматриваемая в обзоре статья содержит более подробное описание различных методов.

A Simple But Efficient Real-time Voice Activity Detection Algorithm [7]**.** В статье предложен алгоритм детектирования голоса, который опирается на следующие характеристики: энергия фрейма, спектральная плоскостность, номер полосы с наибольшим преобладанием энергии. Для каждой характеристики задается свой начальный порог на основе анализа первых фреймов. Применение каждой характеристики обосновано. Алгоритм подробно описан на псевдокоде. В конце приводятся результаты тестирования на тестовых данных. Статья примечательна тем, что в ней показан пример, как можно скомбинировать различные подходы к VAD. Еще одной особенностью описанного алгоритма является то, что в нем применен иной подход вычисления порогового значения для характеристик (по сравнению с подходом, описанным Comparison of Voice Activity Detection Algorithms for VoIP).

Глава 1. Интеграция с внешней ИС

1.1. Сбор требований

Все филиалы работают по схожей схеме. Процесс тестирования можно разделить на этапы:

1. Подача заявки на тестирование;
2. Регистрация тестируемого в базе **СТКПлюс** и формирование группы;
3. Прохождение теста;
4. Проверка ответов тестируемого;
5. Оглашение результата;
6. Изготовление сертификата;
7. Выдача сертификата.

Подача заявки на тестирование. Процесс тестирования начинается с того, что лицо, желающее пройти тестирование, в рабочее время филиала лично подает заявление на экзамен. Организатор выбирает подходящие дату и время для проведения экзамена, уведомляет об основных правилах и выдает направление.

Регистрация тестируемого в базе СТКПлюс. В назначенный срок лицо приходит в класс для тестирования и отдает оператору тестирования направление. Оператор сверяет данные и, если все правильно, вручную вводит персональные данные тестируемого. Как только набирается достаточное число тестируемых, они добавляются в группу тестируемых (лица, проходящие тестирование одновременно). Каждому тестируемому в группе назначается свободный компьютер в классе. Оператор знакомит тестируемого с правилами.

Прохождение теста. Когда процесс формирования группы окончен, тестируемые садятся за компьютер. Оператор объясняет, как правильно заполнять ответы и инициализирует тестирование. Тестируемые отвечают на вопросы. Вопросы отображаются на сенсорном мониторе.

Вопросы делятся на следующие типы:

1. Вопросы, подразумевающие выбор варианта среди предложенных. Тестируемый отвечает на них, касаясь желаемого ответа в мониторе;
2. Вопросы, требующие письменного ответа. На такие вопросы тестируемый отвечают на бумаге;
3. Вопросы, требующие ответа в микрофон («Говорение»).

Для тестируемого экзамен заканчивается, когда он ответил на все вопросы или когда закончилось время, отведенное на тест. Тестируемый освобождает компьютер и ждет результатов.

Проверка ответов тестируемого. Вопросы, подразумевающие выбор варианта ответа, проверяются системой автоматически. Письменные ответы относятся проверяющему. Проверяющий работает в модуле **DataEditor**. Письменные ответы он считывает с бумаги, а ответы субтеста «Говорение», прослушиваются через **DataEditor**. Результаты тестирования выставляются также через **DataEditor**.

Оглашение результата. Как только у тестируемого закончились вопросы для проверки, считается, что экзамен завершен. Оператор озвучивает результат. Если экзамен пройден, то начинается процесс изготовления сертификата. В противном случае на месте заполняется отказ в выдаче сертификата.

Изготовление сертификата. Филиалы могут находиться в разных городах, но все сертификаты печатаются в Санкт-Петербурге в СПБГУ. Оператор уведомляет тестируемого о том, когда будет готов сертификат. По

окончании дня оператор отправляет пакет данных об экзаменах и о необходимости в печати сертификатов в центр по защищенному каналу.

Как отмечалось ранее, помимо тестирования в обычных филиалах, специалисты СПбГУ также проводят тестирование в Едином Центре Документов. Тестирование проводится с другим программным комплексом, который интегрирован в единую информационную сеть Единого Центра Документов.

При переходе на **СТКПлюс** специалисты Единого Центра Документов выдвинули следующие требования: регистрация тестируемого должна проходить в ИС Единого Центра Документов, и результаты тестирования должны быть отправлены обратно в эту ИС. Предполагается, что после регистрации тестируемого на экзамен ему будет выдано направление на тестирование, которое содержит:

1. ФИО;
2. Организацию (если есть);
3. Данные паспорта;
4. Дату экзамена;
5. Номер договора;
6. Штрихкод с закодированным номером договора.

При этом сервер филиала в ЕЦД не должен хранить и обрабатывать персональные данные. Идентификация тестируемого должна проходить по номеру договора.

Такое требование связано, во-первых, с тем, что в ЕЦД большое количество тестируемых, и ввод персональных данных каждого тестируемого оператором занимал бы много времени. Во-вторых, с требованиями законодательства по защите персональных данных в Российской Федерации. В соответствии с постановлением правительства от 1 ноября 2012 г. N 1119 оператору персональных данных необходимо обеспечить сохранность

носителей персональных данных [8]. Сервер филиала в ЕЦД находится не на территории СПбГУ и не может быть защищен. Вдобавок к этому, передача данных должна проходить по защищенному каналу.

1.2. Создание центральной базы данных

Для того, чтобы собирать данные с филиалов в центр, необходимо выбрать базу данных для их хранения. В разрабатываемом решении эта база данных в том числе содержит персональные данные тестируемых, поступающие из **ИС ЕЦД**.

Автором были рассмотрены следующие варианты:

1. Использовать существующую базу данных модуля **БД Сертификаты**;
2. Создать новую базу данных со структурой как в филиалах.

В итоге был выбран второй вариант по следующим причинам:

1. Структура базы сертификатов не рассчитана на хранение всех данных, требующихся Центру Языкового Тестирования;
2. При использовании базы данных со структурой как в филиалах, можно использовать существующий **Модуль управления DataEditor** для просмотра и анализа этих данных.

Далее эта база данных будет обозначаться как **Центральная База**.

1.3. Анализ решений для интеграции

Исходя из требований Единого Центра Документов возникает необходимость “связать” разные информационные системы — **ИС ЕЦД** и **СТКПлюс**, т. е. провести интеграцию: обеспечить обмен данными и наладить рабочие процессы [9].

Существует множество классификаций решений для интеграции информационных систем. Например, в статье [10] выделяется 4 класса. При

этом классы называются также ступенями развития. Каждая следующая ступень появилась позже предыдущей и решает какую-либо ее проблему. При этом ни одна из ступеней не является “устаревшей”. Каждая решает свой набор задач. Основные ступени следующие:

1. **Соединение типа точка-точка с использованием низкоуровневого API**.

   Самый первый тип интеграции систем и приложений, появившийся в начале 90-х годов. Характеризуется тем, что каждое приложение интегрируется с другим каким-то своим способом. Нет каких-либо стандартных протоколов и соглашений. Очевидный недостаток такого подхода: сложность в поддержке, трата лишних ресурсов на добавление нового приложения (требуется писать много нового кода для соединения приложений).

2. **Enterprise Application Integration (EAI)**.

   Данный подход характеризуется тем, что появился некоторый центральный интеграционный хаб, т. е. связующее программное обеспечение, через которое можно подключиться к любой системе. Этот хаб обеспечивает следующее [11] :

   - Механизмы, через которые приложения могут делиться своей функциональностью и данными;

   - Механизмы, через которые приложения координируют бизнес-процессы.

   Подход существенно отличается от первого тем, что при добавлении новой системы не нужно писать дополнительный код для того, чтобы она могла подключаться к существующим.

3. **SOA**. На данной ступени появился стандарт обмена SOAP (начало 2000-х). Ключевая особенность SOA заключается в том,

что к существующим сервисам могут подключиться другие приложения через Service Exposure Gateway, используя стандарт SOAP. Сервисы между собою обмениваются данными через единую шину сообщений (ESB).

4. **Развитие Web API и REST**. В связи с ростом числа смартфонов к 2007 году появилась потребность в решении, которое позволило бы смартфонам и другим клиентам получать информацию с сервисов максимально быстро и удобно. Нельзя сказать, что ранее стандарт SOAP не использовался для этих целей, но он во многом не подходил по причине “громоздкости” и сложности формата. Требовалось более легковесное и удобное решение. Появился Web API и архитектурный стиль REST. Один из основателей подхода REST, Рой Филдинг, выделяет шесть основных принципов REST архитектуры [12]: Модель клиент-сервер; Отсутствие состояния; Кэширование; Единообразие интерфейса; Слои; Код по требованию (необязательное ограничение). Для веб-служб, которые не нарушают это правило применяется термин «RESTful». Большинство RESTful-реализаций используют стандарты, такие как HTTP, URL, JSON и XML. Таким образом, данный этап характеризуется большим количеств API открытых «наружу» и наличием легковесного стиля взаимодействия REST.

Есть и другая классификация, используемая в книге Enterprise Integration Patterns [3]. В отличие от предыдущего варианта, здесь классы называются шаблонами.

1. **Передача файла**. Взаимодействие между приложениями осуществляется с помощью файлов, в которые помещаются данные.

2. **Общая база данных**. Взаимодействие между приложениями осуществляется с помощью базы данных, в которой сохраняется общая информация.
3. **Обмен сообщениями**. Взаимодействие между приложениями осуществляется с помощью системы обмена сообщениями, которые используются для обмена данными и выполнения действий.

Такая классификация является более низкоуровневой по сравнению с предыдущей. Например, в основе многих EAI продуктов лежит шаблон обмена сообщениями.

Можно сказать, что количество интеграционных задач с каждым годом становится больше. Появляется больше различных приложений и информационных систем, которые необходимо связать. Для решения таких задач в настоящее время существует множество решений и подходов, среди которых есть как готовые продукты, так и отдельные инструменты и стандарты для построения интеграции. Выбор конечного решения зависит от конкретной задачи. Один из подходов будет применен для решения задачи интеграции **СТКПлюс** с внешней информационной системой.

1.4. Архитектура решения

Интеграция **ИС ЕЦД** и **СТКПлюс** — это интеграция информационных систем разных организаций. Часто для такого рода интеграции выбирается технология WebAPI [10].

Не существует четкого определения, что такое Web API. В данной работе под этим подразумевается распространенное определение, что это сервис, который осуществляет связь через стандартные веб протоколы и отправляет и получает данные в заранее определенном формате [13]. Важной особенностью Web API является использование именно стандартных веб

протоколов и форматов данных. Благодаря этому формат передачи данных между организациями легко согласовать.

Совместно с разработчиками **ИС ЕЦД** было принято решение использовать именно Web API. В СПбГУ должен быть развернут сервис (**Сервис интеграции**), предоставляющий методы для отправки и получения данных, связанных с тестированием. Основные требования к функциональности сервиса следующие:

1. Сервис должен принимать данные о тестируемом и предстоящем экзамене;
2. Сервис должен возвращать данные об экзамене.

Обращаться к сервису должен агент **ИС ЕЦД**.

В качестве архитектурного стиля обмена данными был выбран REST. При построении решения на основе Restful сервиса часто оперируют терминами ресурс и представление. Ресурс — это то, что может быть сохранено в компьютере: документ, строка базы данных, результат выполнения алгоритма. При использовании HTTP каждый ресурс имеет свой URL. Под представлением подразумевается любой машиночитаемый документ, содержащий какую-либо информацию о ресурсе [2].

В случае обмена данными с Единым Центром Документов при построении сервиса в СПбГУ было согласовано использование протокола HTTP и формата данных XML для представлений ресурсов:

Далее были согласованы два ресурса и их XML-схемы:

1. Новый тестируемый;
2. Экзамен.

Новый тестируемый. Под этим ресурсом понимаются данные тестируемого, включая персональные, и данные о назначенном экзамене:

1. Уровень назначенного экзамена;
2. ФИО на русском языке;

3. ФИО на английском языке;
4. Дата рождения;
5. Тип документа, удостоверяющего личность;
6. Серия паспорта;
7. Номер паспорта;
8. Телефон;
9. Пол;
10. Гражданство;
11. Уникальный идентификатор договора. Этот же идентификатор печатается на самом направлении.

Экзамен. Под этим ресурсом понимаются результаты экзамена. Представление ресурса содержит следующие информацию:

1. Идентификатор направления;
2. Информацию о том, завершен ли экзамен;
3. Дата тестирования;
4. Дата сертификата;
5. Номер сертификата;
6. Организация, выдавшая сертификат;

При построении REST сервиса на основе HTTP за каждым ресурсом необходимо закрепить конкретный URL [2]. Часто предлагается использовать следующие рекомендации:

1. Скрывать расширения файлов серверных сценариев (.jsp, .php, .asp), если таковые используются, чтобы можно было выполнить портирование приложений на другую технологию без изменения URI;
2. Использовать только строчные буквы;
3. Заменять пробелы дефисами или знаками подчеркивания (чем-то одним);

4. Стараться максимально избегать использования строк запросов;
5. Вместо использования кода 404 Not Found для URI, указывающих неполный путь, всегда предоставлять в качестве ответа ресурс или страницу по умолчанию.

При этом для создания ресурса необходимо использовать метод POST, а для получения GET.

В итоге были согласованы следующие методы:

1. POST /api/examinees для создания нового тестируемого
2. GET /api/examens/{contractNumber} для получения результатов экзамена, где {contractNumber} — это номер договора, который был указан при создании тестируемого

1.5. Разработка сервиса

Существующие модули **СТКПлюс** написаны на языке C# и платформе .NET Framework. При разработке нового сервиса было принято решение продолжить использовать эти технологии. Это позволило использовать существующую кодовую базу в новом проекте.

Существует минимум два способа построить WebAPI сервис на платформе .NET Framework — используя либо WCF Framework, либо ASP.NET Web API Framework. Оба фреймворка разрабатываются и поддерживаются Microsoft. В официальной документации [14] представлено сравнение фреймворков. В целом для построения именно HTTP Rest API больше подходит ASP.NET Web API Framework. Именно с применением этой технологии был разработан сервис.

В соответствии со статьей [15] было принято решение использовать многоуровневую архитектуру разрабатываемого приложения. В основе архитектуры лежат три проекта Visual Studio:

1. DTO;

2. Services;
3. WebAPI.Console.

DTO. Для того, чтобы отфильтровать данные из базы и привести их к оговоренному формату, был применен паттерн программирования Data Transfer Object (DTO) [16]. Проект Integration.DTO содержит определения Data Transfer Object.

Services. Слой реализует паттерн “Слой служб” (Service Layers). Этот слой инкапсулирует код отвечающий за бизнес логику приложения и делает его доступным из разных мест [17]. Применение этого паттерна может упростить дальнейшее развитие всей системы в целом. Для нужд обмена данными с ЕЦД было разработано два сервиса:

1. Examinee Service — сервис отвечающий за создание сущности Новый тестируемый;
2. Examen Service — сервис для возврата данных о тестируемом.

WebAPI.Console. Центральный компонент архитектуры. Обрабатывает запросы пользователей. Возвращает сериализованные объекты DTO и HTTP коды.

Также в приложении были задействованы следующие паттерны и технологии:

1. Autofac. Фреймворк для реализации принципа Инверсия управления;
2. eXpressPersistent Objects (XPO). ORM фреймворк от компании DevExpress. Фреймворк уже активно применялся в **Модуле Управления** и поэтому был использовать и в разработанном сервисе с использованием существующих наработок;
3. Automapper для преобразования объектов в DTO.

1.6. Нагрузочное тестирование

Автором работы было проведено нагрузочное тестирование разработанного сервиса. Цель — собрать показатели производительности и время отклика приложения под нагрузкой с разным количеством одновременных подключений (пользователей).

Тестирование проводилось следующим образом:

1. В облачной платформе Azure была создана виртуальная машина со следующими характеристиками:
   - 2 виртуальные ЦП;
   - 8 гб оперативной памяти;
   - 16 гб памяти SSD;
2. В созданной виртуальной машине развернут сервис для интеграции с внешней ИС;
3. Средствами Visual Studio был создан тест имитирующий создание нового тестируемого (Webtest в терминологии Visual Studio) и сбор данных о тестируемых;
4. Средствами Visual Studio было запущено тестирование. Одновременное число пользователей на старте — 10. Далее число пользователей увеличивается на 10 каждую минуту до 100. Время тестирования: 15 минут.

Виртуальная машина примерно соответствует характеристикам сервера, где предполагается установка сервиса. Тестирование позволит проанализировать как будет вести себя сервис, если **ИС ЕЦД** будет проводить асинхронные запросы по созданию пользователей, и если помимо ЕЦД подобным образом будут интегрированы другие филиалы. Помимо этого такое тестирование позволяет определить освобождается ли память приложением после спада нагрузки, т. е. проанализировать приложение на

предмет утечек памяти. Такой вид нагрузочного тестирования часто называется тестированием стабильности или надежности [18]. Итоги тестирования представлены на графиках (Рис. 2, Рис. 3, Рис. 4).

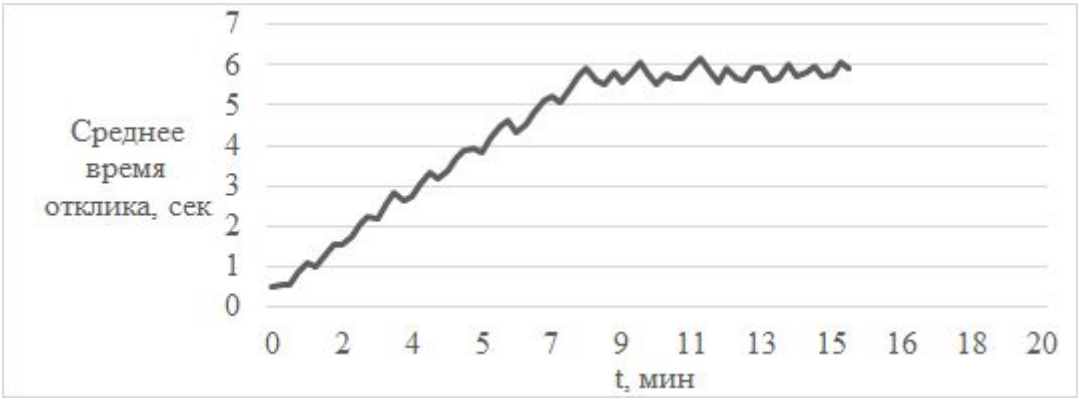

Рис. 2. График среднего времени отклика

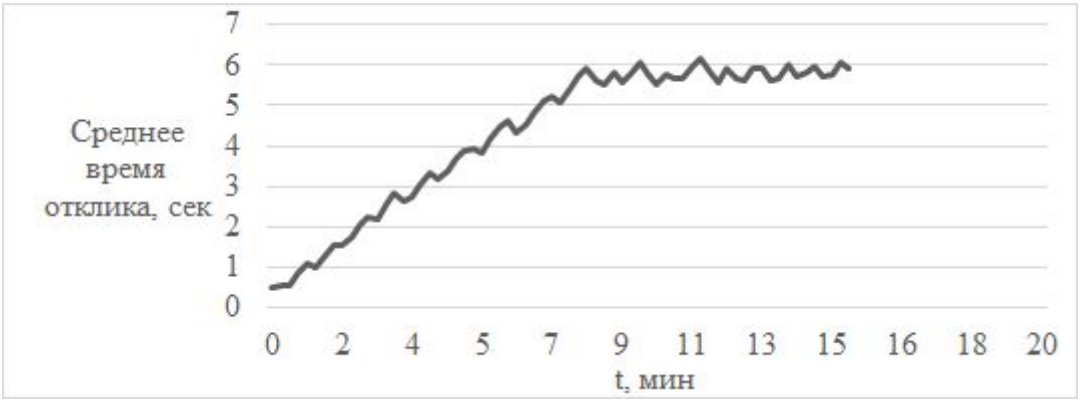

Рис. 3. График расхода памяти

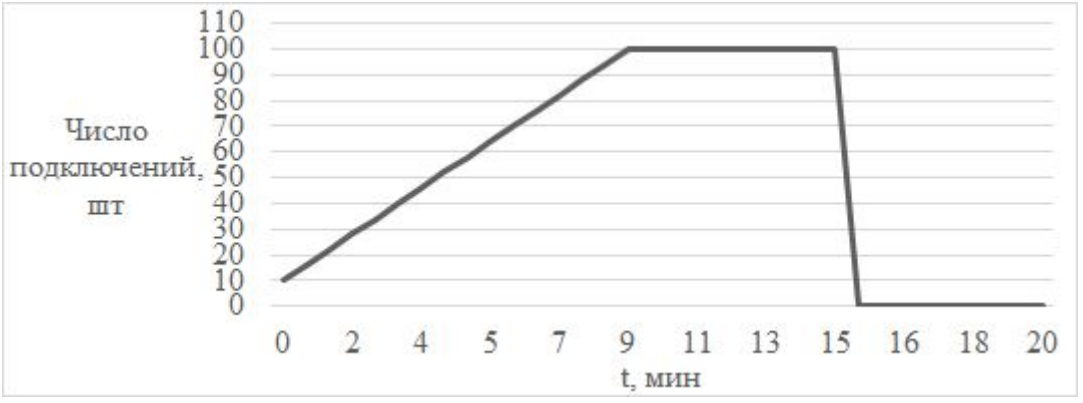

Рис. 4. График числа одновременных подключений

Таким образом были выявлены два показателя производительности. Эти данные могут быть в дальнейшем использованы при согласовании

работы модулей ЕЦД. На первом графике виден закономерный рост среднего времени отклика с увеличением числа пользователей. Память после спада нагрузки высвобождается, что говорит о корректной работе приложения. Также следует отметить, что на все запросы сервис вернул корректный результат (создал тестируемого или вернул данные экзамена).

1.7. Достигнутые результаты

Итогом работы, описанной в данной главе, являются следующие результаты:

1. Изучены существующие средства интеграции;
2. Разработан сервис для интеграции **СТКПлюс** и ЕЦД (**Сервис интеграции**);
3. Проведено нагрузочное тестирование разработанного сервиса.

Глава 2. Обмен данными с классом

2.1. Обзор бизнес-процессов

В текущей реализации передача результатов тестирования в Центр для изготовления сертификата осуществляется следующим образом.

1. **Формирование пакета**

   По окончании дня оператор отправляет пакет данных об экзаменах. Создание пакета осуществляется в **Модуле управления DataEditor**. Оператор должен выбрать какие именно экзамены необходимо выгрузить. Сам пакет представляет собой CSV файл и содержит основную информацию о каждом экзамене: ФИО, паспортные данные, тип экзамена, общая оценка, результаты за каждый субтест.

2. **Подключение к удаленному рабочему столу в СПбГУ**

   Оператор подключается к заранее настроенной VPN сети СПбГУ и заходит на удаленный рабочий стол со своим персональным логином и паролем.

3. **Загрузка пакета в Центр**

   Загрузка CSV пакета осуществляется через модуль **Сертификаты для контрагентов**. Модуль загружает каждый экзамен в пакете в общую базу данных — БД Сертификаты.

4. **Проверка загруженных экзаменов**

   С помощью модуля **Сертификаты для сотрудников ЦЯТ** сотрудник Центра Языкового Тестирования регулярно проверяет загруженные экзамены на предмет корректности введенных данных. Если все в порядке, экзамен помечается как готовый к

печати. В противном случае сотрудник ЦЯТ связывается с контрагентом и уведомляет о найденных ошибках.

5. **Печать сертификата**

   В удобное время сотрудник ЦЯТ печатает сертификаты для новых экзаменов.

В рамках главы стоит задача избавить систему от ручной передачи данных из филиала в центр и организовать передачу обезличенных данных о тестировании в класс ЕЦД. Т. е. необходимо найти решение для синхронизации баз центра и филиала. При этом под базой Центра подразумевается **Центральная База Данных**, предложенная в предыдущей главе.

2.2. Типы данных и маршруты передачи

Можно выделить следующие сущности **СТКПлюс**, участвующие в обмене:

1. *Examinee*. Основная информация о тестируемом.
2. *Examen*. Общая информация об экзамене.
3. *Result*. Результаты тестируемого за субтест.
4. *Contract*. Информация о договоре.
5. *TestSessionQuestion.* Информация об ответе тестируемого на вопрос.

На Рис. 5 показана диаграмма с сущностями, указанными выше. Некоторые поля были опущены для краткости.

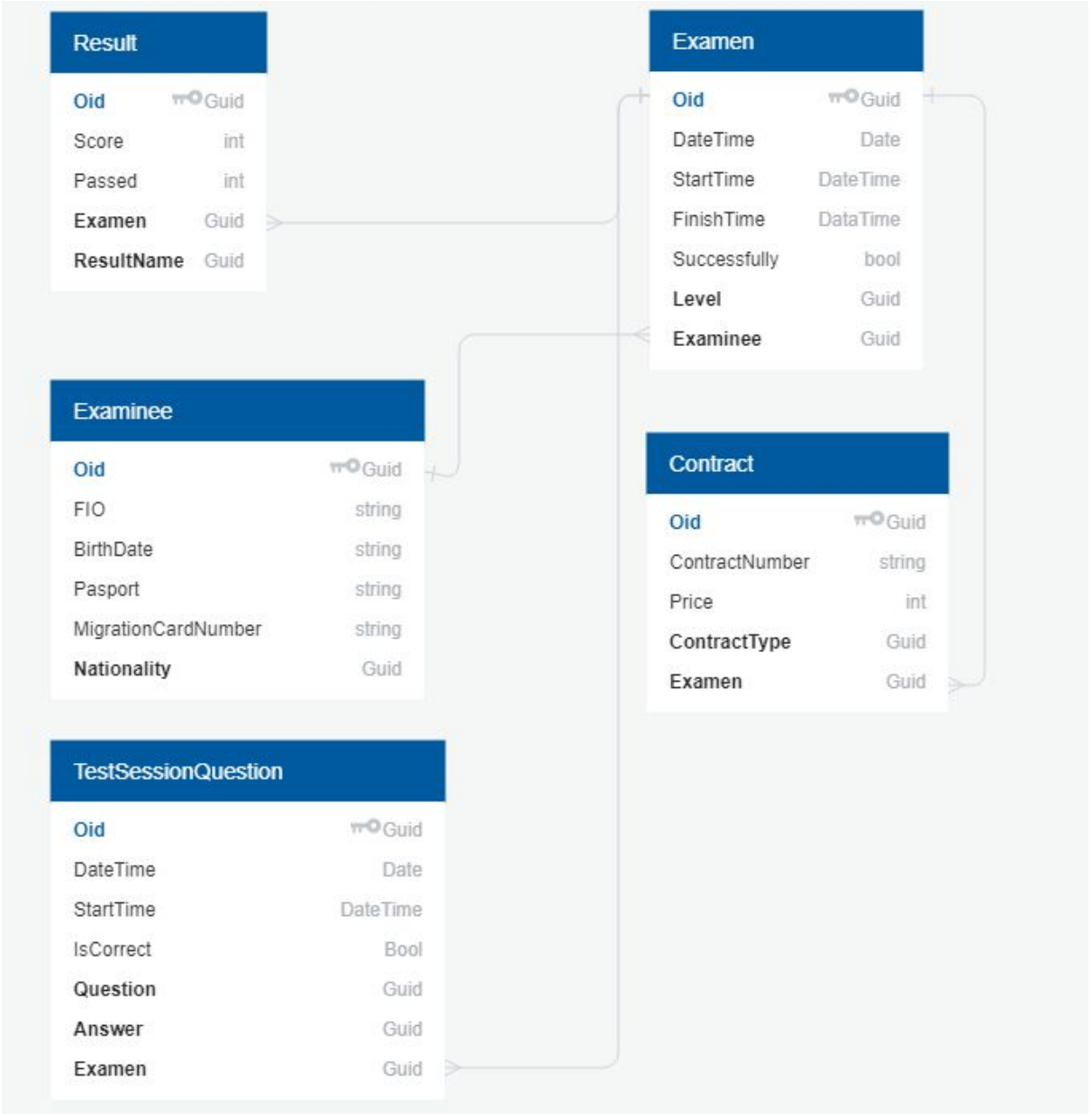

Рис. 5. Диаграмма сущностей, участвующих в обмене

В таблице 1 представлена сводная информация о направлениях передачи указанных сущностей.

	Филиал → Центр	Центр → Филиал
Examinee (Экзаменуемый)	✓	□
Examen (Экзамене)	✓	✓
Contract (Договор)	✓	✓
Result (Результат субтеста)	✓	□
TestSessionQuestion (Ответ)	✓	□

Табл. 1. Направления передачи

Как видно, в центр из филиала должны передаваться все сущности. Из центра в филиал только информация о предстоящем экзамене и договор.

При решении задачи автор опирался на следующие требования:

1. Центр должен сам контролировать очередь передачи данных с филиалов;
2. Данные, которые уже переданы, можно передать повторно;
3. Поддержка передачи аудио-файлов;

   В будущем сотрудники Центра языкового тестирования должны иметь возможность слушать аудио-ответы тестируемых с филиалов;
4. Устойчивость решения к непостоянному интернет-соединению;
5. Гибкая настройка времени синхронизации каждого филиала с центром.

2.3. Анализ инструментов

Ниже представлен обзор нескольких инструментов, которые можно применить для решения поставленной задачи.

Репликация Microsoft SQL Server. Репликация — это набор технологий копирования и распространения данных и объектов баз данных между базами данных, а также синхронизации баз данных для поддержания

согласованности [19]. Поскольку сейчас во всех филиалах и центре установлен Microsoft SQL Server 2014 Express, предлагается рассмотреть механизмы репликации, которые встроены в эту СУБД. В репликации часто используется терминология издательского дела: издатель и подписчик [20]. Издатель распространяет данные на подписчиков или забирает их с них.

В Microsoft SQL Server есть три типа репликации [21]:

1. **Репликация моментальных снимков**. В данном типе издатель передает подписчику целиком набор данных, участвующих в репликации. При этом старые данные перезаписываются.
2. **Репликация транзакций**. Вначале создается полная копия данных издателя на подписчике. Впоследствии подписчику передаются только измененные данные (используется журнал транзакций на издателе).
3. **Репликация слиянием**. Аналогично типу выше, вначале создается полная копия данных издателя на подписчике. Однако впоследствии не только изменения с издателя поступают на подписчика, но и наоборот.

Требованию двунаправленной синхронизации соответствует только последний тип репликации.

Важно заметить, что при настройке репликации старые данные подписчика удаляются.

XAF Delta. Модуль управления, используемый в филиале и центре, использует фреймворк XAF от компании DevExpress. Для XAF существует модуль XafDelta, представляющий собой систему асинхронной репликации [22]. В отличие от репликации SQL Server, XafDelta организует репликацию не на уровне базы данных, а на уровне объектной модели приложения. В этом случае протоколировать и воспроизводить необходимо не операции с

записями БД, а операции с объектами модели. С августа 2012 года модуль стал бесплатным.

В процессе репликации XafDelta оперирует так называемыми пакетами, содержащими изменения. Пакеты могут быть широковещательными — те, которые должны быть отправлены на все узлы, и целевыми — те, которые должны быть отправлены в определенный узел. Передача пакетов может проходить либо через общую базу данных, либо с использованием протоколов WCF, HTTP, HTTPS, FTP.

Еще одним важным понятием в XafDelta является ключ репликации — строковое значение, однозначно идентифицирующее объект в сети репликации. XafDelta не использует первичный ключ объекта, поскольку в разных базах они могут совпадать. Ключ используется для разрешения некоторых видов коллизий. Подробный пример содержится в официальной документации.

Особенностью XafDelta является то, что создание пакета инициируется самим пользователем через приложение.

Основным недостатком XafDelta является то, что его разработка прекращена и последняя поддерживаемая версия XAF — 11.1.9, в то время как в **СТКПлюс** используется версия 16.2.10. Поэтому официальный установщик выдает ошибку. Исходные коды XafDelta доступны на GitHub [23]. Однако скомпилировать их автору тоже не удалось, поскольку XafDelta вызывает методы XAF, которых либо уже нет в новых версиях, либо поменялась их сигнатура.

Microsoft Sync Framework. Фреймворк от Microsoft, представляющие собой решение для синхронизации данных из разных источников. Sync Framework помогает решить следующие распространенные проблемы в области синхронизации данных: интеграция различных типов данных,

обнаружение и разрешение конфликтов, работа в ненадежных сетях и другие [24]. Sync Framework поддерживает 3 типа узла для синхронизации:

1. Источники данных с поддержкой ADO.NET;
2. RSS- и Atom-каналы;
3. Файлы и папки.

Стандартный набор можно дополнить, написав собственный провайдер.

Остановимся подробнее на возможностях синхронизации баз данных. Для подключения к базам данных приложение должно создавать два провайдера синхронизации — по одному для конкретной базы данных. Подключение к базам данных происходит напрямую через строку подключения. Перед началом синхронизации необходимо совершить ее предварительную настройку (создать Scope в терминах Sync Framework). Scope описывает какие именно таблицы и столбцы должны участвовать в синхронизации [25]. Также можно настроить фильтрацию. Scope хранится непосредственно в базе данных и включает в себя:

1. Отслеживающую таблицу для каждой таблицы, участвующей в синхронизации. Такая таблица содержит временную метку последнего изменения, информацию о том какой именно участник сделал изменение;
2. Дополнительную запись в таблице scope_info (и саму таблицу, если ее нет). Запись содержит метаданные Scope;
3. Дополнительную запись в таблице scope_config (и саму таблицу, если ее нет). Запись содержит конфигурацию Scope в xml-формате;
4. Триггер на каждую таблицу, участвующую в синхронизации. Триггер необходим для актуализации данных в отслеживающей таблице;

Полный список содержится в официальной документации. В базе данных может быть множество Scope. За счет этого можно достичь гибкой синхронизации между базами.

При синхронизации папок с файлами, все метаданные содержатся в директории в специальном формате [26].

Последняя версия фреймворка вышла в 2010 году [27]. Новых провайдеров синхронизации для различных источников создано не было.

Отслеживание изменений в SQL Server. Технология, доступная начиная с SQL Server 2008, позволяющая отслеживать изменения в базе данных [28]. С помощью отслеживания изменений можно отследить, какая операция была произведена над строкой: вставка, изменение или удаление. Опционально можно также получить информацию и о том, какие столбцы были изменены. Данную технологию можно использовать при построении приложений для синхронизации баз.

Для того, чтобы отслеживание заработало, его необходимо включить [29]. Сделать это можно либо через DDL инструкцию T-SQL ALTER DATABASE, либо через утилиту SQL Server Management Studio. Возможно выбрать конкретные таблицы, для которых отслеживание изменений будет включено.

Сразу необходимо ввести понятие версии изменений. Как только отслеживание в базе включено, текущая версия изменений в базе данных равна 0. При каждой DML операции она будет увеличиваться на 1.

При помощи специальной функции Transact-SQL можно запросить, какие строки были изменены, и получить сведения об этих изменениях [30]. Можно выбрать изменения начиная с некоторой версии.

Изнутри технология работает благодаря созданию скрытой системной таблицы с именем [sys].[change_tracking_<table_id>] для каждой отслеживаемой таблицы [31].

XAF Audit Trail Module. Модуль для XAF, который обеспечивает отслеживание изменений в базе данных XAF-приложения [32]. Важно отметить, что отслеживаются только те изменения, которые были сделаны через приложение. При использовании модуля в базу приложения добавляются следующие таблицы:

1. AuditDataItemPersistent. Содержит список изменений всех объектов. При этом содержится не только сам факт изменения, но также старое и новое значение каждого поля;
2. XPWeakReference. Список объектов, которые изменились;
3. AuditedObjectWeakReference. Содержит информацию об объектах, которые изменились, и об объектах, которые приняли участие в этих изменениях.

Модуль можно настроить таким образом, чтобы отслеживать изменения только в некоторых таблицах [33].

XAF Middle Tier Security. XAF предоставляет инструмент XAF Middle Tier Security для создания защищенного WCF сервиса для доступа к данным приложения (при использовании XPO как ORM) [34]. С помощью сервиса можно получать защищенные данные с базы приложения, не имея прямого подключения. Это полезно с точки зрения безопасности (сервис требует ввода логина и пароля, сервис может фильтровать передаваемые данные) и с точки зрения организации сети (сервис и клиент могут быть даже не в локальной сети).

По сути, сервис представляет собой дополнительный слой между базой данных и приложением. При этом важно отметить, что ORM фреймворк, используемый в XAF — XPO — может подключаться к такому сервису так, как будто бы соединение установлено с базой напрямую.

XPO OData Service. Помимо WCF сервиса, XPO предоставляет возможность быстрого создания OData сервиса для доступа к данным

приложения [35]. OData (Open Data Protocol) — это стандарт построения RESTful API. OData позволяет выполнять запросы к данным, выполнять CRUD операции. OData избавляет программиста от необходимости писать множество одинаковых REST методов для доступа к данным.

Стандарт OData имеет 4 версии [36]. XPO поддерживает создание OData сервиса для третьей версии стандарта. Однако в отличие от XAF Middle Tier Security (WCF), XPO не может подключаться к OData как к источнику данных. Но Visual Studio предоставляет удобный инструмент для автоматического создания класса-клиента для доступа к службе [37].

Выводы. Автором был рассмотрен ряд инструментов, которые могут быть использованы при синхронизации данных (баз данных и файлов). В конечном итоге, из этого списка был выбран инструмент XPO OData Service для доступа к данным. Остальную часть, связанную с синхронизацией, было решено разработать самостоятельно.

Отказ от репликации и технологии отслеживания изменений в MS SQL Server SQL Server был обусловлен нежеланием привязывать решение к конкретной СУБД. И хотя сейчас в филиалах установлен MS SQL Server Express, в перспективе в филиалах возможен переход на более легковесные СУБД.

Microsoft Sync Framework создает множество дополнительной информации в базе данных. К тому же, фреймворк не обновлялся с 2010 года. Еще одним недостатком Microsoft Sync Framework в контексте решаемой задачи является необходимость прямого подключения к обеим базам (источник и приемник).

XAF Audit Module, как и встроенный инструмент в MS SQL Server, позволяет отслеживать изменения в таблицах. Использование этого модуля было бы оправдано, если данные могли бы как меняться, так и создаваться. Но в **СТКПлюс** все данные могут только создаваться без последующего

редактирования, что упрощает задачу и избавляет от необходимости использовать подобные решения. Более того, XAF Audit Module отслеживает изменения, сделанные только через XAF приложение (**Модуль управления**), а в системе **СТКПлюс** данные создаются и другими модулями.

Из двух прослоек к базе данных — XPO OData Service и XAF Middle Tier Security — была выбрана первая, как более привычная автору. OData можно легко настроить и для отдачи файлов (это будет полезно при передачи файлов с филиалов в центр).

2.4. Архитектура решения

Спроектированная архитектура состоит из двух основных компонент:

1. **Компонент доступа к данным.** Служба, установленная в филиале, представляющая собой OData сервис. Сервис используется для получения данных с филиала и загрузки новых данных в филиал.
2. **Модуль синхронизации**. Представляет собой службу, развернутую в центре, которая по расписанию проводит синхронизацию данных с филиалами. Для связи с филиалом приложение обращается к методам сервиса филиала. Для связи с Центром приложение напрямую обращается к базе данных Центра.

На Рис. 6 представлена обновленная схема системы.

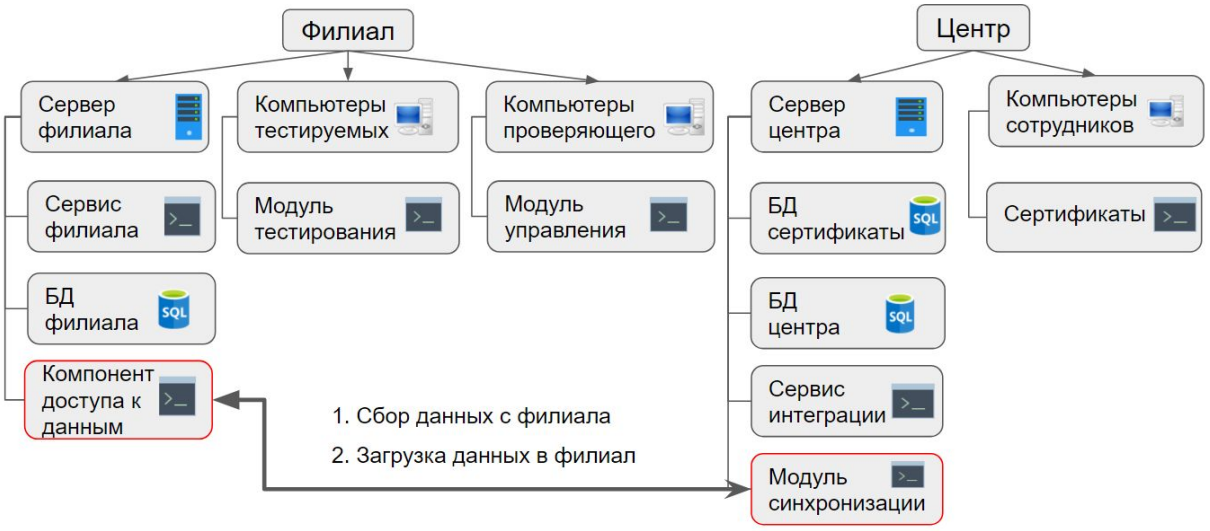

Рис. 6. Обновленная схема системы

Вся разработка велась на языке C# и платформе .NET. Для хранения кода и планирования задач был применен набор облачных инструментов Visual Studio Team Services (VSTS).

Для гибкой настройки времени синхронизации филиалов модуль синхронизации позволяет добавлять задачи по синхронизации отдельно для каждого филиала. Добавление задач происходит через **модуль управления** в центре.

Задачи делятся на два типа:

1. Сбор данных с филиала;
2. Загрузка данных в филиал.

Для каждой задачи можно указать периодичность ее выполнения в cron-формате (распространенный способ описания времени и периодичности действий [38]). Наличие двух типов задач позволяет выполнять сбор данных и их загрузку с разной периодичностью. Это полезно в случае синхронизации с ЕЦД, когда вносить данные о новых тестируемых нужно чаще, чем забирать.

Для планирования и выполнения задач использовался планировщик Hangfire. Помимо запуска задач с определенным интервалом выполнения, Hangfire предоставляет веб-интерфейс для отслеживания задач [39]. Стоит

отметить, что Hangfire — не единственный планировщик задач для платформы .NET. Как минимум, можно выделить библиотеку Quartz.NET [40] и Task Scheduler Managed Wrapper [41]. Выбор на Hangfire пал из-за наличия встроенного веб-интерфейса для наблюдения за задачами.

Сбор данных осуществляется пакетами. Пакет содержит экзамен и связанную информацию. Благодаря гибкому языку запросов OData пакет загружается одним запросом к филиалу.

Для того, чтобы помечать, какие экзамены уже были переданы в центр (класс), для сущности Examen в ORM слое было добавлено поле типа DateTime — *SyncDate* (может быть null - отсутствующее значение). Отсутствие значения в центре (филиале) сигнализирует о необходимости передачи экзамена и связанных сущностей в филиал (центр). Когда объект передан, значение SyncDate устанавливается равным текущей дате как на источнике, так и на приемнике. Следует заметить, что часто в распределенных системах используются более сложные механизмы упорядочивания событий и вычисления разницы данных для передачи. Например, логические часы [4].

На рисунке 7 показана диаграмма последовательности для сбора данных с филиала.

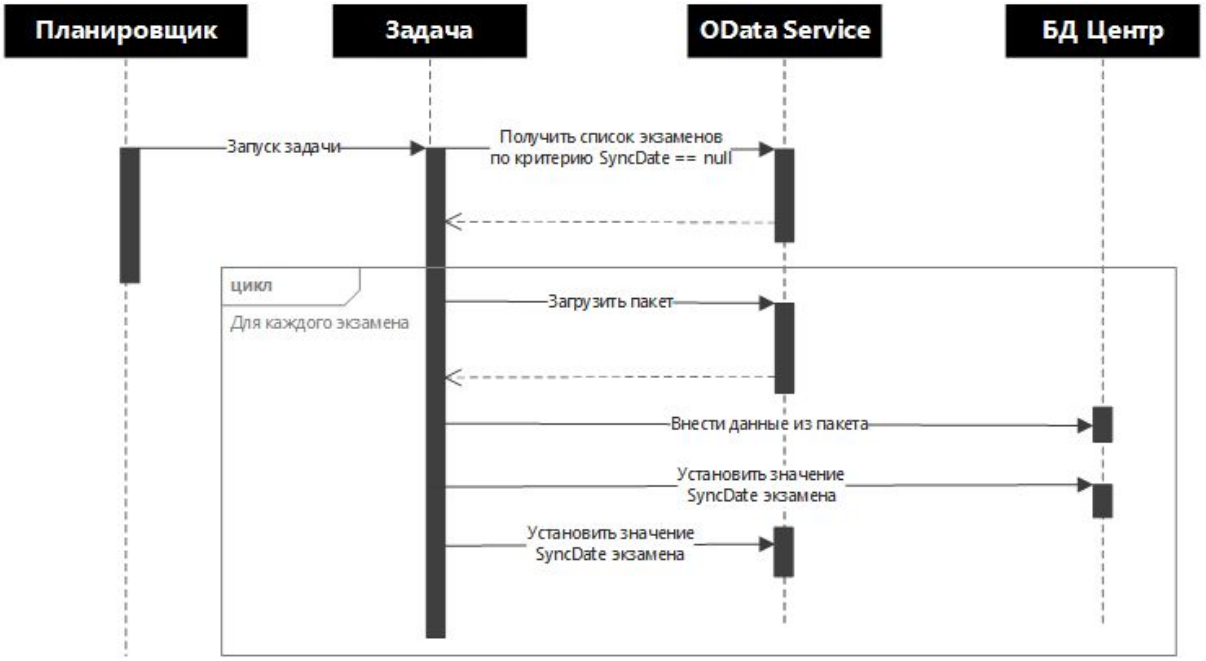

Рис. 7. Сбор данных с филиала

Как показано на диаграмме, пометка о том, что экзамен передан, устанавливается в самом конце цикла. Благодаря этому, в случае какой-либо ошибки при загрузке данных об экзамене в центр, при следующем запуске задачи будет произведена повторная попытка загрузки экзамена.

На рисунке 8 показана диаграмма последовательности для задачи по загрузке данных в филиал.

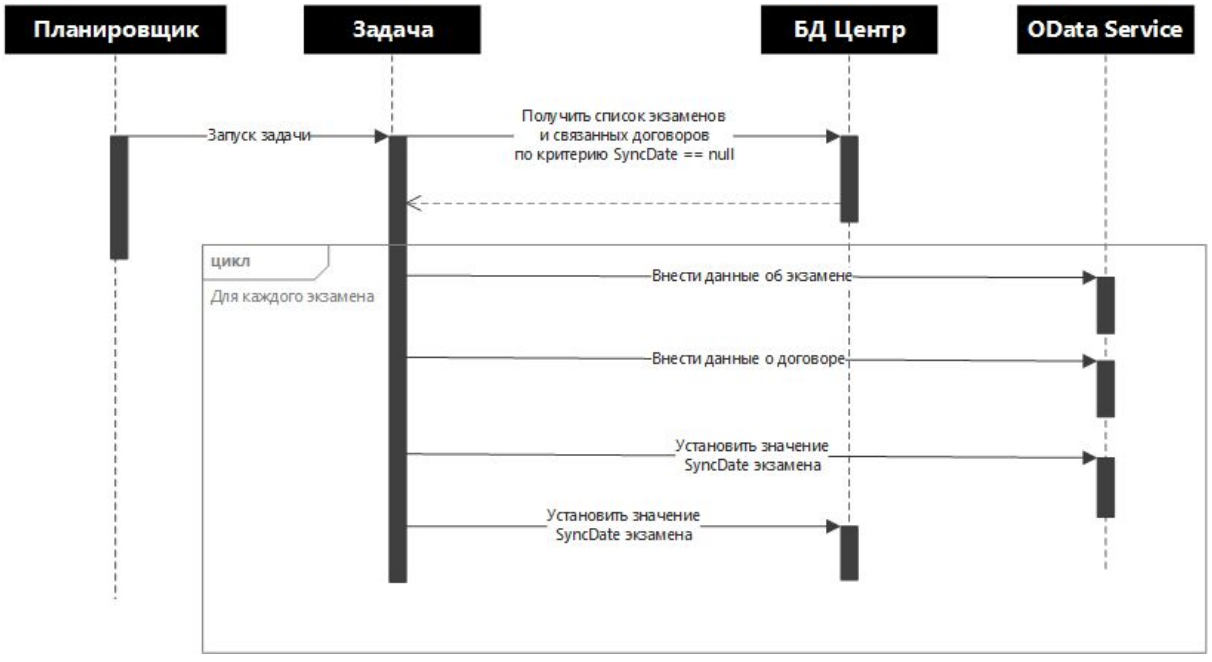

Рис. 8. Загрузка данных в филиал

Данная задача также подразумевает повторную попытку передачи данных в филиал в случае ошибки.

2.5. Тестирование

Автором было выполнено ручное интеграционное тестирование разработанных приложений. С помощью Continuous Delivery (подробнее описано в следующей главе) были развернуты **Модуль синхронизации** на одном компьютере (центр) и компонент доступа к данным на другом (филиал), а также вся необходимая инфраструктура. Далее было проведено тестирование с помощью **Модуля тестирования**. В соответствии с требованиями к системе, было проверено следующее: **Модуль управления** корректно добавляет задачу по синхронизации в очередь, задача работает с заданным интервалом, данные с филиала передаются в центр, повторная передача работает. Также была проверена работоспособность передачи данных в филиал, путем создания соответствующего экзамена в центре.

Дальнейшее улучшение тестирования состоит в автоматизации интеграционных тестов. Это позволит исключить возможность человеческой ошибки и повысить скорость тестирования.

2.6. Достигнутые результаты

Итогом работы, описанной в данной главе, являются следующие результаты:

1. Были проанализированы типы данных и маршруты их передачи;
2. Был рассмотрен ряд инструментов, которые могут быть использованы при синхронизации данных;
3. Разработано и протестировано решение для синхронизации центра и филиалов.

Глава 3. Применение Continuous Integration, Delivery, Deployment

3.1. Анализ инструментов

При разработке сервиса для интеграции с внешней информационной системой и сервиса для обмена данными с классом были применены некоторые из техник Continuous Integration (непрерывная интеграция), Continuous Delivery (непрерывная доставка), Continuous Deployment (непрерывное развертывание).

Применение Continuous Integration включает в себя регулярное слияние рабочих копий проекта в основную ветвь и использование отдельного сервера для выполнения сборок проекта. Continuous Integration позволяет находить ошибки в программном обеспечении как можно раньше [42].

Continuous Delivery является дополнением к Continuous Integration. После каждой успешной сборки проекта он автоматически развертывается в среде, приближенной к реальной.

CI/CD подразумевает использование большого количества автоматизированных тестов [42].

Существует также техника Continuous Deployment, которая заключается в том, что если Continuous Integration и Continuous Delivery прошли успешно и тесты пройдены, то приложение автоматически развертывается в рабочем окружении.

Далее автором были рассмотрены некоторые инструменты для Continuous Integration, Continuous Delivery, Continuous Deployment.

Jenkins. Jenkins — ПО для непрерывной интеграции с открытым исходным кодом, написанный на Java. Является ответвлением проекта Hudson, принадлежащего компании Oracle. Четких правил как и для чего его

использовать нет. Его можно применять для Continuous Integration, Continuous Delivery и Continuous Deployment . Возможны топологические конфигурации, когда используется, например, два отдельных сервера Jenkins, один для Continuous Integration, другой для Continuous Deployment.

Jenkins можно установить как на Unix/Linux, так и на Windows. После установки вся настройка проектов происходит через веб панель. Связь с другими компьютерами (на которых может проходить сборка или доставка) возможна 3 способами [43]:

1. Сервер подключается к клиентам UNIX по SSH (на ведомых машинах требуются только SSH и JRE);
2. Для Windows используется Distributed Component Object Model (DCOM);
3. Установка дополнительного клиента.

Для поддержки сборки проекта, размещенного в TFS, существует специальный плагин [44]. В целом, важной особенностью Jenkins является расширяемость. Существует множество плагинов, дополняющих стандартный функционал.

Службы CI/CD в Visual Studio Team Services. Visual Studio Team Services (VSTS) представляет ряд сервисов для настройки Continuous Integration и Continuous Delivery [45]. Вся настройка проектов происходит через web интерфейс http://visualstudio.com, поэтому установка дополнительного ПО в качестве средства управления не требуется. Сборка проектов может быть осуществлена как на серверах, предоставляемых Microsoft, так и на выделенном (агент в терминологии VSTS). Для настройки агента необходимо запустить специальный скрипт в PowerShell. Поддерживаются как Unix/Linux, так и Windows.

Список шагов для развертывания настраивается через веб-панель. Существуют готовые шаги для многих типичных действий, например для установки и запуска службы.

Характерной особенностью служб CI/CD в VSTS является плотная интеграция с другими инструментами и экосистемой Microsoft.

Octopus Deploy. Octopus Deploy предназначен для Continuous Deployment .NET приложений. Устанавливается в виде сервиса, предоставляющего веб-панель для управления.

На компьютеры, на которых будет проходить развертывание, устанавливается специальный агент.

Список шагов для развертывания настраивается через веб-панель. Существуют готовые шаги для многих типичных действий, например для установки и запуска службы.

Схема работы Octopus следующая:

1. Разработчик осуществляет изменение в репозитории;
2. Какой-то другой инструмент осуществляет CI/CD;
3. Если предыдущий шаг был успешен, приложение отправляется на сервер Octopus Deploy;
4. Приложение развертывается на определенные компьютеры в соответствии с настройкой Octopus.

Выводы. Jenkins и Службы CI/CD в Visual Studio Team Services можно использовать для решения сразу трех задач: Continuous Integration, Continuous Delivery, Continuous Deployment. В то время, как Octopus Deploy предназначен именно для Continuous Deployment. Характерной особенностью Jenkins является то, что это проект с открытым исходным кодом . К тому же к нему написано большое количество расширений. Службы CI/CD в Visual Studio Team Services и Octopus Deploy плотно интегрированы в экосистему

Microsoft и содержат большое количество встроенных шагов для этой экосистемы.

3.2. Настройка служб CI/CD в Visual Studio Team Services

Для Continuous Integration, Continuous Delivery, Continuous Deployment автором работы был выбран VSTS. Основная причина — Visual Studio Team Services уже используется для хранения кода.

Список действий для сборки проекта в терминологии VSTS называется «Определением сборки». Список действий для Continuous Delivery — «Определением релиза».

Автором была настроена автоматическая сборка для следующих компонент: **Сервиса филиала**, **Сервиса интеграции, Компонента доступа к данным, Модуля синхронизации**. «Определение сборки» содержит два основных шага:

1. Восстановление зависимостей в проекте;
2. Сборка проекта.

Также автором был настроен список шагов Continuous Delivery, среди которых основными являются следующие:

1. Остановка службы;
2. Обновление базы;
3. Копирование файлов службы новой версии;
4. Запуск службы.

Благодаря этому стало возможным:

1. Автоматическая проверка возможности сборки проекта;
2. Тестирование приложений без необходимости развертывать их вручную.

Для обновления базы используются инструменты SQL Server Data Tools.

Следует заметить, что, как уже было сказано, для полноценных Continuous Integration и Continuous Delivery необходимо также наличие большого количества автоматизированных тестов.

Сценарий Continuous Delivery был использован и для настройки развертывания приложений в Production (Continuous Deployment). На момент написания работы происходит апробация служб CI/CD в Visual Studio Team Services для Continuous Deployment в одном из филиалов.

3.3. Достигнутые результаты

В рамках данной главы было выполнено следующее:

1. Рассмотрены некоторые из инструментов Continuous Integration, Continuous Delivery, Continuous Deployment ;
2. Применены службы CI/CD Visual Studio Team Services.

Глава 4. Обнаружение голосовой активности в аудио-ответах

Аудио-ответы, записанные экзаменуемыми, помимо голоса зачастую содержат много тишины. Как правило, тишина содержится перед ответом (тестируемый читал вопрос и обдумывал ответ) и после него. Вследствие этого проверяющему приходится тратить время на прослушивание той части ответа, которая не содержит голоса. В филиалах с большим числом тестируемых это замедляет скорость проверки.

На Рис. 9 представлен пример формы сигнала реального аудио-ответа. Полезная информация (голос) занимает не больше 30 %.

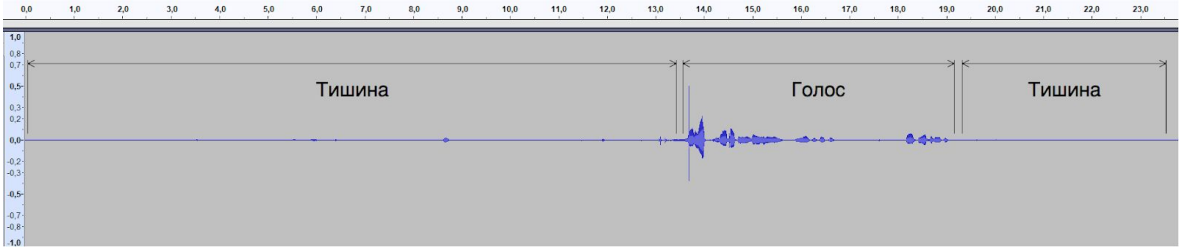

Рис 9. Пример формы аудио-ответа

Стоит заметить, что некоторые аудио-ответы могут вообще не содержать голоса — тестируемый не знал, что ответить, и молчал.

Поэтому еще одной задачей по адаптации системы тестирования является обнаружение голосовой активности в аудио-ответах. Предполагается, что **Модуль управления** должен отображать проверяющему метки, показывающие, где начинается и заканчивается аудио-ответ. Проверяющий по своему усмотрению может слушать ответ, начиная с метки.

Отображение начала и конца ответа — не единственное применение технологии обнаружения голосовой активности в системе тестирования. **СТКПлюс** устроен таким образом, что запись ведется не только в вопросах, требующих устный ответ и последующую проверку, но и во всех остальных.

При этом такие ответы, как правило, не должны содержать голос вообще. Если анализировать такие ответы системой обнаружения голосовой активности, можно обнаружить, были ли даны тестируемому подсказки голосом. Поэтому предполагается, что **Модуль управления** должен как-то отмечать такие ответы, чтобы сотрудники центра тестирования могли проверить, были ли подсказки.

Задача обнаружения голосовой активности в литературе часто называется Voice Activity Detection (VAD) или Speech Activity Detection (SAD). Традиционно VAD используется для уменьшения количества передаваемой информации между источником и приемником путем удаления речевых пауз, которые не несут полезной информации.

Основная проблема, с которой сталкивается любой VAD — наличие посторонних шумов в аудио. Отсюда вытекает одно из важных требований, предъявляемых к VAD алгоритму, — адаптивность к меняющемуся шуму [5].

Как правило, любой VAD алгоритм начинается с деления аудио-сигнала на короткие участки (фреймы). Затем для каждого фрейма VAD алгоритм должен определить, содержит ли он речь [5]. Как правило, длина фрейма выбирается в пределах от 5 до 40 мс. Фрейм с голосом далее называется активным, фрейм без голоса — неактивным.

4.1. Обзор подходов VAD

Ниже представлен обзор основных подходов к обнаружение голосовой активности. Данные подходы описаны во многих статьях. Автор работы опирался на [5], [6].

Линейный детектор на основе энергии. Данный подход основан на вычислении энергии каждого фрейма. Пусть x_i — i-ый сэмпл фрейма, n —

длина фрейма. Энергия фрейма определяется следующим образом: $E = \frac{1}{n}\sum_{i=1}^{n} x_i^2$.

Правило классификации определяется следующим образом: если выполняется равенство $E > kE_r$, то фрейм активный. k выступает в роли параметра — чувствительность алгоритма. E_r отражает энергию неактивных фреймов. Начальное значение E_r как правило вычисляется как среднее арифметическое энергий первых m фреймов. Первые фреймы в любом аудио, как правило, содержат тишину, т. е. являются неактивными.

E_r пересчитывается, как только найден неактивный фрейм, по формуле: $E_r = (1-p)E_{ro} + pE$, где E_{ro} — прошлое значение E_r, E — энергия текущего фрейма (он не содержит голос), $p = 0.2$. Таким образом, E_r пересчитывается как выпуклая комбинация E_{ro} и E. p выбирается из следующих соображений. Формула для вычисления E_r рассматривается как фильтр, где E_r — выход $y[n]$, E — вход $u[n]$. Далее ищется Z-преобразование фильтра: $Y(z) = (1-p)z^{-1}Y(z) + pU(z)$, где $Y(z)$ и $U(z)$ соответствуют Z-преобразованиям выхода и входа. Следовательно, передаточная функция $H(z) = \frac{Y(z)}{X(z)} = \frac{p}{1-(1-p)z^{-1}}$. Откуда, импульсная характеристика $h[n] = p(1-p)^n u[n]$. Время спада импульсной характеристики при различных p показано на Рис. 10 . Таким образом, при выборе $p = 0.2$ существенное влияние на E_r оказывает 15 последних фреймов, а короткие незначительные паузы между словами не оказывают существенного воздействия на E_r. В последующих алгоритмах, где используется параметр p, он выбирается из аналогичных соображений.

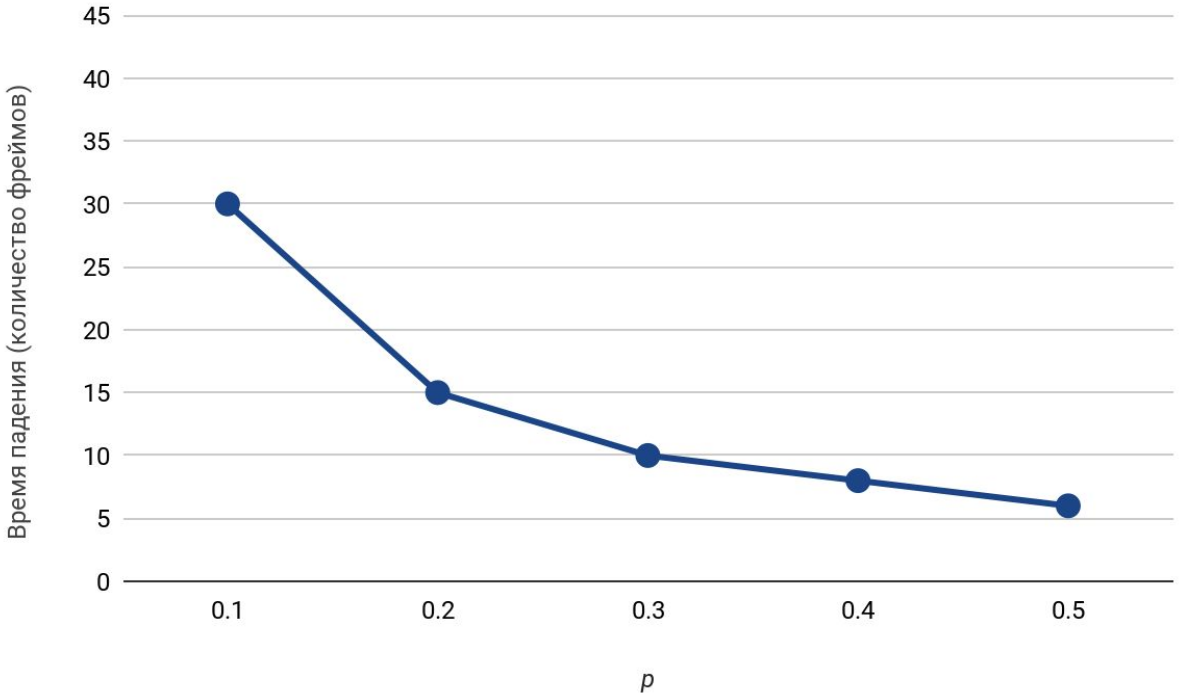

Рис. 10. Время падения импульсной характеристики при различных p

Недостатком данного алгоритма является слабая устойчивость к меняющемуся шуму.

Адаптивный линейный детектор на основе энергии. Является модификацией простого линейного детектора. Основное отличие заключается в том, что параметр p корректируется, когда найден фрейм без голоса. Для этого алгоритм подразумевает наличие буфера, состоящего из m элементов — энергий неактивных фреймов.

Корректировка p происходит следующим образом. Когда найден фрейм без голоса, вычисляется дисперсия буфера σ_{old}. Затем из буфера удаляется самый ранний элемент и добавляется новый (энергия фрейма без голоса). Далее считается дисперсия обновленного буфера σ_{new}. Значение p пересчитывается по таблице 2:

Условие	**Значение** p
$\sigma_{new}/\ \sigma_{old} \geq 1.25$	0.25
$1.10 \leq \sigma_{new}/\ \sigma_{old} < 1.25$	0.20
$1 \leq \sigma_{new}/\ \sigma_{old} < 1.10$	0.15
$\sigma_{new}/\ \sigma_{old} < 1$	0.10

Табл. 2. Пересчет p

Таким образом, алгоритм реагирует на изменяющийся шум.

Детектор на основе числа пересечений нуля. Классификатор подхода основан на том, что во фрейме длиной 10 мс сигнал пересекает ноль от 5 до 15 раз. Из этого факта вытекает правило классификации. Детектор пересечений ноля, как правило, используется вместе с одним из алгоритмов, описанных выше.

Линейный детектор на основе энергий полос. Данный подход является модификацией простого линейного детектора на основе энергии и иногда дает лучший результат. Спектр фрейма делится на несколько полос, например 0-1 kHz, 1-2 kHz, 2-3 kHz, 3-4 kHz. В каждой полосе считается энергия E_n, где n- номер полосы. Считается, что полоса активна, если $E_n > kE_{rn}$. E_{rn} отражает энергию n-ой полосы в неактивных фреймах. По аналогии с простым линейным детектором, E_{rn} пересчитывается, как только найден неактивный фрейм, по формуле: $E_{rn} = (1-p)E_{rno} + pE_n$, где E_{rno} — прошлое значение E_{rn}. Начальное значение E_{rn} вычисляется как среднее значений энергий полосы в первых m фреймах.

Правило для всего фрейма формулируется так: фрейм активен, если полоса, отвечающая самым низким частотам, активна, и любые два полосы из оставшихся активны.

Детектор на основе спектральной плоскостности. Спектральная плоскостность показывает, насколько спектр сигнала плоский. Т. е. насколько его график похож на прямую линию. Известно, что среди

распространенных сигналов, максимальную спектральную плоскостность имеет белый шум. На Рис. 11 показан спектр белого шума.

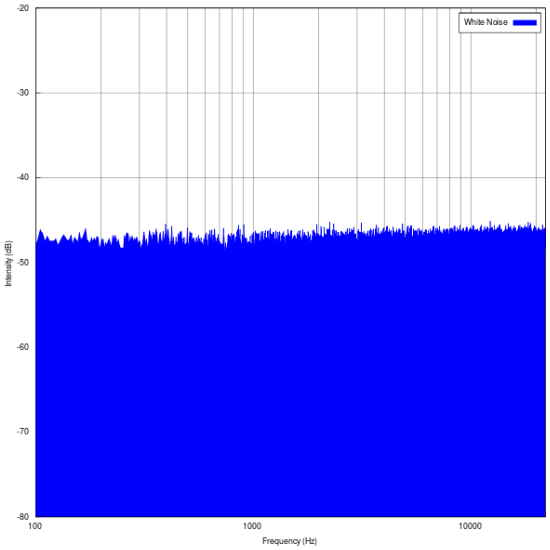

Рис. 11. Спектр белого шума

Поэтому высокая дисперсия спектра фрейма (σ) свидетельствует о том, что фрейм активен. Строгое правило выглядит так: фрейм активен, если $\sigma > (1-p)\sigma_{ro} + p\sigma$. Порог, который пересчитывается, как только найден неактивный фрейм. σ_{ro} - старое значение σ_r .

4.2. Комбинация подходов

На практике часто используют комбинацию подходов, описанных выше. Например, в статье [7] предлагается для каждого фрейма высчитывать три характеристики: энергию, спектральную плоскостность и номер полосы с наибольшим преобладанием энергии. Для каждой характеристики задается свой начальный порог на основе анализа первых фреймов. Если для фрейма значение двух характеристик превысило пороговое значение, то фрейм помечается как активный, и пересчитывается порог.

4.3. Описание выбранного подхода

Было принято решение использовать комбинацию из трех подходов: адаптивный детектор на основе энергии, детектор на основе пересечений нуля и детектор на основе спектральной плоскостности.

Длина фрейма выбрана равной 10 мс. Начальное значение E_r вычисляется как среднее арифметическое энергий первых 100 фреймов.

Сначала решение о принадлежности фрейма выносится адаптивным детектором на основе энергии. Если фрейм признан активным, алгоритм переходит к следующему фрейму. Если нет, то фрейм анализируется двумя оставшимися подходами. Если оба подхода классифицируют фрейм как активный, то фрейм помечается как активный.

Наличие в подходе классификаторов, основанных не на энергии, позволяет отличать активные фреймы от неактивных даже в условиях малого отношения полезного сигнала к шуму.

Менее 20 подряд идущих сэмплов с голосом игнорируются. Как правило, это случайные стуки в микрофон.

На рисунке 12 показана блок-схема VAD алгоритма. Размер фрейма выбран равным 10 мс.

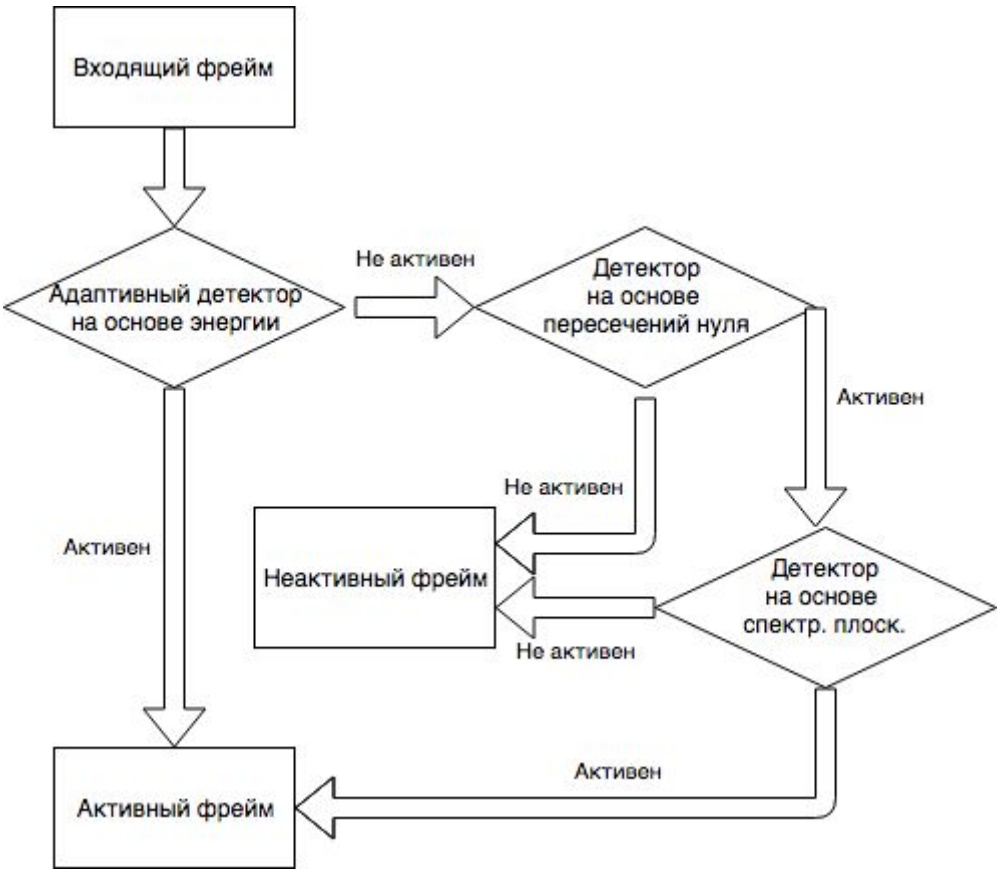

Рис. 12. Используемый VAD алгоритм

В конце работы алгоритм возвращает номер первого сэмпла первого активного фрейма — начало голосовой активности, и номер последнего сэмпла последнего активного фрейма — конец голосовой активности. Если алгоритм вернул пару нулей, значит голос не был найден.

Алгоритм был реализован в среде Matlab. Исходный код прототипа доступен на GitHub по адресу https://github.com/cantti/ExaminatorVAD.

4.4. Тестирование

Для тестирования алгоритма с сервера одного из филиалов было загружено 100 аудио-ответов, соответствующих вопросу «Куда вы поедете летом?». Далее для каждого аудио-ответа было вручную отмечено начало и конец голосовой активности. Алгоритм был применен к каждому файлу. Тестирование проводилось при различных значениях параметра k, используемого в адаптивном детекторе на основе энергии. По результатам

тестирования можно выбрать оптимальное значение параметра. Допустимая ошибка — 1 секунда. Использованы следующие метрики:

1. Отношение числа верно размеченных файлов к общему числу файлов в процентах (R1);
2. Отношение числа тех файлов, в которых алгоритм удалил голос, к общему числу файлов в процентах (R2);
3. Отношение числа тех файлов, в которых алгоритм оставил шум, к общему числу файлов в процентах (R3).

Подобные характеристики используются в упомянутых ранее статьях [5,7]. Очевидно, что чем выше значение R1, тем алгоритм работает лучше. Значения R2 и R3 стремятся минимизировать. Однако, как правило, уменьшение метрики R3 ведет к увеличению R2. В случае системы тестирования в приоритете уменьшение R2, пусть и при увеличении R3. Т. е. лучше оставить немного шума, чем удалить голос. При этом, как уже было сказано, предполагается, что в конечном решении голос не удаляется, а происходит только разметка аудио-волны, отображаемой проверяющему. Поэтому проверяющий всегда может прослушать любую часть исходного аудио-ответа.

Результаты тестирования представлены на Рис. 13.

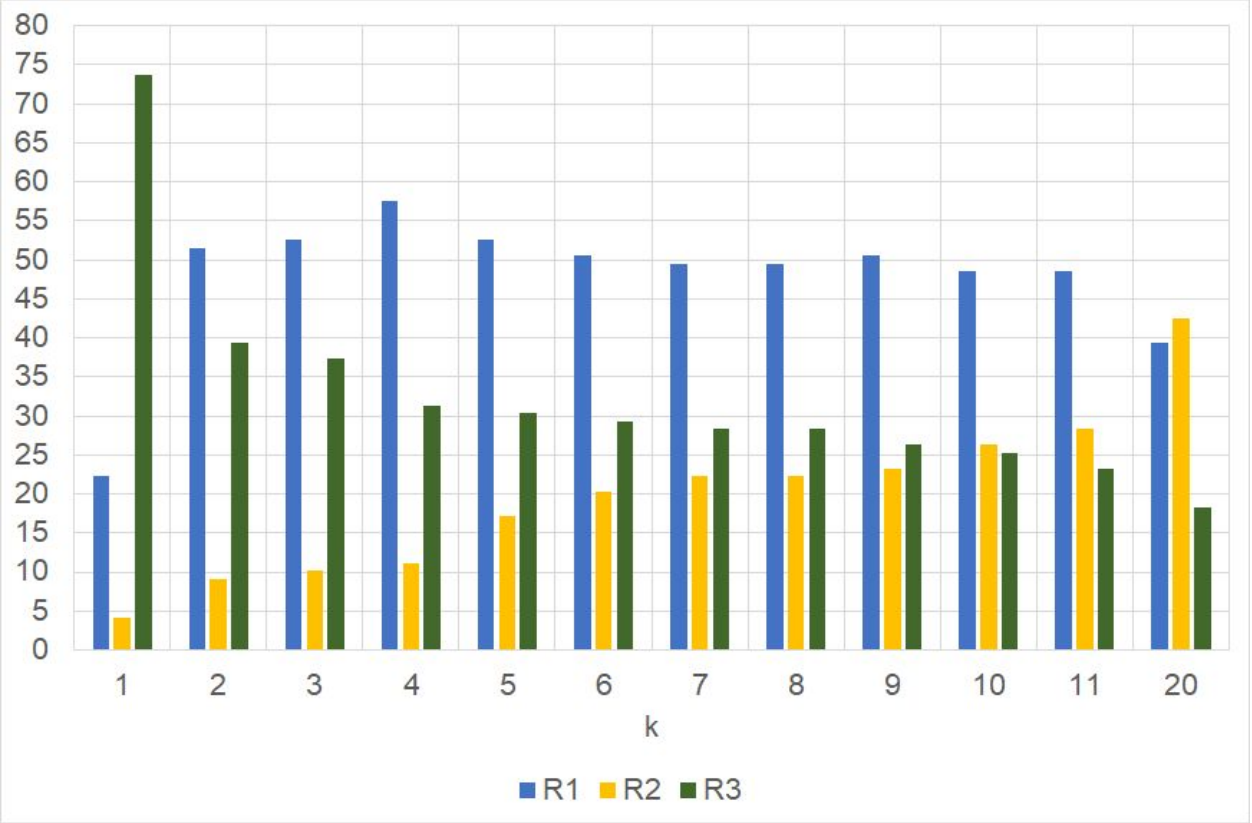

Рис. 13. Результаты тестирования при различных значениях k.

Максимальное значение R1 достигается при $k = 4$. При этом значение R2 на достаточно низком уровне (около 10 %). При $k = 5$ значение R2 уже равно 17 %. Очевидно, что при очень больших k линейный детектор на основе энергии перестает работать, и алгоритм работает только за счет двух других детекторов.

Стоит отметить, что во всех аудио-ответах, где была только тишина (шум), алгоритм верно выдал результат.

4.5. Достигнутые результаты

Итогом работы, описанной в главе, стало следующее:

1. Проанализировано, как могут быть применены техники VAD в системе **СТКПлюс**;
2. Проанализирован ряд подходов VAD;
3. Разработан прототип компонента детектирования голоса в среде Matlab;
4. Подготовлены тестовые данные на основе реальных ответов тестируемых;
5. Протестирован прототип компонента.

Полученные результаты могут быть использованы при разработке конечного решения по разметке аудио-ответов и обнаружению нежелательной голосовой активности.

Заключение

В рамках данной работы был совершен ряд улучшений системы тестирования **СТКПлюс**, благодаря которому она может работать в большем количестве филиалов с минимальным количеством ручного труда.

Для этого были выполнены задачи по интеграции **СТКПлюс** с внешней информационной системой ЕЦД, среди которых: анализ существующих средств интеграции ПО, разработка и тестирование модулей интеграции.

Также был разработан и протестирован модуль для синхронизации данных между филиалом и центром. Его наличие избавляет сотрудников от необходимости ручной передачи данных. Предварительно были рассмотрены инструменты, применяемые при синхронизации данных.

Далее были применены техники Continuous Integration, Continuous Delivery, Continuous Deployment для некоторых модулей системы. Было рассмотрено несколько инструментов из области. Благодаря данным техникам стали возможными более быстрый выпуск новых версий и более раннее устранение ошибок. Также это избавило от необходимости ручного развертывания модулей в филиалах.

Также был разработан прототип компонента, устанавливающего метки начала и конца ответа в аудио-волне. Предварительно были изучены существующие подходы к детектированию аудио. В дальнейшем подобный компонент позволит проверяющим меньше тратить время на проверку аудио-ответов и выявлять нарушения при тестировании.

Дальнейшее развитие системы тестирования может производиться в следующих направлениях:

1. Создание автоматизированных тестов всех типов для всех компонентов системы;

2. Доработка и внедрение компонента детектирования голоса (необходимо также проанализировать все его возможные применения);
3. Расширение перечня синхронизируемых данных между филиалом и центром. В этот перечень должны входить, как минимум, тесты и файлы аудио-ответов.

Список литературы

1. Integration architecture: Comparing web APIs with service-oriented architecture and enterprise application integration [Электронный ресурс]. URL: https://www.ibm.com/developerworks/websphere/library/techarticles/1503_clark/1305_clark.html (дата обращения: 16.05.2018).
2. Richardson L., Amundsen M., Ruby S. RESTful Web APIs: Services for a Changing World. «O'Reilly Media, Inc.», 2013. С. 29.
3. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, 2012. 735 с.
4. Косяков М.С. Введение в распределенные вычисления. НИУ ИТМО, 2014. С. 65.
5. Venkatesha Prasad R. и др. Comparison of voice activity detection algorithms for VoIP // Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications.
6. Md. Sahidullah G.S. Comparison of Speech Activity Detection Techniques for Speaker Recognition // Cornell University Library. 2012.
7. Moattar M.H., Homayounpour M.M. A simple but efficient real-time voice activity detection algorithm. 2010.
8. Постановление от 1 ноября 2012 г. N 1119 об утверждении требований к защите персональных данных при их обработке в информационных системах персональных данных [Электронный ресурс]. URL: http://pravo.gov.ru/proxy/ips/?docbody=&nd=102160483 (дата обращения: 16.05.2018).
9. What is Application Integration: Definition | Informatica US [Электронный ресурс]. URL: https://www.informatica.com/services-and-training/glossary-of-terms/applicati

on-integration-definition.html (дата обращения: 16.05.2018).

10. Integration architecture: Comparing web APIs with service-oriented architecture and enterprise application integration [Электронный ресурс]. URL: https://www.ibm.com/developerworks/websphere/library/techarticles/1503_clark/1305_clark.html (дата обращения: 16.05.2018).
11. Ruh W.A., Maginnis F.X., Brown W.J. Enterprise Application Integration: A Wiley Tech Brief. John Wiley & Sons, 2002. C. 224.
12. Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST) [Электронный ресурс]. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата обращения: 16.05.2018).
13. What is a Web API? Learn about Web APIs from Reich Web Consulting [Электронный ресурс] // Reich Web Consulting. URL: https://www.reich-consulting.net/services/api-development/what-is-a-web-api/ (дата обращения: 16.05.2018).
14. WCF and ASP.NET Web API [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/dotnet/framework/wcf/wcf-and-aspnet-web-api#choosing-which-technology-to-use (дата обращения: 16.05.2018).
15. Christos S. ASP.NET MVC Solution Architecture – Best Practices [Электронный ресурс] // chsakell’s Blog. 2015. URL: https://chsakell.com/2015/02/15/asp-net-mvc-solution-architecture-best-practices/ (дата обращения: 16.05.2018).
16. MikeWasson. Create Data Transfer Objects (DTOs) [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5 (дата обращения: 16.05.2018).
17. Apex Enterprise Patterns - Service Layer - developer.force.com [Электронный

ресурс]. URL: https://developer.salesforce.com/page/Apex_Enterprise_Patterns_-_Service_Layer (дата обращения: 16.05.2018).
18. Bulat A. Про Тестинг - Тестирование - Виды Тестирования ПО - Виды нагрузочных тестов и тестирования производительности [Электронный ресурс]. URL: http://www.protesting.ru/testing/types/loadtesttypes.html (дата обращения: 16.05.2018).
19. MashaMSFT. Репликация SQL Server [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/relational-databases/replication/sql-server-replication?view=sql-server-2017 (дата обращения: 16.05.2018).
20. MashaMSFT. Обзор модели публикации репликации [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/relational-databases/replication/publish/replication-publishing-model-overview?view=sql-server-2017 (дата обращения: 16.05.2018).
21. MashaMSFT. Типы репликации [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/relational-databases/replication/types-of-replication?view=sql-server-2017 (дата обращения: 16.05.2018).
22. XafDelta documentation [Электронный ресурс]. URL: http://xafdelta.narod.ru (дата обращения: 16.05.2018).
23. xafdelta. xafdelta/xafdelta [Электронный ресурс] // GitHub. URL: https://github.com/xafdelta/xafdelta (дата обращения: 23.05.2018).
24. Преимущества использования Sync Framework [Электронный ресурс]. URL: https://msdn.microsoft.com/ru-ru/library/dd918617(v=sql.110).aspx (дата обращения: 16.05.2018).
25. How to: Use Synchronization Scopes [Электронный ресурс]. URL: https://msdn.microsoft.com/en-us/library/hh882039(v=sql.110).aspx (дата обращения: 16.05.2018).

26. Канжилал Д. Создание провайдеров синхронизации с помощью Sync Framework [Электронный ресурс]. URL: http://www.oszone.net/13887/Sync-Framework (дата обращения: 16.05.2018).
27. Пакет SDK для платформы Microsoft Sync Framework [Электронный ресурс] // Microsoft Download Center. URL: https://www.microsoft.com/ru-ru/download/details.aspx?id=23217 (дата обращения: 16.05.2018).
28. rothja. Об отслеживании изменений (SQL Server) [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/relational-databases/track-changes/about-change-tracking-sql-server?view=sql-server-2017 (дата обращения: 16.05.2018).
29. rothja. Включение и отключение отслеживания изменений (SQL Server) [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/relational-databases/track-changes/enable-and-disable-change-tracking-sql-server?view=sql-server-2017 (дата обращения: 16.05.2018).
30. rothja. Функции (Transact-SQL) отслеживания изменений [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/relational-databases/system-functions/change-tracking-functions-transact-sql?view=sql-server-2017 (дата обращения: 16.05.2018).
31. Olontsev S. Начинаем работу с технологией Change Tracking | Олонцев Сергей [Электронный ресурс]. URL: https://olontsev.ru/2012/10/introducing-change-tracking/ (дата обращения: 16.05.2018).
32. Audit Trail Module Overview | eXpressApp Framework (XAF) | eXpressApp

Framework | DevExpress Help [Электронный ресурс]. URL: https://documentation.devexpress.com/eXpressAppFramework/112782/Concepts/Extra-Modules/Audit-Trail/Audit-Trail-Module-Overview (дата обращения: 16.05.2018).

33. Customize the Audit Trail System | eXpressApp Framework (XAF) | eXpressApp Framework | DevExpress Help [Электронный ресурс]. URL: https://documentation.devexpress.com/eXpressAppFramework/112783/Concepts/Extra-Modules/Audit-Trail/Customize-the-Audit-Trail-System (дата обращения: 16.05.2018).
34. Middle Tier Security - WCF Service | eXpressApp Framework (XAF) | eXpressApp Framework | DevExpress Help [Электронный ресурс]. URL: https://documentation.devexpress.com/eXpressAppFramework/113439/Concepts/Security-System/Middle-Tier-Security-WCF-Service (дата обращения: 16.05.2018).
35. OData Service Wizard | eXpress Persistent Objects (XPO) | Cross-Platform Core Libraries | DevExpress Help [Электронный ресурс]. URL: https://documentation.devexpress.com/CoreLibraries/14812/DevExpress-ORM-Tool/Design-Time-Features/OData-Service-Wizard (дата обращения: 16.05.2018).
36. Documentation · OData - the Best Way to REST [Электронный ресурс]. URL: http://www.odata.org/documentation/ (дата обращения: 16.05.2018).
37. MikeWasson. Вызов службы OData из клиента .NET (C#) [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/aspnet/web-api/overview/odata-support-in-aspnet-web-api/odata-v3/calling-an-odata-service-from-a-net-client (дата обращения: 16.05.2018).
38. Cron-формат [Электронный ресурс]. URL: http://www.nncron.ru/nncronlt/help/RU/working/cron-format.htm (дата

обращения: 16.05.2018).

39. Hangfire – Background Jobs for .NET and .NET Core [Электронный ресурс]. URL: http://hangfire.io (дата обращения: 16.05.2018).
40. Lahma M. Quartz Enterprise Scheduler .NET | Quartz.NET Documentation [Электронный ресурс]. URL: https://www.quartz-scheduler.net (дата обращения: 16.05.2018).
41. dahall. dahall/TaskScheduler [Электронный ресурс] // GitHub. URL: https://github.com/dahall/TaskScheduler (дата обращения: 16.05.2018).
42. Ellingwood J. An Introduction to Continuous Integration, Delivery, and Deployment | DigitalOcean [Электронный ресурс] // DigitalOcean. DigitalOcean, 2017. URL: https://www.digitalocean.com/community/tutorials/an-introduction-to-continuous-integration-delivery-and-deployment (дата обращения: 17.05.2018).
43. Distributed builds - Jenkins - Jenkins Wiki [Электронный ресурс]. URL: https://wiki.jenkins.io/display/JENKINS/Distributed+builds (дата обращения: 17.05.2018).
44. jenkinsci. jenkinsci/tfs-plugin [Электронный ресурс] // GitHub. URL: https://github.com/jenkinsci/tfs-plugin (дата обращения: 17.05.2018).
45. andyjlewis. Overview of Build and Release [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/vsts/build-release/overview?view=vsts (дата обращения: 17.05.2018).