

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФГБОУ ВО «БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

Кафедра «Информатика и программное обеспечение»

«У Т В Е Р Ж Д А Ю»

Зав. кафедрой «И и ПО», к.т.н., доцент

_____ Подвесовский А.Г.

« ____ » _____ 2017 г.

ПРОЕКТИРОВАНИЕ, РЕАЛИЗАЦИЯ И РАЗВЕРТЫВАНИЕ
ИНТЕГРАЦИОННОЙ ПЛАТФОРМЫ И БАЗОВЫХ СЕРВИСОВ
КОМПЛЕКСНОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ КАФЕДРЫ
«ИНФОРМАТИКА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

М А Г И С Т Е Р С К А Я Д И С С Е Р Т А Ц И Я

Всего ____ листов

Научный руководитель

_____ к.т.н., доц. Лагереv Д.Г.

« ____ » _____ 2017 г.

Нормоконтролер

_____ к.т.н., доц. Лагереv Д.Г.

« ____ » _____ 2017 г.

Магистрант

_____ Васин А.В.

« ____ » _____ 2017 г.

БРЯНСК 2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФГБОУ ВО «БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»
КАФЕДРА «ИНФОРМАТИКА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

Направление 09.04.04 – Программная инженерия
Магистерская программа
«Проектирование программно-информационных систем»

«У Т В Е Р Ж Д А Ю»

Зав. кафедрой «И и ПО», к.т.н., доцент

_____ Подвесовский А.Г.
« ____ » _____ 2017 г.

З А Д А Н И Е
на выполнение магистерской диссертации

магистранту группы 15ПРИ-мг

Васину Антону Вадимовичу

1. Тема работы Проектирование, реализация и развертывание интеграционной платформы и базовых сервисов комплексной информационной системы кафедры «Информатика и программное обеспечение»

Утверждена приказом по БГТУ № 408-3 от 15 мая 2017 г.

2. Срок предоставления законченной диссертации июнь 2017 г.

3. Исходные данные

3.1. Цель работы Проектирование и разработка программной платформы информационной системы кафедры.

3.2. Методы исследования Для достижения поставленной цели в процессе решения обозначенных задач использовались методы системного анализа, метод анализа иерархий, метод мозгового штурма, методы проектирования сервис-ориентированных систем, методы проектирования RESTful API.

3.3. Инструментальные средства Microsoft Visio, Microsoft Project, Star UML, OpenLDAP, PostgreSQL, Node.js, Visual Studio Code, pgAdmin 4, LdapAdmin, Putty, Debian 8, WSO2 Enterprise Service Bus, Atlassian Jira, Atlassian Confluence, Atlassian SourceTree, GitLab, Eclipse.

4. Имеющийся задел Установленное и настроенное программное обеспечение Debian 8, OpenLDAP, PostgreSQL, WSO2 Enterprise Service Bus. Разработаны сервисы авторизации, аутентификации и учёта контингента.

5. Перечень вопросов, подлежащих разработке _____
Исследование путей автоматизации деятельности выпускающей кафедры информационного профиля, формулирование основных требований к платформе. Изучение современных архитектур программного обеспечения, выбор архитектуры платформы, формулирование технического задания. Проектирование и разработка сервисов платформы, подключение их к сервисной шине предприятия. Интеграция Atlassian Jira, Atlassian Confluence и GitLab в платформу информационной системы кафедры.

Дата выдачи задания: 10 мая 2017 г.

Научный руководитель, к.т.н., доцент _____ Лагереv Д.Г.

Задание принял к исполнению _____ Васин А.В.

АННОТАЦИЯ

Объектом исследования является информационное обеспечение учебной, учебно-методической и организационно-методической деятельности выпускающей кафедры.

Предметом исследования являются средства программной поддержки указанных видов деятельности.

Целью данной работы является проектирование и разработка программной платформы информационной системы кафедры, то есть ее фундамента. Главная особенность целевого решения – это наличие сервисов, источников данных, политик безопасности и правил для встраивания сторонних решений, предназначенных для автоматизации прикладных задач.

Теоретическая значимость работы заключается в исследовании современных технологий разработки информационных систем в контексте автоматизации информационных процессов учебного заведения, подготавливающего IT-специалистов.

Практическая значимость работы заключается в создании программной платформы, предназначенной для использования студентами, магистрантами и аспирантами в рамках учебных проектов по автоматизации кафедральных бизнес-процессов.

По теме магистерской диссертации опубликовано 5 работ, ещё одна в настоящий момент находится в печати. Также было принято участие в региональном этапе программы «У.М.Н.И.К.».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1. ПУТИ КОМПЛЕКСНОЙ АВТОМАТИЗАЦИИ ДЕЯТЕЛЬНОСТИ ВЫПУСКАЮЩЕЙ КАФЕДРЫ ИНФОРМАЦИОННОГО ПРОФИЛЯ.....	8
1.1. Предварительный системный анализ.....	8
1.1.1. Потребность в автоматизации	9
1.1.2. Цели проекта	9
1.1.3. Потребности и цели сотрудников	10
1.1.4. Потребности и цели других заинтересованных сторон.....	13
1.2. Общие требования к информационной системе.....	13
1.3. Выводы.....	14
2. ИССЛЕДОВАНИЕ АРХИТЕКТУР	16
2.1. Анализ архитектур	16
2.1.1. Выбор альтернатив	16
2.1.2. Выбор критериев.....	17
2.1.3. Коллективная оценка критериев	18
2.1.4. Модель принятия решения	20
2.2. Техническое задание.....	24
2.2.1. Основания для разработки.....	24
2.2.2. Назначение разработки	24
2.2.3. Общие требования	24
2.2.4. Функциональные требования	25
2.2.5. Требования к архитектуре	26
2.2.6. Требование к аппаратному и программному обеспечению	26
2.3. Выводы по главе.....	26
3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПЛАТФОРМЫ	28
3.1. Общий вид архитектуры.....	29
3.2. Сервисная шина предприятия.....	31
3.3. Сервис контингента кафедры	32

3.3.1. Служба каталогов OpenLDAP	33
3.3.2. СУБД и ORM.....	34
3.3.3. Нереляционная модель данных	35
3.3.4. Реляционная модель данных	36
3.3.5. Интерфейс RESTful API.....	37
3.3.6. Hypermedia Application Language.....	38
3.3.7. Системные события и интеграция с ESB	46
3.4. Общая концепция безопасности в платформе	48
3.5. Микросервис аутентификации	50
3.5.1. Форма аутентификации.....	51
3.5.2. REST API микросервиса аутентификации	51
3.6. Сервис авторизации	52
3.6.1. Attribute Based Access Control.....	52
3.6.2. Общий вид политик безопасности.....	54
3.6.3. Политика создания персон.....	54
3.6.4. Политика доступа на чтение персональных данных	55
3.6.5. Политика доступа на модификацию персональных данных.....	56
3.6.6. Политика редактирования учебных групп.....	57
3.7. Совместная интеграция трёх служб	58
3.8. Интеграция платформы с внешними системами	59
3.8.1. Atlassian Jira.....	60
3.8.2. Atlassian Confluence	61
3.8.3. GitLab	62
3.9. Правила интеграции разработанных служб	63
3.10. Планы развития системы.....	65
3.11. Выводы по главе.....	65
4. УПРАВЛЕНИЕ ПРОЕКТОМ	66
4.1. Комбинированная матрица ответственности.....	66
4.2. Календарный план работ.....	66
4.3. Управление рисками проекта	69

4.3.1. Идентификация рисков	69
4.3.2. Определение вероятности и степени воздействия рисков	69
ЗАКЛЮЧЕНИЕ	72
СПИСОК ЛИТЕРАТУРЫ	74

ВВЕДЕНИЕ

Кафедра «Информатика и программное обеспечение» является одной из крупнейших выпускающих кафедр в Брянском государственном техническом университете. Кафедра исполняет широкий круг обязанностей, поэтому она испытывает потребность в автоматизации своей деятельности современными программными средствами.

К сожалению, для нужд кафедры не существуют идеально подходящих программных продуктов: готовые решения не учитывают всех особенности предметной области, а заказные обладают высокой стоимостью. Руководство кафедры, понимая это, приняло решение разработать информационную систему собственными силами.

В результате были разработаны мобильные и веб-приложения, которые позволяют учитывать посещаемость и сдачу работ студентов. Результаты показали себя с положительной стороны: студенты, разрабатывавшие эти веб-приложения, приобрели опыт участия в разработке и поддержке по-настоящему востребованных систем, а кафедра получила необходимое ей программное обеспечение.

Стратегия автоматизации собственными силами, как и планировалось, оказалась выгодной для обеих сторон. Однако, для планомерной автоматизации бизнес-процессов кафедры, недостаточно просто иметь набор независимых приложений. Эти приложения должна объединять единая инфраструктура, т.е. общие источники данных, правила безопасности и выделенные вычислительные ресурсы, иначе говоря, должна существовать программная платформа, которая позволит новым приложениям легко встраиваться и использовать общие для всей системы ресурсы. Платформа и базирующиеся на ней прикладные приложения позволят создать полноценную информационную систему, автоматизирующую бизнес-процессы кафедры.

Объектом исследования является информационное обеспечение учебной, учебно-методической и организационно-методической деятельности выпускающей кафедры.

Предметом исследования являются средства программной поддержки указанных видов деятельности.

Целью данной работы является проектирование и разработка программной платформы информационной системы кафедры, то есть ее фундамента. Главная особенность целевого решения – это наличие сервисов, источников данных, политик безопасности и правил для встраивания сторонних решений, предназначенных для автоматизации прикладных задач.

Теоретическая значимость работы заключается в исследовании современных технологий разработки информационных систем в контексте автоматизации информационных процессов учебного заведения, подготавливающего IT-специалистов.

Практическая значимость работы заключается в создании программной платформы, предназначенной для использования студентами, магистрантами и аспирантами в рамках учебных проектов по автоматизации кафедральных бизнес-процессов.

В главе 1 «Пути комплексной автоматизации деятельности выпускающей кафедры информационного профиля» отражены результаты системного анализа деятельности кафедры, определены потребности заинтересованных сторон, сформированы общие требования к программной платформе.

В главе 2 «Исследование архитектур» приводятся результаты исследования современных архитектур программного обеспечения, а также характеристики и причины выбора окончательной архитектуры. Приведено окончательное техническое задание, сформулированное с учётом выбранной архитектуры.

В главе 3 «Проектирование и разработка платформы» приводятся результаты детального проектирования и реализации базовых сервисов платформы, а также результаты интеграции с внешними системами.

В главе 4 «Управление проектом» представлена иерархическая структура работ, календарный план работ и план управления рисками.

1. ПУТИ КОМПЛЕКСНОЙ АВТОМАТИЗАЦИИ ДЕЯТЕЛЬНОСТИ ВЫПУСКАЮЩЕЙ КАФЕДРЫ ИНФОРМАЦИОННОГО ПРОФИЛЯ

Рынок труда в сфере информационных технологий требует, чтобы выпускники соответствующих специальностей владели современными методологиями, технологиями и средствами разработки программного обеспечения. В современной IT-сфере работа в команде является одним из важных навыков, которым должны обладать будущие IT-специалисты [33].

Отслеживая ситуацию на рынке труда, Министерство образования и науки РФ постоянно обновляет стандарты образования, где чётко указаны навыки, какими должны обладать будущие IT-специалисты, чтобы удовлетворить текущий спрос на рынке труда [35, 34, 36, 38, 37].

Кафедра «Информатика и программное обеспечение» является одной из ведущих в г. Брянске по выпуску IT-специалистов. Чтобы её выпускники ценились на рынке труда, кафедра должна постоянно осваивать новые технологии и подходы к обучению.

Всё вышеописанное означает, что кафедра исполняет очень широкий круг обязанностей, что вызывает потребность в автоматизации информационных процессов на разных уровнях. К сожалению, существующее программное обеспечение, такое как «1С: Университет» решают либо задачи всего университета на уровне руководства, либо автоматизируют узкую область в образовательном процессе.

1.1. Предварительный системный анализ

В ходе работы был проведён системный анализ предметной области, в рамках которого проводилось общение с руководством кафедры и её сотрудниками. Целью этого системного анализа было формулирование потребностей кафедры, а также выявление общих требования к требуемому программному обеспечению, с помощью глубокого изучения проблематики

предметной области с точки зрения руководства, сотрудников и учащихся кафедры.

Далее по тексту используется *терминология системного анализа* [32].

1.1.1. Потребность в автоматизации

Руководство кафедры на основе обратной связи от сотрудников и учащихся осознало **потребность** в единой информационной системе, которая позволит:

- а) хранить максимально подробную историю и комментарии преподавателей об учебной деятельности студентов;
- б) автоматизировать документооборот (генерировать типовые документы, хранить в кафедральном репозитории учебные планы и рабочие программы);
- в) упростить организацию и планирование, как кафедральных событий, так и отдельных учебных проектов;
- г) выявлять проблемы в учебном процессе на ранних стадиях;
- д) разрабатывать новые подсистемы, автоматизирующие информационные процессы кафедры, с привлечением студентов, магистров и аспирантов, которые смогут получить опыт работы над крупным проектом.

Таким образом, можно сформулировать **желание** руководства – предоставить сотрудникам такую информационную систему. Однако, существуют следующие **проблемы**:

- а) не существует программной системы, которая бы удовлетворяла всем потребностям;
- б) большая часть информации хранится в виде электронных документов офисных пакетов.

1.1.2. Цели проекта

Чтобы удовлетворить вышеописанные потребности, реализовать желание и решить указанные проблемы, текущему проекту ставятся следующие **цели**:

- а) разработать кафедральную информационную систему, которая удовлетворит потребности, сформулированные руководством кафедры;
- б) интегрировать существующие и сторонние решения посредством выделения общих частей;
- в) создать кафедральные информационные ресурсы, которые станут основным источником данных для приложений и сотрудников, вместо электронных документов.

1.1.3. Потребности и цели сотрудников

В ходе общения с сотрудниками стало ясно, что каждый из них часто выполняет несколько обязанностей на кафедре. Например, сотрудник может одновременно выполнять обязанности преподавателя, куратора и руководителя выпускной квалификационной работы. Анализ показал, что потребности сотрудников в рамках идентичных обязанностей не являются принципиально противоречивыми и часто дополняют друг друга.

Таким образом, в качестве **стейкхолдеров** корректнее вместо отдельных сотрудников рассматривать их обязанности в качестве ролей, а именно: преподаватель, куратор, методист, руководство кафедры, студент-пользователь и студент-разработчик.

Цели, которые преследуют стейкхолдеры, изображены в схеме на рис. 1.

Верхний узел на схеме – это основная цель, зелёные узлы – это стейкхолдеры, оранжевые – главные цели, которые стейкхолдеры преследуют. Белые узлы ниже – это подцели, без которых не достичь главных целей.

Рассмотрим подробнее главные цели стейкхолдеров на данной схеме.

«Разработка подсистем в качестве курсовых и дипломных работ» позволит заинтересованным студентам получать опыт в участии над крупным проектом.

«Использование персонального органайзера» позволит обычным студентам лучше контролировать свою учебную деятельность:

- а) узнавать о сроках сдачи и о состоянии лабораторных, курсовых и других работ;

- б) во время работы над дипломным проектом быть в курсе организационных событий (сроки на выполнения каждого этапа работы, даты встреч с руководителем и пр.);
- в) узнавать расписание преподавателей;
- г) быть уведомленным о тех или иных кафедральных событиях (конференции, конкурсы, встречи).

«Улучшить контроль над успеваемостью студентов» – цель, достижение которой, позволит кураторам получать оперативную информацию о состоянии каждого студента, в частности, об отставание от сроков сдачи работ. Это позволит заранее принимать соответствующие меры.

«Отказ от традиционных методов учёта и анализа успеваемости». Существуют следующие проблемы использования бумажных ведомостей:

- а) их можно потерять;
- б) они хранят информацию только в формате «был/не был» и «сдал/не сдал»;
- в) данные в них легко фальсифицируемы;
- г) в случае, если студент был восстановлен, невозможно точно определить какие работы были сданы.

«Помощь при разработке рабочих программ». Возникают ситуации, когда из-за определенных обстоятельств (праздники, болезни преподавателя) из курса могут выпасть темы. Реструктуризация курса может помочь хотя бы частично их затронуть, в случае возникновения этих обстоятельств.

«Подробный анализ деятельности кафедры» позволит руководству оперативно выявлять проблемы на своём уровне.

«Генерация типовых документов» позволит избежать монотонной и рутинной работы по созданию приказов, отчётов и других документов.

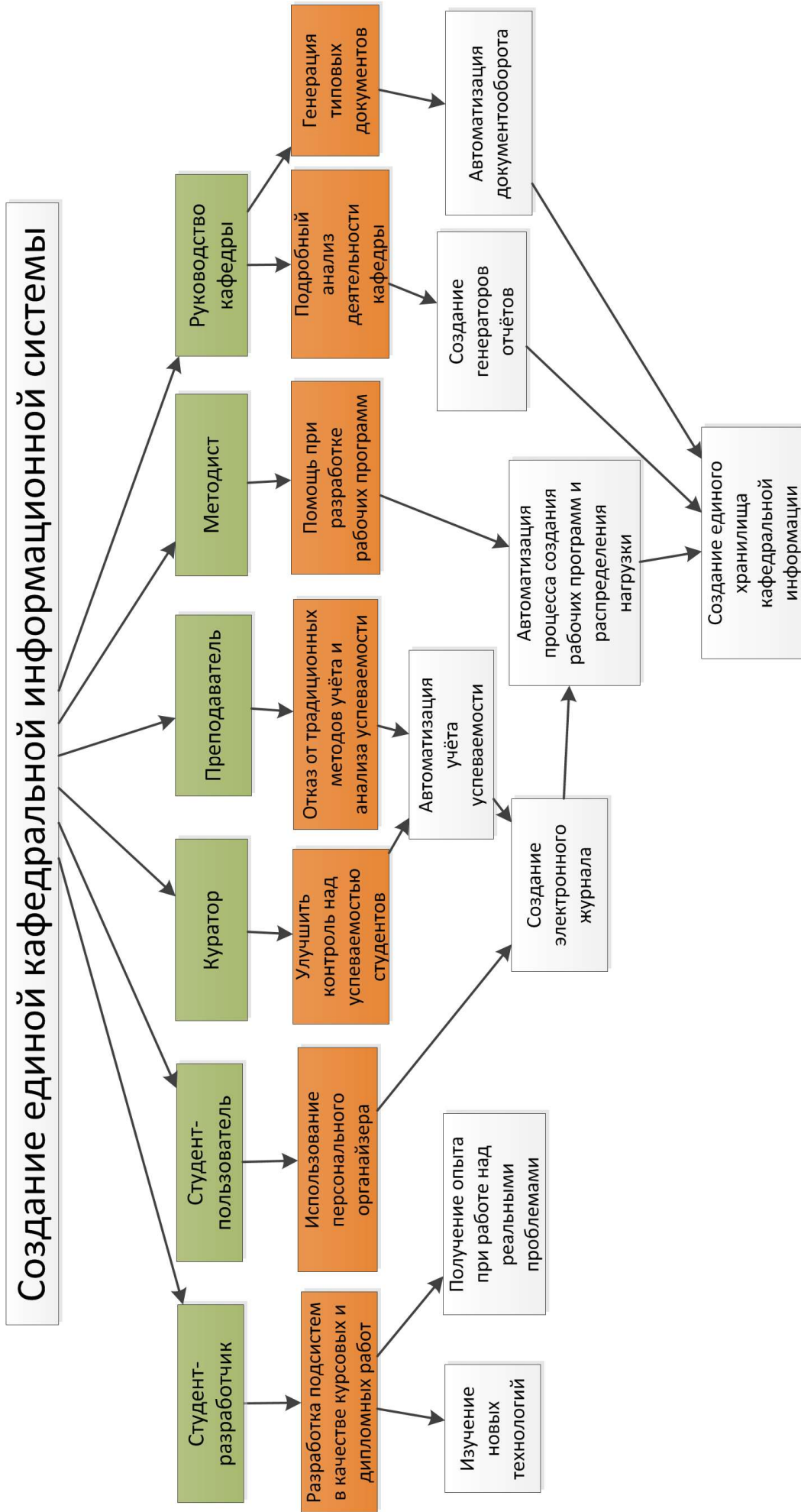


Рис. 1. Схема целей с позиции стейкхолдеров

1.1.4. Потребности и цели других заинтересованных сторон

Кафедра «Информатика и программное обеспечение» занимается подготовкой по следующим направлениям:

1. «02.03.03 Математическое обеспечение и администрирование информационных систем (уровень бакалавриата)».
2. «09.03.01 Информатика и вычислительная техника (уровень бакалавриата)».
3. «09.03.04 Программная инженерия (уровень бакалавриата)»
4. «09.04.01 Информатика и вычислительная техника (уровень магистратуры)».
5. «09.04.04 Программная инженерия (уровень магистратуры)»

В разделе «Общекультурные компетенции» во всех пяти федеральных государственных образовательных стандартах описано требование, что выпускники должны обладать навыками работы в команде в рамках профессиональной деятельности [35, 34, 36, 38, 37]. Это требование точно соответствует современному состоянию рынка труда, так как IT-специалистам редко приходится разрабатывать проекты с нуля в одиночку.

Поэтому, чтобы соблюсти требования федеральных стандартов, а также удовлетворить спрос на рынке труда, кафедра должна подготавливать выпускников, обладающих этим навыком.

1.2. Общие требования к информационной системе

По результатам предварительного системного анализа предметной области, руководство кафедры поставило задачу спроектировать информационную систему кафедры и реализовать её основную часть – программную платформу, на базе которой студенты, магистранты и аспиранты в качестве разработчиков смогут в рамках единой архитектуры автоматизировать кафедральные процессы, попутно получая опыт работы над крупным проектом.

Согласно выявленным потребностям руководства и сотрудников в рамках системного анализа, были сформулированы общие требования к программному обеспечению. Разрабатываемая информационная система должна:

- а) позволять вести учёт успеваемости и посещаемости студентов;
- б) по запросу генерировать типовые документы;
- в) предоставить кафедральный репозиторий учебных планов, рабочих программ и прочих документов;
- г) предоставлять платформу для разработки учащимися подсистем, автоматизирующих кафедральные процессы.

1.3. Выводы

Объектом исследования является информационное обеспечение учебной, учебно-методической и организационно-методической деятельности выпускающей кафедры.

Предметом исследования являются средства программной поддержки указанных видов деятельности.

Целью данной работы является проектирование и разработка программной платформы информационной системы кафедры, то есть ее фундамента. Главная особенность целевого решения – это наличие сервисов, источников данных, политик безопасности и правил для встраивания сторонних решений, предназначенных для автоматизации прикладных задач.

Для успешного достижения поставленной цели необходимо выполнить следующие задачи:

1. Выполнить системный анализ учебной, учебно-методической и организационно-методической деятельности выпускающей кафедры.
2. Исследовать современные архитектуры программного обеспечения и с помощью метода анализа иерархий выбрать подходящую.
3. По результатам исследования архитектур разработать техническое задание на разработку кафедральной программной платформы.

4. Развернуть базовую программную инфраструктуру на кафедральном сервере.
5. Разработать основные сервисы платформы: сервис учёта контингента, сервис аутентификации, сервис авторизации.
6. Определить структуру работ в проекте, идентифицировать проектные риски и разработать методы их контроля.

Теоретическая значимость работы заключается в исследовании современных технологий разработки информационных систем в контексте автоматизации информационных процессов учебного заведения, подготавливающего IT-специалистов.

Практическая значимость работы заключается в создании программной платформы, предназначенной для использования студентами, магистрантами и аспирантами в рамках учебных проектов по автоматизации кафедральных бизнес-процессов.

Актуальность данной работы заключается в использовании современных программных архитектур и инструментов разработки для построения систем комплексной автоматизации деятельности организации.

Полученные практические результаты могут использоваться не только в Брянском государственном техническом университете, но также и в других вузах страны, занимающихся подготовкой IT-специалистов.

2. ИССЛЕДОВАНИЕ АРХИТЕКТУР

Для программных систем такого масштаба вопрос архитектуры стоит крайне остро. Архитектуру через несколько лет будет очень проблематично изменить, если та окажется непригодной. Следовательно, система придёт в упадок, что крайне нежелательно для всех стейкхолдеров. Для решения вопроса такой важности были проведены групповые общения с экспертами, работающие техническими лидерами и архитекторами в IT-компаниях.

Целью данного исследования является выбор оптимальной архитектуры будущего программного комплекса.

В задачи данного исследования входят:

- а) изучение современных архитектур программного обеспечения и выбор подходящих альтернатив;
- б) формулирование критериев сравнения архитектур;
- в) построение модели принятия решения;
- г) выбор конечной архитектуры с помощью экспертов.

2.1. Анализ архитектур

Обзорное исследование распространённых в данное время архитектур крупных систем позволило выделить 4 основные альтернативы.

2.1.1. Выбор альтернатив

Монолитная архитектура – трёхзвенная архитектура, чаще всего включающая в себя один тип клиента, например, браузер, одно серверное приложение и одну базу данных [12].

Сервис-ориентированная архитектура (Service-oriented architecture, SOA) – множество серверных приложений (веб-сервисы, веб-службы), множество клиентов (браузеры, мобильные приложения, другие веб-сервисы). Веб-служба может иметь доступ только к собственной базе данных. Возможен вариант, когда

служба не имеет собственной базы данных, так как она использует данные из других служб [18].

Микросервисная архитектура – Частный случай SOA. Множество клиентов, множество небольших служб с минимальным, но строго определенным набором функций (основные принципы разработки операционной системы UNIX) [1]. Микросервисная архитектура является экстремальным случаем SOA: размер сервисов уменьшен настолько, что каждый из них можно запустить в количестве нескольких экземпляров, распределяя нагрузку через балансировщик [31].

Сервис-ориентированная архитектура с сервисной шиной предприятия (SOA + ESB) – SOA или микросервисная архитектура с общим для всех служб событийным каналом связи. В чистой SOA или микросервисной архитектуре службы общаются напрямую. В случае отказа службы-адресата, сообщение просто не дойдёт до него. ESB при необходимости может гарантировать доставку, сохранив сообщение и доставить его позже. Сама сервисная шина может обрабатывать проходящие через неё сообщения произвольным образом. Логика обработки запросов и ответов задаётся через административную консоль ESB [5].

2.1.2. Выбор критериев

Для того чтобы осуществить выбор подходящей архитектуры программного обеспечения, был подготовлен следующий список качественных критериев:

- а) отказоустойчивость;
- б) использование одновременно нескольких технологий и языков;
- в) скорость разработки и внедрения нового функционала;
- г) сложность добавления незапланированного при проектировании функционала;
- д) сложность восприятия программного кода;
- е) связность модулей;
- ж) масштабируемость;
- з) порог вхождения в проект для разработчика;

- и) простота покрытия существенного объема кода тестами;
- к) простота ручного тестирования целостной системы;
- л) простота документирования;
- м) необходимость в высокоуровневых архитектурных абстракциях и шаблонах проектирования;
- н) простота рефакторинга и модификации архитектуры;
- о) время обработки пользовательского запроса;
- п) сложность интеграции уже имеющихся и работающих систем в проект.

2.1.3. Коллективная оценка критериев

Оценка альтернатив по указанным критериям с помощью ранжирования альтернатив показала, что сами критерии являются непригодными, а их количество избыточно.

Чтобы уменьшить количество критериев и увеличить их качество, был проведён мозговой штурм. Ниже приведён зафиксированный участниками неструктурированный список мыслей и идей, озвученных во время штурма:

- а) возможность участия студентов, начиная с курсовых по БД;
- б) интеграция с LDAP или Active Directory;
- в) возможность разворачивать тестовые площадки легко и быстро;
- г) лёгкое добавление новых сущностей и функций;
- д) поддержка Data Warehouse;
- е) использование нескольких технологий и языков;
- ж) отказоустойчивость;
- з) скорость получения рабочего серверного прототипа.

Результат структуризации идей, полученных в результате мозгового штурма – это граф главных целей этапа разработки платформы рис. 2.

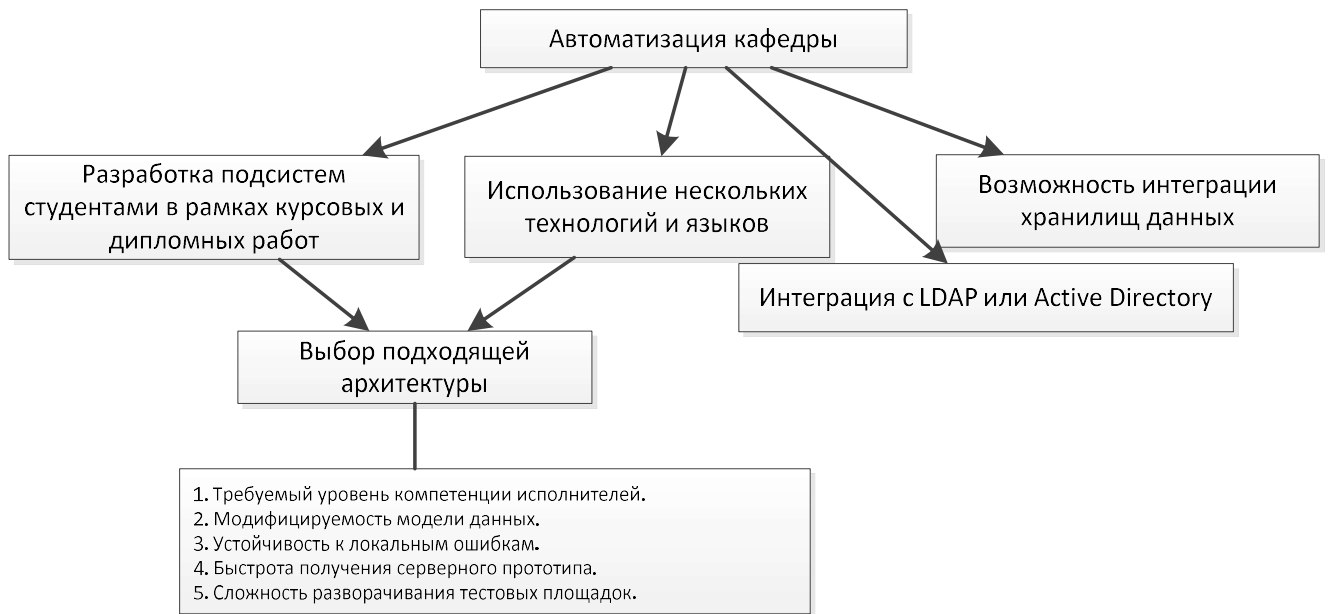


Рис. 2. Структурированные результаты мозгового штурма

Список из пяти пунктов в нижнем блоке – это список критериев, по которым стоит оценивать архитектуры.

Критерий «Требуемый уровень компетенции исполнителей» показывает, какой уровень знаний и опыта требуется от конечных исполнителей, а именно, студентов.

Критерий «Модифицируемость модели данных» отражает насколько трудоёмко модифицировать существующую структуру модели данных, особенно, когда она достаточно сложна.

Критерий «Устойчивость к локальным ошибкам» отражает способность всей системы оставаться работоспособной даже в том случае, когда некачественные модули вызывают частые сбои (вплоть до аварийного завершения) монолитного приложения или служб.

Критерий «Быстрота получение серверного прототипа» показывает сколько нужно затратить времени, чтобы создать серверное приложение, хранящее основные справочники (студенты, преподаватели, группы и т.д.).

Критерий «Сложность разворачивания тестовых площадок» отражает трудоёмкость администрирования тестовых площадок, например, периодическое восстановление состояния различных СУБД, а также трудоёмкость конфигурирования и разворачивания серверного приложения.

2.1.4. Модель принятия решения

Оценка альтернатив производилась с помощью метода анализа иерархий [39]. Для этого была построена иерархическая модель на рис. 3, где промежуточный уровень представляют критерии. Оценка производилась коллективно экспертами.

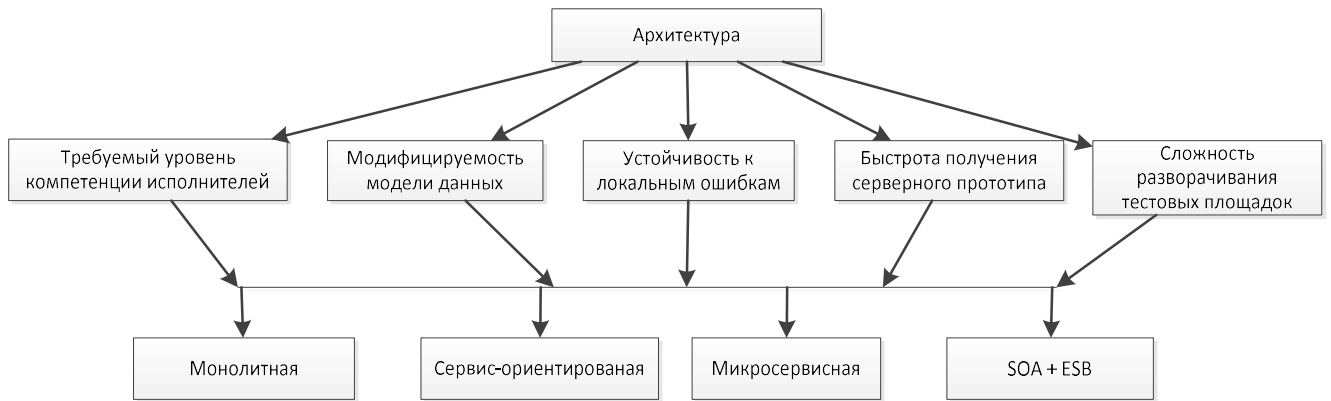


Рис. 3. Иерархия модели принятия решения

Сначала определим веса критериев с помощью метода парных сравнений.

$$D0 = \begin{pmatrix} 1 & \frac{1}{3} & \frac{1}{5} & 1 & 3 \\ 3 & 1 & 3 & 7 & 5 \\ 5 & \frac{1}{3} & 1 & 5 & 3 \\ 1 & \frac{1}{7} & \frac{1}{5} & 1 & 1 \\ \frac{1}{3} & \frac{1}{5} & \frac{1}{3} & 1 & 1 \end{pmatrix} \quad u = \begin{pmatrix} 0.196 \\ 0.8196 \\ 0.51 \\ 0.123 \\ 0.12 \end{pmatrix} \quad w = \begin{pmatrix} 0.11 \\ 0.463 \\ 0.288 \\ 0.069 \\ 0.068 \end{pmatrix}.$$

$$\lambda_{max} = 5.36 \quad CI = 0.0908 \quad CR = 0.081.$$

Данное сравнение показало, что критерий «Модифицируемость модели данных» оказался на первом месте, а вторым по важности стал критерий «Устойчивость к локальным ошибкам». В целом, оценка совпала с интуитивными ожиданиями. Ожидалось, однако, что оба лидера будут примерно равны по весу.

Далее была произведена оценка альтернатив с точки зрения критерия «Требуемый уровень компетенции исполнителей».

$$D1 = \begin{pmatrix} 1 & \frac{1}{3} & \frac{1}{5} & \frac{1}{7} \\ 3 & 1 & \frac{1}{3} & \frac{1}{5} \\ 5 & 3 & 1 & \frac{1}{3} \\ 7 & 5 & 3 & 1 \end{pmatrix} u = \begin{pmatrix} 0.087 \\ 0.185 \\ 0.0412 \\ 0.888 \end{pmatrix} w = \begin{pmatrix} 0.055 \\ 0.117 \\ 0.262 \\ 0.565 \end{pmatrix}.$$

$$\lambda_{max} = 4.12 \text{ CI} = 0.0389 \text{ CR} = 0.0433.$$

В данном случае победила альтернатива «SOA + ESB». Она требует высокого уровня компетентности от архитектора и администратора, однако разработчикам, реализующие решения прикладных задач, она позволит сфокусироваться на решении этих самых задач, не заботясь об архитектуре всего комплекса. Монолитная архитектура проиграла, так как профессиональный опыт экспертов подсказывает, что перед решением прикладной задачи, необходимо детально изучить подсистему, куда решение будет встроено. Микросервисная архитектура опередила SOA просто потому что, эксперты сошлись во мнении, что принципы разработки для операционной системы UNIX, позволяющей разрабатывать качественное ПО, допустимо перенести в веб-среду.

Была произведена оценка альтернатив с точки зрения критерия «Модифицируемость модели данных».

$$D2 = \begin{pmatrix} 1 & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ 5 & 1 & \frac{1}{2} & \frac{1}{5} \\ 6 & 2 & 1 & \frac{1}{6} \\ 9 & 5 & 3 & 1 \end{pmatrix} u = \begin{pmatrix} 0.066 \\ 0.222 \\ 0.366 \\ 0.9 \end{pmatrix} w = \begin{pmatrix} 0.04 \\ 0.143 \\ 0.235 \\ 0.579 \end{pmatrix}.$$

$$\lambda_{max} = 4.115 \text{ CI} = 0.0383 \text{ CR} = 0.0426.$$

Монолитная архитектура опять проиграла по причине сильных зависимостей от структуры данных внутри монолитной программы. В случае с «SOA+ESB» при модификации модели данных, можно не изменять клиенты, а просто настроить правила трансформации сообщений внутри административной консоли ESB. Следовательно, клиентские приложения могут работать старой и новой моделями данных.

Далее была произведена оценка альтернатив с точки зрения критерия «Устойчивость к локальным ошибкам».

$$D3 = \begin{pmatrix} 1 & \frac{1}{5} & \frac{1}{5} & \frac{1}{7} \\ 5 & 1 & 1 & \frac{1}{3} \\ 5 & 1 & 1 & \frac{1}{3} \\ 7 & 3 & 3 & 1 \end{pmatrix} u = \begin{pmatrix} 0.085 \\ 0.345 \\ 0.344 \\ 0.869 \end{pmatrix} w = \begin{pmatrix} 0.05 \\ 0.21 \\ 0.21 \\ 0.53 \end{pmatrix}.$$

$$\lambda_{max} = 4.07 \quad CI = 0.02386 \quad CR = 0.0265.$$

В данном случае однозначно побеждает «SOA+ESB». Причина в том, что всё ПО, которое входит в класс «Сервисные шины предприятия», имеет опцию гарантированной доставки сообщения, а также дополнительные средства контроля над устойчивостью подсистем.

Была произведена оценка альтернатив с точки зрения критерия «Быстрота получения серверного прототипа».

$$D4 = \begin{pmatrix} 1 & 3 & 2 & \frac{1}{4} \\ \frac{1}{3} & 1 & 1 & \frac{1}{3} \\ \frac{1}{2} & 1 & 1 & \frac{1}{3} \\ 4 & 3 & 3 & 1 \end{pmatrix} u = \begin{pmatrix} 0.41 \\ 0.21 \\ 0.221 \\ 0.861 \end{pmatrix} w = \begin{pmatrix} 0.24 \\ 0.12 \\ 0.13 \\ 0.51 \end{pmatrix}.$$

$$\lambda_{max} = 4.13 \quad CI = 0.043 \quad CR = 0.0477.$$

Здесь монолитная опередила SOA и микросервисную архитектуру, потому что вследствие простоты монолитной архитектуры, разработка на начальных этапах даётся проще и быстрее. Однако, согласно опыту экспертов, такой прототип в дальнейшем будет сложно поддерживать и развивать.

Была произведена оценка альтернатив с точки зрения критерия «Сложность разворачивания тестовых площадок».

$$D5 = \begin{pmatrix} 1 & 4 & 5 & 2 \\ \frac{1}{4} & 1 & 3 & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{3} & 1 & \frac{1}{5} \\ \frac{1}{2} & 5 & 5 & 1 \end{pmatrix} u = \begin{pmatrix} 0.776 \\ 0.196 \\ 0.105 \\ 0.59 \end{pmatrix} w = \begin{pmatrix} 0.465 \\ 0.117 \\ 0.063 \\ 0.354 \end{pmatrix}.$$

$$\lambda_{max} = 4.21 \text{ CI} = 0.06985 \text{ CR} = 0.07761.$$

Монолитная архитектура победила остальные альтернативы, потому что её объективно проще развернуть на любой операционной системе: в приложении просто указывается адрес тестовой БД. Для всех сервис-ориентированных архитектур придётся либо отдельно настраивать каждый сервис, либо разворачивать копию всей инфраструктуры на тестовом сервере.

Теперь необходимо оценить глобальные приоритеты альтернатив. Для этого построим матрицу, где i -й столбец – вектор приоритетов альтернатив относительно i -го критерия. Далее умножим матрицу на приоритеты критериев.

$$w = \begin{pmatrix} 0.0553 & 0.0424 & 0.0519 & 0.2405 & 0.4654 \\ 0.1175 & 0.1431 & 0.2101 & 0.1214 & 0.1174 \\ 0.2622 & 0.2355 & 0.2097 & 0.1307 & 0.0632 \\ 0.565 & 0.579 & 0.5283 & 0.5074 & 0.3539 \end{pmatrix} * \begin{pmatrix} 0.111 \\ 0.463 \\ 0.2885 \\ 0.069 \\ 0.0676 \end{pmatrix} = \begin{pmatrix} 0.0889 \\ 0.1563 \\ 0.2121 \\ 0.5426 \end{pmatrix}.$$

Таким образом, альтернатива «SOA+ESB» получила наивысшее место, «Микросервисная архитектура» получила второе место, «Сервис-ориентированная архитектура» третье, и на последнем месте оказалась «Монолитная архитектура».

Докажем общую согласованность модели.

$$C = CI^0 + w^{0,i} * CI^i = 0.0908 + \begin{pmatrix} 0.111 \\ 0.4635 \\ 0.2885 \\ 0.0694 \\ 0.0676 \end{pmatrix} \cdot \begin{pmatrix} 0.0389 \\ 0.0383 \\ 0.024 \\ 0.043 \\ 0.07 \end{pmatrix} = 0.127.$$

Так как на первом уровне иерархии всегда по 4 альтернативы, значит независимо от весов на нулевом уровне, которые в сумме дают единицу, средневзвешенное значение индекса будет равно 0.9.

$$CS = CIS^0 + 0.9 = 1.12 + 0.9 = 2.02.$$

$$R = \frac{C}{CS} = \frac{0.127}{2.02} = 0.063 < 0.1.$$

Таким образом, модель согласована, что и требовалось доказать.

В результате, было принято решение использовать сервис-ориентированную архитектуру совместно с сервисной шиной предприятия. На основе полученных данных, можно сформулировать подробное техническое задание, учитывающее этот вид архитектуры.

2.2. Техническое задание

Необходимо спроектировать программную платформу и разработать базовые сервисы, которые будут использоваться для автоматизации деятельности кафедры «Информатика и программное обеспечение».

2.2.1. Основания для разработки

Основанием для разработки является задание на выполнение магистерской диссертации, выданное доцентом кафедры «Информатика и программное обеспечение» Лагеревым Д.Г. на основе приказа по БГТУ № 408-3 от 15.05.2017.

2.2.2. Назначение разработки

Разработка должна определить общую архитектуру будущей кафедральной системы, а также реализовать основные элементы платформы:

- а) основные сервисы, т.е. ядро;
- б) правила взаимодействия и интеграции служб;
- в) правила безопасности.

2.2.3. Общие требования

Главное требование, предъявляемые к разрабатываемому ядру – наличие службы учёта контингента кафедры. Служба должна хранить информацию о преподавателях и студентах, позволяя остальным службам ссылаться на них для решения прикладных задач или хранения дополнительной информации.

Основные данные, которые необходимо хранить о каждой личности:

- а) ID;
- б) ФИО;
- в) логин/пароль (для интеграции со сторонними системами);
- г) статус (студент, преподаватель);
- д) рабочая почта, рабочий телефон (публичная информация)
- е) домашние адреса постоянного и временного проживания (закрытая информация);
- ж) учебная группа (для студентов).

2.2.4. Функциональные требования

Требования к службе учёта контингента.

1. Хранение вышеперечисленных данных.
2. Предоставление REST API для работы с данными:
 - а) для запроса открытых данных;
 - б) для поиска по вышеперечисленным полям;
 - в) для авторизованного добавления студентов и преподавателей;
 - г) для авторизованного добавления групп;
 - д) для авторизованного обновления данных;
 - е) для авторизованного запроса к закрытым данным.
3. Запрещение удаления данных через API.
4. Инициирование широковещательных системных событий, посылаемых в шину, при изменении данных.

Помимо основных задач, необходимо уделить внимание безопасности. Механизмы авторизации всех кафедральных служб, в том числе будущих, должны быть вынесены в отдельный сервис авторизации, в задачи которого входит принятие решения о выдаче доступа к тому или иному ресурсу.

2.2.5. Требования к архитектуре

Архитектура должна быть спроектирована с учётом сервис-ориентированного подхода. Центральным звеном архитектуры должна стать сервисная шина предприятия в качестве единственно возможного канала связи между службами. Доступ к службам должен быть проксирован для обеспечения их работы в рамках одного домена *esb.iipo.tu-bryansk.ru*.

2.2.6. Требование к аппаратному и программному обеспечению

Разработанное программное обеспечение должно работать под управлением операционной системы Debian 8. Платформа должна быть развёрнута на кафедральном сервере внутри виртуальной машины. Характеристики виртуальной машины:

1. Процессор 2 ГГц.
2. ОЗУ 4 Гб.
3. 200 Гб дискового пространства.

Клиентские приложения и сервисы должны иметь доступ по сети к домену *esb.iipo.tu-bryansk.ru* по протоколам HTTP/HTTPS.

2.3. Выводы по главе

Результаты проделанной работы по исследованию архитектур позволяют прийти к выводу, что классические монолитные архитектуры неприменимы для разработки систем такой сложности. Для успешного развития системы в течение долгого срока, необходимо использовать сервис-ориентированную архитектуру.

Для сервис-ориентированной архитектуры важны только те бизнес-задачи, которые способны выполнять сервисы, независимо от того где и как они размещены, какова их внутренняя структура и на каких технологиях они построены. Такой подход позволяет при построении сервисов использовать любые языки программирования и другие технологии, если они отлично подходят

для решения определённой бизнес-задачи. Распределенная структура позволит в целом увеличить надёжность всей системы.

Такой класс программного обеспечения как сервисные шины предприятия специально были разработаны для предоставления вспомогательных инструментов и функций для разработки систем на базе сервис-ориентированной архитектуры.

3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПЛАТФОРМЫ

Как уже отмечалось выше, отличительной особенностью сервис-ориентированной архитектуры является возможность комбинировать сразу несколько технологий без больших трудозатрат. Каждый сервис чаще всего представляет собой автономное серверное приложение и в основном общается с другими сервисами по протоколу HTTP. В данном случае очень хорошо просматривается аналогия с некоторыми правилами разработки UNIX [1]:

1. «Пишите программы, которые делают что-то одно, но хорошо».
2. «Пишите программы, которые бы работали вместе».
3. «Пишите программы, которые бы поддерживали текстовые потоки, поскольку это универсальный интерфейс».

Веб-службы могут общаться с клиентами и друг с другом напрямую. Однако, когда служб много, или в систему необходимо интегрировать службы, разработанные в другое время, например, унаследованные службы, или другими компаниями, например, облачные сервисы от крупных IT-корпораций, возникает необходимость сохранения общей согласованности потоков данных. Для упорядочивания и обеспечения общей согласованности приходит на помощь такой класс промежуточного программного обеспечения как сервисная шина предприятия (Enterprise Service Bus, ESB).

Брянский государственный технический университет уже имеет большое количество стороннего («1С: Университет», «Deductor») и разработанного собственными силами программного обеспечения. Каждое решает задачи вуза, но ни одно из них не взаимодействует друг с другом напрямую. Для совместной интеграции подобных систем как раз и существует такой класс промежуточного программного обеспечения, как сервисные шины предприятия.

Основные функции шины – это посредничество («mediation»), интеграция («integration») и обеспечение взаимосвязанности («interconnectivity») слабосвязанных служб [5]. Функция посредничества, возможно, наиболее важная функция шины. Она позволяет трансформировать и обрабатывать сообщения, проходящие через шину, по произвольной логике. Функция интеграции позволяет

связывать службы, работающие даже по разным протоколам, т.е. которые изначально не были спроектированы для совместной работы. Впрочем, это довольно экстремальные варианты использования шины, основные же сводятся к организации и упорядочиванию потоков данных, проходящих через неё, предварительной обработке и конвертации сообщений, балансировки нагрузки, широковещательной рассылки событий и гарантированной доставки сообщений.

3.1. Общий вид архитектуры

Согласно общим требованиям к платформе, а также существующим потребностям в различных сервисах, можно сформировать общий вид разрабатываемой системы (см. рис. 4). На данной схеме красной рамкой выделены подсистемы, которые непосредственно разрабатываются или интегрируются как сторонние в рамках текущей работы. Системы, уже разрабатываемые или запланированные для будущей разработки в течение краткосрочного и долгосрочного периода, изображены в соответствующих рамках.

Сервис учёта успеваемости позволяет преподавателям учитывать сданные лабораторные, расчётно-графические и курсовые работы студентов за всё время обучения, в том числе после восстановления или академического отпуска студента.

Сервис учёта посещаемости позволяет контролировать присутствие студента на лекциях, лабораторных работах и семинарах.

Сервис распределения студентов по руководителям ВКР (выпускных квалификационных работ) будет использовать математический аппарат теории игр, что позволит оптимальным образом достичь компромисса между желаниями руководителей и студентов при распределении.

Сервис учёта достижений будет учитывать такие достижения студентов, как участие в конференциях, конкурсах, олимпиадах, публикации в журналах, а также использовать данные из сервисов успеваемости и посещаемости для формирования рейтинга в таблице лидеров семестра.

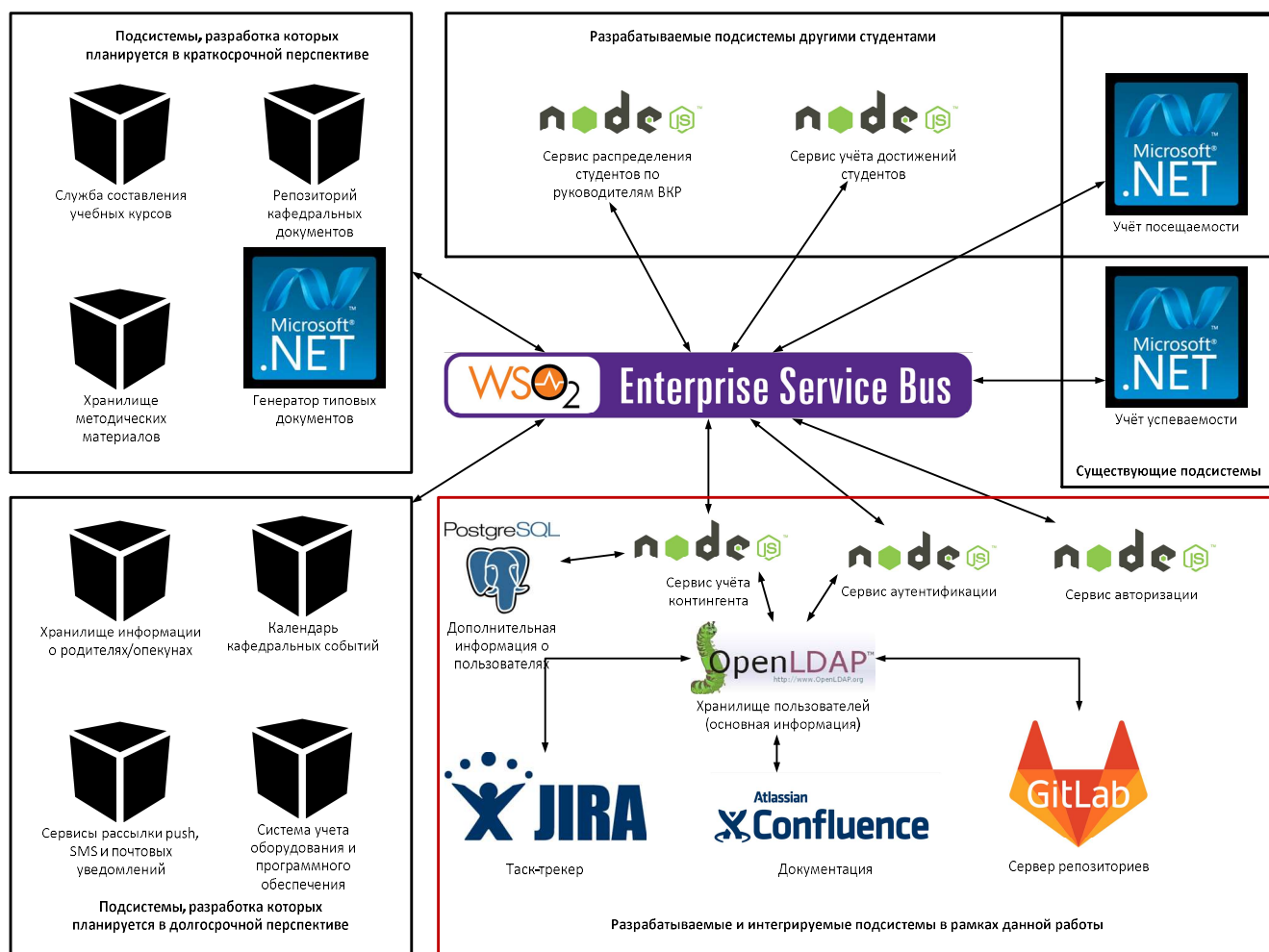


Рис. 4. Общий вид архитектуры кафедральной системы

Генератор типовых документов позволит генерировать шаблонные документы, используя данные из других сервисов.

Репозиторий кафедральных документов должен будет хранить учебные планы, рабочие программ, прочие документы, попутно осуществлять их версионирование и резервное копирование.

Хранилище методических материалов позволит любому нуждающемуся студенту скачать из него материал, необходимый для выполнения своей работы.

Хранилище информации о родителях и опекунах позволит преподавателям и руководству кафедры в случае серьезных проблем связаться с родителями/опекунами студента.

Календарь кафедральных событий позволит заинтересованным лицам быть уведомленным об определенных событиях (конечные сроки сдачи работ, переносы пар, конференции и прочие).

Система учёта оборудования и программного обеспечения позволит другим сервисам определять, наличие нужного оборудования и программ в аудиториях, что является важным при планировании кафедральных событий.

Сервис рассылки push, SMS и почтовых уведомлений позволит рассылать любые информационные сообщения персоналу вуза, студентам и их родителям/опекунам.

3.2. Сервисная шина предприятия

В качестве центрального звена используется сервисная шина предприятия Enterprise Service Bus от компании WSO2. Ниже перечислены причины выбора её выбора:

- а) лицензия Apache 2.0;
- б) подробная документация;
- в) развитое сообщество;
- г) большая база знаний на сайте компании;
- д) разнообразный каталог плагинов для интеграции сторонних сервисов;
- е) обширный спектр доступных функций для работы в рамках SOA;
- ж) кроссплатформенность.

В качестве основной платформы разработки рассматривались .NET и Node.js. Эти альтернативы были выбраны из следующих соображений:

- а) языки, которые подробно изучаются студентами на кафедре, это C++, C# и JavaScript;
- б) из-за особенностей рынка труда города Брянска, большинство студентов-старшекурсников работают в компаниях, занимающихся разработкой веб и мобильных приложений, т.е. используют языки C#, JavaScript, Java, PHP, Objective C, Swift.

Для разработки ядра выбор сделан в пользу платформы Node.js, так как с её помощью можно прототипировать службы с существенно меньшими трудозатратами, в сравнении с платформой .NET, тем самым позволив сократить время на проверку различных архитектурных решений. Поэтому, службы

контингенты, аутентификации и авторизации были разработаны на базе этой платформы.

В качестве основной операционной системы используется Debian 8, согласно требованиям в техническом задании.

3.3. Сервис контингента кафедры

Исходя из требований, сервис контингента должен хранить в себе общую информацию о студентах кафедры:

- а) ФИО;
- б) контакты (адреса, телефоны, email, контакты мессенджеров);
- в) группы, в которых учится студент.

Есть несколько очень важных аспектов, которые должны быть учтены.

Первый – это независимость данных о личности от ФИО. Т.е. персона должна однозначно идентифицироваться не по имени, а по какому-либо идентификатору. Номера паспортов, СНИЛС, ИНН не являются подходящими, т.к. в университете учатся студенты из других стран, следовательно, они могут не иметь этих идентификаторов. Форматы номеров их паспортов также отличаются. Следовательно, нужно генерировать идентификатор самостоятельно, например, использовать целое число с автоинкрементом или UUID.

Второй – система должна хранить актуальное, а также все имена, которые когда-либо были у персоны, т.к. на разных этапах обучения в разных документах могут фигурировать различные фамилии. Особенно актуально для девушек, несколько раз выходявших замуж за время обучения.

Третий аспект заключается в необходимости хранить всю информацию о персоне даже после отчисления, на случай, если студент решит восстановиться, что ещё раз подчёркивает в необходимости однозначно и уникально идентифицировать персону.

3.3.1. Служба каталогов OpenLDAP

Для учёта вышеописанных особенностей предметной области отлично подходят службы каталогов, такие как Active Directory и OpenLDAP. Обе реализации позволяют хранить в заранее заготовленных схемах данных информацию в формате «ключ – несколько значений». Службы каталогов обладают следующим набором полезных функций для кафедральной платформы:

1. Хранение полей в учётной записи пользователя:
 - а) фамилия, имя, отчество и отображаемое имя;
 - б) мобильный, городской телефоны, адреса, email;
 - в) фотография;
 - г) должность или положение;
 - д) логин и хеш пароля;
 - е) другие поля, определенные схемой данных inetOrgPerson [9].
2. Одному полю может соответствовать несколько значений. Это позволяет в явном виде хранить несколько фамилий, телефонов, адресов, и должностей, что немаловажно.
3. Реализуют протокол LDAP, поддерживающий операцию аутентификации. Данный протокол поддерживают практически любое программное обеспечение, нацеленное на коллективную работу (системы управления проектами, системы контроля версий, почтовые сервера, системы документирования).
4. Так как системы предназначены для хранения в основном текстовой информации, они производят индексацию по всем текстовым полям. Сразу после развёртывания доступен регистронезависимый поиск, в том числе по маске.

Так как службы каталогов уже решают довольно большой круг задач, вполне очевидна необходимость в их эксплуатации. Так как OpenLDAP распространяется по свободной лицензии и является бесплатным, выбор естественным образом пал на эту службу каталогов.

Для взаимодействия со службой каталогов напрямую, сервис учёта контингента использует библиотеку `ldapjs` из репозитория пакетов `npm` [10].

Дополнительный анализ предметной области показал, что некоторые студенты становятся преподавателями. Следовательно, чтобы сохранять всю информацию о человеке, нужно абстрагироваться от сущностей «Студент» и «Преподаватель» и перейти к сущности «Персона». Схема данных *inetOrgPerson* позволяет хранить в поле *title* должность (даже несколько), тем самым однозначно определяя положение пользователя на кафедре. Это поле также полезно в случаях, когда персона выполняет несколько ролей, например, является аспирантом и занимает должность ассистента одновременно, или же является заведующим кафедрой и в тоже время доцентом. Все эти аспекты предметной области можно уже отразить в поле *title* в службе каталогов.

3.3.2. СУБД и ORM

Служба каталогов OpenLDAP не является реляционным хранилищем, обеспечивающим контроль целостности данных. Однако потребность в таковом имеется, например, для хранения отношения много-ко-многим сущностей «Студент» – «Группа» (студент может учиться сразу в нескольких группах). К тому же, схемы OpenLDAP не позволяют хранить такие данные как день рождения, активен ли пользователь и др. Для хранения подобного рода данных в рамках сервиса контингента, использовалась СУБД PostgreSQL. Была выбрана исключительно из соображений бесплатности и универсальности в сравнении с MySQL. В качестве ORM (Object-Relational Mapping) библиотеки для работы с реляционной базой данных была выбрана Sequelize [17].

OpenLDAP не поддерживает автоинкремент и полноценные транзакции. Следовательно, для идентификации персон вместо целого числа необходимо использовать UUID.

3.3.3. Нереляционная модель данных

Как уже отмечалось выше, основная информация о персонах хранится в схеме данных `inetOrgPerson`. Список полей уже используемых приведён в таблице 1. Список полей, которые могут пригодиться в будущем приведён в таблице 2. Полное описание схемы данных с комментариями к каждому полю и примерами доступно в официальном репозитории OpenLDAP [30].

Таблица 1

Краткий список доступных полей в схеме `inetOrgPerson`

Название	Описание
<i>givenName</i>	Основное имя пользователя
<i>sn</i>	Все фамилии (surname) пользователя
<i>initials</i>	Инициалы всех остальных имён [29], используется сервисом в основном для хранения отчества
<i>displayName</i>	Актуальное полное имя. В основном это ФИО. Подразумевается, что будет заполняться вручную, так как это имя будет использоваться в официальных документах: для девушек, вышедших замуж, оно содержит актуальную фамилию, для иностранных студентов с длинными и сложными именами хранит допустимо сокращенную версию имени.
<i>uid</i>	Уникальный идентификатор пользователя в формате UUID v4
<i>cn</i>	Логин пользователя для аутентификации в приложениях
<i>userPassword</i>	Хеш пароля пользователя
<i>mail, homePhone, mobile, postalAddress</i>	Основные контактные данные. LDAP не накладывает требований на формат хранения. Хранятся в произвольном виде
<i>title</i>	Должность пользователя. На данный момент принимает следующие значения: «Студент», «Преподаватель», «Доцент», «Ассистент», «Декан», «Заведующий кафедрой».
<i>photo, jpegPhoto</i>	Фотография пользователя

Поля, интересные для будущего использования

Название	Описание
<i>employeeType</i>	Оригинальная схема данных предлагает использовать для хранения отношения вида отношения работодателя с работником (штатный, временный, внешний). Может быть использовано для хранения информации о статусе студентов, например, «учится по программе обмена».
<i>country</i>	Страна, откуда прибыл пользователь.
<i>departmentNumber</i>	Факультет, к которому принадлежит пользователь. Будет актуально, если система будет использоваться не только в рамках кафедры.

3.3.4. Реляционная модель данных

Схемы *inetOrgPerson* в целом достаточно, чтобы хранить информацию о студентах и преподавателях. К сожалению, эта схема данных в OpenLDAP, в отличие от аналогичной, но расширенной в Active Directory, не позволяет сохранить дату рождения и статус активности пользователя. К тому же OpenLDAP не обеспечивает целостность данных в отношении много-ко-многим. Поэтому такого рода реляционную информацию приходится хранить в БД. Общая модель данных сервиса учёта контингента представлена на рис. 5.

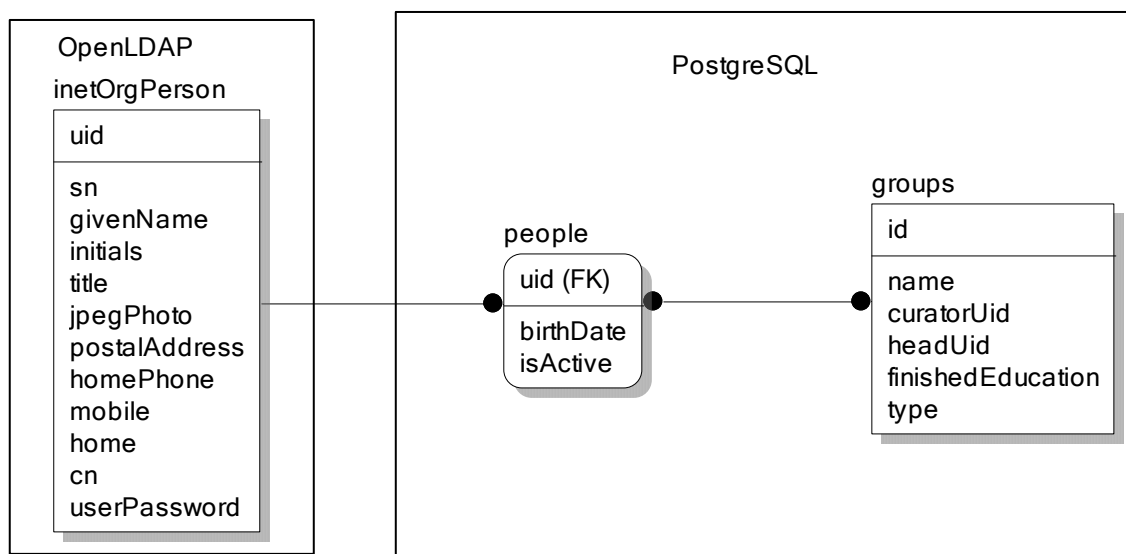


Рис. 5. Модель данных сервиса учёта контингента

В процессе развития системы модель реляционных и нереляционных данных и будут неизбежно усложняться, следовательно, в будущем необходимо сервис, разделить их на два различных сервиса.

3.3.5. Интерфейс *RESTful API*

Внешний доступ к данным обеспечивается посредством REST API. Формат данных передачи – JSON. Для внешних клиентов служба предоставляет два ресурса: «people» и «groups». Список разрешенных URL и HTTP методов представлен в таблице 3. URL оформлены уже с учётом интеграции в сервисную шину предприятия (префиксы «core», означающий название службы и «v1», означающее версию API).

Таблица 3

Ресурсы, предоставляемые сервисом учёта контингента

Ресурс	HTTP метод	Комментарий
1	2	3
/core/v1/	GET	Точка входа в службу, согласно концепции Hypermedia Application Language. Не требует авторизации.
/core/v1/people{?cn, givenName,sn,initials,mail}	GET	Предоставляет список всех пользователей (поля uid, givenName, sn, initials, displayName, cn), с возможностью фильтрации по маске одного из полей, указанных в фигурных скобках. Не требует авторизации.
/core/v1/people	POST	Создание нового пользователя. Поля sn и givenName обязательны, метод требует авторизации.
/core/v1/people/:uid/	GET	Предоставляет публичные данные о пользователе: поля uid, sn, givenName, initials, displayName, mail, title, cn; а также список учебных групп, в которых учится пользователь.

1	2	3
/core/v1/people/:uid/profile	GET	То же что и предыдущий метод, плюс, персональные данные. Метод требует авторизации.
	PATCH	Заменяет предоставленные в запросе поля новыми данными. Метод требует авторизации.
/core/v1/groups{?name}	GET	Предоставляет список учебных групп (поля id и name) с возможностью фильтрации по маске поля name. Не требует авторизации.
/core/v1/groups	POST	Создает новую группу. Требуется авторизация.
/core/v1/groups/:id	GET	Возвращает id, name группы и список её студентов (поля uid, givenName, sn, initials, displayName). Не требует авторизации.
	PATCH	Редактирует поле name группы. Требуется авторизация.
/core/v1/groups/:groupId/students/:uid	PUT	Добавляет студента в группу. Требуется авторизация.
/core/v1/groups/:groupId/students/:uid	DELETE	Удаляет студента из группы. Требуется авторизация.

Для описания обработчиков различных URL используется Node.js фреймворк Express [6].

3.3.6. Hypermedia Application Language

Семантике ресурсов службы учёта контингента и их методам уделено особое внимание. Однако в таком виде API предоставляет собой только CRUD (Create, Read, Update, Delete) интерфейс, который не отражает текущее состояние бизнес-процесса, т.е. так называемый stateless интерфейс [7]. Он обязывает разработчика клиентского приложения точно знать структуру ответов и когда какой метод можно вызывать.

Концептуальной особенностью данного сервиса является применение Hypermedia Application Language (HAL) [41]. Технически, это всего лишь дополнительная метайнформация в ответах сервера, необязательная к прочтению. Однако, данная информация позволит клиентским приложениям получать в

ответах корректные ссылки с учётом текущего состояния бизнес-процесса. Например, если клиент запросит информацию о студенте, в метайнформации ответа будут присутствовать ссылки на допустимые действия с данным студентом в данный момент времени для данного клиента. Например, ссылку на команду отчисления, ссылку на полный профиль персоны, ссылку на список всех групп, в которых студент учится и т.д. «Hypermedia» в идеале должна позволить разрабатывать службы с «самодокументирующимся» API, т.е. интуитивно понятным для разработчиков клиентских приложений. Интуитивности также способствует следующее правило в «Hypermedia» – наличие корневого ресурса (точки входа), который позволит пользователю, при наличии соответствующих прав, из неё добраться до любого ресурса в службе. Интерфейс может в полной мере считаться RESTful, если он реализует Hypermedia в том или ином виде [15].

Главная точка входа в сервис учёта контингента находится по адресу `/core/v1/`. Она носит исключительно описательный характер для разработчиков и клиентских приложений. Ответ сервера приведён в листинге 1.

```
{
  "description": "core service of IIPO v1",
  "_links": {
    "self": {
      "href": "/core/v1"
    },
    "people": {
      "href":
"/core/v1/people{?cn,givenName,sn,initials,mail,title}",
      "templated": true,
      "method": "GET"
    },
    "groups": {
      "href": "/core/v1/groups{?name}",
      "templated": true,
      "method": "GET"
    }
  }
}
```

Листинг 1. Точка входа службы учёта контингента

Данная точка входа уже описывает доступные ресурсы:

- а) `people` – список всех персон на факультете;
- б) `groups` – список учебных групп.

Согласно *рекомендациям* стандарта, Hypermedia Application Language [41], документ в формате HAL должен содержать ссылку на самого же себя в поле *_links.self.href*. Данная рекомендация соблюдена для всех ресурсов, предоставляемых службой.

Поле *method* никак не регламентируется стандартом HAL, однако было принято решение внедрить его во все ответы сервера. Это поле внесёт большую ясность того, что можно сделать с каждым ресурсом для конечных разработчиков, тем самым позволяя API быть самодокументирующимся.

Поле *templated* является стандартным и опциональным и означает, что в поле *href* на том же уровне представлена не полноценная ссылка, а её шаблон. Например, для ресурса *people* допустима необязательная часть из различного набора поисковых запросов. Например, можно запросить список всех преподавателей на кафедре, фамилия которых начинается на букву П, обратившись по следующему URL: */core/v1/people?title=Преподаватель&sn=П**

Если перейти по указанному URL, то сервер отдаст HAL документ, приведенный в листинге 2. Настоящие имена персон убраны, часть данных удалена для краткости.

```
{
  "_links": {
    "self": {
      "href":
"/core/v1/people?title=%D0%9F%D1%80%D0%B5%D0%BF%D0%BE%D0%B4%D0%B0%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C&sn=%D0%9F*"
    },
    "create": {
      "href": "/core/v1/people",
      "method": "POST"
    },
    "find": {
      "href":
"/core/v1/people{?uid,givenName,sn,initials,mail,title,cn}",
      "templated": true,
      "method": "GET"
    }
  },
  "_embedded": {
    "people": [
      {
        "_links": {
          "self": {
```

```

        "href": "/core/v1/people/1afd5c6d-cf2b-46ee-805d-
25ff27dd6f14"
      },
      "sn": "...",
      "givenName": "...",
      "initials": "...",
      "cn": "...",
      "title": [
        "Доцент",
        "Преподаватель"
      ],
      "uid": "1afd5c6d-cf2b-46ee-805d-25ff27dd6f14",
      "displayName": "..."
    },
    {
      "_links": {
        "self": {
          "href": "/core/v1/people/1b3b8494-dfac-4f00-8adc-
6980b64064e0"
        }
      },
      "sn": "...",
      "givenName": "...",
      "initials": "...",
      "cn": "...",
      "title": [
        "Ассистент",
        "Преподаватель"
      ],
      "uid": "1b3b8494-dfac-4f00-8adc-6980b64064e0",
      "displayName": "..."
    }
  ]
}

```

*Листинг 2. Пример ответа ресурса /core/v1/people?title=Преподаватель&sn=П**

Данный HAL документ можно рассматривать результат поиска по коллекции *people*. Эта коллекция содержит в себе вложенные сущности, следовательно, согласно правилам Hypermedia Application Language, ссылки на полноценные ресурсы с основными данными хранятся в массиве *_embedded*. Допустимые операции над всей коллекцией *people*, находятся в поле *_links*, а именно, создание персон и поиск. Важно отметить, что было принято решение запрета на прямое удаление пользователей через API. Удаление производится только в ручном режиме, только администратором и только в исключительных

случаях. Информация о персоне должна собираться как можно полнее и храниться как можно дольше.

Используя коллекцию `people`, можно найти любую персону, зарегистрированную на факультете и перейти на страницу с кратким профилем, которая доступна для широкого круга лиц и содержит несекретные персональные данные. Пример краткого профиля приведён в листинге 3.

```
{
  "title": "Студент",
  "uid": "9bd4defb-e9da-4b12-9a86-3176095f6f75",
  "mail": [
    "bstu@vasin.space",
    "penfriend2@yandex.ru"
  ],
  "displayName": "Васин Антон Вадимович",
  "sn": "Васин",
  "initials": "Вадимович",
  "cn": "avasin",
  "givenName": "Антон",
  "isActive": true,
  "birthDate": "25.4.1994",
  "_links": {
    "self": {
      "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-3176095f6f75"
    },
    "profile": {
      "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-3176095f6f75/profile"
    },
    "groups": [
      {
        "id": 37,
        "name": "15-При (мг) ",
        "links": {
          "self": "/core/v1/groups/37"
        }
      }
    ]
  }
}
```

Листинг 3. Краткий профиль студента

Для получения подробного профиля (ссылка описана в поле `_links.profile.href`) нужно обладать соответствующими правами. На данный момент, профили персон могут смотреть только их владельцы и преподаватели, не являющиеся тестовыми. Пример полного профиля приведён в листинге 4.

```
{
```

```

"homePhone": "(4832) 65-...-...",
"postalAddress": "г. Брянск, ул. ..., д. ..., кв. ...",
"mobile": "+7-906-...-...-...",
"title": "Студент",
"uid": "9bd4defb-e9da-4b12-9a86-3176095f6f75",
"mail": [
  "bstu@vasin.space",
  "penfriend2@yandex.ru"
],
"displayName": "Васин Антон Вадимович",
"sn": "Васин",
"initials": "Вадимович",
"cn": "avasin",
"givenName": "Антон",
"jpegPhoto": "",
"birthDate": "25.4.1994",
"isActive": true,
"_links": {
  "self": {
    "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-3176095f6f75/profile"
  },
  "shortProfile": {
    "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-3176095f6f75/"
  },
  "update": {
    "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-3176095f6f75/profile",
    "method": "PATCH"
  },
  "groups": [
    {
      "id": 37,
      "name": "15-При (мг)",
      "links": {
        "self": "/core/v1/groups/37"
      }
    }
  ]
}
}

```

Листинг 4. Полный профиль персоны

Краткий и полный профили содержат ссылки на группы, в которых учится персона. Документ HAL, описывающий группу со списком студентов (все кроме одного убраны для краткости), куратором и старостой, представлен в листинге 5.

```

{
  "id": 37,
  "name": "15-При (мг)",
  "curatorUid": "5e25d715-e5e0-42d2-b81e-ab59349a783a",
  "headUid": "9bd4defb-e9da-4b12-9a86-3176095f6f75",

```

```

"finishedEducation": false,
"type": "Магистратура",
"_links": {
  "self": {
    "href": "/core/v1/groups/37"
  },
  "update": {
    "href": "/core/v1/groups/37",
    "method": "PATCH"
  },
  "includeStudent": {
    "href": "/core/v1/groups/37/students/{uid}",
    "method": "POST",
    "template": true,
    "emptyBody": true
  },
  "assignHead": {
    "href": "/core/v1/groups/37/head/{uid}",
    "method": "POST",
    "template": true,
    "emptyBody": true
  },
  "assignCurator": {
    "href": "/core/v1/groups/37/curator/{uid}",
    "method": "POST",
    "template": true,
    "emptyBody": true
  }
},
"_embedded": {
  "students": [
    {
      "_links": {
        "self": {
          "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-3176095f6f75"
        },
        "profile": {
          "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-3176095f6f75/profile",
          "method": "GET"
        },
        "exclude": {
          "href": "/core/v1/groups/37/students/9bd4defb-e9da-4b12-9a86-3176095f6f75",
          "method": "DELETE"
        }
      },
      "uid": "9bd4defb-e9da-4b12-9a86-3176095f6f75",
      "givenName": "Антон",
      "sn": "Васин",
      "initials": "Вадимович",
      "displayName": "Васин Антон Вадимович"
    }
  ]
}

```

```

],
"curator": [
  {
    "_links": {
      "self": {
        "href": "/core/v1/people/5e25d715-e5e0-42d2-b81e-
ab59349a783a"
      },
      "profile": {
        "href": "/core/v1/people/5e25d715-e5e0-42d2-b81e-
ab59349a783a/profile",
        "method": "GET"
      },
      "remove": {
        "href": "/core/v1/groups/37/curator",
        "method": "DELETE"
      }
    },
    "uid": "5e25d715-e5e0-42d2-b81e-ab59349a783a",
    "givenName": "...",
    "sn": "...",
    "initials": "...",
    "displayName": "..."
  }
],
"head": [
  {
    "_links": {
      "self": {
        "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-
3176095f6f75"
      },
      "profile": {
        "href": "/core/v1/people/9bd4defb-e9da-4b12-9a86-
3176095f6f75/profile",
        "method": "GET"
      },
      "remove": {
        "href": "/core/v1/groups/37/head",
        "method": "DELETE"
      }
    },
    "uid": "9bd4defb-e9da-4b12-9a86-3176095f6f75",
    "givenName": "Антон",
    "sn": "Васин",
    "initials": "Вадимович",
    "displayName": "Васин Антон Вадимович"
  }
]
}
}

```

Листинг 5. Описание учебной группы

Данный документ уже содержит информацию о том, как можно включить студента в группу, назначить старосту, назначить куратора и исключить из группы конкретного студента.

Всё вышеописанное демонстрирует, как концепция Hypermedia Application Language была применена на практике и позволила создать самодокументирующийся программный интерфейс службы учёта контингента. Интерфейс отражает текущее состояние бизнес-процесса, а также содержит корневой документ, позволяющий прийти в любую точку API.

3.3.7. Системные события и интеграция с ESB

Сервисная шина предприятия имеет функцию широковещательной рассылки сообщений с помощью механизма «Темы и события» [21]. Служба учёта контингента посылает сообщения в шину после наступления следующих событий:

- а) создание пользователя (тема «`/core/people/created`»);
- б) изменение полей пользователя («`/core/people/modified`»);
- в) создание учебной группы («`/core/group/created`»);
- г) редактирование учебной группы («`/core/group/modified`»);
- д) добавление студента в учебную группу («`/core/group/student-added`»);
- е) удаление студента из группы («`/core/group/student-excluded`»).

Подписка на эти события позволит заинтересованным сервисам в реактивном режиме отслеживать изменения данных. Также было учтено, что зависимый сервис по каким-либо причинам может быть недоступен (завершился в аварийном режиме, находится в обслуживании, возникли проблемы с сетью). Для таких случаев в шине имеется функция гарантированной доставки сообщений при помощи механизма хранилищ сообщений [16].

В общем виде процесс интеграции нового подписчика выглядит так:

- а) служба-подписчик предоставляет окончечную точку, обрабатывающую POST-запрос;

- б) администратор шины создает хранилище сообщений и связывает его с оконечной точкой подписчика;
- в) происходит регистрация очередной службы в соответствующей теме.

Сам сервис контингента зарегистрирован в шине под названием «core» при помощи механизмов для работы с API [19].

На данный момент в платформу успешно интегрирована уже существующая система учёта посещаемости, которая уведомляется при изменении данных в службе контингента. Если сервис недоступен, шина пытается доставить сообщение раз в минуту в течении 10 дней. После этого момента, попытки доставки прекращаются, но сообщения продолжают сохраняться в хранилище сообщений. Администратор шины может в любой момент включить процессор сообщений и передать скопившийся массив стороннему сервису.

Формат передаваемых сообщений следующий:

- а) subject – uid пользователя, инициировавшего событие;
- б) timestamp – время, когда событие было инициировано;
- в) topic – название события;
- г) message – сообщение.

Формат сообщения зависит от типа события, но обычно имеет следующую структуру: id созданного или измененного ресурса, список изменений (старое и новые значения полей). Примеры сообщений приведены в листингах 6 и 7.

```
{
  "subject": "22ae2828-ab2f-4ef3-80c2-d065e61ca2b5",
  "timestamp": "2017-05-23T07:46:53.356Z",
  "topic": "core/group/name-changed",
  "message": {
    "groupId": "61",
    "changes": {
      "name": {
        "old": "16-Тест (мг) ",
        "new": "16-Тест"
      }
    }
  }
}
```

Листинг 6. Уведомление об изменении имени группы

```
{
  "subject": "e3499d81-74e2-4834-acb1-8dcac722d85b",
  "timestamp": "2017-05-19T21:42:06.462Z",
```

```

"topic": "core/people/created",
"message": {
  "uid": "026b8e32-b5f7-433d-8031-4d958ae43199"
}
}

```

Листинг 7. Уведомление о создании нового пользователя

3.4. Общая концепция безопасности в платформе

В начале проектирования предполагалось, что службы будут разрабатываться как некие коллекции ресурсов, т.е. служба в чистом виде не должна проверять доступ субъектов. Планировалось, что контроль доступа будет переложен на шину согласно упрощенной версии паттерна безопасности «Constrained Access Delegation» [3]. Схематично паттерн изображен на диаграмме последовательности на рис. 6. Этот паттерн имеет следующие плюсы:

- а) централизация всех политик безопасности в одном месте;
- б) существенно упрощена логика работы сервисов;
- в) можно интегрировать сторонние сервисы и наложить на них ограничения, согласно политикам безопасности.

Однако в процессе разработки выяснилось, что такой подход не является универсальным. Более того, данный паттерн принципиально не подходит именно для сервиса учёта контингента, так как при обращении к его ресурсам возникают циклические зависимости. Это происходит потому, что для каждого защищённого ресурса в сервисе учёта контингента должна быть описана своя политика, определяемая набором атрибутов субъекта, ресурса и действия над ним. А для того чтобы получить атрибуты субъекта, нужно обратиться к сервису учёта контингента. В итоге, если обращаются именно к ресурсам сервиса учёта контингента, сервис авторизации в процессе решения будет вынужден опять обратиться к сервису учёта контингента. Таким образом, получаем цикл.

Для решения проблемы циклических зависимостей было принято решение снять обязанность обращения к службе авторизации с шины на сам сервис. Если идёт доступ к публичному ресурсу, никаких политик не вычисляется, если идёт

доступ к защищенному ресурсу (см. рис. 7), то политики вычисляются на основе атрибутов из открытых ресурсов (см. рис. 8).

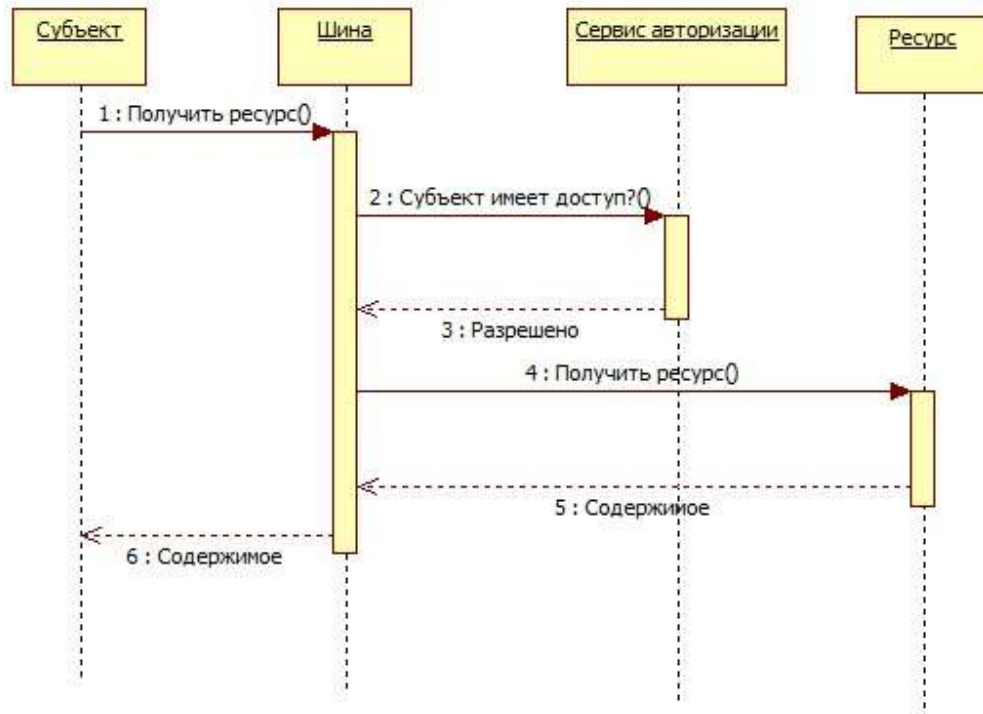


Рис. 6. Последовательность событий в упрощенной версии паттерна «Constrained Access Delegation»

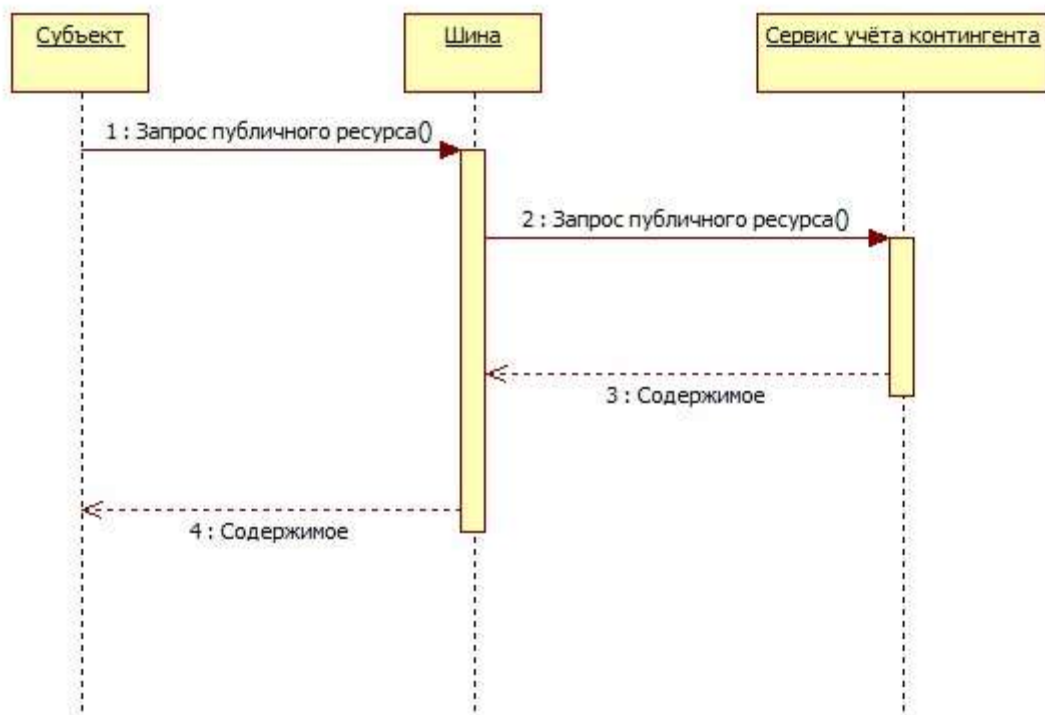


Рис. 7. Запрос публичных ресурсов

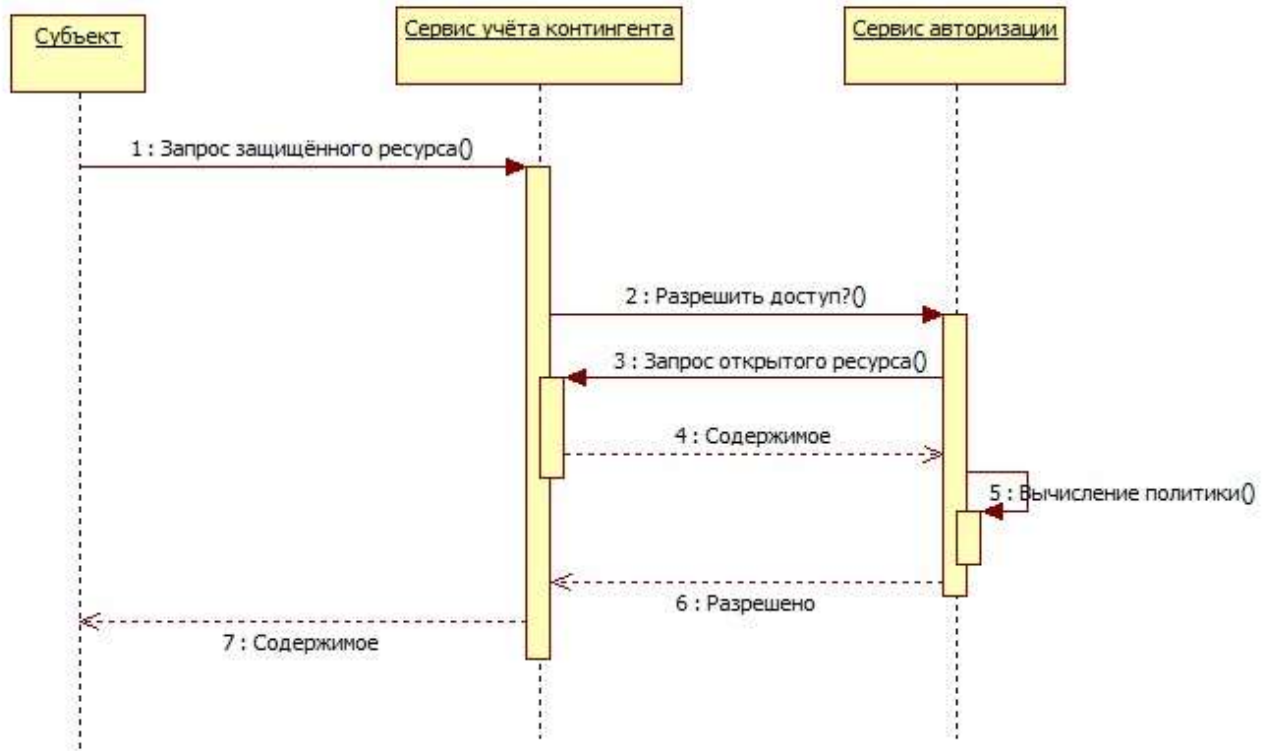


Рис. 8. Запрос ресурсов, требующих авторизованный доступ (шина скрыта для краткости)

3.5. Микросервис аутентификации

Главная функция, которую выполняет микросервис аутентификации – это выдача и валидация токенов в формате JSON Web Tokens (JWT). Токены выдаются после успешной проверки пары логин-пароль, через обращение к OpenLDAP, и подписываются симметричной цифровой подписью [40]. Секретный ключ генерируется случайно при старте микросервиса, оставаясь известным только ему. Именно поэтому только эта служба может произвести валидацию токена. Также планируется, что с течением времени в службу будут добавлены функции отзыва токенов. Эта функция полезна как с точки зрения общей безопасности, так и для конечных пользователей, желающих выйти на всех устройствах.

3.5.1. Форма аутентификации

Ко всему прочему, был реализован прототип единой формы аутентификации (см. рис. 9). Форма предназначена для аутентификации в кафедральных веб-приложениях, т. е. размещённых на основном домене. Сама форма представляет статичное веб-приложение, которое запрашивает токен у службы аутентификации и сохраняет его в локальное хранилище (*localStorage*) браузера в поле *user-token*. Этот токен может использоваться другими приложениями, размещённых на том же домене. Если аутентификация прошла успешно, форма перенаправляет браузер по указанному URL в параметре *redirect*.

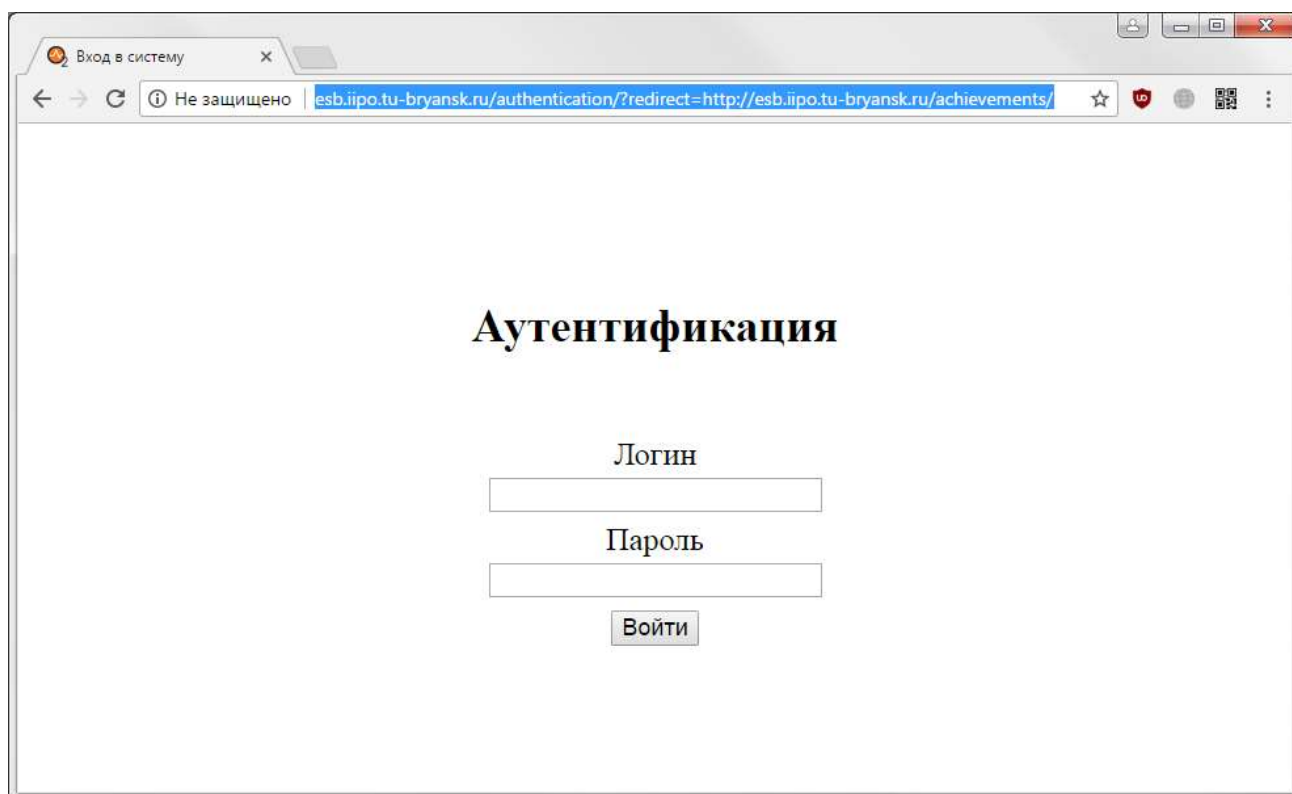
The image is a screenshot of a web browser window. The address bar shows the URL 'esb.iipo.tu-bryansk.ru/authentication/?redirect=http://esb.iipo.tu-bryansk.ru/achievements/'. The page content is centered and features the title 'Аутентификация' in a large, bold, black serif font. Below the title are two input fields: the first is labeled 'Логин' (Login) and the second is labeled 'Пароль' (Password). Below these fields is a button labeled 'Войти' (Login). The browser's address bar also shows a security warning 'Не защищено' (Not secure) and a tab titled 'Вход в систему' (System login).

Рис. 9. Единая форма аутентификации в кафедральных веб-приложениях

3.5.2. REST API микросервиса аутентификации

Список методов, который предоставляет микросервис аутентификации, приведён в таблице 4.

Список методов, предоставляемых микросервисом аутентификации

Ресурс	HTTP метод	Комментарий
/authentication/[index.html]	GET	Веб-страница аутентификации
/authentication/authenticate/	POST	Запрос на выдачу токена по паре логин-пароль
/authentication/change-password/	POST	Метод, задающий новый пароль пользователя, используя старый.
/authentication/validate/	POST	Проверяет валидность выданного токена.

3.6. Сервис авторизации

Сервис авторизации служит для централизованного управления политиками всей системы. Его основная функция ответить на вопрос: «Имеет ли субъект А право совершить действие В над ресурсом С?».

3.6.1. Attribute Based Access Control

Существует распространённая модель разграничения прав под названием «Role Based Access Control». В рамках этой модели субъектам назначаются роли, а самим ролям присваиваются действия, которые эти роли могут совершать. В итоге задача поиска ответа на вышеописанный вопрос выглядит так: нужно выяснить, имеется ли запрашиваемое действие разрешенным для хотя бы одной роли, к которым принадлежит пользователь [16].

Данная модель имеет главный плюс – она очень простая и высокопроизводительная. Однако у неё есть ряд ограничений. Например, с её помощью нельзя адекватным образом описать бизнес-правило: «засчитать лабораторную работу может только тот преподаватель, который в текущем семестре ведёт назначенный ему предмет у назначенной ему группы».

Для описания подобных бизнес-правил был изобретён подход «Attribute Based Access Control» и описывающий его стандарт XACML [11].

Стандарт является детально проработанным. С его помощью можно описывать политики, учитывающие даже параметры окружающей среды, например, текущее время. Сам стандарт поддерживает такие сущности как «Правило», «Политика», «Группа политик», «Алгоритм комбинации правил», «Алгоритм комбинации политик» и др.

Пример группы политик, которую можно описать с помощью этого стандарта, изображен на рис. 10.

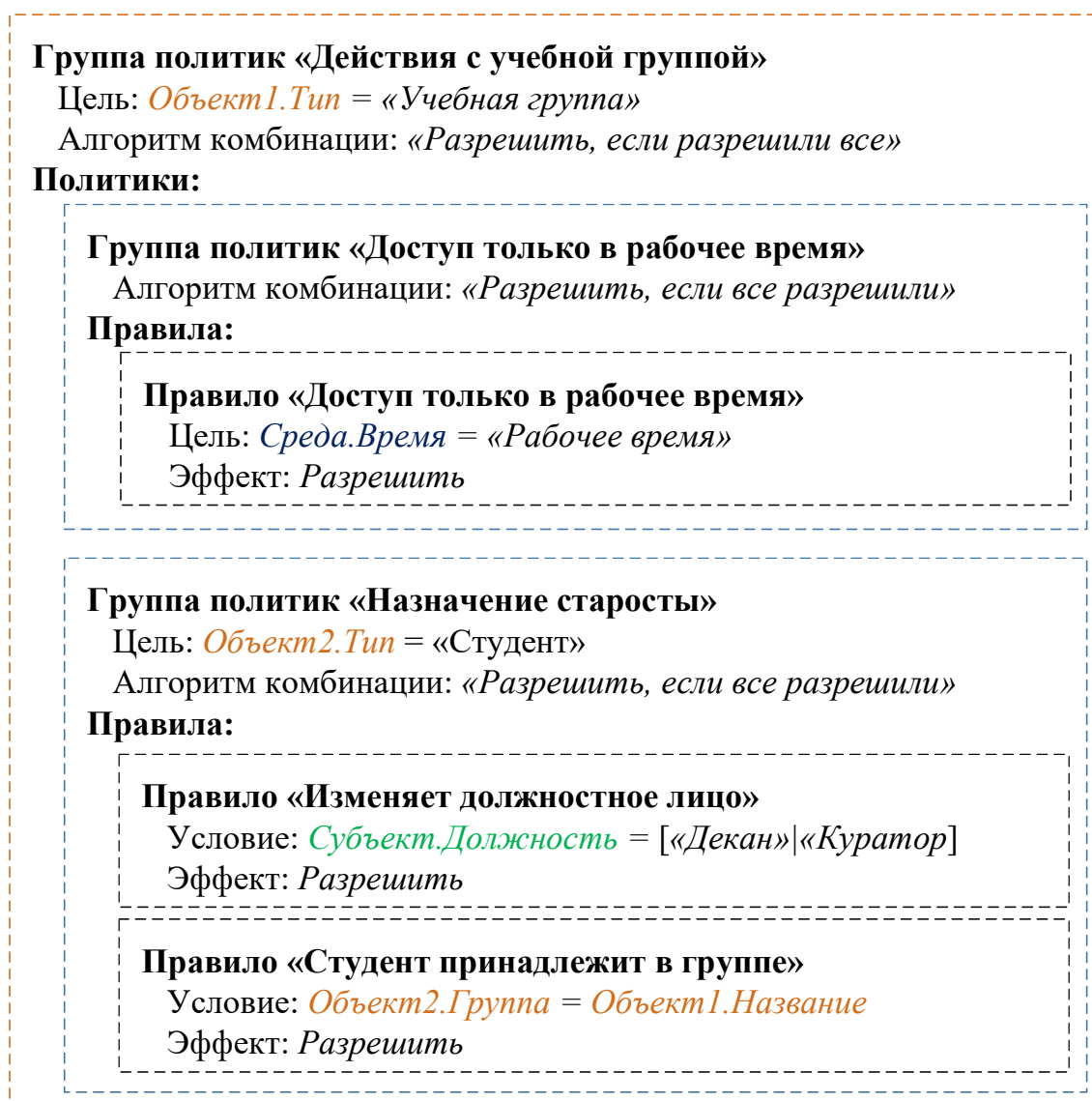


Рис. 10. Пример бизнес-правил, которых сложно описать с помощью подхода RBAC

Компания «WSO2» имеет в линейке своих продуктов службу «Identity Server» [22], который в полной мере реализует стандарт XACML. Сервисная шина «Enterprise Service Bus» от той же компании, даже имеет специальный медиатор «Entitlement» [4], который может обращаться к «Identity Server», запрашивая

разрешение. Однако, интеграция этих двух продуктов задача сама по себе довольно обширная и трудоёмкая. В качестве более простой альтернативы, был разработан очень упрощённый вариант сервиса авторизации, политики которого хранятся не в XACML формате, а в виде обычных асинхронных JavaScript функций.

3.6.2. Общий вид политик безопасности

Политики представляют собой обычные JavaScript функции, которые возвращают объект типа *Promise* [14]. Это необходимо потому, что политика может не обязательно выполняется мгновенно, в зависимости от бизнес-правила, политика может сделать дополнительные асинхронные запросы на различные сервисы для получения дополнительной информации. Пример политик безопасности, которые описывают возможные действия над персонами в службе контингента, приведён в листинге 8.

```
async function modifyProfile(subject) {
  const isTeacher = checkTeacher(subject);
  if (isTeacher && !isTest(subject))
    return {decision: 'allow'};

  return {
    decision: 'deny',
    reason: 'only real teachers have write access to profiles'
  }
}

module.exports = {
  "create person": createPerson,
  "get person's private profile": readProfile,
  "modify person's private profile": modifyProfile
};
```

Листинг 8. Фрагмент кода, содержащий одну из политик безопасности

Для увеличения ясности кода было принято решение давать максимально понятные и подробные для человека названия политикам.

3.6.3. Политика создания персон

На данном этапе развития системы, политика создания новых персон в общем хранилище описывается следующим бизнес-правилом: «Только настоящие

преподаватели могут создавать пользователей». Код политики представлен в листинге 9.

```
function checkTeacher(subject) {
    return utils.contains(subject.title, 'Преподаватель');
}

function isTest(subject) {
    return utils.contains(subject.title, 'тест');
}

async function createPerson(subject) {
    const isTeacher = checkTeacher(subject);
    if (isTeacher && !isTest(subject))
        return {decision: 'allow'};

    return {
        decision: 'deny',
        reason: 'only real teachers can create persons'
    }
}
```

Листинг 9. Код политики безопасности создания новых персон.

3.6.4. Политика доступа на чтение персональных данных

Текущая политика безопасности описывается следующим правилом.

Субъект имеет доступ на чтение полного профиля, если выполняется одно из следующих условий:

- а) субъект является владельцем профиля;
- б) субъект является настоящим преподавателем;
- в) субъект является тестовым преподавателем и запрашивает доступ к тестовому профилю.

Иные запросы считаются не авторизованными, например, запрос доступа на чтение тестовым преподавателем к доступу профиля настоящего студента.

Код вышеописанной политики безопасности приведен в листинге 10.

```
const Person =
require('../.../pip/PolicyInformationPoint').Person;

async function readProfile(subject, resources) {
    const isMe = utils.commonElements(subject.uid,
resources.profile).length > 0;
    if (isMe) return {decision: 'allow'};
```

```

const isTeacher = checkTeacher(subject);
if (isTeacher) {
  if (!isTest(subject))
    return {decision: 'allow'};

  const profile = await Person.findById(resources.profile);
  if (isTest(profile))
    return {decision: 'allow'};

  return {
    decision: 'deny',
    reason: 'test teachers have read access only to test
students'
  };
}

return {
  decision: 'deny',
  reason: 'only real teachers and owners have read access to
profile'
};
}

```

Листинг 10. Политика безопасности на чтение персональных данных пользователей

3.6.5. Политика доступа на модификацию персональных данных

Эта политика описывается следующим правилом: «Персональные данные могут редактировать только настоящие преподаватели». Стоит отметить, что владельцам профилей, редактирование персональных данных запрещено во избежание внесения ими заведомо ложных данных или эксплуатации потенциальных уязвимостей, связанных с повышением собственных привилегий.

Код политики приведён в листинге 11.

```

async function modifyProfile(subject) {
  const isTeacher = checkTeacher(subject);
  if (isTeacher && !isTest(subject))
    return {decision: 'allow'};

  return {
    decision: 'deny',
    reason: 'only real teachers have write access to profiles'
  }
}

```

Листинг 11. Политика безопасности на редактирование профилей студентов

3.6.6. Политика редактирования учебных групп

Сейчас сервис учёта контингента поддерживает следующие манипуляции над списком учебных групп и над каждой группой по отдельности:

- а) создать учебную группу;
- б) редактировать группу (название, uid куратора, uid старосты);
- в) добавление в группу студента;
- г) исключение студента из группы;
- д) назначение старосты;
- е) назначение куратора.

На данном этапе развития системы допустимо все эти действия объединить в одну политику, при которой любой настоящий преподаватель может выполнить вышеописанные манипуляции. В дальнейшем, политики безопасности для каждого действия будут сформированы отдельно по мере уточнения деталей бизнес-процессов.

Код единой политики безопасности для всех этих операций приведён в листинге 12.

```
async function teacherAccess(subject, resources) {
  const isTeacher = checkTeacher(subject);
  if (!isTeacher)
    return {
      decision: 'deny',
      reason: 'only teachers can modify groups'
    };

  if (!isTest(subject))
    return {decision: 'allow'};

  if (!('group' in resources)) {
    return {
      decision: 'deny',
      reason: 'test teachers not allowed to create groups'
    }
  }

  const group = await Group.findById(resources.group);
  if (group.name.toLowerCase().includes('тест'))
    return {decision: 'allow'};

  return {
    decision: 'deny',
```

```

    reason: 'test teachers can modify only test groups'
  }
}

module.exports = {
  "create group": teacherAccess,
  "patch group": teacherAccess,
  "include student into group": teacherAccess,
  "exclude student from group": teacherAccess,
  "assign curator to group": teacherAccess,
  "assign head to group": teacherAccess
};

```

Листинг 12. Политика безопасности редактирования учебных групп

3.7. Совместная интеграция трёх служб

Как уже было отмечено выше, предоставляя доступ к защищённым ресурсам, служба учёта контингента должна обратиться к сервису авторизации за разрешением. Общая схема работы представлена на рис. 11.

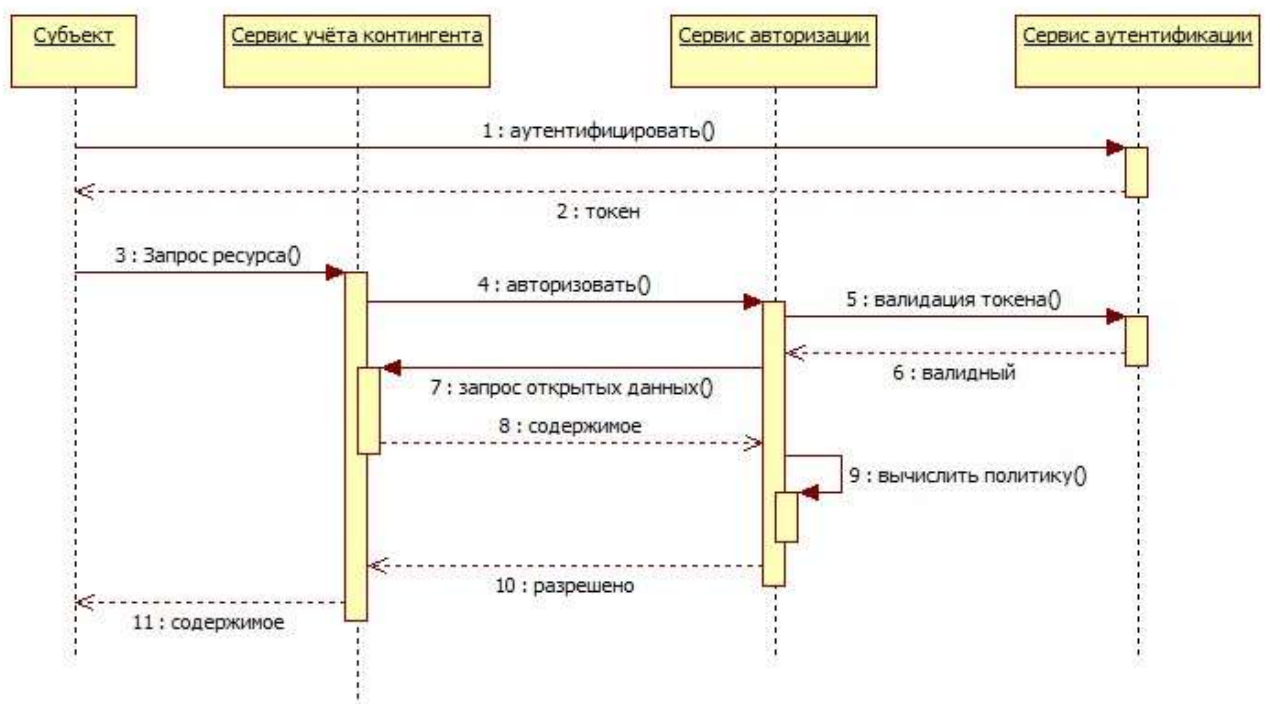


Рис. 11. Общая схема обработки запроса к ресурсу (шина скрыта для краткости)

Общий шаблон кода запроса разрешения на примере метода назначения куратора группе приведён в листинге.

```

// Общая функция, осуществляющая запрос авторизации к сервису
// авторизации
async function authorizeRequest(token, rule, resources) {
  if (resources === undefined)

```

```

    resources = {};

    if (!token)
        return {
            decision: 'deny',
            reason: 'provide jwt token inside Authorization header'
        };

    let response = await fetch(global.config['authorization-
service'], {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({
            token,
            rule,
            resources
        })
    });
    if (!response.ok)
        throw new Error("authorization request failed");

    return await response.json();
}

// Пример обработчика REST ресурса
app.express.post('/v1/groups/:groupId/curator/:uid', async (req,
res) => {
    try {
        let authResult = await authorization.authorizeRequest(
            req.headers.authorization, //здесь хранится JWT токен
            'assign curator to group', /*имя политики*/ {
                group: req.params.groupId, // атрибуты...
                curator: req.params.uid    // ...политики
            });

        if (authResult.decision !== 'allow')
            return res.status(401).json({error: authResult.reason});
        //
        // бизнес-логика обработки запроса
        //
    }
    catch (err) {
        res.status(500).json({error: err.message});
    }
});
});

```

Листинг 13. Общий шаблон запроса авторизации

3.8. Интеграция платформы с внешними системами

Как уже было отмечено выше, важной особенностью данного решения является использование службы каталогов OpenLDAP, реализующий протокол

LDAP. Этот протокол наиболее распространён в многопользовательских системах, которые поддерживают внешние хранилища пользователей. Для придания инфраструктуре законченного вида, в платформу были интегрированы такие системы как Atlassian Jira, Atlassian Confluence и GitLab.

3.8.1. Atlassian Jira

Atlassian Jira – это система управления проектами. Она позволяет создавать проекты, задачи, назначать исполнителей и контролировать сроки. Система будет использоваться при разработке различных выпускных квалификационных работ.

Также у Jira есть свой собственный API, что позволит в дальнейшем более тесно интегрировать её с календарем кафедральных событий и системой учёта успеваемости.

Jira полностью загружает из кафедрального OpenLDAP сервера информацию о группах пользователей (студенты, преподаватели, администраторы), а также информацию об отдельном пользователе. Пример загруженного профиля студента приведён на рис. 12.

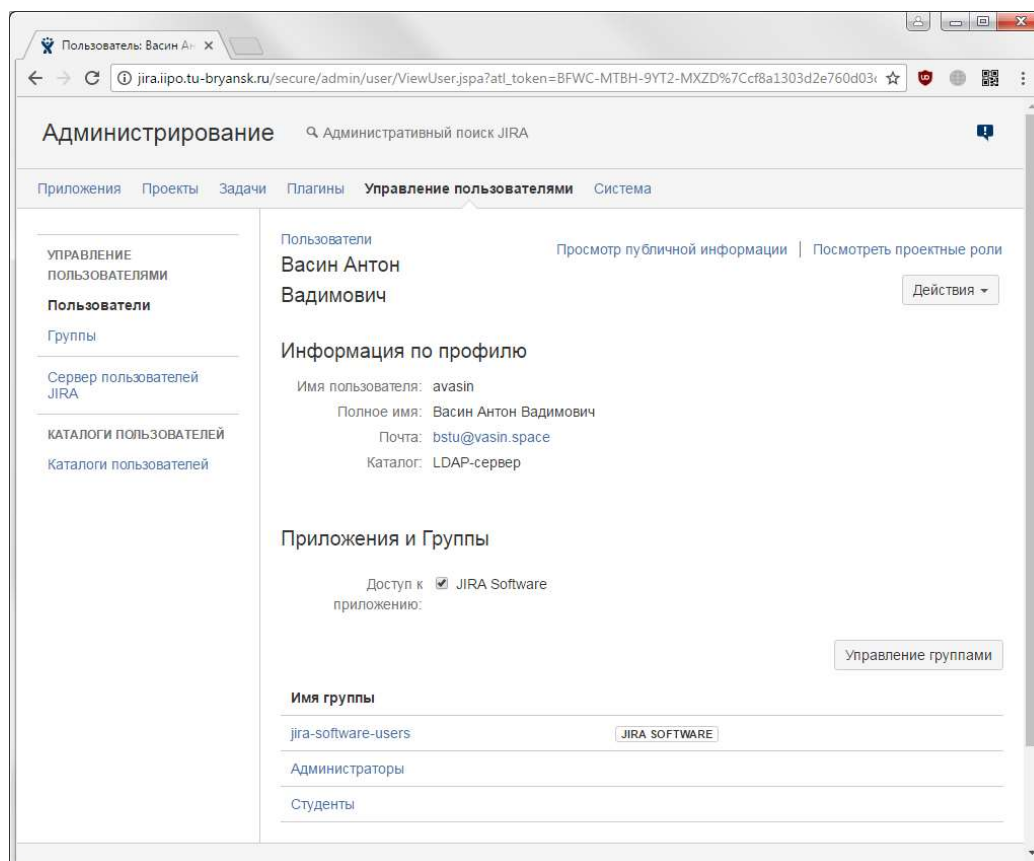


Рис. 12. Профиль студента, загруженный из OpenLDAP

3.8.2. Atlassian Confluence

Atlassian Confluence является системой управления документацией с поддержкой версионирования. Тесно интегрируется с Jira и LDAP системами.

Аналогично Jira, использует информацию о группах и персонах из OpenLDAP (см. рис. 13).

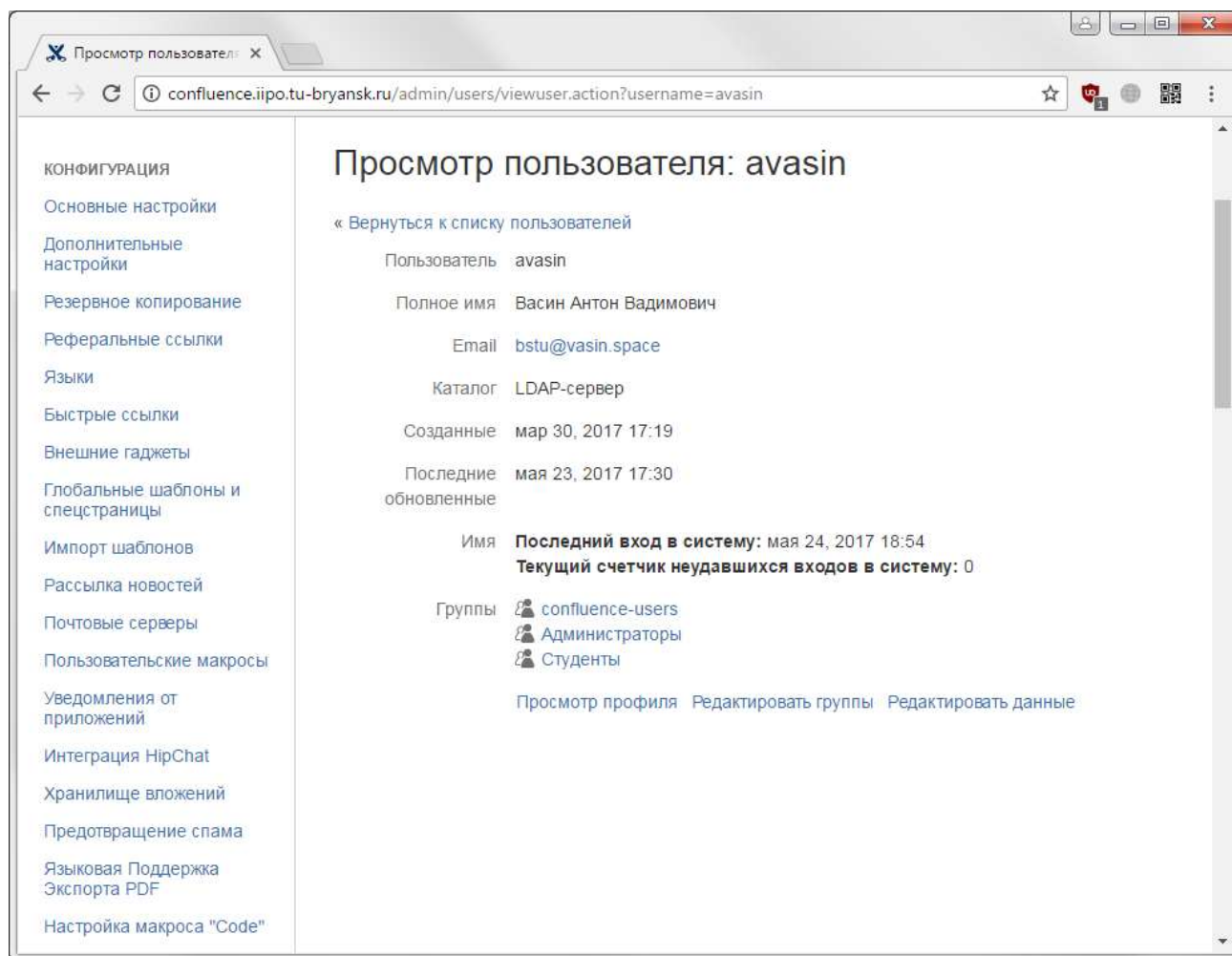


Рис. 13. Confluence, как и Jira поддерживает протокол LDAP

Jira и Confluence также были интегрированы совместно. Например, в ленте активности Jira отображается активность сразу в двух системах (см. рис. 14).

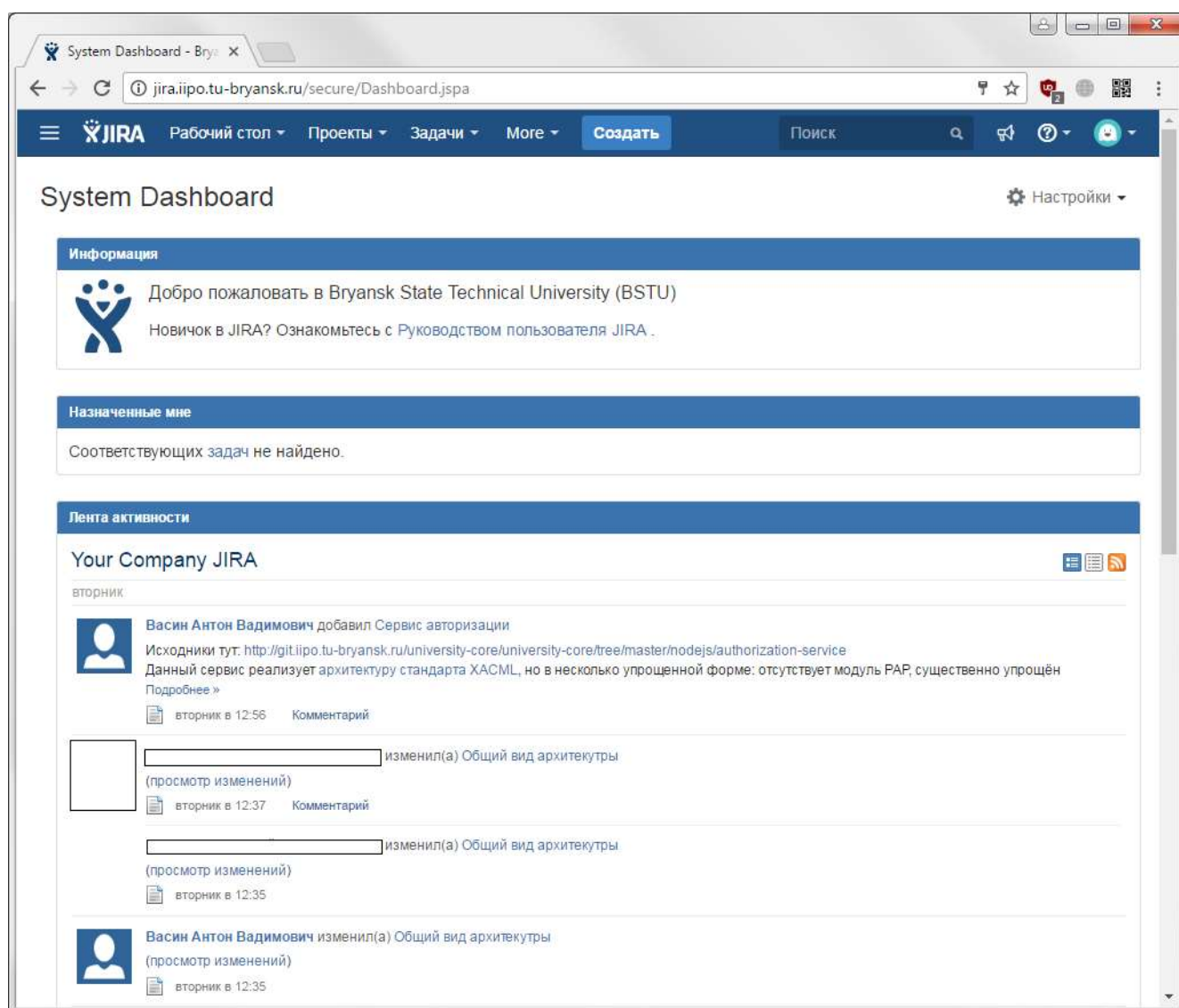


Рис. 14. Jira в ленте активности отображает работу в системе Confluence

3.8.3. GitLab

На кафедральном сервере также был развёрнут GitLab – сервер git репозиторий. Данный сервис уже используется для хранения исходных кодов платформы и конфигурационных файлов сервисной шины предприятия. В дальнейшем GitLab будет использоваться студентами для хранения кодовой базы своих учебных проектов. Он тоже подключен к OpenLDAP серверу (см. рис. 15).

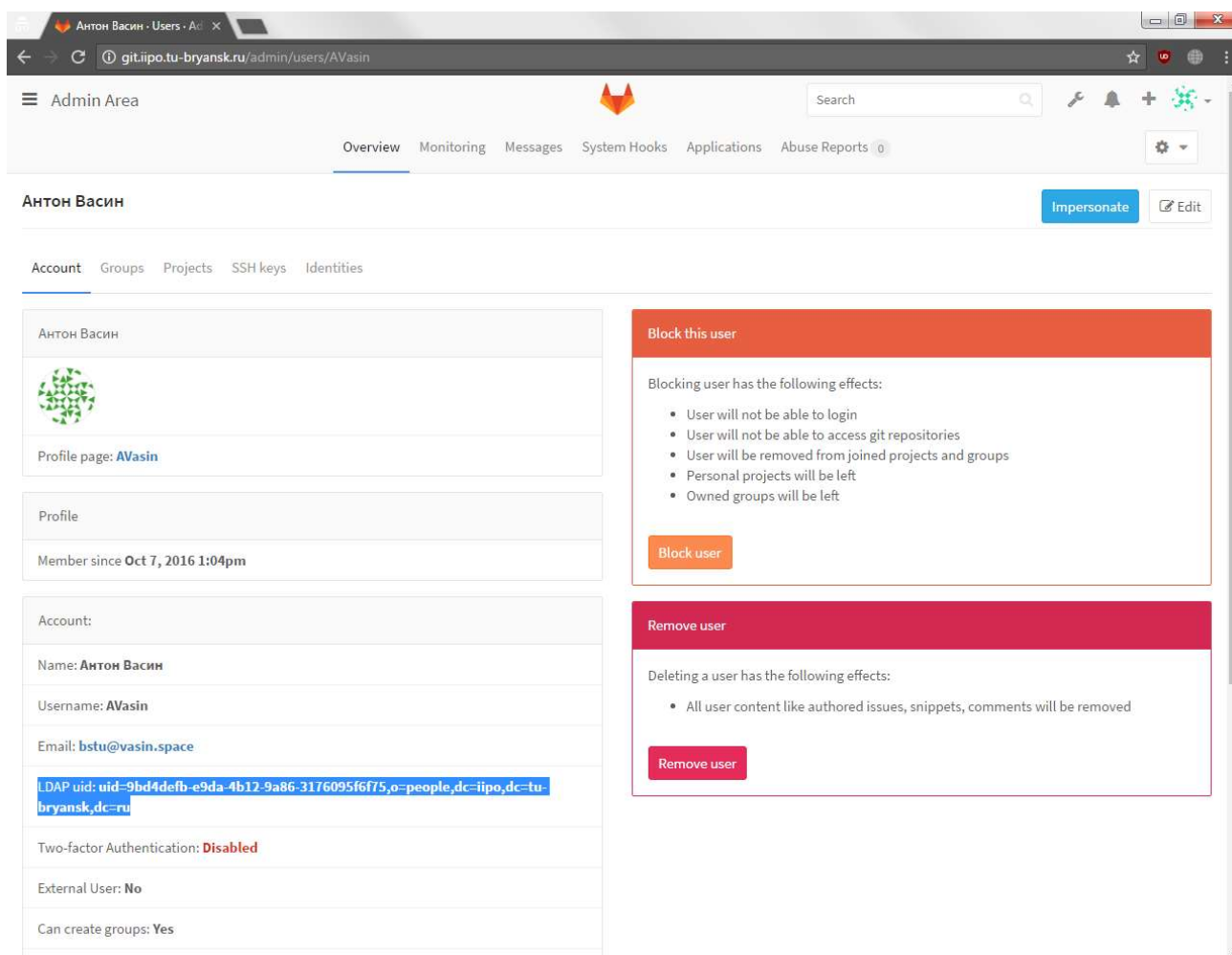


Рис. 15. GitLab использует кафедральный OpenLDAP как основной источник аккаунтов

3.9. Правила интеграции разработанных служб

На данный момент возможны несколько способов разработки и интеграции будущих сервисов и веб-приложений, разработанных специально для кафедры.

Первый способ заключается в разработке службы с незащищенными ресурсами, т.е. она будет предоставлять REST API не требующее авторизации. Разработчик должен будет клонировать репозиторий с сервисом авторизации, описывать новые политики безопасности, протестировать их на локальной версии сервиса авторизации. После успешного тестирования, разработчик в GitLab оформляет запрос на слияние в GitLab, с помощью функции Merge request [8]. После чего администратор проверяет политики на правильность и наличие уязвимостей, сливает изменения разработчика и обновляет службу авторизации. Когда служба разработчика объявляется законченной, администратор разворачивает её на кафедральном сервере и подключает её к шине, оборачивая её

незащищённые ресурсы средствами шины используя паттерн безопасности «Claim based Authorization» [2]. Диаграмма последовательности доступа к ресурсу согласно этому паттерну представлена на рис. 16. Если субъект не имеет доступа к ресурсу, то сервис авторизации в доступе откажет и действия 4, 5 и 6 выполнены не будут и субъект получит 401 ошибку протокола HTTP «Unauthorized». Данный способ является наиболее предпочтительным, в случае, когда прикладной сервис предоставляет небольшое количество ресурсов и является довольно простым.

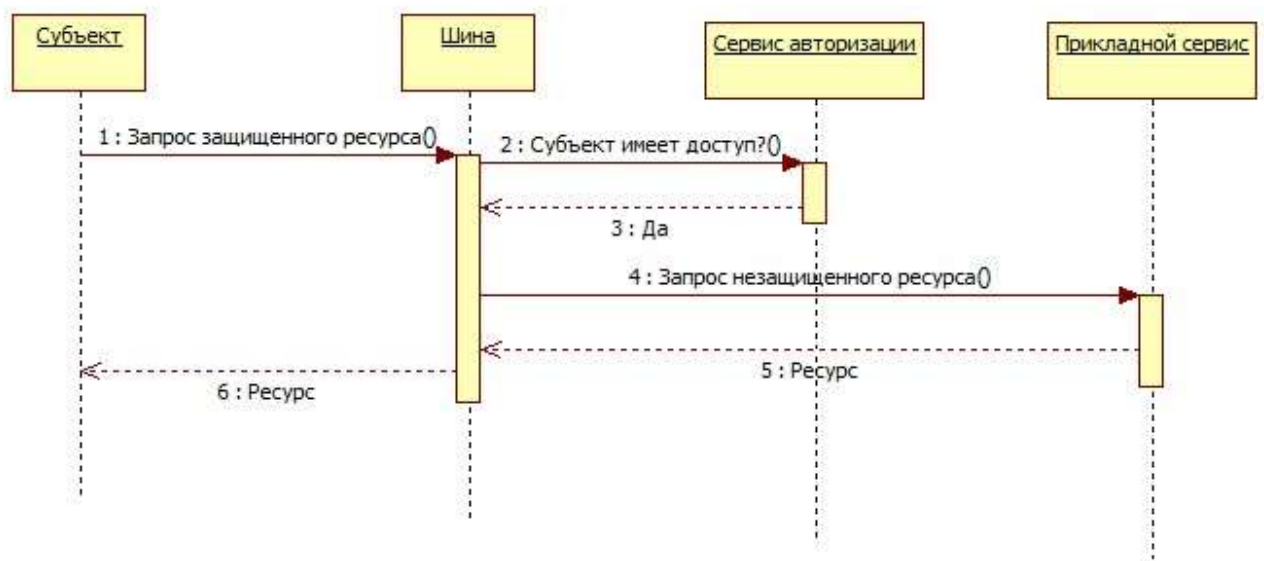


Рис. 16. Принцип работы паттерна «Claim based Authorization»

Второй способ является вариантом предыдущего. Отличие заключается в том, разработчик предоставляет уже защищенные ресурсы, т.е. его служба сама опрашивает сервис авторизации. Этот способ уже был применён в сервисе учёта контингента при разрешении циклических зависимостей. Общий принцип работы уже был представлен на рис. 11. Данный способ рекомендуется использовать в сервисах, предоставляющих большое количество ресурсов и обладающих сложной внутренней структурой.

Третий способ является крайне нежелательным, так как не позволит централизованно контролировать политики, но является допустимым, если необходимость обоснована. В этом случае сервис сам принимает решения о доступе пользователям к его ресурсам. Так, например, работают сторонние решения Jira, Confluence и GitLab: права доступа приходится настраивать в этих в

каждой системе по отдельности. Рекомендуется использовать этот способ только в случае крайней необходимости.

3.10. Планы развития системы

Первостепенные задачи, которые требуется решить на следующих этапах развития системы – это интеграция существующих кафедральных приложений учёта успеваемости и посещаемости в платформу, интеграция с системами деканата для синхронизации списков студентов и групп, а также разработка новых сервисов в рамках курсовых и дипломных работ, которые уже были объявлены как планируемые в краткосрочной перспективе в разделе 3.1.

Кроме того, нужно обеспечить более качественную защиту персональных данных, хранящихся в сервисе учёта контингента. Данная задача является достаточно объемной по требуемым ресурсам, поэтому в данной работе не рассматривалась.

3.11. Выводы по главе

В результате проделанной работы была разработана первая версия платформы, содержащая следующие базовые службы:

- а) сервис учёта контингента, хранящий общую и наиболее востребованную для будущих прикладных служб информацию о факультете;
- б) сервис аутентификации, обслуживающий токены безопасности;
- в) сервис авторизации, хранящий политики для всей платформы.

Были предложены основные правила интеграции новых сервисов в платформу, и защиты их ресурсов от неавторизованного доступа.

С данной платформой были интегрированы система управления проектами Jira, система управления документацией Confluence и сервер git-репозитория GitLab.

Таким образом, была разработана и развёрнута на кафедре полноценная экосистема, которая позволит системно автоматизировать бизнес-задачи кафедры.

4. УПРАВЛЕНИЕ ПРОЕКТОМ

Проведенный системный анализ проблематики предметной области позволил определить следующие этапы развития проекта:

- а) проектирование программной платформы;
- б) создание инфраструктуры;
- в) реализация базовых сервисов платформы;
- г) тестирование и внедрение платформы на кафедру.

Для успешного прохождения этих этапов необходимо привлечь исполнителей, выполняющие следующие роли: архитектор, системный администратор, разработчик серверных приложений, разработчик веб-приложений, специалист по тестированию.

4.1. Комбинированная матрица ответственности

Так как этапы и роли исполнителей достаточно чётко определены, можно сразу изобразить их в матрице ответственности проекта. Комбинированная с иерархической структурой работ и организационной структурой проекта матрица ответственности приведена на рис. 17.

4.2. Календарный план работ

Общий план проекта представлен на рис. 18. Стоит отметить, что ресурсы, задействованные на проекте, не могут работать более 10 часов в неделю из-за того, что большую часть своего времени исполнители посвящают непосредственному обучению и работе по специальности. Эта особенность отражена в виде 25% нагрузки каждого ресурса в плане работ.

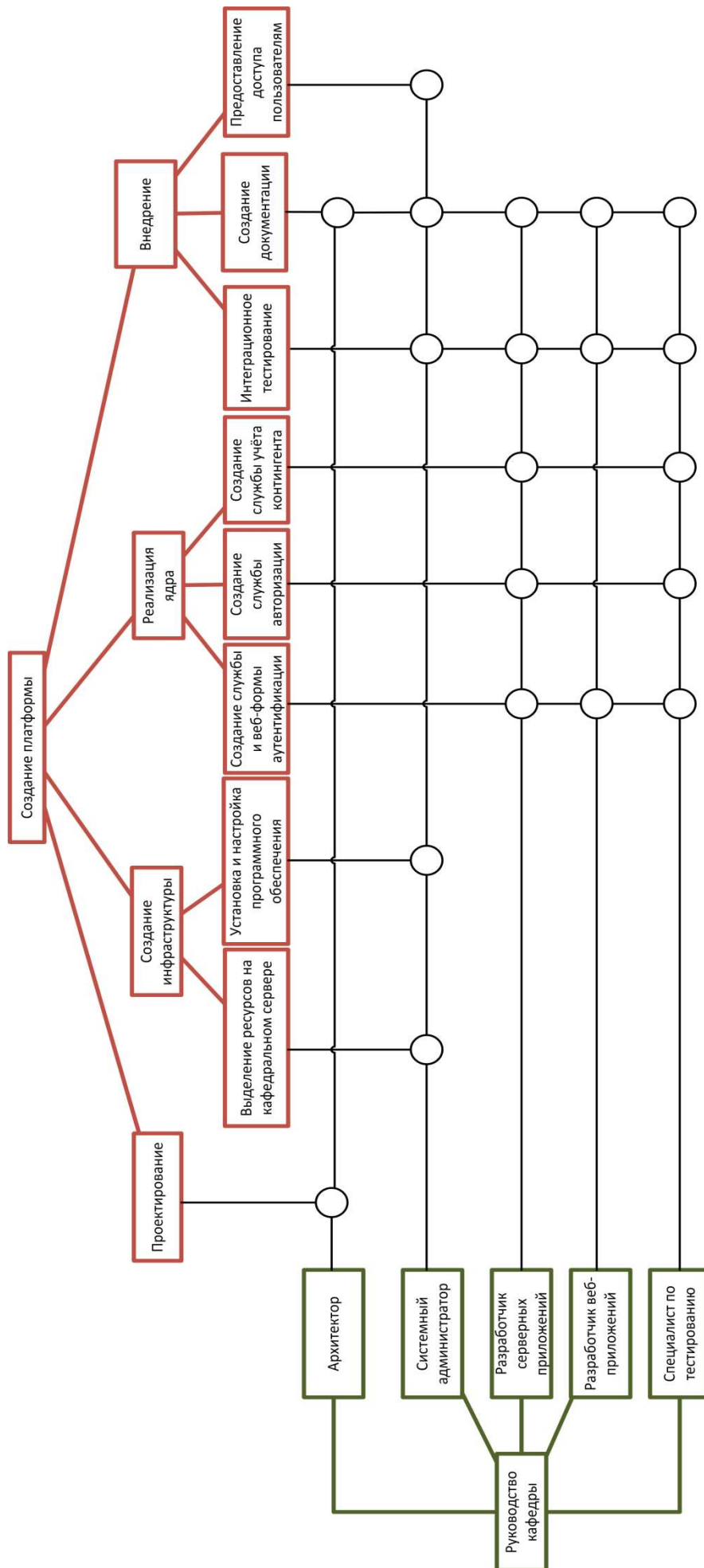


Рис. 17. Матрица ответственности проекта

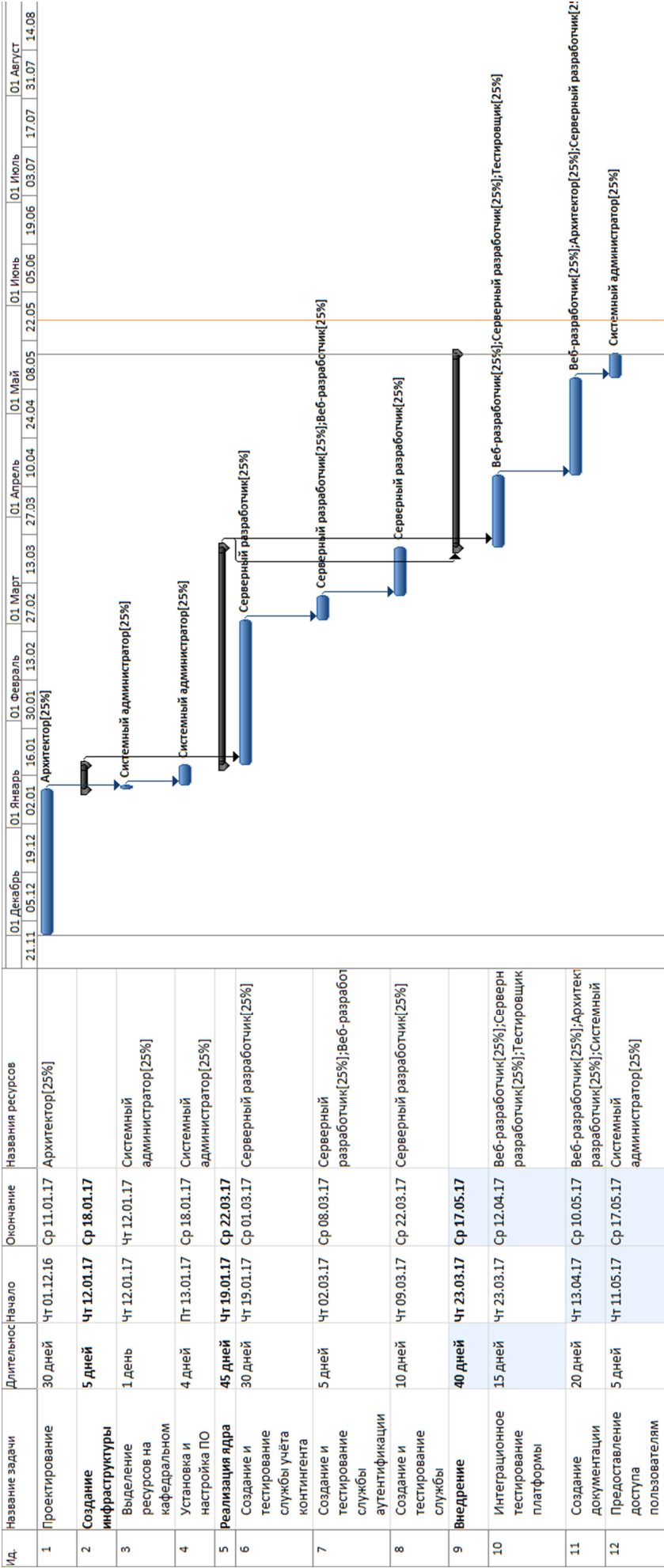


Рис. 18. План управления проектом по разработке кафедральной платформы

4.3. Управление рисками проекта

Согласно описанным рекомендациям в PMBOK [13], был проведён анализ рисков текущего проекта с целью их контроля. В задачи анализа входили:

- а) идентификация рисков;
- б) определения вероятности и степени воздействия каждого риска;
- в) определение порогового значения, для отсеивания малозначимых рисков
- г) определение мер воздействия на существенные риски.

4.3.1. Идентификация рисков

Анализ предметной области позволил определить следующие риски:

- а) возможна нехватка ресурсов интернет канала для связи с платформой извне;
- б) вероятен уход главного архитектора системы из проекта;
- в) может сложиться ситуация, в которой принцип работы ядра будет известен ограниченному кругу лиц;
- г) имеется шанс включения в проект недостаточно квалифицированных разработчиков;
- д) сервисная шина предприятия может оказаться узким местом в системе
- е) вероятно дублирования данных в службах;
- ж) возможно получение несогласованных данных из-за их распределенной природы.

4.3.2. Определение вероятности и степени воздействия рисков

Детальное изучение выявленных рисков позволило определить вероятности возникновения рисков, степень их воздействия, а также посчитать интегральную оценку. Результаты оценки приведены в таблице 5. Риски, интегральная оценка которых не превышает 0.5, принимаются; для остальных прорабатываются меры воздействия.

Таблица 5

Оценка характеристик идентифицированных рисков

Риск	Вероятность	Степень воздействия	Интегральная оценка
Возможна нехватка ресурсов интернет канала для связи с системой извне.	0,5	0,8	0,4
Вероятен уход главного архитектора системы из проекта	0,9	0,9	0,81
Может сложиться ситуация, в которой принцип работы ядра будет известен ограниченному кругу лиц	0,7	0,8	0,56
Имеется шанс включения в проект недостаточно квалифицированных разработчиков	0,7	0,8	0,56
ESB может оказаться узким местом в системе	0,5	0,8	0,4
Вероятно дублирования данных в службах	0,5	0,3	0,15
Возможно получение не консистентных данных из-за их распределенной природы.	0,8	0,9	0,72

Таким образом, были определены четыре основных риска. Для каждого из них были разработаны меры воздействия. Результаты приведены в таблице 6.

Меры воздействия на риски

Риск	Мера воздействия на риск
Вероятен уход главного архитектора системы из проекта	Минимизация: подготовка разработчиков, готовых перенять эту роль.
Может сложиться ситуация, в которой принцип работы ядра будет известен ограниченному кругу лиц	Минимизация: создание подробной документации о ядре и базы знаний о технологиях.
Вероятно включение в проект недостаточно квалифицированных разработчиков	Избежание: отбор ответственных и способных студентов
Возможно получение не консистентных данных из-за их распределенной природы.	Избежание: применение распределенных транзакций. Минимизация: применение механизмов гарантированной доставки сообщений в ESB. Минимизация: разработка служб с атомарными методами.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы была исследована учебная, учебно-методическая и организационно-методическая деятельность выпускающей кафедры. Сформулированы общие требования к программной платформе кафедральной информационной системы.

Исследованы современные архитектуры программного обеспечения и с помощью метода анализа иерархий выбрана архитектура кафедральной информационной системы. По результатам исследования сформулировано техническое задание.

На кафедральном сервере была развёрнута сервисная шина предприятия WSO2 Enterprise Service Bus, служба каталогов OpenLDAP, СУБД PostgreSQL, программная платформа Node.js.

В качестве основных сервисов платформы спроектированы, разработаны и введены в эксплуатацию сервисы авторизации, аутентификации и учёта контингента. Также в платформу интегрированы система управления проектами Atlassian Jira, система управления документацией Atlassian Confluence, сервер репозитория исходного кода GitLab.

Реализованное решение позволяет интегрировать существующие на кафедре приложения через сервисную шину предприятия, заставив их использовать единый источник данных о студентах и группах. Базовые сервисы платформы позволяют будущим разработчикам сконцентрировать свои силы на автоматизации непосредственно прикладных задач, так как часто используемые функции уже реализованы в сервисах аутентификации, авторизации и учёта контингента. Применение механизмов гарантированной рассылки сообщений шиной позволяет сервисам всегда быть в курсе изменений данных в других частях информационной системы. Таким образом, за счёт предоставления в виде отдельных сервисов часто востребованной функциональности, разработанная платформа облегчает процесс разработки новых сервисов, ускоряя тем самым процесс автоматизации отдельных бизнес-задач. Наличие шины в архитектуре повышает общую надёжность системы за счёт механизмов гарантированной

доставки сообщений. Рассылка системных событий дополнительно снижает вероятность рассинхронизации данных в сервисах, подключенных к платформе.

В рамках управления проектом идентифицированы основные риски, которые могут возникнуть в краткосрочной перспективе в процессе развития информационной системы. Были предложены меры их минимизации и избегания.

Основные результаты, полученные в диссертационной работе, докладывались на научных конференциях и были опубликованы в сборниках этих конференций [23, 24, 25, 26, 27].

Работа докладывалась на региональном этапе программы «Участник молодежного научно-инновационного конкурса» («У.М.Н.И.К.») [28].

Дальнейшие перспективы развития единой кафедральной системы заключаются в улучшении качества защиты персональных данных, интеграции существующих приложений кафедры и деканата, разработке новых сервисов, использующих возможности платформы, студентами, магистрантами и аспирантами.

СПИСОК ЛИТЕРАТУРЫ

1. Basics of the Unix Philosophy. – 2017. – Режим доступа: http://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html
2. Claim based Authorization. Security Patterns with WSO2 ESB. – 2017. – Режим доступа: <https://www.slideshare.net/wso2.org/security-patterns-with-wso2-esb-1>
3. Constrained access delegation. – 2017. – Режим доступа: https://www.slideshare.net/wso2.org/security-patterns-with-wso2-esb/26-Chanelling_appointments_Name_Time_7
4. Entitlement Mediator. WSO2 Enterprise Service Bus Documentation. – 2017. – Режим доступа: <https://docs.wso2.com/display/ESB500/Entitlement+Mediator>
5. Exploring the Enterprise Service Bus, Part 1: Discover how an ESB can help you meet the requirements for your SOA solution. – 2017. – Режим доступа: <https://www.ibm.com/developerworks/library/ar-esbpat1/ar-esbpat1-pdf.pdf>
6. Express. Fast, unopinionated, minimalist web framework for Node.js. – 2017. – Режим доступа: <http://expressjs.com/>
7. High Quality CRUD using RESTful Services in DB2 for z/OS. – 2017. – Режим доступа: <https://developer.ibm.com/recipes/tutorials/high-quality-crud-using-restful-services-in-db2-for-zos/>
8. How to create a merge request. GitLab Documentation. – 2017. – Режим доступа: <https://docs.gitlab.com/ee/gitlab-basics/add-merge-request.html>
9. inetOrgPerson class (Windows). – 2017. – Режим доступа: [https://msdn.microsoft.com/en-us/library/ms682282\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms682282(v=vs.85).aspx)
10. ldapjs Client API. – 2017. – Режим доступа: <http://ldapjs.org/client.html>
11. OASIS eXtensible Access Control Markup Language (XACML). – 2017. – Режим доступа: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
12. Pattern: Monolithic Architecture. – 2017. – Режим доступа: <http://microservices.io/patterns/monolithic.html>
13. PMBOK® Guide and Standards, 2017. – Режим доступа. – <http://www.pmi.org/pmbok-guide-standards/foundational/pmbok>

- 14.Promise – JavaScript | MDN. – 2017. Режим доступа:
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/
Global_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)
- 15.Richardson Maturity Model. Steps toward the glory of REST. – 2017. – Режим
доступа: <https://martinfowler.com/articles/richardsonMaturityModel.html>
- 16.ROLE BASED ACCESS CONTROL (RBAC) AND ROLE BASED SECURITY.–
2017. – Режим доступа: <http://csrc.nist.gov/groups/SNS/rbac/>
- 17.Sequelize | The Node.js / io.js ORM for PostgreSQL, MySQL, SQLite and
MSSQL. – 2017. – Режим доступа: <http://docs.sequelizejs.com/en/v3/>
- 18.Service, architecture, governance, and business terms. – 2017. – Режим доступа:
<https://www.ibm.com/developerworks/webservices/library/ws-soa-term1/index.html>
- 19.Working with APIs. – 2017. – Режим доступа:
<https://docs.wso2.com/display/ESB500/Working+with+APIs>
- 20.Working with Message Stores and Message Processors. – 2017. – Режим доступа:
[https://docs.wso2.com/display/ESB500/Working+with+Message+Stores+and+Mess
age+Processors](https://docs.wso2.com/display/ESB500/Working+with+Message+Stores+and+Mess
age+Processors)
- 21.Working with Topics and Events. – 2017. – Режим доступа:
<https://docs.wso2.com/display/ESB500/Working+with+Topics+and+Events>
- 22.WSO2 Identity Server Documentation. – 2017. – Режим доступа:
<https://docs.wso2.com/display/IS530/WSO2+Identity+Server+Documentation>
- 23.Васин, А.В. Программная платформа автоматизации деятельности кафедры
«Информатика и программное обеспечение» / А.В. Васин, Д.Г. Лагереv //
Новые горизонты: Материалы международной конференции-конкурса. –
Брянск: БГТУ, 2016. – 268с.
- 24.Васин, А.В. Проектирование архитектуры программного комплекса для
автоматизации кафедры «Информатика и программное обеспечение» /
А.В. Васин, Д.Г. Лагереv // Научно-технический вестник БГУ. –
Брянск: БГУ, 2017. – в печати.
- 25.Васин, А.В. Разработка сервис-ориентированной информационной системы
для кафедры «ИиПО» на базе связующего программного обеспечения /

- А.В. Васин, Д.Г. Лагереv // Материалы 72-й студенческой научной конференции. – Брянск: БГТУ, 2017. – С. 64-70.
26. Васин, А.В. Программная платформа автоматизации деятельности кафедры «Информатика и программное обеспечение» / А.В. Васин, Д.Г. Лагереv // Материалы VIII международной научно-практической конференции «Достижения молодых ученых в развитии инновационных процессов в экономике, науке и образовании». – Брянск: БГТУ, 2016. – 310с.
27. Васин, А.В. Программная платформа для автоматизации деятельности крупной выпускающей кафедры // А.В. Васин, Д.Г. Лагереv, Материалы 71-й студенческой научной конференции – Брянск: БГТУ, 2016. – 1244 с.
28. Васин, А.В. Разработка программной платформы автоматизации деятельности кафедр высших учебных заведений / А.В. Васин, Д.Г. Лагереv // Материалы II региональной научно-практической конференции «Инновации 2016». Современное состояние и перспективы развития инновационной экономики». – Брянск: БГТУ, 2016. – 135 с.
29. Исходный код основных схем проекта OpenLDAP. – 2017. – Режим доступа: <https://github.com/openldap/openldap/blob/master/servers/slapd/schema/core.schema>
30. Исходный код схемы inetOrgPerson проекта OpenLDAP. – 2017. – Режим доступа: <https://github.com/openldap/openldap/blob/master/servers/slapd/schema/inetorgperson.schema>
31. Микрослужбы в действии: Часть 1. Введение в микрослужбы. – 2017. – Режим доступа: <https://www.ibm.com/developerworks/ru/library/cl-bluemix-microservices-in-action-part-1-trs/>
32. Перегудов Ф.И., Основы системного анализа: Учеб. 2-е изд., доп. / Ф.И. Перегудов, Ф.П. Тарасенко. – Томск: НТЛ, 1997. – 396 с.
33. Подвесовский, А.Г. Особенности обучения будущих программистов принципам и навыкам командной разработки программного обеспечения / А.Г. Подвесовский, Д.Г. Лагереv, Д.А. Коростелеv // Вестник славянских вузов. – 2015. – № 4. – С. 64-70.

- 34.Приказ Минобрнауки России от 12.01.2016 №5 «Об утверждении федерального государственного образовательного стандарта высшего образования по направлению подготовки 09.03.01 Информатика и вычислительная техника (уровень бакалавриата)». – Портал Федеральных государственных образовательных стандартов высшего образования, 2017. – Режим доступа: <http://fgosvo.ru/uploadfiles/fgosvob/090301.pdf>
- 35.Приказ Минобрнауки России от 12.03.2015 №222 «Об утверждении федерального государственного образовательного стандарта высшего образования по направлению подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем (уровень бакалавриата)». – Портал Федеральных государственных образовательных стандартов высшего образования, 2017. – Режим доступа: <http://fgosvo.ru/uploadfiles/fgosvob/020303.pdf>
- 36.Приказ Минобрнауки России от 12.03.2015 №229 «Об утверждении федерального государственного образовательного стандарта высшего образования по направлению подготовки 09.03.04 Программная инженерия (уровень бакалавриата)» – Портал Федеральных государственных образовательных стандартов высшего образования, 2017. – <http://fgosvo.ru/uploadfiles/fgosvob/090304.pdf>
- 37.Приказ Минобрнауки России от 30.10.2014 №1406 «Об утверждении федерального государственного образовательного стандарта высшего образования по направлению подготовки 09.04.04 Программная инженерия (уровень магистратуры)». – Портал Федеральных государственных образовательных стандартов высшего образования, 2017. – http://fgosvo.ru/uploadfiles/fgosvom/090404_progrimg.pdf
- 38.Приказ Минобрнауки России от 30.10.2014 №1420 «Об утверждении федерального государственного образовательного стандарта высшего образования по направлению подготовки 09.03.01 Информатика и вычислительная техника (уровень магистратуры)». – Портал Федеральных

- государственных образовательных стандартов высшего образования, 2017. – http://fgosvo.ru/uploadfiles/fgosvom/090401_informatikaivych.pdf
- 39.Саати, Т. Принятие решений. Метод анализа иерархий. – М.: Радио и связь, 1993. – 278 с.
- 40.Стандарт JSON Web Token (JWT). – 2017. – Режим доступа: <https://tools.ietf.org/html/rfc7519>
- 41.Черновик стандарта JSON Hypertext Application Language. – 2017. – Режим доступа: <https://tools.ietf.org/html/draft-kelly-json-hal-06>