ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| ассистент |  |  |  | Зянчурин А. Э. |
| --- | --- | --- | --- | --- |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| --- |
| Разработка требований. Моделирование  предметной области. Проектирование программного обеспечения |
| по курсу: |
|  |
| Основы программной инженерии |

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ гр. № | 4332 |  |  |  | Топорина А. В.  Каташов В.  Лютов А. А, |
| --- | --- | --- | --- | --- | --- |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

**СОДЕРЖАНИЕ**

[1 Документ о концепции и границах проекта 4](#_g66kq6q72cqv)

[1.1 Описание предметной области 4](#_nlbhh37l2ut2)

[1.2 Цель и постановка задачи разработки автоматизированной/ информационной системы разработки 4](#_spgz7iysb33l)

[1.3 Словарь данных предметной области 5](#_fjswmlajnv8b)

[1.4 Нежелательные эффекты в предметной области 8](#_19k3b9msp2c)

[1.5 Диаграмма вариантов использования 10](#_w1doioz1mtot)

[1.6 Дерево функций информационной системы 11](#_q91o55bxzz8x)

[1.7 Заключение и выводы 12](#_exzmxltg34bc)

[2 Спецификация требований к программному обеспечению 14](#_n6alwoe8pmiy)

[2.1 Определение бизнес-требований 14](#_40xg55oak1v3)

[2.2 Определение функциональных требований 16](#_atas6dvhd6ap)

[2.3 Определение нефункциональных требований 20](#_iumzcdbzhskn)

[2.4 Заключение и выводы 21](#_1y8wsxp3ynmn)

[3 Модель предметной области 23](#_lgffgd7jnav9)

[3.1 Диаграмма прецедентов 23](#_otcdjph7h1eg)

[3.2 Диаграммы объектов 24](#_3p31pylpt16t)

[3.2.1 Диаграмма записи на электив 24](#_hcl688m10g33)

[3.2.2 Диаграмма занятия 25](#_rfb5wjchbpqw)

[3.2.3 Диаграмма ведомость 26](#_utcvi6m5acp2)

[3.2.4 Диаграмма регистрация приказа 27](#_hddge5jig8yq)

[3.3 Диаграмма бизнес-процессов 28](#_f0870sz128td)

[3.3.1 Диаграмма зачисление нового студента 28](#_bprzq6tz4tij)

[3.3.2 Диаграмма запись на электив 28](#_k36v5mfzg7et)

[3.3.3 Диаграмма составления расписания 29](#_4dvgfr8gygme)

[3.3.4 Диаграмма бронирования аудитории 29](#_8meme45imlyo)

[3.3.5 Диаграмма выставления оценок и формирования ведомостей 30](#_6fwl5bcrw3h7)

[3.3.6 Диаграмма управления приказами 30](#_ihyz8atxttm7)

[3.3.7 Диаграмма формирования отчётов 31](#_6pp3dvx2ry0v)

[3.4 Диаграмма состояний 31](#_95pg30ximzdg)

[3.4.1 Диаграмма запись на электив 31](#_updx38v2kyc4)

[3.4.2 Диаграмма занятие 32](#_6h1b735tofrc)

[3.4.3 Диаграмма ведомость 33](#_v9r0hsjgmf1u)

[3.4.4 Диаграмма приказ 34](#_ph9bxlavafaz)

[3.5 Заключение и выводы 35](#_rc0zytj47zcz)

[4 Архитектура программного обеспечения 38](#_kboga2p4fsh7)

[4.1 Определение состава структурных элементов программного обеспечения автоматизированной информационной системы, их назначения и интерфейсов 38](#_yzwc6kuc0noa)

[4.2 Определение варианта архитектуры и ее разработка 38](#_6dcl8agimbev)

[4.3 Заключение и выводы 38](#_lbz0tjucma5i)

[5 Выводы 39](#_ew1mt9eyw0qm)

1. **Документ о концепции и границах проекта**
   1. **Описание предметной области**

В настоящее время учебный процесс и административная деятельность университета поддерживается набором разрозненных, неинтегрированных между собой систем и процессов, многие из которых в значительной степени формализованы с использованием бумажного документооборота или простых локальных электронных таблиц.

Процесс планирования учебного процесса является длительным и сложным. Методисты факультетов вручную согласовывают с кафедрами преподавательский состав, аудиторный фонд и расписание, что часто приводит к конфликтам («накладкам») и нерациональному использованию ресурсов. На изменение расписания, вызванное болезнью преподавателя или иной причиной, уходит несколько часов телефонных звонков и рассылок электронной почты.

Студенты сталкиваются с трудностями при получении актуальной информации: расписание обновляется несвоевременно и публикуется в виде статичных файлов на разных сайтах факультетов; процесс записи на элективные курсы представляет собой сбор бумажных заявлений; информация об академических задолженностях и приказах доходит через старост групп с задержкой.

Преподаватели тратят значительное время на формирование ведомостей и отчетов по успеваемости вручную, а также на выяснение текущего состава учебных групп и подгрупп. Учебная часть испытывает сложности с формированием сводной отчетности для ректората и министерства, так как данные приходится собирать и консолидировать вручную из десятков различных источников.

Администрация университета не имеет оперативного инструмента для анализа ключевых показателей эффективности (KPI), таких как средний балл, процент успеваемости, нагрузка преподавателей и аудиторий, что затрудняет принятие своевременных управленческих решений.

В результате, все участники образовательного процесса тратят чрезмерное время на рутинные, неавтоматизированные операции, что снижает общую эффективность работы университета, увеличивает вероятность человеческих ошибок и приводит к неудовлетворенности как сотрудников, так и студентов.

* 1. **Цель и постановка задачи разработки автоматизированной/ информационной системы разработки**

Цель разработки: создание единой информационной системы университета, которая обеспечит повышение эффективности учебного процесса и административной деятельности за счет сокращения временных издержек на рутинные операции не менее чем на 40%, снижения количества ошибок, связанных с человеческим фактором, и повышения оперативности и достоверности информации для всех участников образовательного процесса.

Постановка задачи разработки: в настоящее время рейтинг удовлетворенности студентов и сотрудников университета процедурами организации учебного процесса остается низким, а операционные издержки — высокими. Это напрямую связано с отсутствием у пользователей адекватных интегрированных инструментов, позволяющих эффективно выполнять свои задачи.

* Для методистов и учебной части отсутствие автоматизированного инструмента планирования приводит к тому, что процесс составления и актуализации расписания занимает дни, а не часы, и сопровождается постоянными конфликтами ресурсов (аудитории, преподаватели). Это делает учебный процесс негибким и неэффективным.
* Для студентов отсутствие единого и актуального источника информации о расписании, академическом прогрессе и приказах, а также необходимость физической подачи заявлений для выбора курсов создает барьеры для комфортного обучения и снижает академическую мобильность.
* Для преподавателей рутинные операции по формированию отчетности и выяснению состава групп отнимают время, которое могло бы быть направлено на научную и преподавательскую деятельность, что снижает их профессиональную удовлетворенность.
* Для администрации отсутствие оперативной аналитической отчетности и ключевых показателей эффективности (KPI) не позволяет проводить своевременный мониторинг состояния учебного процесса и принимать обоснованные управленческие решения, что негативно сказывается на конкурентоспособности университета в целом.
  1. **Словарь данных предметной области**

Формирование словаря данных направлено на выделение и описание ключевых объектов предметной области, участвующих в учебном и административном процессе университета.

Каждый объект представлен как структура данных, включающая простейшие элементы — атрибуты, необходимые для корректного функционирования автоматизированной информационной системы.

Таблица №1 отражает основные сущности системы, описание, атрибуты и типы данных

Таблица №1. Словарь данных

| Элемент данных | Описание | Структура или тип данных | Количество | Пример значения |
| --- | --- | --- | --- | --- |
| **Студент** | |  | Участник образовательного процесса, обучающийся в университете. | | --- | --- | | Снилс + {ФИО} + Номер телефона + Почта + Номер студ. Билета + Номер зачетной книжки + Номер группы + Номер специальности + Форма обучения + Институт + Уровень образования + Курс + 1:n {приказы} |  |  |
| **Сотрудник** | Работник университета. Может быть преподавателем, методистом, администратором и т.д. | Снилс + {ФИО} + Номер телефона + Почта + Должность + Каферда/отдел + Ученая степень + Ставка + 1:n {приказы} |  |  |
| **Группа** | Учебный коллектив студентов. | Номер группы + Год создания + Курс + Статус группы + Кол-во студентов + Специальность |  |  |
| **Дисциплина** | Учебный предмет, изучаемый студентами. | Код дисциплины + Название дисциплины + Кафедра + Количество часов + Семестр + Тип дисциплины |  |  |
| **Расписание** | План учебных занятий. | Идентификатор записи + Дата + Время начала + Дисциплина + Преподаватель + Группы + Аудитория + Тип занятия |  |  |
| **Успеваемость** | Результат сдачи студентом дисциплины. | Идентификатор записи + Студент + Дисциплина + Оценка + Преподаватель + Дата сдачи + Тип контроля + Семестр |  |  |
| **Приказ** | Официальный распорядительный документ. | Номер приказа + Тип приказа + Дата подписания + Содержание |  |  |
| **Аудитория** | Помещение для проведения занятий. | Идентификатор аудитории + Адрес + Номер аудитории + Вместимость + Тип аудитории + Оснащение |  |  |
| **Учебный план** | Набор дисциплин для конкретной специальности и года поступления. | Идентификатор плана + Специальность + Год набора + Форма обучения + 1:n {Дисциплина} |  |  |
| Снилс | страховой номер индивидуального лицевого счета | Число (формат: XXX-XXX-XXX XX) | 11 | 123-345-567-78 |
| **ФИО** | Фамилия имя отчество | Фамилия + Имя + Отчество |  |  |
| Фамилия | фамилия | строка |  | Иванов |
| Имя | имя | строка |  | Иван |
| Отчество | отчество | строка |  | Иванович |
| Номер студ. билета | Уникальный номер документа студента. | Строка |  | 345346 |
| Номер зачетной книжки | Контактный номер телефона. | Строка |  | 234123 |
| Номер телефона | Контактный номер телефона. | Число |  | 234556768 |
| Почта | Адрес электронной почты. | Строка |  | person@university.ru |
| Институт | Структурное подразделение университета. | Строка |  | 4 институт |
| Форма обучения | Форма освоения образовательной программы. |  |  | Очное |
| Уровень образования | Уровень получаемого образования. | Строка |  | Магистратура |
| Должность | Занимаемая сотрудником должность. Определяет его роль (преподаватель, методист, декан и т.д.). | Строка |  | Доцент |
| Кафедра/отдел | Структурное подразделение сотрудника. | Строка |  | 43 кафедра |
| Ставка | Размер учебной нагрузки преподавателя | Число с плавающей точкой |  | 25,5 |
| Номер группы | Учебный номер группы. | Строка |  | 4332 |
| Курс | Текущий курс обучения группы. | Целое | 1 | 3 |
| Статус активности | Статус группы (активна/закрыта). | Булевый |  | true |
| Специальность | Направление подготовки группы. | Код специальность + Название специальности |  |  |
| Оценка | Результат аттестации | Строка |  | Зачет |
| Тип занятия | Вид учебного занятия | Строка |  | Лекция |
| Тип контроля | Форма контроля знаний | Строка |  | Зачет |
| Код специальности | Код, который присваивается образовательной программе | Число (формат: XX.XX.XX) | 6 | 09.03.04 |
| Название специальности | Название специальности | Строка |  | Программная инженерия |

* 1. **Нежелательные эффекты в предметной области**

Проектирование новой информационной системы университета требует выявления и устранения противоречий, возникающих между элементами существующей системы. Для идентификации этих противоречий была построена таблица взаимодействия (Таблица №2), отражающая наличие конфликтов между ключевыми элементами предметной области.

Таблица №2.

| Элементы | Студент | Сотрудник | Группа | Дисциплина | Расписание | Успеваемость | Приказ | Учебный план |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент |  | + | - | + | + | - | + | - |
| Сотрудник |  |  | - | + | + | + | + | + |
| Группа |  |  |  | - | + | - | + | - |
| Дисциплина |  |  |  |  | - | + | - | - |
| Расписание |  |  |  |  |  | - | - | - |
| Успеваемость |  |  |  |  |  |  | - | - |
| Приказ |  |  |  |  |  |  |  | - |
| Учебный план |  |  |  |  |  |  |  |  |

Пояснение к выявленным конфликтам:

* Студент – Сотрудник: отсутствие эффективного канала коммуникации
* Студент – Дисциплина: трудности с доступом к материалам и записям на курсы
* Студент – Расписание: несвоевременное получение актуального расписания и сложность его изменения
* Студент – Приказ: неоперативное информирование о приказах
* Сотрудник – Дисциплина: трудоемкость процессов, связанных с веденим дисциплины(отчетность)
* Сотрудник – Расписание: низкая эффективность и гибкость процесса взаимодействия с расписанием
* Сотрудник – Успеваемость: преобладание ручного процесса работы с оценками
* Сотрудник – Приказ: несвоевременное доведение решений до сотрудников
* Сотрудник – Учебный план: сложность в получении актуальной информации и формировании отчетности на основе учебных планов
* Группа – Расписание: нерациональное использование ресурсов группы и создание неудобного для обучения расписания
* Группа – Приказ: задержка в доведении административных решений до студентов
* Дисциплина – Успеваемость: не стандартизированный процесс внесения и учета оценок по дисциплине
  1. **Диаграмма вариантов использования**

Для выявления полноты функциональных требований к разрабатываемой системе была построена матрица CRUD (Create, Read, Update, Delete), которая позволяет сопоставить варианты использования с основными сущностями системы и операциями над ними.

Таблица №3. Матрица CRUD

| Варианты использования | Студент | Сотрудник | Группа | Дисциплина | Расписание | Успеваемость | Приказ | Аудитория | Учебный план |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Зачислить студента | C | - | R,U | - | - | - | C,R | - | R |
| Изменить данные о студенте | U,D | - | R,U | - | - | - | C,R | - | R |
| Оформление сотрудника | - | C | - | - | - | - | C,R | - | - |
| Изменение данных о сотруднике | - | U,D | R,U | R,U | R,U | R,U | C,R | - | - |
| Создание группы | - | - | C | - | - | - | C,R | - | R |
| Изменение группы | R,U | R | U,D | R | R,U | - | C,R | - | R |
| Создание дисциплины | - | - | - | C | - | - | - | - | R,U |
| Изменение дисциплины | - | R | - | U,D | R,U | - | - | - | R,U |
| Создание расписания | R | R | R | R | C | - | - | R,U | R |
| Изменение расписания | R | R | R | R | U,D | - | - | R,U | R |
| Выставление оценки | R | R | - | R | - | C,U | - | - | R |
| Изменение оценки | R | R | - | R | - | U,D | - | - | R |
| Создание учебного плана | - | R | R | R,U | - | - | - | - | C |
| Изменение учебного плана | - | R | R | R,U | R,U | - | - | - | U,D |
| Просмотр ведомости | R | R | R | R | - | R | - | - | R |
| Бронирование аудитории | - | R | - | - | C,U | - | - | R,U | - |
| Изменение аудитории | - | R | - | - | R,U | - | - | U,D | - |

* 1. **Дерево функций информационной системы**

Для построение дерева функций программной системы определим основные функции, отражающие ключевые варианты использования и направленные на устранение нежелательных эффектов предметной области:

* Управление учебным процессом
* Управление персоналом
* Планирование и оптимизация расписания
* Управление успеваемостью
* Административно-организационное управление
* Аналитика и отчетность
* Интеграция и системные функции

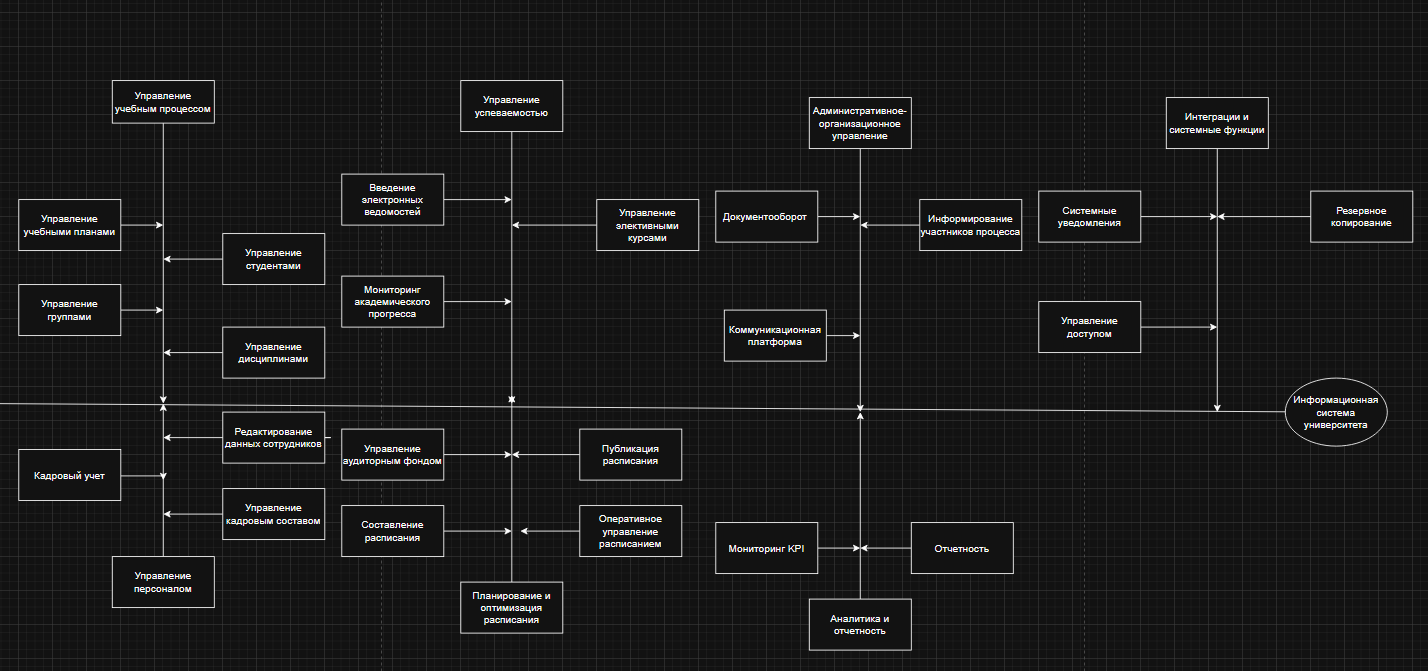


Рис. 1 Дерево функций программной системы

* 1. **Заключение и выводы**

В результате проведенного анализа предметной области и постановки задачи разработки автоматизированной информационной системы университета можно сделать следующие выводы:

1. В настоящее время учебный процесс и административная деятельность университета характеризуются высокой трудоемкостью, низкой гибкостью и значительной зависимостью от ручного труда, что приводит к увеличению временных издержек и вероятности ошибок.
2. Существующие разрозненные системы и бумажный документооборот не обеспечивают оперативного обмена информацией между студентами, преподавателями и сотрудниками учебной части, что снижает эффективность управления учебным процессом.
3. Процессы планирования расписания, формирования отчетности, учета успеваемости и приказов осуществляются преимущественно вручную, что замедляет работу всех участников образовательного процесса и создаёт трудности в координации ресурсов.
4. Студенты сталкиваются с проблемой получения актуальной информации о расписании, академическом прогрессе и приказах, что ограничивает их академическую мобильность и снижает уровень удовлетворенности образовательным процессом.
5. Преподаватели тратят значительное время на рутинные операции, такие как формирование ведомостей, учет успеваемости и проверка состава учебных групп, что уменьшает их возможности для научной и преподавательской деятельности.
6. Методисты и сотрудники учебной части испытывают сложности при составлении и корректировке расписания, а также при анализе учебной нагрузки, что приводит к конфликтам ресурсов и нерациональному использованию аудитории и преподавателей.
7. Администрация университета не располагает оперативными инструментами для анализа ключевых показателей эффективности (KPI), что ограничивает возможности своевременного принятия управленческих решений.
8. Разработка единой информационной системы университета позволит автоматизировать основные процессы, сократить время на рутинные операции не менее чем на 40%, снизить количество ошибок, связанных с человеческим фактором, и обеспечить актуальность информации для всех участников образовательного процесса.
9. Основные функции разрабатываемой системы — управление учебным процессом, управление персоналом, планирование и оптимизация расписания, управление успеваемостью, административно-организационное управление, аналитика и отчетность, интеграция и системные функции — направлены на устранение выявленных конфликтов и нежелательных эффектов предметной области.
10. Реализация предложенной системы создаст условия для повышения эффективности работы университета, улучшения качества образовательного процесса и повышения удовлетворенности студентов, преподавателей и сотрудников администрации.
11. **Спецификация требований к программному обеспечению**
    1. **Определение бизнес-требований**

2.1.1 Структура системы

Система должна быть реализована в виде единой централизованной информационной системы университета, обеспечивающей сквозную автоматизацию ключевых процессов. Архитектура системы должна включать следующие высокоуровневые компоненты:

* Веб-портал / Пользовательские интерфейсы: Единая точка доступа для всех категорий пользователей (студенты, преподаватели, методисты, администрация) с персонализированным рабочим пространством в зависимости от роли.
* Ядро системы (Backend): Централизованная база данных, содержащая все актуальные данные (согласно Словарю данных), и серверная логика для обработки бизнес-процессов.
* Подсистема планирования и оптимизации расписания: Интеллектуальный модуль для автоматизированного составления и корректировки расписания с учетом всех ресурсов (аудитории, преподаватели, группы).
* Подсистема управления учебным процессом: Модули для ведения учебных планов, дисциплин, групп и записи на элективные курсы.
* Подсистема управления успеваемостью: Модуль для ввода, хранения и анализа оценок студентов.
* Подсистема отчетности и аналитики: Модуль для автоматического формирования стандартной отчетности и расчета KPI для администрации.
* API для интеграции: Обеспечение возможности безопасного обмена данными с существующими или будущими внешними системами (при необходимости).

2.1.2 Требования к режимам функционирования системы

* Штатный режим: Круглосуточная работа системы в режиме 24/7 с исключением плановых технологических окон в периоды низкой нагрузки (например, ночное время).
* Пиковые режимы: Система должна быть рассчитана на пиковые нагрузки в периоды начала семестра (формирование расписания, запись на курсы), сессии (ввод оценок) и подачи заявлений.
* Аварийный режим: В случае сбоя система должна обеспечивать сохранность всех данных. Время восстановления после сбоя не должно превышать 1 часа.

2.1.3 Требования к персоналу

* Численность: Система должна быть рассчитана на одновременную работу всех сотрудников университета (методисты, преподаватели, администрация) и студентов.
* Квалификация: Для работы с системой не должно требоваться специальных технических знаний, выходящих за рамки базовой компьютерной грамотности. Интерфейс должен быть интуитивно понятным.
* Режим работы: Доступ к системе предоставляется в соответствии с ролью пользователя (студент, преподаватель и т.д.).

2.1.4 Требования к надежности

* Производительность: Время отклика для 95% операций: просмотр данных — до 2 сек., изменение данных — до 3 сек., формирование отчетов — до 30 сек.
* Доступность: Система должна демонстрировать уровень доступности, при котором время простоя в работе системы не превышает 1 часа в сутки.
* Вероятность сбоя: Система должна демонстрировать уровень надёжности, при котором вероятность сбоя при обращении к её функциям не превышает 5%.

2.1.5 Требования к безопасности

* Аутентификация и авторизация: Обязательная авторизация всех пользователей по индивидуальным учетным записям с разграничением прав доступа на основе ролевой модели (RBAC).
* Защита данных: Шифрование передаваемых и хранимых персональных данных в соответствии с 152-ФЗ.
* Аудит и мониторинг: Логирование всех критических операций и несанкционированных попыток доступа.

2.1.6 Требования к эргономике и технической эстетике

* Пользовательский интерфейс должен быть современным, интуитивно понятным и адаптивным для различных устройств (ПК, планшеты).
* Навигация по системе должна быть простой и логичной, исключающей возможность "потеряться" в интерфейсе.
* Интерфейс должен поддерживать персонализацию в зависимости от роли пользователя.

2.1.7 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

* Система должна предоставлять инструменты для администраторов для управления пользователями, ролями, мониторинга производительности и резервного копирования.
* Процедуры резервного копирования и восстановления должны быть автоматизированы.
* Регулярное тестирование процедур восстановления должно проводиться не реже 1 раза в квартал.

2.1.8 Требования к защите информации от несанкционированного доступа

* Запрет доступа к данным и функционалу, не относящемуся к компетенции пользователя.
* Использование механизмов шифрования для защиты конфиденциальных данных.
* Регулярный аудит прав доступа и подозрительной активности.

2.1.9 Требования к сохранности информации при авариях

* Обязательное регулярное резервное копирование всех данных системы.

2.1.10 Требования к защите от влияния внешних воздействий

* Система должна быть отказоустойчивой к сбоям в работе сетевой инфраструктуры.
* Обеспечение graceful degradation (постепенного снижения функциональности)
* Защита от DDoS-атак и других внешних угроз.

2.1.11 Требования к патентной чистоте

* Используемое программное обеспечение и технологии не должны нарушать патентные права третьих лиц.
* Все сторонние библиотеки и компоненты должны иметь соответствующие лицензии.

2.1.12 Требования к стандартизации и унификации

* Система должна использовать общепринятые отраслевые стандарты и протоколы (REST API, SQL).
* Структура данных должна быть унифицирована для исключения дублирования и обеспечения согласованности.
* Использование российских стандартов для совместимости с внешними системами.
  1. **Определение функциональных требований**

Функциональные требования определяют операции и действия, которые должны выполняться системой для автоматизации учебного процесса и административной деятельности университета.

2.2.1 Функциональные требования подсистемы управления учебным процессом

2.2.1.1 Управление студентами

* FR-001: Система должна предоставлять возможность зачисления нового студента с заполнением всех обязательных атрибутов (ФИО, контактные данные, специальность, форма обучения)
* FR-002: Система должна обеспечивать редактирование данных о студенте (изменение персональной информации, перевод между группами/специальностями)
* FR-003: Система должна предоставлять возможность отчисления студента с формированием соответствующего приказа
* FR-004: Система должна обеспечивать поиск студентов по различным критериям (ФИО, группа, специальность, институт)

2.2.1.2 Управление учебными группами

* FR-005: Система должна предоставлять возможность создания учебных групп с указанием номера, специальности, куратора
* FR-006: Система должна обеспечивать редактирование состава и параметров учебной группы
* FR-007: Система должна автоматически рассчитывать количество студентов в группе

2.2.1.3 Управление дисциплинами и учебными планами

* FR-008: Система должна предоставлять возможность создания и редактирования дисциплин с указанием кода, названия, кафедры, количества часов
* FR-009: Система должна обеспечивать формирование учебных планов для специальностей с привязкой дисциплин
* FR-010: Система должна предоставлять возможность записи студентов на элективные курсы в онлайн-режиме

2.2.2 Функциональные требования подсистемы управления персоналом

2.2.2.1 Управление сотрудниками

* FR-011: Система должна предоставлять возможность оформления нового сотрудника с заполнением всех обязательных атрибутов
* FR-012: Система должна обеспечивать редактирование данных о сотруднике (изменение должности, кафедры, ставки)
* FR-013: Система должна предоставлять возможность распределения учебной нагрузки между преподавателями

2.2.2.2 Управление ролями и доступом

* FR-014: Система должна обеспечивать назначение ролей пользователям (студент, преподаватель, методист, администратор)
* FR-015: Система должна разграничивать права доступа в соответствии с ролевой моделью

2.2.3 Функциональные требования подсистемы планирования расписания

2.2.3.1 Составление расписания

* FR-016: Система должна предоставлять возможность автоматического составления расписания с учетом:
  + Доступности аудиторий
  + Занятости преподавателей
  + Учебных планов групп
* FR-017: Система должна обнаруживать и предотвращать конфликты в расписании (накладки по аудиториям, преподавателям, группам)
* FR-018: Система должна предоставлять инструменты для ручной корректировки автоматически составленного расписания

2.2.3.2 Управление изменениями расписания

* FR-019: Система должна обеспечивать оперативное внесение изменений в расписание (замена преподавателя, перенос занятия)
* FR-020: Система должна автоматически уведомлять всех затронутых участников об изменениях в расписании

2.2.3.3 Бронирование аудиторий

* FR-021: Система должна предоставлять возможность бронирования аудиторий для внеучебных мероприятий
* FR-022: Система должна отображать занятость аудиторий в реальном времени

2.2.4 Функциональные требования подсистемы управления успеваемостью

2.2.4.1 Ведение успеваемости

* FR-023: Система должна предоставлять возможность ввода и редактирования оценок студентов по дисциплинам
* FR-024: Система должна обеспечивать формирование ведомостей успеваемости в автоматическом режиме
* FR-025: Система должна рассчитывать средний балл и академический рейтинг студентов

2.2.4.2 Контроль учебного процесса

* FR-026: Система должна предоставлять возможность отслеживания академических задолженностей студентов
* FR-027: Система должна автоматически формировать списки студентов для отчисления по академической неуспеваемости

2.2.5 Функциональные требования подсистемы административно-организационного управления

2.2.5.1 Управление приказами

* FR-028: Система должна обеспечивать создание, регистрацию и хранение приказов различных типов
* FR-029: Система должна автоматически уведомлять сотрудников и студентов о новых приказах, касающихся их непосредственно
* FR-030: Система должна предоставлять возможность поиска приказов по номеру, типу, дате

2.2.5.2 Коммуникации

* FR-031: Система должна предоставлять каналы коммуникации между студентами и преподавателями
* FR-032: Система должна обеспечивать рассылку официальных объявлений и уведомлений

2.2.6 Функциональные требования подсистемы аналитики и отчетности

2.2.6.1 Формирование отчетности

* FR-033: Система должна автоматически формировать стандартные отчеты для министерства и ректората
* FR-034: Система должна предоставлять возможность формирования пользовательских отчетов по произвольным параметрам

2.2.6.2 Аналитика KPI

* FR-035: Система должна рассчитывать ключевые показатели эффективности:
  + Средний балл успеваемости
  + Процент успеваемости по группам и специальностям
  + Нагрузка преподавателей и аудиторий
* FR-036: Система должна визуализировать аналитические данные в виде графиков и диаграмм

2.2.7 Функциональные требования подсистемы интеграции и системных функций

2.2.7.1 Интеграционные возможности

* FR-037: Система должна предоставлять REST API для интеграции с внешними системами
* FR-038: Система должна обеспечивать импорт/экспорт данных в стандартных форматах (Excel, CSV)

2.2.7.2 Системные функции

* FR-039: Система должна обеспечивать управление пользователями и их аутентификацию
* FR-040: Система должна предоставлять инструменты резервного копирования и восстановления данных
* FR-041: Система должна вести журнал аудита для отслеживания действий пользователей

Все функциональные требования направлены на устранение выявленных нежелательных эффектов предметной области и обеспечение достижения целевых показателей эффективности, определенных в бизнес-требованиях.

* 1. **Определение нефункциональных требований**

2.3.1 Ограничения

2.3.1.1 Технологические ограничения

* NFR-001: Система должна использовать веб-технологии с поддержкой кроссплатформенной работы
* NFR-002: Клиентская часть должна поддерживать браузеры Chrome, Firefox, Safari

2.3.1.2 Аппаратные ограничения

* NFR-003: Клиентские рабочие места: 2-ядерный CPU, 4 ГБ RAM, разрешение 1280×720

2.3.2 Внешние интерфейсы

2.3.2.1 Интерфейсы интеграции

* NFR-004: REST API с документацией в формате OpenAPI 3.0
* NFR-005: Поддержка аутентификации OAuth 2.0 и JWT-токенов
* NFR-006: Импорт данных из XLSX, CSV, JSON
* NFR-007: Экспорт отчетов в PDF, XLSX, DOCX

2.3.2.2 Системные интерфейсы

* NFR-008: Интеграция с системой электронной подписи
* NFR-009: Отправка email-уведомлений через SMTP
* NFR-010: Запись логов в системные журналы (syslog)

2.3.3 Информационные требования к пользовательским интерфейсам

2.3.3.1 Требования к отображению данных

* NFR-011: Отображение актуального расписания с подсветкой изменений
* NFR-012: Визуализация успеваемости студентов
* NFR-013: Дашборды KPI для администрации
* NFR-014: Уведомления о новых приказах и событиях

2.3.3.2 Требования к структуре информации

* NFR-015: Организация данных по ролевой модели пользователя
* NFR-016: Персонализированное представление данных
* NFR-017: Единообразное представление данных во всех модулях

2.3.4 Функциональные требования к пользовательским интерфейсам

2.3.4.1 Требования к взаимодействию

* NFR-018: Поддержка drag-and-drop для управления расписанием
* NFR-019: Функция "умного поиска" с подсказками

2.3.4.2 Требования к навигации

* NFR-020: Единая навигационная панель с быстрым доступом

2.3.4.3 Требования к адаптивности

* NFR-021: Корректное отображение на различных разрешениях экрана
* NFR-022: Упрощенный интерфейс для мобильных устройств

2.3.4.4 Требования к доступности

* NFR-023: Работа с клавиатуры без использования мыши
* NFR-024: Возможность увеличения шрифтов без потери функциональности

2.3.5 Требования к сопровождению

* NFR-025: Инструменты мониторинга производительности в реальном времени

Все нефункциональные требования направлены на обеспечение стабильной, безопасной и эффективной работы системы в условиях учебного заведения.

* 1. **Заключение и выводы**

На основании формирования требований к информационной системе университета были сделаны следующие выводы:

1. Разработана комплексная система бизнес-требований, определяющая целевую архитектуру системы как единой централизованной платформы, обеспечивающей сквозную автоматизацию ключевых процессов университета.
2. Определен полный перечень функциональных требований, охватывающий все основные направления деятельности университета: управление учебным процессом, персоналом, расписанием, успеваемостью, административно-организационной деятельностью и аналитикой.
3. Разработаны нефункциональные требования, обеспечивающие высокую производительность, надежность, безопасность и удобство использования системы, включая требования к времени отклика, масштабируемости, защите данных и адаптивности интерфейсов.
4. Определены требования к пользовательским интерфейсам, направленные на обеспечение интуитивной навигации, адаптивности для различных устройств и доступности для всех категорий пользователей.
5. Установлены требования к эксплуатационным характеристикам системы, включая мониторинг производительности, резервное копирование, восстановление после сбоев и защиту от внешних угроз.
6. Сформирована комплексная система требований, достаточная для перехода к этапу проектирования и реализации информационной системы университета, которая позволит достичь поставленных целевых показателей эффективности и устранить выявленные проблемы предметной области.
7. Сформулированы конкретные целевые показатели эффективности системы, включая сокращение временных затрат на рутинные операции на 40%, обеспечение коэффициента доступности 99,5% и поддержку 1000+ одновременных пользователей.
8. **Модель предметной области**
   1. **Диаграмма прецедентов**

Исходя из требований (п.2.1 – структура системы, п.2.2 – функциональные требования) основные роли, взаимодействующие с системой это реальные пользователи и внешние системы:

* Студент – конечный пользователь: просмотр расписания/успеваемости, запись на элективы (FR-010, FR-001..FR-004).
* Преподаватель – ввод оценок, просмотр ведомостей, бронирование аудитории, участие в расписании (FR-023..FR-025, FR-021).
* Методист – формирование учебных планов, генерация/утверждение расписания, управление группами (FR-005..FR-009, FR-016..FR-019).
* Администратор – управление пользователями, приказами, отчётностью, резервным копированием (FR-028..FR-041).
* Внешние системы – интеграция с Минобразования, LMS, сервисами почты/ЭЦП/аутентификации (FR-037..FR-038, NFR-008..NFR-010).
* Сервис планирования – выделена как актор, т.к. планирование может быть автономным модулем с API (FR-016, NFR-004).

Прецеденты отражают конкретные (в иерархии) функции из требований: управление студентами, расписанием, успеваемостью, приказами, интеграция, уведомления, отчётность, бронирование аудиторий, авторизация. Они покрывают весь набор FR-001..FR-041 и ключевые NFR (аутентификация, уведомления, API, импорт/экспорт и т.д.).

Диаграмма прецедентов представлена на Рисунке 3.1.

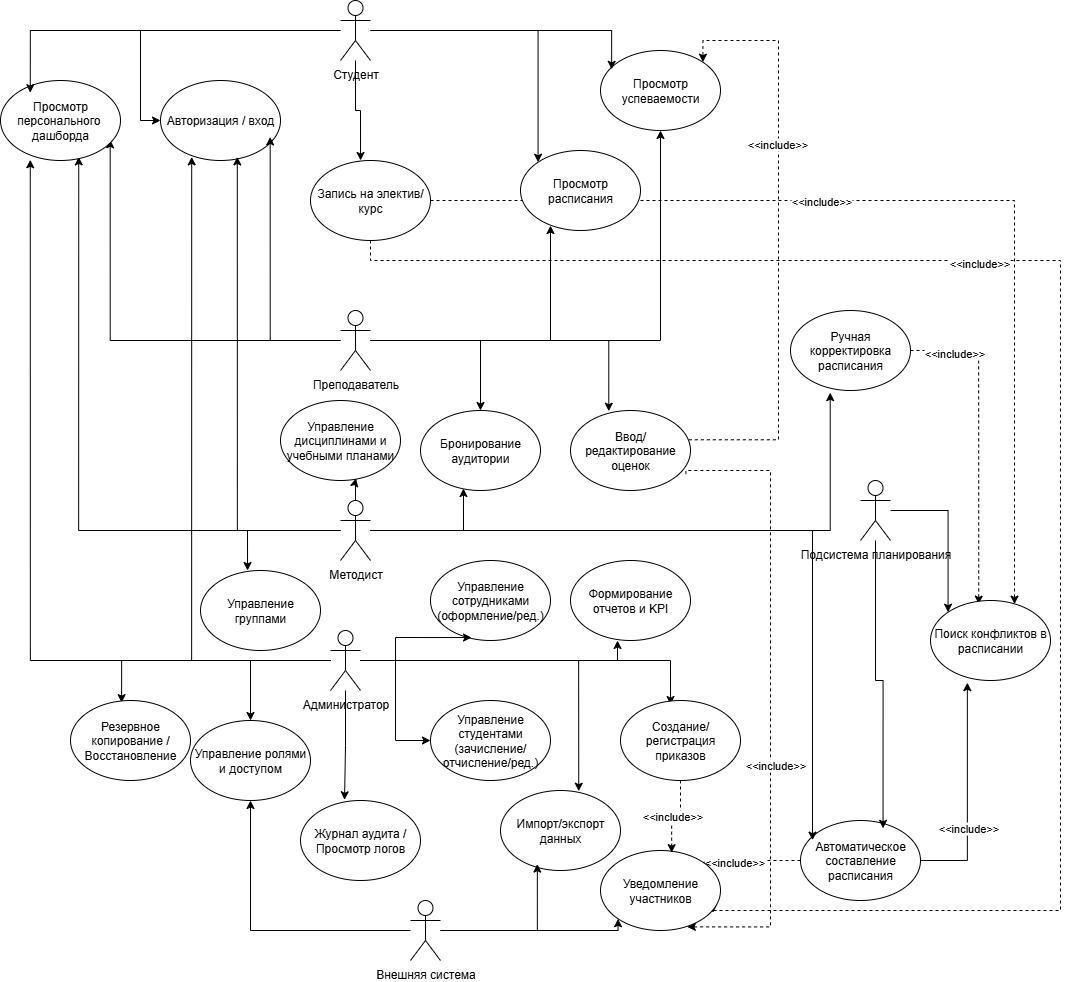


Рисунок 3.1 - Диаграмма прецедентов

* 1. **Диаграммы объектов**
     1. **Диаграмма записи на электив**

Цель: показать объекты и их связи при записи студента на электив (Enrollment). Помогает проверить логику проверки конфликтов, лист ожидания и связь с учебным планом.

Объекты: Student, Group, StudyPlan, Discipline, Enrollment, SeatQuota

* Student и Group — фундаментальные сущности.
* StudyPlan и Discipline — нужны для проверки прав на запись (соответствие учебному плану).
* Enrollment —объект, отражающий состояние заявки/записи. Атрибут status показывает жизненный цикл.
* SeatQuota — модель ресурса, влияющего на логику записи (места/квоты).

Диаграмма записи на электив представлена на Рисунке 3.2.1

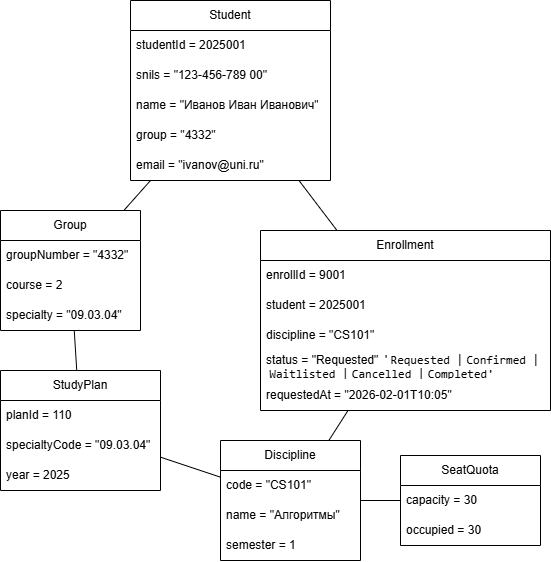


Рисунок 3.2.1 - Диаграмма записи на электив

* + 1. **Диаграмма занятия**

Цель: показать объекты, участвующие в одном занятии: аудитория, преподаватель, группа, дисциплина и запись расписания. Отражает конфликтные связи (аудитория/преподаватель/группа).

Объекты: ScheduleEntry, Classroom, Staff, Discipline, Group, AttendanceList

* ScheduleEntry — центральный объект в сценарии планирования (жизненный цикл: Draft → Scheduled → Confirmed → Completed / Cancelled).
* Classroom и Staff — ресурсы, по которым возможны конфликты.
* AttendanceList — дополнительный объект для сценариев учёта посещаемости/ведомости.

Диаграмма занятия представлена на Рисунке 3.2.2

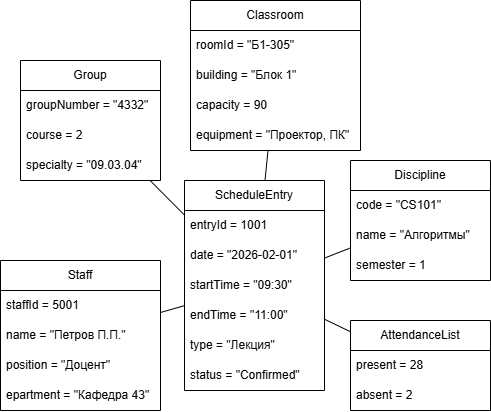


Рисунок 3.2.2 - Диаграмма занятия

* + 1. **Диаграмма ведомость**

Цель: показать объекты ведомости и оценки, связь с дисциплиной и студентами, расчёт среднего.

Объекты: Grade, GradeBook, Student, Staff

* Grade как индивидуальная запись оценки; GradeBook —ведомость.
* Staff — ответственный за ведомость.

Диаграмма ведомость представлена на Рисунке 3.2.3

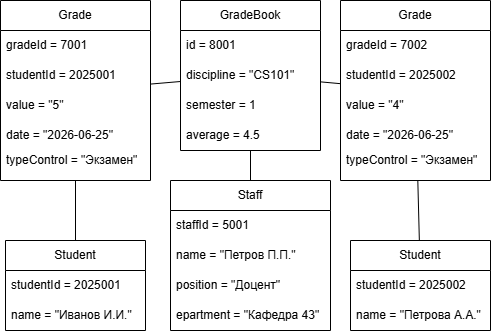


Рисунок 3.2.3 – Диаграмма ведомость

* + 1. **Диаграмма регистрация приказа**

Цель: показать взаимодействие приказа с объектами (студенты, сотрудники, группы) и запись в реестре.

Объекты: Order, Student, Staff, AuditLog, DocSignature

* Order — ключевой административный объект; связан с адресатами (студент/сотрудник).
* DocSignature — объект, моделирующий факт ЭЦП для соблюдения NFR-008.
* AuditLog — покрывает требования по аудиту (FR-041).

Диаграмма регистрация приказа представлена на Рисунке 3.2.4

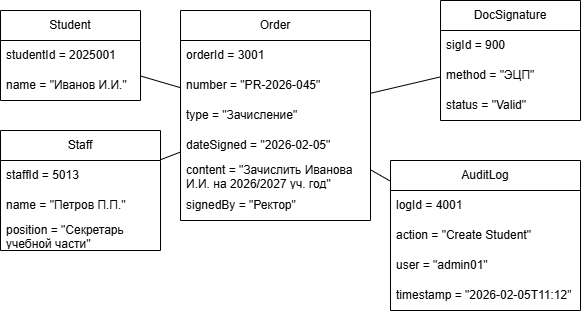


Рисунок 3.2.4 – Диаграмма регистрация приказа

* 1. **Диаграмма бизнес-процессов**
     1. **Диаграмма зачисление нового студента**

Назначение. Формализовать приём заявлений, проверку документов, создание карточки студента и формирование приказа о зачислении.

Диаграмма зачисление нового студента представлена на Рисунке 3.3.1

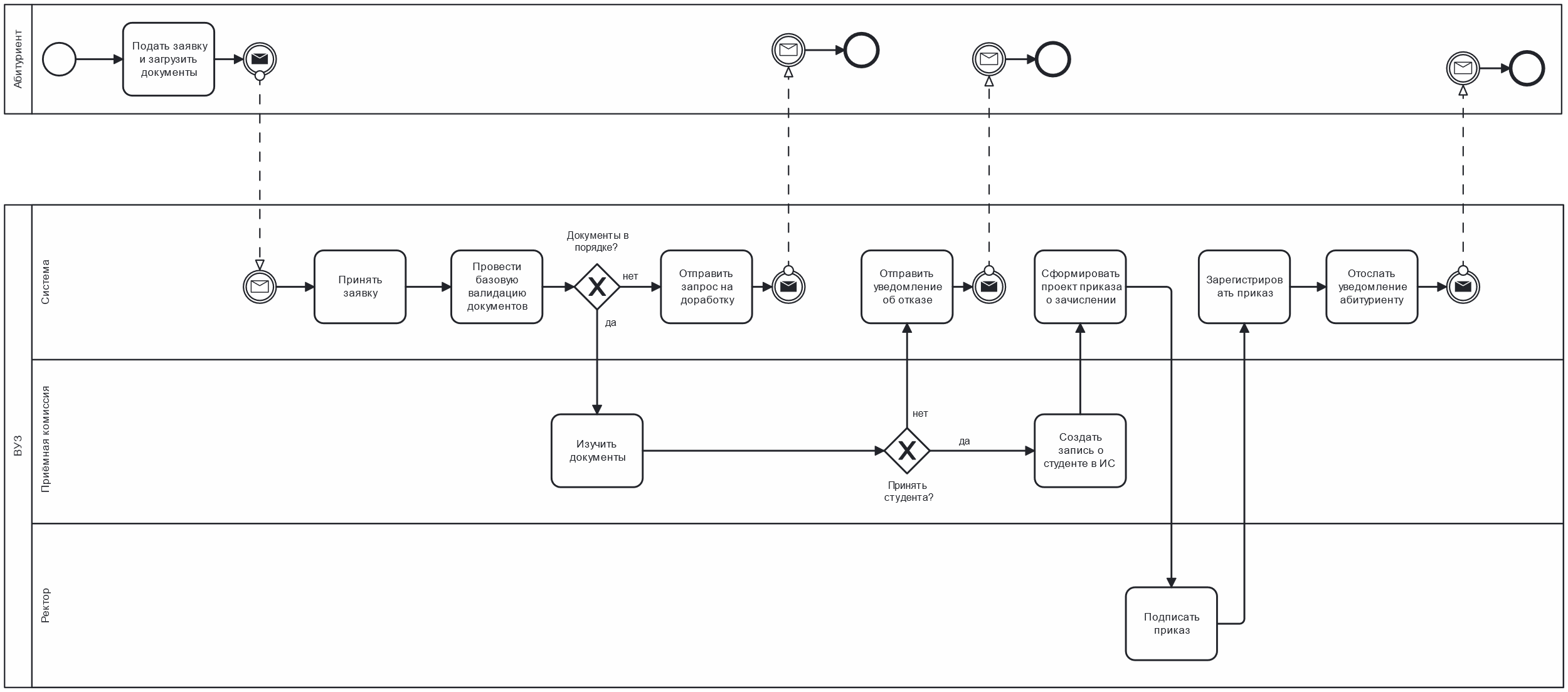


Рисунок 3.3.1 – Диаграмма зачисление нового студента

* + 1. **Диаграмма запись на электив**

Назначение. Онлайн-запись студентов на элективы с проверкой конфликтов расписания и квот, управление листом ожидания.

Диаграмма запись на электив представлена на Рисунке 3.3.2

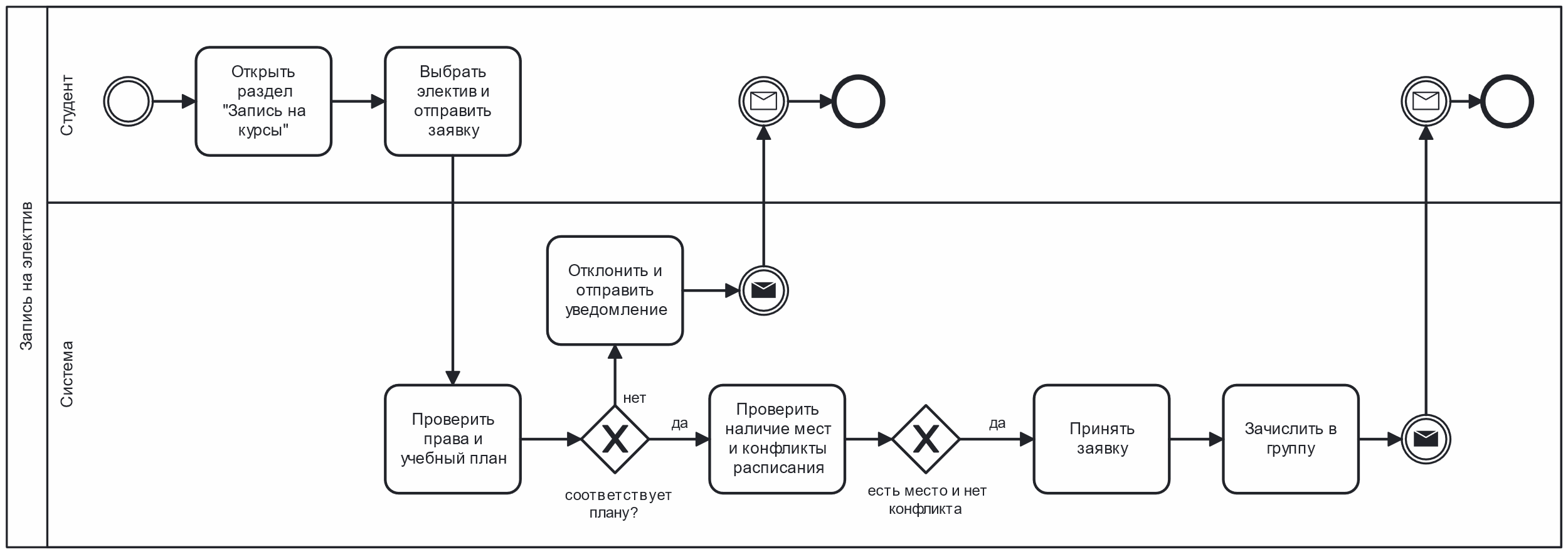


Рисунок 3.3.2 – Диаграмма запись на электив

* + 1. **Диаграмма составления расписания**

Назначение. Автоматическая генерация предварительного расписания с учётом ограничений (аудитории, преподаватели, учебные планы), последующая валидация и утверждение.

Диаграмма составления расписания представлена на Рисунке 3.3.3

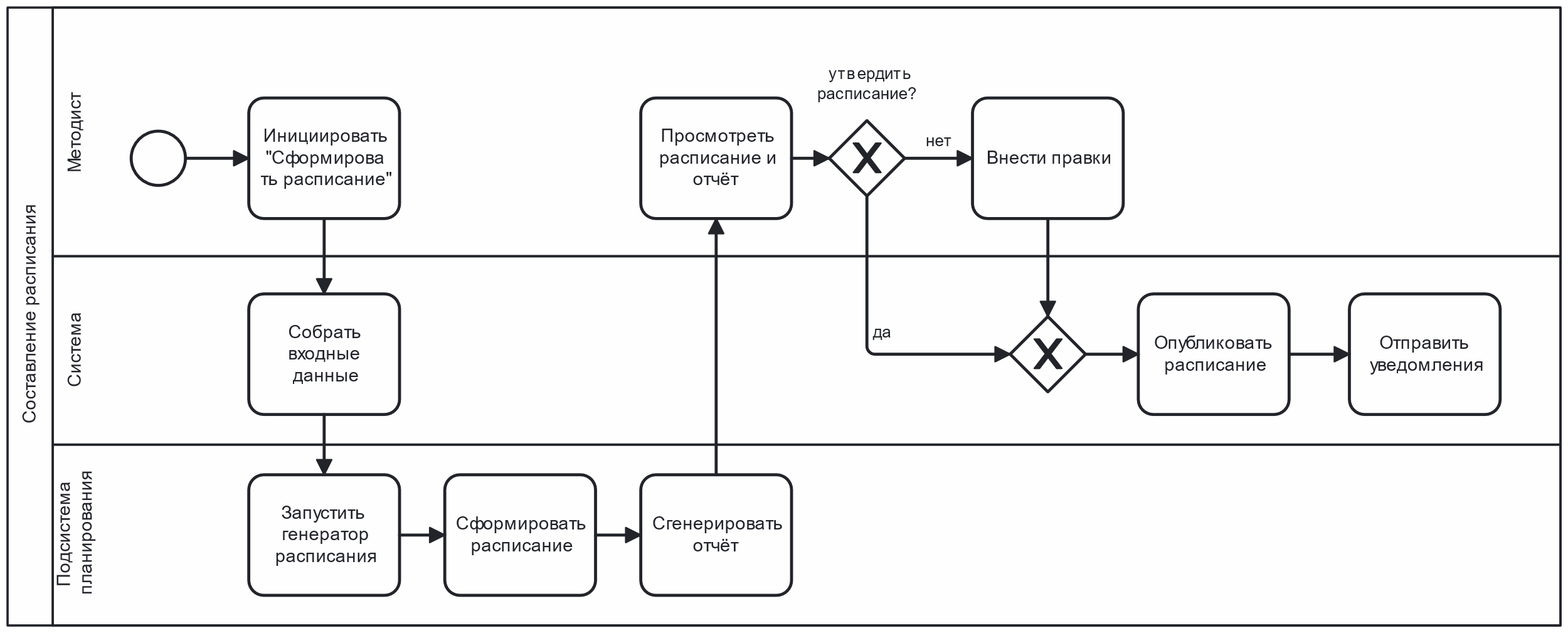


Рисунок 3.3.3 – Диаграмма составления расписания

* + 1. **Диаграмма бронирования аудитории**

Назначение. Обработка заявок на бронирование аудиторий для внеучебных мероприятий с проверкой занятости, утверждением (при необходимости) и подтверждением.

Диаграмма бронирования аудитории представлена на Рисунке 3.3.4

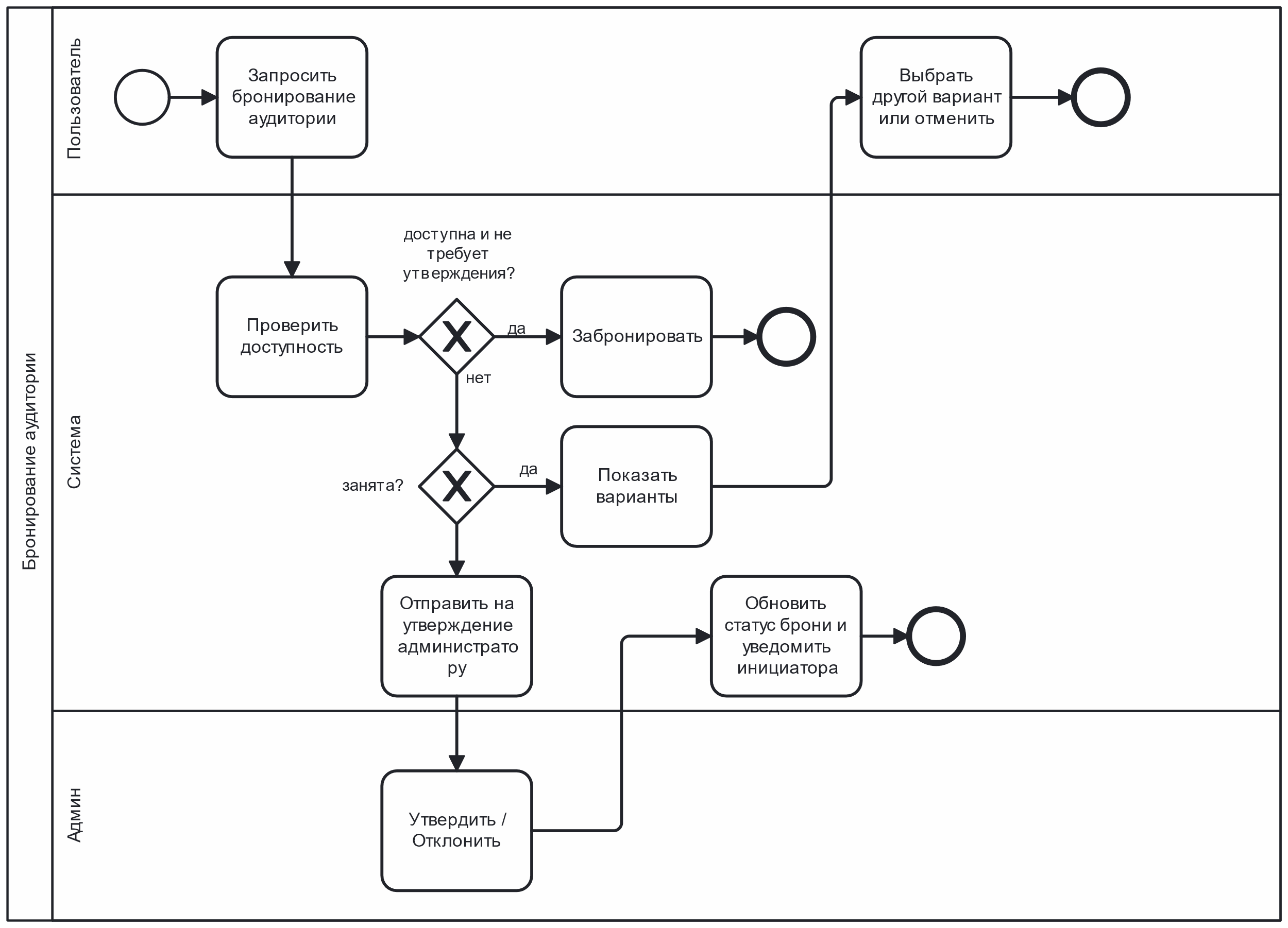


Рисунок 3.3.4 – Диаграмма бронирования аудитории

* + 1. **Диаграмма выставления оценок и формирования ведомостей**

Назначение. Преподаватели выставляют оценки, система формирует ведомости, рассчитывает средний и триггерит списки должников/отчисления.

Диаграмма выставления оценок и формирования ведомостей представлена на Рисунке 3.3.5

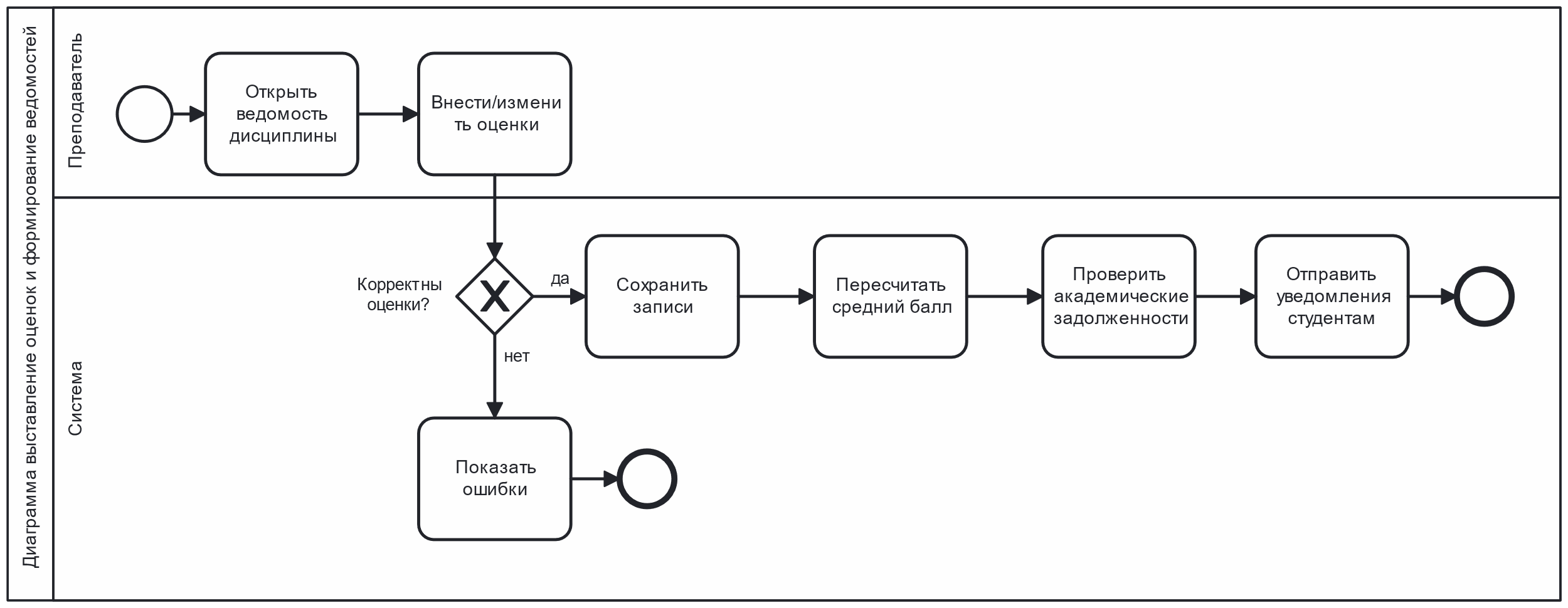


Рисунок 3.3.5 – Диаграмма выставления оценок и формирования ведомостей

* + 1. **Диаграмма управления приказами**

Назначение. Создание, согласование, подписание и публикация приказов (зачисления, отчисления, переводы, кадровые распоряжения).

Диаграмма управления приказами представлена на Рисунке 3.3.6

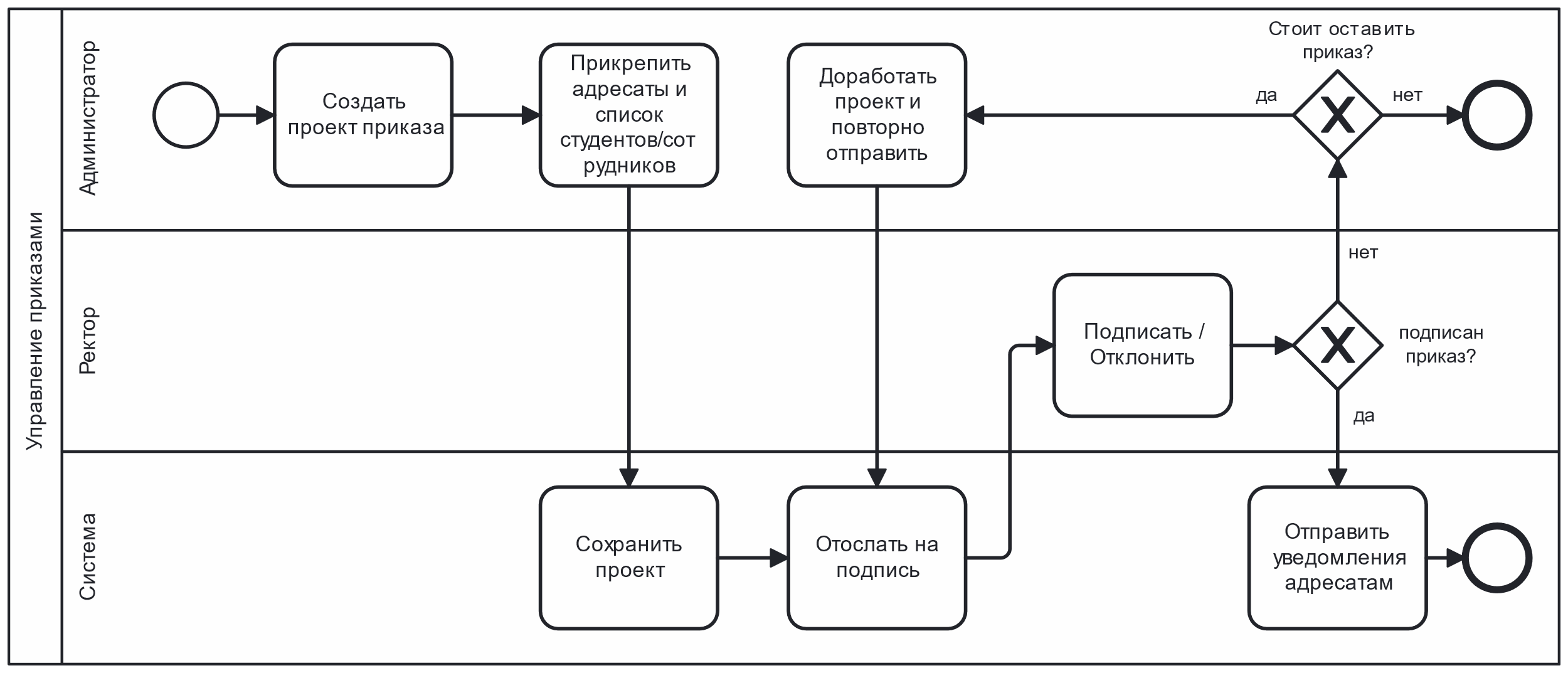


Рисунок 3.3.6 – Диаграмма управления приказами

* + 1. **Диаграмма формирования отчётов**

Назначение. Аггрегация данных, расчёт KPI (средний балл, процент успеваемости, нагрузка), визуализация дашбордов и экспорт отчётов.

Диаграмма формирования отчётов представлена на Рисунке 3.3.7

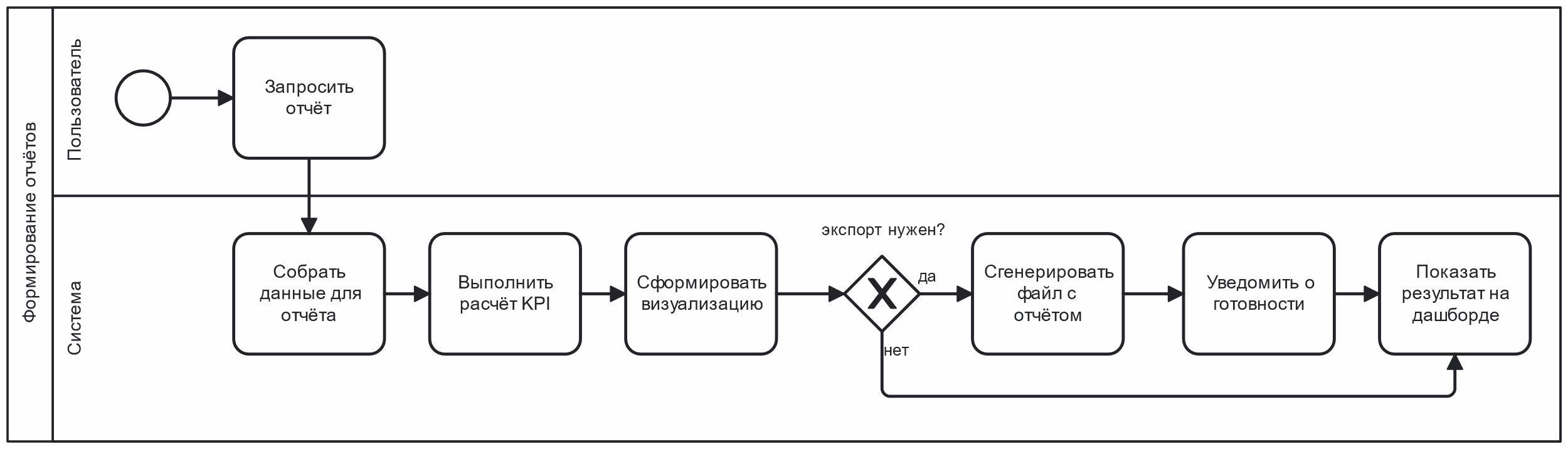


Рисунок 3.3.7 – Диаграмма формирования отчётов

* 1. **Диаграмма состояний**
     1. **Диаграмма запись на электив**

Данная диаграмма описывает жизненный цикл заявки студента на запись на элективный курс. Объект проходит через состояния от подачи заявки до окончательного зачисления или отказа. Учитываются проверки на соответствие учебному плану, наличие мест и конфликты расписания.

Диаграмма запись на электив представлена на Рисунке 3.4.1

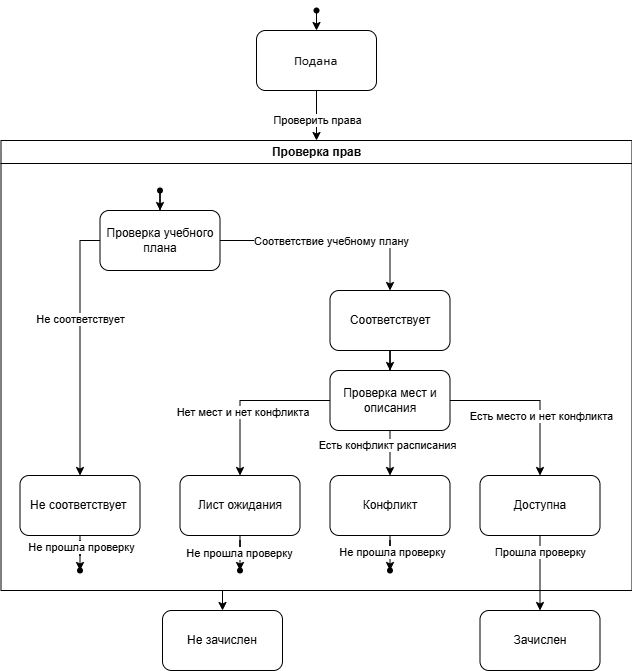


Рисунок 3.4.1 – Диаграмма запись на электив

* + 1. **Диаграмма занятие**

Эта диаграмма описывает жизненный цикл учебного занятия в расписании. Объект проходит через состояния от создания до завершения или отмены, включая проверку конфликтов и учет посещаемости.

Диаграмма занятие представлена на Рисунке 3.4.2

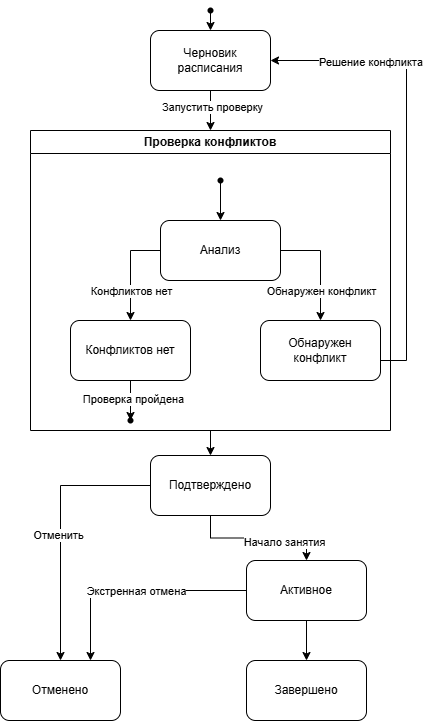


Рисунок 3.4.2 – Диаграмма занятие

* + 1. **Диаграмма ведомость**

Данная диаграмма описывает жизненный цикл ведомости успеваемости по дисциплине. Ведомость создается, затем заполняется оценками, проверяется, и после закрытия становится архивным документом.

Диаграмма ведомость представлена на Рисунке 3.4.3

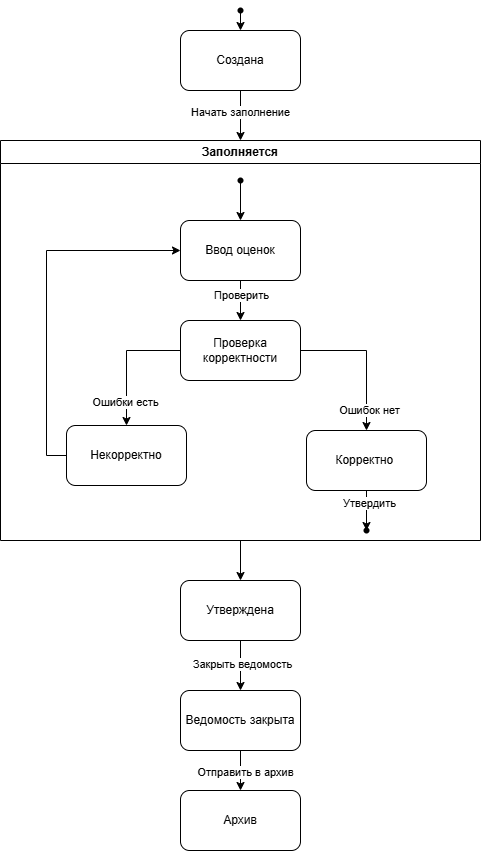


Рисунок 3.4.3 – Диаграмма ведомость

* + 1. **Диаграмма приказ**

Данная диаграмма описывает жизненный цикл приказа в системе. Приказ создается как проект, затем проходит этапы согласования, подписания и, наконец, исполнения. Может быть отозван или отменен на некоторых этапах.

Диаграмма приказ представлена на Рисунке 3.4.4

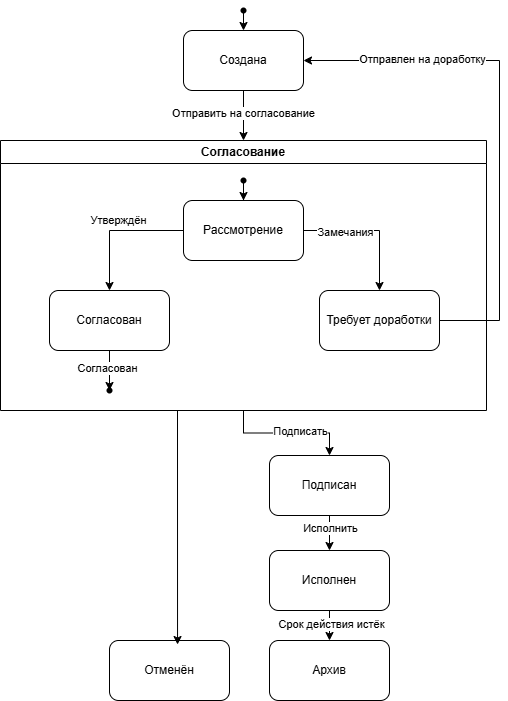


Рисунок 3.4.4 – Диаграмма приказ

* 1. **Заключение и выводы**

1. В результате построения диаграммы прецедентов была определена полная совокупность функциональных возможностей системы с точки зрения конечных пользователей (Студент, Преподаватель, Методист, Администратор) и внешних систем. Диаграмма наглядно демонстрирует, как варианты использования покрывают все сформулированные функциональные требования (FR-001 – FR-041) и обеспечивают взаимодействие между различными ролями в рамках единой информационной среды.
2. Разработанные диаграммы объектов (записи на электив, занятия, ведомости, регистрации приказа) позволили детализировать структуру данных и отношения между ключевыми сущностями предметной области на уровне конкретных сценариев.
3. Построение диаграмм бизнес-процессов (зачисление студента, запись на электив, составление расписания, бронирование аудитории, выставление оценок, управление приказами, формирование отчетов) обеспечило формализацию и оптимизацию ключевых административных и учебных процедур. Диаграммы выявили точки принятия решений, участников процессов и позволили визуализировать переход от ручных, бумажных операций к автоматизированным, что является основой для достижения целевого показателя по сокращению временных издержек.
4. Разработанные диаграммы состояний для сущностей "Запись на электив", "Занятие", "Ведомость" и "Приказ" формализовали их жизненные циклы, определили возможные состояния, переходы между ними и инициирующие события. Это моделирование является фундаментом для корректной реализации логики работы системы, обеспечения согласованности данных и управления бизнес-процессами на всех этапах.
5. Комплекс проведенного моделирования (прецеденты, объекты, процессы, состояния) обеспечил непротиворечивое и полное представление предметной области, что создает надежную основу для перехода к этапу проектирования архитектуры программного обеспечения. Все модели согласованы между собой и с ранее сформулированными требованиями, подтверждая возможность создания системы, которая эффективно устранит выявленные в предметной области нежелательные эффекты.
6. **Архитектура программного обеспечения**
   1. **Определение состава структурных элементов программного обеспечения автоматизированной информационной системы, их назначения и интерфейсов**

Архитектура автоматизированной информационной системы университета представляет собой гибридный подход, сочетающий принципы микросервисной (MSA) и сервисно-ориентированной архитектуры (SOA). Система разделена на ряд независимых микросервисов, каждый из которых реализует отдельную бизнес-функцию, в то время как ядро сервера выполняет функции оркестрации, маршрутизации и авторизации.

Структурные элементы системы делятся на три основных уровня:  
1. **Уровень представления (Клиент):** Веб-браузер пользователей.

Веб-портал (SPA) - единый интерфейс для всех пользователей

Технологии: React, TypeScript, Material-UI, HTML5, CSS3, JavaScript

Назначение: Визуализация данных и взаимодействие с пользователем

2. **Уровень бизнес-логики(Сервер приложений):** Обрабатывает запросы от клиентов, выполняет бизнес-правила и взаимодействует с уровнем данных.

**API Gateway:**

Единая точка входа

Аутентификация и авторизация

Маршрутизация запросов  
**Технологии**:  
Nginx / Kong - высокопроизводительный прокси и маршрутизация

JWT + OAuth 2.0 - безопасная аутентификация и авторизация

Rate Limiting - защита от DDoS и ограничение нагрузки

SSL Termination - обработка HTTPS соединений

**Бизнес-сервисы:**

**Academic Managment Service** - управление студентами, группами, учебными планами, элективами  
**Модули**:

Student Module - управление студентами (зачисление, перевод, отчисление)

Study Group Module - управление учебными группами

Curriculum & Discipline Module - учебные планы и дисциплины

Elective Courses Module - элективные курсы и запись

**Технологии:** Spring Boot, PostgreSQL, Hibernate, REST API

**Scheduling Service** - составление расписания, бронирование аудиторий  
**Модули:**  
Timetable Generator - автоматическое составление расписания

Schedule Management - ручное редактирование расписания

Room Booking - бронирование аудиторий

Conflict Detection - обнаружение конфликтов

**Технологии**: Python, FastAPI, Redis, алгоритмы оптимизации

**Gradebook Service** - ведение успеваемости, оценки, аналитика

**Модули:**  
Grading Module - ввод и управление оценками

Academic Analytics - анализ успеваемости и статистика  
**Технологии:** Java, Spring Data, PostgreSQL, Chart.js

Emloyment Service - управление сотрудниками, кафедрами.

**Модули:**

Employee Module - управление сотрудниками

Workload Module - учебная нагрузка преподавателей

Department Management Module - кафедры и отделы

**Технологии:** Node.js, Express, MongoDB, JWT

**Administrative Service** - приказы, уведомления

**Модули:**

Order Management - приказы и распоряжения

Notification Engine - система уведомлений

Announcements - объявления и новости,

**Технологии**: NET Core, Entity Framework, SQL Server

**Системные сервисы (Инфраструктура)**

**Auth Service -** аутентификация, проверка прав доступа

**Модули:**Authentication Module - аутентификация пользователей

Authorization Module - проверка прав доступа

User Session Module - управление сессиями

RBAC Management Module - ролевая модель  
**Технологии:** Node.js, Express, JWT, OAuth 2.0, Redis

**Reporting Service - Отчетность, кпи**

**Модули:**Standard Reports Module - стандартная отчетность

KPI Module - ключевые показатели эффективности

Data Visualization Module - визуализация данных

**Технологии:** Python, Pandas, ClickHouse, Grafana

**Integration Service -** управление внешними api  
**Модули:**  
API Management Module - управление внешними API

Data Import/Export Module - импорт/экспорт данных

External Systems Adapter Module - адаптеры внешних систем

**Технологии:** Java, Apache Camel, REST/SOAP, XML/JSON

**Audit Service -** сбор и агрегация логов

**Модули:**

Log Collection Module - сбор и агрегация логов

Event Tracking Module - отслеживание событий

Security Monitoring Module - мониторинг безопасности

**Технологии**: Prometheus

**Инфраструктурные компоненты**

**API Gateway -** аутентификация, маршрутизация

Основные функции:

Маршрутизация запросов к соответствующим сервисам

Аутентификация и авторизация запросов

Rate Limiting - ограничение частоты запросов

Кэширование ответов - сохраняет частые запросы для ускорения ответов

**Технологии:** NGINX, Kong, Spring Cloud Gateway

**System Core -** распределение, координация

**Основные функции:**

Координация распределенных транзакций

Валидация бизнес-правил между сервисами

Согласованность данных

Оркестрация сложных процессов

Технологии: Java, Spring Boot, Saga Pattern, Event Sourcing

3. **Уровень данных(Сервер БД):** Хранит и управляет данными.

**Хранилища данных**

**PostgreSQL - основное хранилище данных  
Что хранится:**

Основные операционные данные - вся ключевая информация системы

Транзакционные данные - данные, требующие гарантий целостности

Структурированные данные - данные с четкой схемой и связями

**Redis - In-Memory хранилище**

**Что хранится:**

Кэширование данных - ускорение частых запросов

Сессии пользователей - данные активных сессий

Временные данные - данные с ограниченным временем жизни

**ClickHouse**

**Что хранится:**Аналитические данные

Исторические данные

OLAP-запросы  
**Message Broker (RabbitMQ/Kafka)  
Что хранится:**Асинхронная коммуникация

Событийная архитектура

Гарантированная доставка

**Платформа развертывания**

**Kubernetes**Оркестрация контейнеров

Автомасштабирование

Сервис-меш (Istio/Linkerd)  
**Docker**Контейнеризация приложений

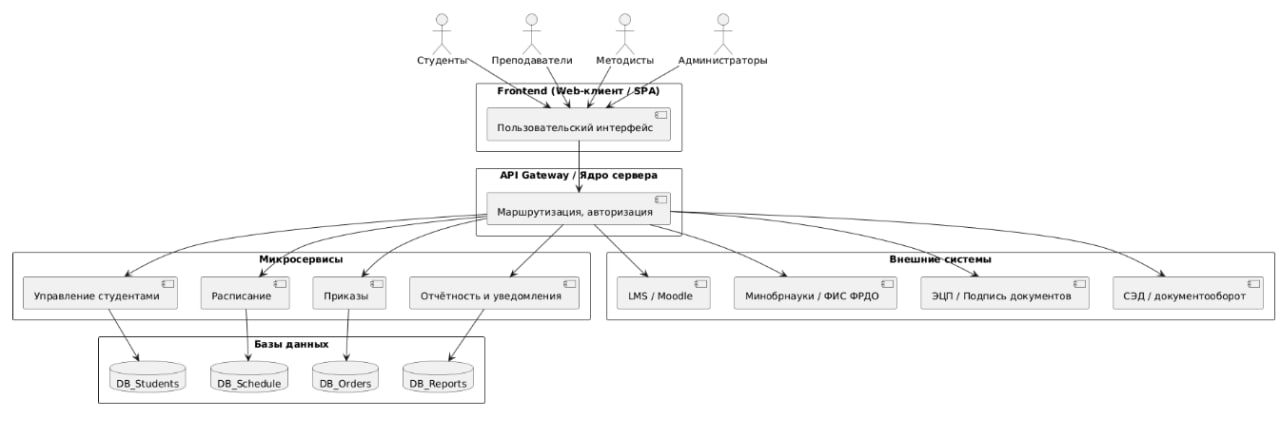
Изоляция зависимостей

**Monitoring Stack**Prometheus + Grafana - метрики и дашборды

Jaeger - распределенная трассировка

Alertmanager - уведомления

Рассмотрим модули и инфраструктуры этих трех уровней.



4.1.1 Поверхностная диаграмма архитектуры  
**4.2 Определение варианта архитектуры и ее разработка**

| **Наименование модуля:** DashboardModule |
| --- |
| **Назначение модуля:** Персонализированная главная страница для разных ролей пользователей. |
| **Варианты использования через состав методов модуля на псевдокоде:**  renderPersonalizedDashboard(userRole) -> dashboardConfig  updateDashboardLayout(userId, newLayout) -> boolean  refreshDashboardData(widgets) -> updatedData  handleUserPreferences(preferences) -> savedPreferences |
| **Структуры и типы данных модуля:**  DashboardConfig {user\_id, role, widgets, layout, theme}  Widget {id, type, title, data\_source, refresh\_interval, position}  UserPreference {user\_id, theme, language, notifications, shortcuts} |
| **Декомпозиция:** являет подмодулем Web Portalсвязан c модулем API Gateway  связан с модулем Auth Service  связан с модулем Redis |
| **Класс или обобщение:**  является подклассом BaseUIModule |

| **Компонент:** DashboardComponent, **Соединитель:** REST API, WebSocket |
| --- |
| **Назначение компонента:** Отображение персонализированной главной страницы с виджетами |
| **Совместно используемые данные:** - |
| **Процесс:**  DashboardInitializationProcess  WidgetDataFetchingProcess  UserInteractionHandler |
| **Параллелизм:**  DashboardRenderThread  RealTimeDataUpdateThread  UserInputHandler  WidgetDataLoader |
| **Клиент-сервер:** Компонент: Клиентский Соединитель: REST API, WebSocket |

| **Наименование модуля:** Schedule UI Module |
| --- |
| **Назначение модуля:** Визуализация и управление расписанием занятий |
| **Варианты использования через состав методов модуля на псевдокоде:**  displaySchedule(viewType, dateRange) -> scheduleData  handleDragAndDrop(lessonId, newTimeslot) -> updateResult  highlightConflicts(scheduleData) -> highlightedSchedule  exportSchedule(format, scheduleId) -> exportFile |
| **Структуры и типы данных модуля:**  ScheduleView {type, date\_range, filters, group\_by}  Lesson {id, discipline, teacher, room, timeslot, group}  Timeslot {day, start\_time, end\_time, duration}  DragDropOperation {source, target, lesson\_data, validation\_rules} |
| **Декомпозиция:** является подмодулем Web Portal  связан с модулем SchedulingService  связан с модулем Academic Service |
| **Класс или обобщение:** |

| **Компонент:** ScheduleComponent, **Соединитель:** REST API, WebSocket |
| --- |
| **Назначение компонента:** Визуальное представление и редактирование расписания с Drag-and-Drop |
| **Совместно используемые данные:** - |
| **Процесс:**  ScheduleRenderingProcess  ConflictDetectionProcess  DragDropValidationProcess  RealTimeUpdateProcess |
| **Параллелизм:**  ScheduleRenderWorker  RealTimeUpdateListener  UserInteractionProcessor  ConflictChecker |
| **Клиент-сервер:** Компонент: Клиентский  Соединитель: REST API, WebSocket для real-time updates |

| **Наименование модуля:** AcademicPerformanceUIModule |
| --- |
| **Назначение модуля:** Отображение и управление успеваемостью студентов |
| **Варианты использования через состав методов модуля на псевдокоде:**  showGrades(studentId, semester) -> gradeSheet  inputGrades(gradeData) -> operationResult  generateProgressChart(studentData) -> chartData  manageGradebook(disciplineId, groupId) -> gradebookView |
| **Структуры и типы данных модуля:**  GradeSheet {student\_id, semester, grades[], gpa, statistics}  GradeInput {discipline\_id, student\_id, grade\_value, grade\_type, date}  ProgressChart {type, data\_points, trends, benchmarks}  GradebookView {discipline\_id, group\_id, students[], grades\_matrix} |
| **Декомпозиция:** является подмодулем WebPortal  связан с модулем Gradebook Service  связан с модулем Academic Service |
| **Класс или обобщение:** |

| **Компонент:** AcademicPerformanceComponent, **Соединитель:** REST API, FileUpload |
| --- |
| **Назначение компонента:** Импорт оценок |
| **Совместно используемые данные:** - |
| **Процесс:**  GradeRenderingProcess  GradeInputValidationProcess  ChartGenerationProcess |
| **Параллелизм:**  - |
| **Клиент-сервер:** Компонент: Клиентский  Соединитель: REST API, WebSocket |

| **Наименование модуля:** UserManagementUIModule |
| --- |
| **Назначение модуля:** Управление пользователями и ролями (для администраторов) |
| **Варианты использования через состав методов модуля на псевдокоде:**  manageUsers(filterCriteria) -> userList  createUser(userData) -> userId  assignRoles(userId, roles) -> operationResult  bulkUserOperations(operation, userList) -> batchResult |
| **Структуры и типы данных модуля:**  UserList {users[], pagination, filters, sort\_order}  UserFormData {personal\_info, contact\_info, role, permissions}  RoleAssignment {user\_id, roles[], effective\_date, expires\_date}  BulkOperation {operation\_type, users[], parameters, results} |
| **Декомпозиция:** является подмодулем WebPortal  связан с модулем AuthSevice  связан с модулем HRService |
| **Класс или обобщение:**  - |

| **Компонент:** UserManagementComponent, **Соединитель:** REST API, WebSocket для real-time updates |
| --- |
| **Назначение компонента:** Визуализация управление пользоватетеялми |
| **Совместно используемые данные: -** |
| **Процесс:**  UserListRenderingProcess  RoleAssignmentWorkflow  BulkOperationProcessor |
| **Параллелизм:**  UserSearchThread  RoleValidationThread  BulkOperationExecutor |
| **Клиент-сервер:** Компонент: Клиентский  Соединитель: REST API, WebSocket |

| **Наименование модуля:** ReportingUIModule |
| --- |
| **Назначение модуля:** Создание и просмотр отчетов и аналитики |
| **Варианты использования через состав методов модуля на псевдокоде:**  generateReport(reportType, parameters) -> reportData  createCustomReport(configuration) -> customReportId  visualizeKPI(kpiCodes, period) -> visualizationData  exportReport(reportId, format) -> downloadLink |
| **Структуры и типы данных модуля:**  ReportConfig {type, parameters, filters, format, visualization}  KPIWidget {kpi\_code, period, value, target, trend, visualization}  CustomReport {id, name, owner, query, charts, access\_rights}  Visualization {type, data, options, interactivity} |
| **Декомпозиция:** является подмодулем WebPortal  связан с модулем Reporting Service  связан с модулем Integration Service |
| **Класс или обобщение:**  - |

| **Компонент:** ReportingComponent, **Соединитель:** REST API, WebSocket для real-time updates |
| --- |
| **Назначение компонента:** Интерфейс для генерации отчетов и визуализации аналитических данных. |
| **Совместно используемые данные: -** |
| **Процесс:**  ReportGenerationProcess  DataVisualizationProcess  ExportHandlerProcess  ChartRenderingProcess |
| **Параллелизм:**  ReportGenerator  ChartRenderer  DataProcessor  FileDownloader |
| **Клиент-сервер:** Компонент: Клиентский  Соединитель: REST API, File Download |

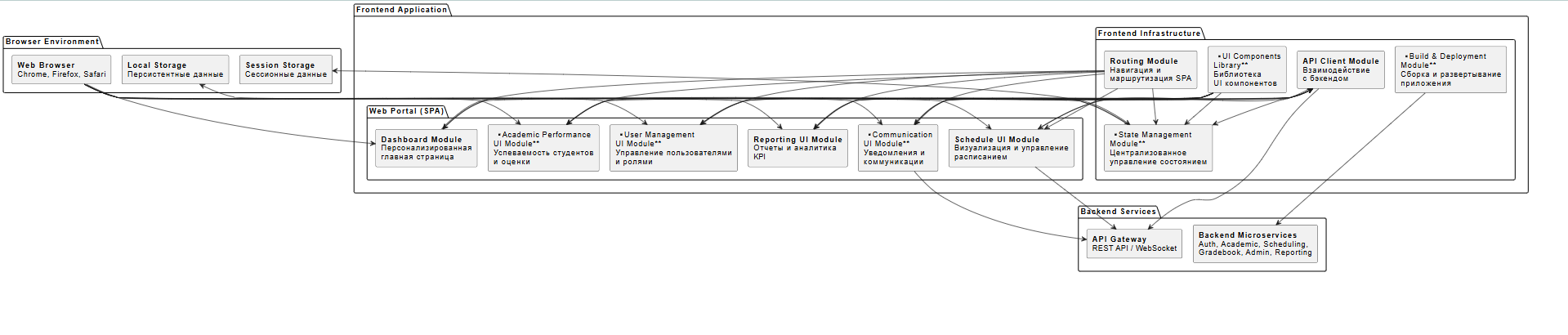
| **Наименование модуля:** CommunicationUIModule |
| --- |
| **Назначение модуля:** Управление уведомлениями и коммуникациями |
| **Варианты использования через состав методов модуля на псевдокоде:**  sendNotification(recipients, message) -> notificationId  manageAnnouncements(announcementData) -> operationResult  handleRealTimeNotifications() -> notificationStream  markAsRead(notificationId) -> status |
| **Структуры и типы данных модуля:**  Notification {id, type, title, message, priority, timestamp, read\_status}  Announcement {id, title, content, audience, publish\_date, expiry}  MessageThread {id, participants[], messages[], last\_activity}  RealTimeEvent {type, data, timestamp, source} |
| **Декомпозиция:** является подмодулем WebPortal  связан с модулем Administrative Service  связан с модулем WebSocket |
| **Класс или обобщение:**  - |

| **Компонент:** CommunicationComponent, **Соединитель:** REST API, WebSocket для real-time updates |
| --- |
| **Назначение компонента:** Система уведомлений и коммуникаций между пользователями |
| **Совместно используемые данные: -** |
| **Процесс:**  NotificationDeliveryProcess  RealTimeEventHandler  MessageRenderingProcess  AnnouncementManagementProcess |
| **Параллелизм:**  WebSocketListener  NotificationProcessor  UIUpdateThread  MessageSender |
| **Клиент-сервер:** Компонент: Клиентский  Соединитель: REST API, WebSocket |

**Инфраструктура Frontend**

| **Наименование модуля:** Routing Module |
| --- |
| **Назначение модуля:** Управление навигацией и маршрутизацией в SPA |
| **Варианты использования через состав методов модуля на псевдокоде:**  navigateTo(route, parameters) -> navigationResult  handleBrowserHistory() -> historyState  protectRoutes(route, userRole) -> accessGranted  lazyLoadRoute(route) -> loadedModule |
| **Структуры и типы данных модуля:**  Route {path, component, guards, data, children}  NavigationState {current\_route, previous\_route, parameters, history}  RouteGuard {can\_activate, can\_deactivate, can\_load} |
| **Декомпозиция:** является подмодулем Frontend Infrastructure  связан с модулем Auth Service  связан со всеми модулями UI |
| **Класс или обобщение:**  - |

| **Компонент:** RouterComponent, **Соединитель:** Browser History API, Hash Routing |
| --- |
| **Назначение компонента:** Маршрутизатор для навигации между страницами приложения |
| **Совместно используемые данные: -** |
| **Процесс:**  RouteMatchingProcess  GuardValidationProcess  LazyLoadingProcess  HistoryManagementProcess |
| **Параллелизм:**  RouteResolver  GuardChecker  ModuleLoader |
| **Клиент-сервер:** Компонент: Клиентский  Соединитель: Browser History API, Hash Routing |



4.1.1 Диаграмма структурных элементов системы уровня представления

**4.1.2 Уровень бизнес-логики(Сервер приложений)**

**Ядро Системы (System Core)**

| **Наименование модуля:** API Gateway |
| --- |
| **Назначение модуля:** Единая точка входа для всех клиентских запросов, маршрутизация и управление трафиком |
| **Варианты использования через состав методов модуля на псевдокоде:**  routeRequest(request) -> response  authenticateRequest(token) -> userClaims  applyRateLimiting(clientId) -> quotaStatus  cacheResponse(cacheKey, data) -> cached |
| **Структуры и типы данных модуля:**  RouteConfig {path, service, methods, authentication}  RateLimit {client\_id, requests\_count, window\_start}  CacheEntry {key, data, expiration, hit\_count} |
| **Декомпозиция:**  связан со всеми микросервисами  связан с модулем Auth Service |
| **Класс или обобщение:**  - |

| **Компонент:** RequestRouter, **Соединитель:** HTTP/HTTPS, WebSocket, gRPC |
| --- |
| **Назначение компонента:** Маршрутизация входящих запросов к соответствующим микросервисам |
| **Совместно используемые данные:**  RoutingTable {routes[], load\_balancing, health\_checks}  RequestContext {request\_id, user, timestamp, headers}  ServiceRegistry {services[], endpoints, status, version} |
| **Процесс:**  RequestRoutingProcess  AuthenticationValidationProcess  RateLimitEnforcementProcess  ResponseAggregationProcess |
| **Параллелизм:**  RequestHandlerThreads  AuthenticationValidator  RateLimitChecker  CacheManager |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: HTTPS, WebSocket, gRPC |

**Микросервис для аутентификации Auth Service**

| **Наименование модуля:** Authentication Module |
| --- |
| **Назначение модуля:** Аутентификация пользователей и управление сессиями |
| **Варианты использования через состав методов модуля на псевдокоде:**  authenticateUser(credentials) -> JWT token  validateToken(token) -> userClaims  refreshToken(oldToken) -> newToken  invalidateSession(sessionId) -> boolean |
| **Структуры и типы данных модуля:**  UserCredentials {user\_id, login, password\_hash, salt}  JWToken {token, user\_id, issued\_at, expires\_at, scope}  UserSession {session\_id, user\_id, ip\_address, created\_at} |
| **Декомпозиция:** является подмодулем Auth Service  связан с модулем User Session Module  связан с PostgreSQL |
| **Класс или обобщение:**  - |

| **Компонент:** AuthManager, **Соединитель:** REST API, Database connections |
| --- |
| **Назначение компонента:** Маршрутизация входящих запросов к соответствующим микросервисам |
| **Совместно используемые данные:**  AuthConfig {jwt\_secret, token\_expiry, refresh\_token\_expiry}  UserSessions {active\_sessions, session\_timeout, max\_sessions}  SecurityPolicy {password\_policy, lockout\_policy, mfa\_required} |
| **Процесс:**  UserAuthenticationProcess  TokenGenerationProcess  SessionManagementProcess  SecurityPolicyEnforcement |
| **Параллелизм:**  AuthenticationWorker  TokenValidator  SessionCleanup |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, Database connections |

| **Наименование модуля:** Authorization Module |
| --- |
| **Назначение модуля:** Проверка прав доступа пользователей к ресурсам системы |
| **Варианты использования через состав методов модуля на псевдокоде:**  checkPermission(userId, resource, action) -> boolean  getUserRoles(userId) -> role[]  getRolePermissions(roleId) -> permission[]  evaluateAccessPolicy(user, resource) -> accessDecision |
| **Структуры и типы данных модуля:**  Permission {id, resource\_type, resource\_id, actions}  Role {id, name, description, permissions}  AccessPolicy {policy\_id, conditions, effect, priority} |
| **Декомпозиция:** является подмодулем Auth Service  связан с модулем RBAC Management Module |
| **Класс или обобщение:**  - |

| **Компонент:** AuthorizationEngine, **Соединитель:** gRPC, Database connections |
| --- |
| **Назначение компонента:** Движок для оценки прав доступа на основе RBAC |
| **Совместно используемые данные:**  RBACModel {roles, permissions, user\_assignments}  PolicySet {policies[], evaluation\_order, default\_decision}  AccessCache {user\_id, permissions\_cache, expiration} |
| **Процесс:**  PermissionEvaluationProcess  RoleResolutionProcess  PolicyEnforcementProcess  AccessDecisionProcess |
| **Параллелизм:**  PermissionEvaluator  RoleResolver  PolicyEngine |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: gRPC, Database connections |

| **Наименование модуля:** User Session Module |
| --- |
| **Назначение модуля:** Управление пользовательскими сессиями и контроль активных подключений |
| **Варианты использования через состав методов модуля на псевдокоде:**  createSession(userId, deviceInfo) -> sessionId  invalidateSession(sessionId) -> boolean  getActiveSessions(userId) -> session[]  refreshSessionActivity(sessionId) -> updatedSession |
| **Структуры и типы данных модуля:**  UserSession {session\_id, user\_id, created\_at, last\_activity, expires\_at, device\_info}  DeviceInfo {ip, user\_agent, device\_type, location, browser}  SessionActivity {session\_id, actions[], timestamps, resource\_access} |
| **Декомпозиция:** является подмодулем Auth Service  связан с модулем Authentication Module  связан с Authorization Module |
| **Класс или обобщение:**  является подклассом BaseSessionService |

| **Компонент:** SessionManager, **Соединитель:** Database connections, Cache, REST API |
| --- |
| **Назначение компонента:** Централизованное управление жизненным циклом пользовательских сессий |
| **Совместно используемые данные:**  SessionStore {active\_sessions, session\_metadata, expiration\_queue}  ConcurrencyControl {max\_sessions\_per\_user, session\_timeout\_policies}  SecurityPolicies {session\_fixation\_protection, concurrent\_login\_rules} |
| **Процесс:**  SessionCreationProcess  SessionValidationProcess  TimeoutEnforcementProcess  CleanupProcess |
| **Параллелизм:**  SessionCreator  SessionValidator  TimeoutMonitor  CleanupWorker |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: Database connections, Cache, REST API |

| **Наименование модуля:** RBAC Management Module |
| --- |
| **Назначение модуля:** Управление ролевой моделью доступа и распределением прав |
| **Варианты использования через состав методов модуля на псевдокоде:**  createRole(roleName, permissions) -> roleId  assignRole(userId, roleId) -> boolean  updateRolePermissions(roleId, permissions) -> boolean  getRoleHierarchy() -> hierarchy |
| **Структуры и типы данных модуля:**  Role {id, name, description, is\_system, permissions, parent\_role}  RoleAssignment {user\_id, role\_id, assigned\_by, effective\_date, expires\_date}  PermissionSet {role\_id, permissions[], constraints, conditions} |
| **Декомпозиция:** является подмодулем Auth Service  связан с System Core  связан с модулем Authorization Module |
| **Класс или обобщение:**  является подклассом BaseRBACService |

| **Компонент:** RoleManager, **Соединитель:** Database connections, REST API, Message Queue |
| --- |
| **Назначение компонента:** Администрирование ролевой модели и политик доступа |
| **Совместно используемые данные:**  RBACModel {roles, permissions, user\_assignments, role\_hierarchy}  PolicyRegistry {access\_policies, inheritance\_rules, conflict\_resolution}  AuditTrail {role\_changes, permission\_updates, assignment\_history} |
| **Процесс:**  RoleManagementProcess  PermissionAssignmentProcess  UserRoleMappingProcess  PolicyEnforcementProcess |
| **Параллелизм:**  RoleProcessor  PermissionManager  AssignmentHandler  PolicyEngine |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: Database connections, REST API, Message Queue |

**Academic Management Service – сервис управления учебным процессом**

| **Наименование модуля:** Student Module |
| --- |
| **Назначение модуля:** Управление данными студентов и академическими процессами |
| **Варианты использования через состав методов модуля на псевдокоде:**  enrollStudent(studentData) -> studentId  updateStudentInfo(studentId, updates) -> boolean  transferStudent(studentId, newGroupId) -> boolean  searchStudents(criteria) -> student[] |
| **Структуры и типы данных модуля:**  Student {id, personal\_info, contact\_info, enrollment\_info}  StudentStatus {status, enrollment\_date, graduation\_date}  AcademicHistory {student\_id, transfers, academic\_breaks} |
| **Декомпозиция:** является подмодулем Academic Management Service  связан с System Core  связан с Study Group Module |
| **Класс или обобщение:**  - |

| **Компонент:** StudentManager, **Соединитель:** REST API, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Движок для оценки прав доступа на основе RBAC |
| **Совместно используемые данные:**  StudentRegistry {students[], status\_updates, audit\_log}  EnrollmentRules {admission\_rules, transfer\_rules, graduation\_rules}  AcademicCalendar {semesters, deadlines, holidays} |
| **Процесс:**  StudentEnrollmentProcess  InformationUpdateProcess  AcademicTransferProcess  StatusManagementProcess |
| **Параллелизм:**  EnrollmentProcessor  DataValidator  StatusUpdater |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, Database connections, Message Queue |

| **Наименование модуля:** Curriculum and Discipline Module |
| --- |
| **Назначение модуля:** Управление учебными планами, дисциплинами и программами обучения |
| **Варианты использования через состав методов модуля на псевдокоде:**  createDiscipline(disciplineData) -> disciplineId  createCurriculum(curriculumData) -> curriculumId  assignDisciplineToCurriculum(curriculumId, disciplineId, semester) -> boolean  validateCurriculumCompleteness(curriculumId) -> validationResult |
| **Структуры и типы данных модуля:**  Discipline {id, code, name, department\_id, credits, hours}  Curriculum {id, specialty\_id, year, status, requirements}  CurriculumItem {curriculum\_id, discipline\_id, semester, hours\_type} |
| **Декомпозиция:** является подмодулем Academic Management Service  связан с System Core  связан с Scheduling Service |
| **Класс или обобщение:**  - |

| **Компонент:** CurriculumEngine, **Соединитель:** REST API, Database connections. |
| --- |
| **Назначение компонента:** Движок для управления и валидации учебных планов |
| **Совместно используемые данные:**  DisciplineCatalog {disciplines[], departments, prerequisites}  CurriculumTemplates {templates[], approval\_workflows, versions}  AcademicStandards {credit\_requirements, hour\_standards, compliance\_rules} |
| **Процесс:**  DisciplineManagementProcess  CurriculumCreationProcess  ComplianceValidationProcess  ApprovalWorkflowProcess |
| **Параллелизм:**  CurriculumValidator  ComplianceChecker  ApprovalProcessor |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, Database connections. |

| **Наименование модуля:** Study Group Module |
| --- |
| **Назначение модуля:** Управление учебными группами и их составом |
| **Варианты использования через состав методов модуля на псевдокоде:**  createGroup(groupData) -> groupId  updateGroup(groupId, updates) -> boolean  addStudentToGroup(studentId, groupId) -> boolean  getGroupStudents(groupId) -> student[] |
| **Структуры и типы данных модуля:**  StudyGroup {id, name, code, specialty\_id, curator\_id, year, status}  GroupMembership {group\_id, student\_id, join\_date, leave\_date, status}  GroupStatistics {group\_id, student\_count, average\_performance, attendance} |
| **Декомпозиция:** является подмодулем Academic Management Service  связан с Student Module  связан с Curriculum Module |
| **Класс или обобщение:**  является подклассом BaseGroupService |

| **Компонент:** GroupOrganizer, **Соединитель:** Соединитель: REST API, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Организация и управление академическими группами |
| **Совместно используемые данные:**  GroupRegistry {groups[], memberships, statistics, history}  FormationRules {group\_size\_limits, specialization\_rules, balancing\_algorithms}  CuratorAssignments {curators, responsibilities, performance\_metrics} |
| **Процесс:**  GroupCreationProcess  MembershipManagementProcess  StatisticsCalculationProcess  CuratorAssignmentProcess |
| **Параллелизм:**  GroupCreator  MembershipManager  Statistician  CuratorAssigner |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: Соединитель: REST API, Database connections, Message Queue |

| **Наименование модуля:** Elective Courses Module |
| --- |
| **Назначение модуля:** Управление элективными курсами и процессом записи студентов |
| **Варианты использования через состав методов модуля на псевдокоде:**  registerForElective(studentId, courseId) -> registrationId  getAvailableElectives(studentId) -> elective[]  cancelElectiveRegistration(registrationId) -> boolean  manageElectiveQuotas(courseId, quotas) -> quotaStatus |
| **Структуры и типы данных модуля:**  ElectiveCourse {id, discipline\_id, teacher\_id, max\_students, current\_students, semester}  ElectiveRegistration {id, student\_id, course\_id, registration\_date, status, priority}  QuotaManagement {course\_id, total\_quota, reserved\_quota, available\_quota} |
| **Декомпозиция:** является подмодулем Academic Management Service  связан с Scheduling Service  связан с Curriculum Module |
| **Класс или обобщение:**  является подклассом BaseElectiveService |

| **Компонент:** ElectiveCoordinator, **Соединитель:** Соединитель: REST API, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Координация процесса записи на элективные курсы |
| **Совместно используемые данные:**  ElectiveCatalog {courses[], availability, prerequisites, schedules}  RegistrationWorkflow {registration\_periods, approval\_process, waitlist\_management}  QuotaSystem {quota\_allocations, reservation\_policies, overflow\_handling} |
| **Процесс:**  CourseRegistrationProcess  QuotaManagementProcess  WaitlistProcessing  ScheduleCoordinationProcess |
| **Параллелизм:**  RegistrationProcessor  QuotaManager  WaitlistHandler  ScheduleCoordinator |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: Соединитель: REST API, Database connections, Message Queue |

**Микросервис Scheduling Service - сервис расписани. Планирование и оптимизация**

| **Наименование модуля:** Timetable Generator Module |
| --- |
| **Назначение модуля:** Автоматическое составление оптимального расписания занятий |
| **Варианты использования через состав методов модуля на псевдокоде:**  generateTimetable(constraints) -> scheduleId  optimizeSchedule(scheduleId, criteria) -> optimizedSchedule  validateScheduleFeasibility(scheduleDraft) -> validationResult  getGenerationProgress(scheduleId) -> progressStatus |
| **Структуры и типы данных модуля:**  ScheduleConstraint {type, weight, parameters, hard\_soft}  OptimizationCriteria {priority\_weights, constraints, objectives}  GenerationResult {schedule\_id, fitness\_score, violations} |
| **Декомпозиция:** является подмодулем Scheduling Service  связан с Conflict Detection Module  связан с System Core |
| **Класс или обобщение:**  - |

| **Компонент:** ScheduleGenerator, **Соединитель:** gRPC, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Алгоритмический движок для генерации расписания |
| **Совместно используемые данные:**  ConstraintSet {hard\_constraints, soft\_constraints, priorities}  ResourcePool {rooms, teachers, time\_slots, equipment}  AlgorithmConfig {parameters, heuristics, termination\_conditions} |
| **Процесс:**  ConstraintSatisfactionProcess  ResourceAllocationProcess  OptimizationProcess  FeasibilityCheckProcess |
| **Параллелизм:**  ConstraintSolver  ResourceAllocator  OptimizationWorker |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: gRPC, Database connections, Message Queue |

| **Наименование модуля:** Conflict Detection Module |
| --- |
| **Назначение модуля:** Обнаружение и разрешение конфликтов в расписании |
| **Варианты использования через состав методов модуля на псевдокоде:**  detectConflicts(scheduleId) -> conflict[]  suggestConflictResolutions(conflictId) -> resolution[]  resolveConflict(conflictId, resolution) -> boolean  monitorRealTimeConflicts() -> conflictAlerts |
| **Структуры и типы данных модуля:**  ScheduleConflict {id, type, severity, entities\_involved, description}  ConflictResolution {conflict\_id, resolution\_type, parameters}  ConflictAlert {conflict\_id, timestamp, urgency, notified\_users} |
| **Декомпозиция:** является подмодулем Scheduling Service  связан с Timetable Generator Module  связан с Room Booking Module |
| **Класс или обобщение:**  - |

| **Компонент:** ConflictResolver, **Соединитель:** REST API, WebSocket, Database connections |
| --- |
| **Назначение компонента:** Система обнаружения и автоматического разрешения конфликтов |
| **Совместно используемые данные:**  ConflictRules {detection\_rules, resolution\_strategies, priorities}  ResolutionHistory {resolved\_conflicts, applied\_solutions, effectiveness}  RealTimeMonitor {active\_monitors, alert\_triggers, notification\_channels} |
| **Процесс:**  ConflictDetectionProcess  ResolutionSuggestionProcess  AutoResolutionProcess  AlertGenerationProcess |
| **Параллелизм:**  ConflictDetector  ResolutionSuggester  AlertGenerator |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, WebSocket, Database connections |

| **Наименование модуля:** Schedule Management Module |
| --- |
| **Назначение модуля:** Ручное управление и корректировка расписания занятий |
| **Варианты использования через состав методов модуля на псевдокоде:**  updateScheduleEntry(entryId, updates) -> boolean  moveLesson(lessonId, newTimeslot) -> boolean  approveSchedule(scheduleId, approverId) -> boolean  getScheduleHistory(scheduleId) -> history |
| **Структуры и типы данных модуля:**  ScheduleEntry {id, group\_id, discipline\_id, teacher\_id, room\_id, timeslot, status}  ScheduleChange {entry\_id, old\_values, new\_values, changed\_by, reason, timestamp}  ApprovalWorkflow {schedule\_id, approvers[], status, comments, deadlines} |
| **Декомпозиция:** является подмодулем Scheduling Service  связан с Timetable Generator Module  связан с Conflict Detection Module |
| **Класс или обобщение:**  является подклассом BaseScheduleService |

| **Компонент:** ScheduleEditor, **Соединитель:** REST API, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Инструменты для ручного редактирования и утверждения расписания |
| **Совместно используемые данные:**  ScheduleVersioning {versions, change\_log, rollback\_points}  ApprovalMatrix {approvers, delegation\_rules, escalation\_policies}  ChangeImpact {impact\_assessment, notification\_triggers, dependency\_analysis} |
| **Процесс:**  ScheduleEditingProcess  ChangeValidationProcess  ApprovalWorkflowProcess  HistoryTrackingProcess |
| **Параллелизм:**  ScheduleEditor  ChangeValidator  ApprovalManager  HistoryTracker |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, Database connections, Message Queue |

| **Наименование модуля:** Room Booking Module |
| --- |
| **Назначение модуля:** Управление бронированием аудиторий для учебных и внеучебных мероприятий |
| **Варианты использования через состав методов модуля на псевдокоде:**  reserveRoom(bookingRequest) -> bookingId  checkRoomAvailability(roomId, timeRange) -> boolean  cancelBooking(bookingId) -> boolean  getRoomSchedule(roomId, date) -> schedule |
| **Структуры и типы данных модуля:**  Room {id, number, building, capacity, equipment, type, status}  Booking {id, room\_id, purpose, organizer\_id, time\_slot, recurring, status}  RoomAvailability {room\_id, date, time\_slots, reservations, conflicts} |
| **Декомпозиция:** является подмодулем Scheduling Service  связан с Conflict Detection Module  связан с System Core |
| **Класс или обобщение:**  является подклассом BaseRoomService |

| **Компонент:** RoomBookingManager, **Соединитель:** REST API, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Система бронирования и управления доступностью помещений |
| **Совместно используемые данные:**  RoomInventory {rooms[], equipment, maintenance\_schedule, availability}  BookingRules {booking\_policies, priority\_rules, conflict\_resolution}  RecurringBookings {recurring\_patterns, exception\_dates, auto\_renewal} |
| **Процесс:**  RoomReservationProcess  AvailabilityCheckProcess  ConflictResolutionProcess  MaintenanceSchedulingProcess |
| **Параллелизм:**  ReservationProcessor  AvailabilityChecker  ConflictResolver  MaintenanceScheduler |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, Database connections, Message Queue |

**Gradebook Service - сервис для ввода оценок, журнал успеваемости и анализа среднего показателя успеваемости**

| **Наименование модуля:** Academic Analytics Module |
| --- |
| **Назначение модуля:** Анализ академической успеваемости и выявление тенденций |
| **Варианты использования через состав методов модуля на псевдокоде:**  calculateGPA(studentId) -> gpa  generateAcademicTranscript(studentId) -> transcript  identifyAtRiskStudents(criteria) -> student[]  analyzePerformanceTrends(parameters) -> trends |
| **Структуры и типы данных модуля:**  AcademicRecord {student\_id, semester, gpa, credits\_earned, standing}  RiskIndicator {student\_id, indicator\_type, severity, description, timestamp}  PerformanceTrend {student\_id, period, metric, value, confidence} |
| **Декомпозиция:** является подмодулем Gradebook Service  связан с Grading Module  связан с System Core |
| **Класс или обобщение:**  является подклассом BaseAnalyticsService |

| **Компонент:** AcademicAnalyzer **Соединитель:** REST API, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Централизованное управление процессом оценивания |
| **Совместно используемые данные:**  AnalyticsModels {prediction\_models, clustering\_algorithms, trend\_analysis}  RiskAssessment {risk\_factors, early\_warning\_indicators, intervention\_triggers}  PerformanceBenchmarks {standards, historical\_comparisons, peer\_analysis} |
| **Процесс:**  GPACalculationProcess  RiskAssessmentProcess  TrendAnalysisProcess  TranscriptGenerationProcess |
| **Параллелизм:**  GPACalculator  RiskAssessor  TrendAnalyzer  TranscriptGenerator |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, WebSocket, Database connections |

| **Наименование модуля:** Grading Module |
| --- |
| **Назначение модуля:** Ввод, хранение и управление академическими оценками |
| **Варианты использования через состав методов модуля на псевдокоде:**  recordGrade(gradeData) -> gradeId  updateGrade(gradeId, newGrade) -> boolean  getStudentGrades(studentId, semester) -> grade[]  generateGradeSheet(disciplineId, groupId) -> gradeSheet |
| **Структуры и типы данных модуля:**  Grade {id, student\_id, discipline\_id, grade\_value, grade\_type, date}  GradeSheet {discipline\_id, group\_id, semester, grades[], statistics}  Assessment {discipline\_id, type, max\_score, weight, deadline} |
| **Декомпозиция:** является подмодулем Gradebook Service  связан с Academic Analytics Module  связан с System Core |
| **Класс или обобщение:**  - |

| **Компонент:** GradeManager **Соединитель:** REST API, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Централизованное управление процессом оценивания |
| **Совместно используемые данные:**  GradeRegistry {grades[], assessments, grade\_scale}  GradingRules {grading\_system, calculation\_rules, rounding\_policy}  AssessmentCalendar {deadlines, submission\_windows, extension\_requests} |
| **Процесс:**  GradeRecordingProcess  GradeValidationProcess  SheetGenerationProcess  AssessmentManagementProcess |
| **Параллелизм:**  GradeRecorder  Validator  SheetGenerator |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, WebSocket, Database connections |

**Administrative Service - сервис для управления административными процессами университета.**

| **Наименование модуля:** Order Management Module |
| --- |
| **Назначение модуля:** Управление приказами, распоряжениями и официальными документами университета |
| **Варианты использования через состав методов модуля на псевдокоде:**  createOrder(orderData) -> orderId  signOrder(orderId, signatoryId) -> boolean  publishOrder(orderId) -> boolean  searchOrders(criteria) -> order[] |
| **Структуры и типы данных модуля:**  Order {id, type, number, date, content, status, signatory\_id, effective\_date}  OrderTemplate {type, template\_content, required\_fields, approval\_workflow}  OrderWorkflow {order\_type, approval\_steps, signatories, notifications} |
| **Декомпозиция:** является подмодулем Administrative Service  связан с Notification Engine Module  связан с System Core |
| **Класс или обобщение:**  является подклассом BaseDocumentService |

| **Компонент:** OrderProcessor **Соединитель:** Message Queue, SMTP, WebSocket, Database connections |
| --- |
| **Назначение компонента:** Обработка жизненного цикла приказов от создания до публикации |
| **Совместно используемые данные:**  OrderRegistry {orders[], templates, workflows, numbering\_sequence}  ApprovalMatrix {approvers, delegation\_rules, escalation\_policies}  DocumentArchive {archived\_orders, retention\_policy, search\_index} |
| **Процесс:**  OrderCreationProcess  ApprovalWorkflowProcess  SigningProcess  PublicationProcess |
| **Параллелизм:**  OrderCreator  ApprovalManager  SigningProcessor  Publisher |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: Message Queue, SMTP, WebSocket, Database connections |

| **Наименование модуля:** Notification Engine Module |
| --- |
| **Назначение модуля:** Управление системой уведомлений и коммуникаций |
| **Варианты использования через состав методов модуля на псевдокоде:**  sendNotification(notification) -> notificationId  scheduleNotification(notification, schedule) -> jobId  getUserNotifications(userId) -> notification[]  markAsRead(notificationId) -> boolean |
| **Структуры и типы данных модуля:**  Notification {id, type, title, message, recipient, priority, status}  NotificationTemplate {type, subject\_template, body\_template, variables}  DeliverySchedule {schedule\_id, trigger\_condition, recurrence} |
| **Декомпозиция:** является подмодулем Administrative Service  связан с Message Broker  связан со всеми микросервисами |
| **Класс или обобщение:**  - |

| **Компонент:** NotificationDispatcher **Соединитель:** Message Queue, SMTP, WebSocket, Database connections |
| --- |
| **Назначение компонента:** Диспетчер для управления доставкой уведомлений через различные каналы |
| **Совместно используемые данные:**  DeliveryChannels {email, sms, push, internal}  NotificationQueue {pending\_notifications, priority\_queue, retry\_count}  TemplateRegistry {templates[], variables, localization} |
| **Процесс:**  NotificationRoutingProcess  TemplateRenderingProcess  DeliverySchedulingProcess  StatusTrackingProcess |
| **Параллелизм:**  NotificationRouter  TemplateRenderer  DeliveryScheduler |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: Message Queue, SMTP, WebSocket, Database connections |

| **Наименование модуля:** Announcements Module |
| --- |
| **Назначение модуля:** Управление официальными объявлениями и новостями университета |
| **Варианты использования через состав методов модуля на псевдокоде:**  createAnnouncement(announcement) -> announcementId  publishAnnouncement(announcementId) -> boolean  getActiveAnnouncements(audience) -> announcement[]  archiveAnnouncement(announcementId) -> boolean |
| **Структуры и типы данных модуля:**  Announcement {id, title, content, author\_id, audience, publish\_date, expiry\_date, priority}  AnnouncementAudience {announcement\_id, target\_type, target\_id, read\_status}  DeliveryStatus {announcement\_id, delivery\_channels, status, metrics} |
| **Декомпозиция:** является подмодулем Administrative Service  связан с Notification Engine Module |
| **Класс или обобщение:**  является подклассом BaseAnnouncementService |

| **Компонент:** AnnouncementManager **Соединитель:** REST API, Database connections, Message Queue |
| --- |
| **Назначение компонента:** Система публикации и распространения официальных объявлений |
| **Совместно используемые данные:**  AnnouncementRegistry {announcements, templates, categories, targeting\_rules}  AudienceManagement {target\_groups, segmentation\_rules, delivery\_preferences}  Analytics {engagement\_metrics, read\_rates, effectiveness\_analysis} |
| **Процесс:**  AnnouncementCreationProcess  TargetingProcess  PublicationProcess  AnalyticsProcess |
| **Параллелизм:**  AnnouncementCreator  TargetingEngine  Publisher  AnalyticsEngine |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, Database connections, Message Queue |

**Reporting service - сервис для формирование отчетов, расчет и мониторинг KPI**

| **Наименование модуля:** Data Visualization Module |
| --- |
| **Назначение модуля:** Визуализация аналитических данных и создание интерактивных дашбордов |
| **Варианты использования через состав методов модуля на псевдокоде:**  createDashboard(dashboardConfig) -> dashboardId  addWidgetToDashboard(dashboardId, widgetConfig) -> widgetId  refreshDashboardData(dashboardId) -> boolean  exportVisualization(data, format) -> exportFile |
| **Структуры и типы данных модуля:**  Dashboard {id, name, layout, owner\_id, shared, refresh\_interval}  Widget {id, type, data\_source, configuration, position, size}  Visualization {type, data, options, interactivity, accessibility} |
| **Декомпозиция:** является подмодулем Reporting Service  связан с KPIModule  связан со всеми микросервисами |
| **Класс или обобщение:**  является подклассом BaseVisualizationService |

| **Наименование модуля:** KPIModule |
| --- |
| **Назначение модуля:** Расчет и мониторинг ключевых показателей эффективности |
| **Варианты использования через состав методов модуля на псевдокоде:**  calculateKPI(kpiCode, period) -> kpiValue  getKPITrend(kpiCode, dateRange) -> trendData  setKPITarget(kpiCode, target) -> boolean  generateKPIReport(period) -> report |
| **Структуры и типы данных модуля:**  KPI {code, name, formula, target\_value, unit, frequency}  KPIValue {kpi\_code, period, actual\_value, target\_value, variance}  KPITrend {kpi\_code, data\_points, trend\_direction, confidence} |
| **Декомпозиция:** является подмодулем Reporting Service  связан с Data Visualization Module  связан со всеми микросервисами |
| **Класс или обобщение:**  - |

| **Компонент:** VisualizationEngine **Соединитель:** Database connections, File System, REST API |
| --- |
| **Назначение компонента:** Движок для создания и рендеринга визуализаций данных |
| **Совместно используемые данные:**  VisualizationLibrary {chart\_types, color\_palettes, templates}  DashboardTemplates {predefined\_layouts, widget\_library, style\_guides}  RenderingCache {precomputed\_visualizations, performance\_optimizations} |
| **Процесс:**  DashboardCreationProcess  WidgetRenderingProcess  DataRefreshProcess  ExportGenerationProcess |
| **Параллелизм:**  DashboardRenderer  WidgetProcessor  DataRefresher  Exporter |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: REST API, Database connections, File System |

| **Компонент:** KPICalculator **Соединитель:** Database connections, Message Queue, REST API |
| --- |
| **Назначение компонента:** Движок для расчета и анализа ключевых показателей |
| **Совместно используемые данные:**  KPIFormulas {formulas[], data\_sources, calculation\_rules}  BenchmarkData {industry\_benchmarks, historical\_averages, targets}  AnalysisConfig {trend\_periods, statistical\_methods, alert\_thresholds} |
| **Процесс:**  KPICalculationProcess  TrendAnalysisProcess  TargetMonitoringProcess  AlertGenerationProcess |
| **Параллелизм:**  KPICalculator  TrendAnalyzer  AlertGenerator |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: Message Queue, SMTP, WebSocket, Database connections |

**Integration Service - сервис для безопасного обмена данными между системой университета и внешними системами.**

| **Наименование модуля:** APIManagement Module |
| --- |
| **Назначение модуля:** Управление внешними API и интеграционными точками |
| **Варианты использования через состав методов модуля на псевдокоде:**  exposeAPI(endpoint, specification) -> apiId  manageAPIAccess(apiId, policies) -> boolean  monitorAPIUsage(apiId, period) -> usageStats  transformData(sourceFormat, targetFormat) -> transformedData |
| **Структуры и типы данных модуля:**  APIEndpoint {id, path, method, handler, rate\_limit, authentication}  APIPolicy {api\_id, access\_rules, throttling, quotas}  DataMapping {source\_field, target\_field, transformation} |
| **Декомпозиция:** является подмодулем Integration Service  связан с Внешними системами  связан с System Core |
| **Класс или обобщение:**  - |

| **Компонент:** KPICalculator **Соединитель:** REST API, gRPC, Message Queue, Database connections |
| --- |
| **Назначение компонента:** Управление внешними API и трансформацией данных |
| **Совместно используемые данные:**  APIRegistry {endpoints[], versions, documentation}  IntegrationConfig {external\_systems, authentication, endpoints}  DataTransformationRules {mappings[], validation\_rules, formats} |
| **Процесс:**  APIExposureProcess  AccessControlProcess  DataTransformationProcess  MonitoringProcess |
| **Параллелизм:**  APIHandler  DataTransformer  Monitor |
| **Клиент-сервер:** Компонент: Серверный  Соединитель:REST API, gRPC, Message Queue, Database connections |

| **Наименование модуля:** Data Import/Export Module |
| --- |
| **Назначение модуля:** Обработка импорта и экспорта данных в различных форматах |
| **Варианты использования через состав методов модуля на псевдокоде:**  importData(source, format, mapping) -> importId  exportData(format, criteria) -> exportId  validateImportFile(file, schema) -> validationResult  getImportTemplates() -> template[] |
| **Структуры и типы данных модуля:**  ImportJob {id, source\_file, format, status, records\_processed, errors}  ExportJob {id, format, criteria, status, download\_url, file\_size}  DataMapping {source\_field, target\_field, transformation, validation} |
| **Декомпозиция:** является подмодулем Integration Service  связан со всеми микросервисами  связан с APIManagement Module |
| **Класс или обобщение:**  является подклассом BaseDataExchangeService |

| **Компонент:** DataExchangeEngine **Соединитель:** File System, REST API, Database connections |
| --- |
| **Назначение компонента:** Движок для преобразования и валидации данных при импорте/экспорте |
| **Совместно используемые данные:**  FormatHandlers {supported\_formats, parsers, serializers, validators}  MappingTemplates {predefined\_mappings, transformation\_rules, schema\_definitions}  JobManagement {import\_queue, export\_queue, progress\_tracking, error\_handling} |
| **Процесс:**  DataImportProcess  DataExportProcess  ValidationProcess  TransformationProcess |
| **Параллелизм:**  Importer  Exporter  Validator  Transformer |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: File System, REST API, Database connections |

| **Наименование модуля:** External Systems Adapter Module |
| --- |
| **Назначение модуля:** Адаптация данных и протоколов для интеграции с внешними системами |
| **Варианты использования через состав методов модуля на псевдокоде:**  transformData(sourceFormat, targetFormat, data) -> transformedData  validateDataCompliance(data, standard) -> complianceResult  synchronizeWithExternalSystem(systemId) -> syncResult  monitorIntegrationHealth() -> healthStatus |
| **Структуры и типы данных модуля:**  ExternalSystem {id, name, type, endpoint, authentication, protocol, version}  DataTransformation {source\_schema, target\_schema, mapping\_rules, validation}  SyncStatus {system\_id, last\_sync, status, errors, records\_processed} |
| **Декомпозиция:** является подмодулем Integration Service  связан с System Core  связан с APIManagement Module |
| **Класс или обобщение:**  является подклассом BaseAdapterService |

| **Компонент:** AdapterEngine **Соединитель:** Various protocols, REST API, Message Queue |
| --- |
| **Назначение компонента:** Адаптация данных и протоколов для внешних систем интеграции |
| **Совместно используемые данные:**  AdapterRegistry {adapters[], protocols, data\_formats, compatibility\_matrix}  ComplianceRules {regulatory\_standards, data\_protection\_rules, format\_requirements}  HealthMonitoring {system\_status, performance\_metrics, alert\_thresholds} |
| **Процесс:**  DataTransformationProcess  ProtocolAdaptationProcess  SynchronizationProcess  HealthMonitoringProcesss |
| **Параллелизм:**  DataTransformer  ProtocolAdapter  Synchronizer  HealthMonitor |
| **Клиент-сервер:** Компонент: Серверный  Соединитель: Various protocols, REST API, Message Queue |

**Audit Service - сервис для просмотра аудита действий, изменений**

| **Наименование модуля:** Log Collection Module |
| --- |
| **Назначение модуля:** Сбор, агрегация и хранение логов со всех микросервисов системы |
| **Варианты использования через состав методов модуля на псевдокоде:**  collectLogs(source, filter) -> logBatch  aggregateLogs(timeRange) -> aggregatedData  searchLogs(query) -> logEntries  archiveOldLogs(retentionPeriod) -> archiveStatus |
| **Структуры и типы данных модуля:**  LogEntry {timestamp, level, service, message, user\_id, ip, correlation\_id}  LogFilter {time\_range, level, service, user\_id, keyword, correlation\_id}  LogArchive {archive\_id, period, size, compression\_ratio} |
| **Декомпозиция:** является подмодулем Audit Service  связан с Event Tracking Module  связан со всеми микросервисами через Message Broker |
| **Класс или обобщение:**  является подклассом BaseLoggingService |

| **Компонент:** LogAggregator **Соединитель:** Message Queue, Database connections, File System |
| --- |
| **Назначение компонента:** Централизованный сбор и обработка логов из распределенной системы |
| **Совместно используемые данные:**  LogStorage {log\_entries, indexes, retention\_policies}  AggregationConfig {time\_windows, aggregation\_functions, grouping\_fields}  SearchIndex {full\_text\_index, field\_indexes, performance\_metrics} |
| **Процесс:**  LogCollectionProcess  LogAggregationProcess  SearchIndexingProcess  ArchiveManagementProcess |
| **Параллелизм:**  LogCollectorThreads  AggregationWorker  IndexBuilder  ArchiveManager |
| **Клиент-сервер:** Компонент: Серверный  Соединитель:REST API, gRPC, Message Queue, Database connections |

| **Наименование модуля:** Event Tracking Module |
| --- |
| **Назначение модуля:** Отслеживание и анализ пользовательских действий и системных событий |
| **Варианты использования через состав методов модуля на псевдокоде:**  trackUserAction(userId, action, resource) -> eventId  getUserActivity(userId, period) -> activity[]  analyzeBehaviorPatterns(criteria) -> patterns  generateAuditTrail(entityType, entityId) -> auditTrail |
| **Структуры и типы данных модуля:**  UserEvent {id, user\_id, action, resource, timestamp, details, outcome}  ActivitySession {user\_id, start\_time, end\_time, actions\_count, session\_data}  AuditTrail {entity\_type, entity\_id, events[], changes[], timeline} |
| **Декомпозиция:** является подмодулем Audit Service  связан с Security Monitoring Module  связан со всеми микросервисами |
| **Класс или обобщение:**  является подклассом BaseEventService |

| **Компонент:** EventTracker **Соединитель:** Message Queue, Database connections, REST API |
| --- |
| **Назначение компонента:** Система отслеживания и корреляции событий для аудита и анализа |
| **Совместно используемые данные:**  EventRegistry {event\_types, schemas, retention\_periods}  BehaviorPatterns {patterns[], detection\_rules, anomaly\_thresholds}  CorrelationEngine {correlation\_rules, timeline\_builder, relationship\_mapper} |
| **Процесс:**  EventCaptureProcess  ActivityCorrelationProcess  PatternAnalysisProcess  AuditTrailGenerationProcess |
| **Параллелизм:**  EventCaptureThread  CorrelationEngine  PatternAnalyzer  TrailGenerator |
| **Клиент-сервер:** Компонент: Серверный  Соединитель:Message Queue, Database connections, REST API |

| **Наименование модуля:** Security Monitoring Module |
| --- |
| **Назначение модуля:** Мониторинг безопасности системы и обнаружение подозрительной активности |
| **Варианты использования через состав методов модуля на псевдокоде:**  detectSuspiciousActivity(patterns) -> alert[]  investigateSecurityIncident(incidentId) -> investigation  generateSecurityReport(period) -> report  monitorCompliance(standards) -> complianceStatus |
| **Структуры и типы данных модуля:**  SecurityAlert {id, type, severity, description, timestamp, status, resolution}  Incident {id, type, severity, start\_time, end\_time, resolution, lessons\_learned}  ThreatPattern {pattern\_id, description, detection\_rules, risk\_level, mitigation} |
| **Декомпозиция:** является подмодулем Audit Service  связан с Event Tracking Module  связан с Auth Service |
| **Класс или обобщение:**  является подклассом BaseSecurityService |

| **Компонент:** SecurityMonitor **Соединитель:** Message Queue, Database connections, REST API |
| --- |
| **Назначение компонента:** Система мониторинга безопасности и реагирования на инциденты |
| **Совместно используемые данные:**  SecurityRules {detection\_rules, alert\_thresholds, response\_playbooks}  IncidentRegistry {active\_incidents, historical\_incidents, statistics}  ComplianceFramework {standards, requirements, evidence\_requirements} |
| **Процесс:**  ThreatDetectionProcess  IncidentResponseProcess  ComplianceMonitoringProcess  SecurityReportingProcess |
| **Параллелизм:**  ThreatDetector  IncidentResponder  ComplianceChecker  ReportGenerator |
| **Клиент-сервер:** Компонент: Серверный  Соединитель:Message Queue, Database connections, REST API |

**Employment Service - Сервис для управления сотрудниками университета**

| **Наименование модуля:** Employee Module |
| --- |
| **Назначение модуля:** Управление данными сотрудников университета и их трудоустройством |
| **Варианты использования через состав методов модуля на псевдокоде:**  hireEmployee(employeeData) -> employeeId  updateEmployeeInfo(employeeId, updates) -> boolean  terminateEmployment(employeeId, reason) -> terminationRecord  searchEmployees(criteria) -> employee[] |
| **Структуры и типы данных модуля:**  Employee {id, personal\_info, position, department, employment\_info, status}  EmploymentHistory {employee\_id, positions[], departments[], date\_changes}  TerminationRecord {employee\_id, termination\_date, reason, exit\_interview} |
| **Декомпозиция:** является подмодулем Emplyment sevirce  связан с Workload Module  связан с System Core |
| **Класс или обобщение:**  является подклассом BaseEmployeeService |

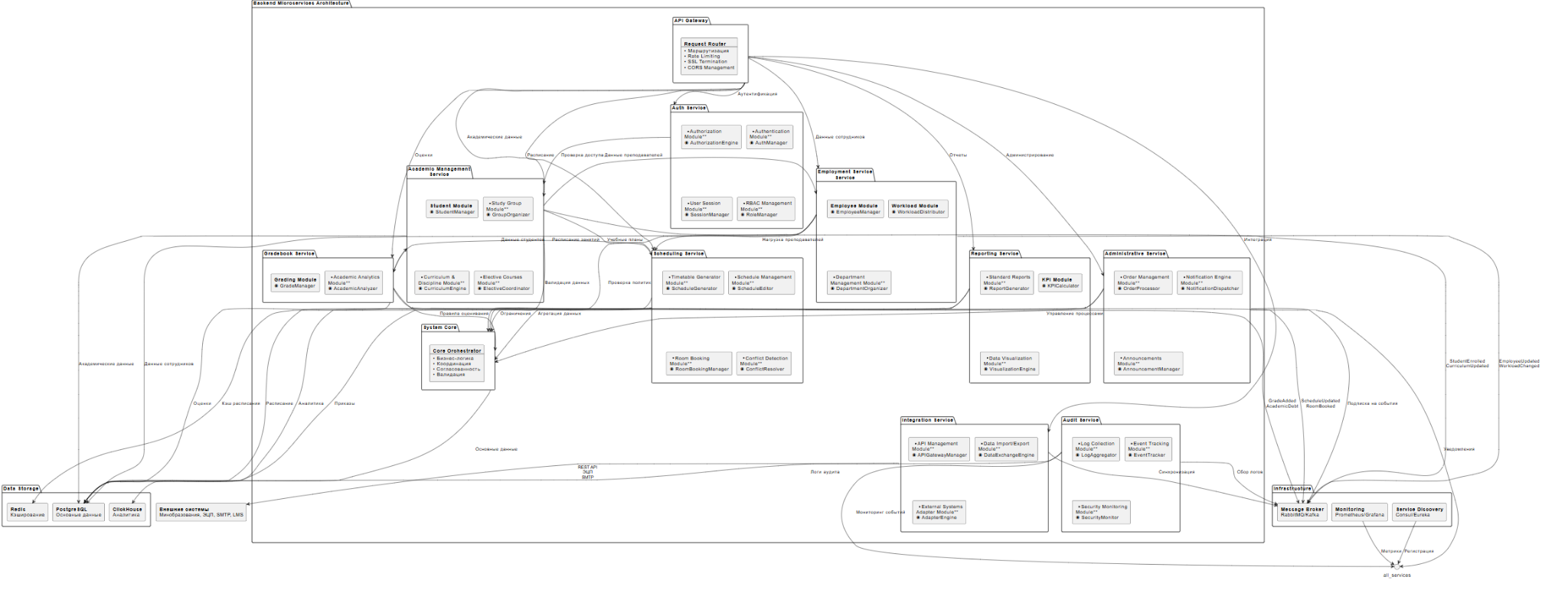
| **Компонент:** EmployeeManager **Соединитель:** Message Queue, Database connections, REST API |
| --- |
| **Назначение компонента:** Управление жизненным циклом сотрудников в университете |
| **Совместно используемые данные:**  EmployeeRegistry {employees[], status\_changes, employment\_history}  HRPolicies {hiring\_policies, termination\_policies, promotion\_policies}  DepartmentStructure {departments, reporting\_lines, organizational\_chart} |
| **Процесс:**  EmployeeOnboardingProcess  InformationUpdateProcess  EmploymentTerminationProcess  OrganizationalManagementProcess |
| **Параллелизм:**  OnboardingProcessor  DataUpdater  TerminationHandler  OrgManager |
| **Клиент-сервер:** Компонент: Серверный  Соединитель:Message Queue, Database connections, REST API |

| **Наименование модуля:** Workload Module |
| --- |
| **Назначение модуля:** Управление учебной нагрузкой преподавателей и распределение обязанностей |
| **Варианты использования через состав методов модуля на псевдокоде:**  assignTeachingLoad(teacherId, load) -> assignmentId  calculateWorkload(employeeId, period) -> workloadSummary  distributeWorkload(departmentId, semester) -> distributionPlan  adjustWorkload(assignmentId, newLoad) -> adjustmentResult |
| **Структуры и типы данных модуля:**  WorkloadAssignment {id, employee\_id, discipline\_id, hours, semester, type}  WorkloadSummary {employee\_id, period, total\_hours, breakdown, capacity}  DistributionPlan {department\_id, semester, assignments[], balance\_metrics} |
| **Декомпозиция:** является подмодулем Emplyment sevirce  связан с Employee Module  связан с Scheduling Service |
| **Класс или обобщение:**  является подклассом BaseWorkloadService |

| **Компонент:** WorkloadDistributor **Соединитель:** Message Queue, Database connections, REST API |
| --- |
| **Назначение компонента:** Система распределения и оптимизации учебной нагрузки |
| **Совместно используемые данные:**  WorkloadRules {max\_hours, min\_hours, preferred\_distribution, constraints}  CapacityPlanning {employee\_capacity, department\_needs, semester\_plan}  BalanceMetrics {fairness\_index, utilization\_rate, satisfaction\_scores} |
| **Процесс:**  WorkloadAssignmentProcess  CapacityCalculationProcess  DistributionOptimizationProcess  BalanceMonitoringProcess |
| **Параллелизм:**  AssignmentProcessor  CapacityCalculator  OptimizationEngine  Monitor |
| **Клиент-сервер:** Компонент: Серверный  Соединитель:Message Queue, Database connections, REST API |

| **Наименование модуля:** Department Management Module |
| --- |
| **Назначение модуля:** Управление организационной структурой университета и кафедрами |
| **Варианты использования через состав методов модуля на псевдокоде:**  createDepartment(departmentData) -> departmentId  updateDepartmentStructure(changes) -> reorganizationId  assignDepartmentHead(departmentId, employeeId) -> assignmentRecord  getDepartmentHierarchy() -> organizationalChart |
| **Структуры и типы данных модуля:**  Department {id, name, code, parent\_id, head\_id, description}  OrganizationalChange {change\_id, type, effective\_date, changes, approval}  DepartmentStats {department\_id, employee\_count, budget, performance} |
| **Декомпозиция:** является подмодулем Emplyment sevirce  связан с Employee Module  связан с System Core |
| **Класс или обобщение:**  является подклассом BaseDepartmentService |

| **Компонент:** DepartmentOrganizer **Соединитель:** Message Queue, Database connections, REST API |
| --- |
| **Назначение компонента:** Управление организационной структурой и иерархией подразделений |
| **Совместно используемые данные:**  OrgStructure {departments, reporting\_hierarchy, matrix\_structure}  ReorganizationHistory {changes[], approvals, impact\_assessment}  BudgetAllocation {department\_budgets, funding\_sources, expenditure\_tracking} |
| **Процесс:**  DepartmentCreationProcess  StructureUpdateProcess  HeadAssignmentProcess  OrganizationalAnalysisProcess |
| **Параллелизм:**  DepartmentManager  StructureUpdater  AssignmentProcessor  AnalyticsEngine |
| **Клиент-сервер:** Компонент: Серверный  Соединитель:Message Queue, Database connections, REST API |



4.1.2 Диаграмма бэкэнд системной архитектуры

**4.1.1.1 Детальное описание архитектуры бэкэнда**

**Бизнес-сервисы (Business Services)**

**Academic Management Service**

Student Module - управление студентами

Study Group Module - управление учебными группами

Curriculum & Discipline Module - учебные планы и дисциплины

Elective Courses Module - элективные курсы и запись на них

**Employment Service**

Employee Module - управление сотрудниками

Workload Module - учебная нагрузка преподавателей

Department Management Module - организационная структура

**Scheduling Service**

Timetable Generator Module - автоматическое составление расписания

Schedule Management Module - ручное управление расписанием

Room Booking Module - бронирование аудиторий

Conflict Detection Module - обнаружение конфликтов

**Gradebook Service**

Grading Module - ввод и управление оценками

Academic Analytics Module - анализ успеваемости

**Administrative Service**

Order Management Module - управление приказами

Notification Engine Module - система уведомлений

Announcements Module - официальные объявления

**Инфраструктурные сервисы (Infrastucture Services)**

**Auth Service**

Authentication Module - аутентификация пользователей

Authorization Module - проверка прав доступа

User Session Module - управление сессиями

RBAC Management Module - ролевая модель

**Reporting Service**

Standard Reports Module - стандартная отчетность

KPI Module - ключевые показатели

Data Visualization Module - визуализация данных

**Integration Service**

API Management Module - управление внешними API

Data Import/Export Module - импорт/экспорт данных

External Systems Adapter Module - адаптеры внешних систем

**Audit Service**

Log Collection Module - сбор и агрегация логов

Event Tracking Module - отслеживание событий

Security Monitoring Module - мониторинг безопасности

**Системное ядро (System Core):**

**Core Orchestrator - централизованная координация бизнес-процессов:**

Валидация данных между сервисами

Координация распределенных транзакций

Обеспечение согласованности данных

Применение сквозных бизнес-правил

**4.3 Заключение и выводы**

На этапе проектирования архитектуры программного обеспечения автоматизированной информационной системы университета были получены следующие ключевые результаты и выводы:

**Определена целевая гибридная архитектура.** В качестве основы была выбрана комбинированная архитектура, интегрирующая принципы микросервисной (MSA) и сервисно-ориентированной (SOA) архитектур. Данный подход позволяет достичь высокой степени модульности, масштабируемости и отказоустойчивости, что является необходимым условием для работы системы с большим количеством пользователей и сложными бизнес-процессами.

**Спроектирована трехуровневая структура системы**. Архитектура четко разделена на уровень представления (веб-портал), уровень бизнес-логики (микросервисы и системное ядро) и уровень данных (распределенные хранилища). Такое разделение обеспечивает слабую связанность компонентов, упрощает разработку, тестирование и развертывание системы.

**Детализирован состав и взаимодействие структурных элементов**. Были выделены и описаны ключевые бизнес-сервисы (Academic Management, Scheduling, Gradebook, Administrative, Employment), каждый из которых отвечает за определенную предметную область и покрывает соответствующие функциональные требования (FR-001 – FR-041). Инфраструктурные сервисы (Auth, Reporting, Integration, Audit) обеспечивают сквозную функциональность, такую как безопасность, аналитика и интеграция.

**Разработана модульная структура сервисов**. Каждый микросервис декомпозирован на модули с четко определенными обязанностями, интерфейсами (REST API, gRPC, Message Queue) и структурами данных. Это позволяет командам разработки работать параллельно над различными модулями, соблюдая согласованность системы в целом.

**Предусмотрены механизмы обеспечения надежности и производительности.** Архитектура включает такие компоненты, как API Gateway для маршрутизации и кэширования, система мониторинга на базе Prometheus/Grafana, использование Redis для кэширования и управления сессиями, а также оркестрация контейнерами через Kubernetes для обеспечения высокой доступности и автоматического масштабирования.

**Обеспечена основа для выполнения нефункциональных требований**. Выбранная архитектура и технологии позволяют достичь целевых показателей производительности (время отклика до 2-3 сек. для 95% операций), доступности (минимизация времени простоя) и безопасности (ролевая модель RBAC, шифрование данных, аудит), сформулированных в разделе 2.

**Создан фундамент для будущего развития системы.** Использование микросервисного подхода и API-ориентированной архитектуры позволяет относительно просто добавлять новые функциональные модули, интегрироваться со сторонними системами (LMS, системы Минобрнауки) и адаптировать систему к изменяющимся требованиям бизнеса.

В результате проведенной работы по проектированию архитектуры была создана комплексная, масштабируемая и технологически современная модель программного обеспечения, которая является надежной основой для последующей успешной реализации единой информационной системы университета.

1. **Выводы**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**