|  |  |
| --- | --- |
| Институт (факультет) | Институт Информационных Технологий |
| Кафедра | Математического и программного обеспечения ЭВМ |

**КУРСОВОЙ ПРОЕКТ**

|  |
| --- |
| по дисциплине Администрирование баз данных |

|  |  |
| --- | --- |
| на тему | Проектирование программного обеспечения «Стипендии ЧГУ»:  серверная часть |

|  |
| --- |
| Выполнил студент группы 1ПИб-01-2оп-21 |
| *группа* |
| направление подготовки (специальности) |
| 09.03.04., Программная инженерия |
| *шифр, наименование* |
| Ульянов Александр Сергеевич |
| фамилия, имя, отчество |

|  |  |
| --- | --- |
| Руководитель | |
| Селяничев Олег Леонидович | |
| *фамилия, имя, отчество* | |
| Доцент, кандидат технических наук | |
| *должность* | |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Оглавление

[Введение 3](#_Toc194944589)

[1. Описание предметной области 5](#_Toc194944590)

[2. Выбор жизненного цикла информационной системы 6](#_Toc194944591)

[2.1. Каскадная модель. 6](#_Toc194944592)

[2.2. Каскадная модель с возвратами. 7](#_Toc194944593)

[2.3. Спиральная модель. 8](#_Toc194944594)

[2.4. V-модель. 9](#_Toc194944595)

[2.5. Итеративная модель. 10](#_Toc194944596)

[2.6. Инкрементная модель. 11](#_Toc194944597)

[2.7. Agile. 12](#_Toc194944598)

[2.8. SCRUM. 13](#_Toc194944599)

[2.9. RAD. 15](#_Toc194944600)

[2.10. DevOps. 15](#_Toc194944601)

[2.11. Выбор методологии Agile 16](#_Toc194944602)

[3. Этапы жизненного цикла информационной системы «Стипендии ЧГУ»: серверная часть. 18](#_Toc194944603)

[3.1. Планирование 18](#_Toc194944604)

[3.2. Проектирование 19](#_Toc194944605)

[3.3. Разработка 22](#_Toc194944606)

[3.4. Тестирование 24](#_Toc194944607)

[3.5. Демонстрация 24](#_Toc194944608)

[Заключение 25](#_Toc194944609)

[Источники 26](#_Toc194944610)

[Приложение 1. Техническое задание 27](#_Toc194944611)

[Приложение 2. Руководство пользователя 37](#_Toc194944612)

[Приложение 3. Текст программы 43](#_Toc194944613)

# Введение

Современные образовательные учреждения, такие как Череповецкий государственный университет, активно внедряют цифровые технологии для автоматизации административных процессов. Одной из важных задач является управление стипендиями, которое включает расчет, распределение и учет выплат для студентов. Эффективное выполнение этой задачи требует создания специализированных программных решений, которые позволят упорядочить и упростить сложные процессы работы с данными.

Задачей данного проекта является разработка программного обеспечения «Стипендии ЧГУ». Проект включает серверную и клиентскую части, предназначенные для использования в Управлении информационных технологий (УИТ) города Череповца.

Целью данного проекта является обеспечить удобный доступ всем участникам процесса, повысить прозрачность и ускорить обработку данных и минимизировать риск ошибок.

Основные задачи разработки включают:

* Создание серверной части для безопасного хранения данных и обеспечения их обработки.
* Разработку клиентской части для удобного взаимодействия сотрудников с системой.
* Интеграцию программного обеспечения с существующими базами данных и учетными системами университета.
* Обеспечение автоматизации расчета и формирования отчетов по стипендиям.
* Настройку прав доступа в зависимости от ролей пользователей (администраторы, жюри, студенты, проверяющие).

Для реализации проекта используются современные технологии: язык программирования Python с использованием Django для серверной части, а для клиентской части используется Views они состоят их HTML, CSS, JS. В качестве системы управления базами данных планируется использовать SQLite, что обеспечит надежное и быстрое хранение данных. Важной частью разработки станет внедрение пользовательского интерфейса, который будет простым, интуитивно понятным и адаптированным для работы на различных устройствах.

Результатом работы станет веб-приложение, которое позволит сотрудникам УИТ эффективно управлять процессом начисления и выплаты стипендий, обеспечивая высокий уровень надежности, безопасности и производительности системы.

В новом подходе будет сравнимо больше плюсов, чем в текущей ситуации. Например, в базе данных специально будут предусмотрены ситуации, с которыми работники и студенты сталкивались ранее, а именно:

* сохранение данные заявлений студентов;
* изменение данные заявлений студентов;
* быстрый поиск данных заявлений студентов;
* защита от потерь файлов заявлений;
* строгая структурность данных;
* масштабирование данных [3].

# Описание предметной области

Предметная область представляет собой часть реального мира, которая изучается с целью организации управления и, в конечном счете, автоматизации. Каждый фрагмент предметной области характеризуется своим набором объектов, процессов и пользователей, которые имеют разные взгляды на эту область [2].

В сфере государственных университетов имеется огромное количество полезной информации, необходимой для пользователей. Например, сотруднику учебного заведения при распределении стипендий требуется быстро получить полную информацию о различных аспектах данной сферы деятельности, включая данные студентов, проверку документов, просмотр истории действий, управление заявлениями и данными этих заявлений, чтобы обеспечить эффективное функционирование и достижение целей.

Ранее эта информация хранилась в электронном виде в Excel-файлах и в базе данных с отсутствующим удобным интерфейсом, что, хотя и просто, но затратно по времени и могло привести к потере важного файла или к ситуации, когда нужный документ был доступен только у одного пользователя и необходимо было его вручную пересылать через сторонние сервисы, потому что он требовался множеству сотрудников.

База данных (БД) — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных [2].

СУБД — комплекс программ, позволяющих создать базу данных (БД) и манипулировать данными (вставлять, обновлять, удалять и выбирать). Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД [3].

# Выбор жизненного цикла информационной системы

При разработке информационной системы (ИС) необходимо учитывать не только технические аспекты, но и процесс её создания. От выбора подходящей модели жизненного цикла зависит успешность проекта, его стоимость, сроки реализации и качество конечного продукта.

Жизненный цикл ИС включает в себя последовательность этапов, начиная от анализа требований и заканчивая сопровождением и развитием системы. Существует множество моделей жизненного цикла, каждая из которых имеет свои особенности, преимущества и недостатки.

Для того чтобы выбрать оптимальную модель ЖЦ, необходимо проанализировать их.

## 2.1. Каскадная модель.

Каскадная модель, или Waterfall, представляет собой строго последовательный процесс разработки, в котором каждая стадия — от анализа требований до тестирования и внедрения — проходит поэтапно, не возвращаясь назад. Этот подход строится на идее полной завершённости одной фазы перед переходом к следующей, что делает её логичной и структурированной. Разработка осуществляется в рамках заранее определённых требований, а процесс документируется на всех этапах.

Сильной стороной каскадной модели является её простота и предсказуемость. Благодаря линейной структуре её легко планировать, контролировать и оценивать по срокам и бюджету. Такая модель особенно хорошо подходит для проектов, где требования известны заранее и не подлежат изменениям. Чёткая документация на каждом этапе облегчает передачу проекта между командами и помогает в обучении новых участников.

Однако основным недостатком каскадного подхода является его негибкость. Если требования изменяются уже после начала работы, это может повлечь за собой значительные переделки и затраты. Проблемы и ошибки, обнаруженные на поздних этапах, могут оказаться критичными, так как отсутствует возможность оперативного реагирования. Кроме того, пользователи видят результат только в самом конце проекта, что затрудняет учёт их пожеланий в процессе разработки. [7]

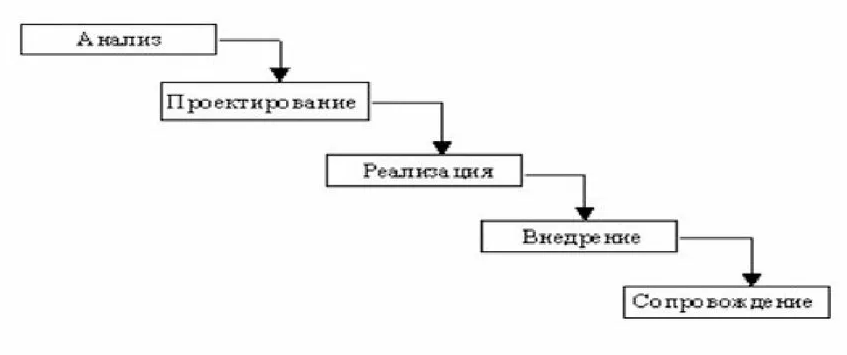


Рис.1. Каскадная модель

## 2.2. Каскадная модель с возвратами.

Эта модель является улучшенной версией классического каскада и предполагает возможность возврата к предыдущим этапам, если были допущены ошибки или изменились требования. Процесс по-прежнему остаётся поэтапным, но при необходимости команда может пересмотреть и откорректировать предыдущие решения. Это делает модель более реалистичной по сравнению с традиционным Waterfall и позволяет учитывать сложность реальных проектов.

Её преимуществом становится гибкость в рамках уже знакомой структуры. Возможность возврата позволяет оперативно устранять ошибки и реагировать на непредвиденные изменения. Команды сохраняют порядок и контроль, характерные для каскада, но при этом приобретают большую устойчивость к изменениям. Такая модель особенно актуальна в тех проектах, где нельзя на 100% предсказать всё с самого начала, но при этом нужна высокая структурированность.

Тем не менее, возвраты усложняют управление проектом. Частые пересмотры могут привести к увеличению сроков и росту бюджета. Кроме того, проект может утратить чёткость плана и контроль над прогрессом, если изменения становятся слишком частыми. Таким образом, несмотря на повышенную гибкость, каскад с возвратами всё ещё ограничен в адаптивности и может создавать сложности при работе с нестабильными требованиями. [7]

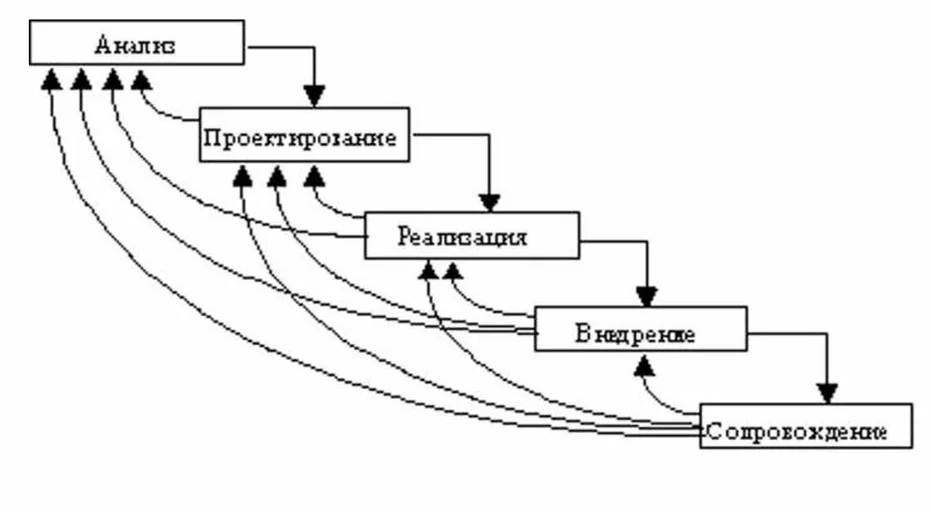


Рис.2. Каскадная модель с возвратами

## 2.3. Спиральная модель.

Спиральная модель сочетает в себе элементы каскадного подхода и итеративной разработки, организуя процесс в виде повторяющихся циклов (или витков спирали), каждый из которых включает планирование, анализ рисков, проектирование, реализацию и тестирование. Она направлена на постоянное уточнение требований и совершенствование продукта с учётом обратной связи и накопленного опыта на каждом витке.

К её основным достоинствам можно отнести высокую гибкость и ориентированность на управление рисками. Эта модель позволяет постепенно уточнять требования и уменьшать неопределённость, а также своевременно выявлять потенциальные проблемы. Благодаря этому она особенно эффективна в сложных и дорогостоящих проектах, где важно избежать ошибок на ранних стадиях. Участники проекта могут регулярно получать рабочие версии и корректировать курс по мере необходимости.

Однако у спиральной модели есть и свои слабые стороны. Она требует тщательного планирования, профессиональной оценки рисков и более высокой квалификации команды. Также процесс может быть дорогим и затяжным, особенно если циклов слишком много. В проектах с ограниченным бюджетом или сроками она может оказаться избыточной, а её сложность нередко делает внедрение затруднительным для небольших команд. [7]

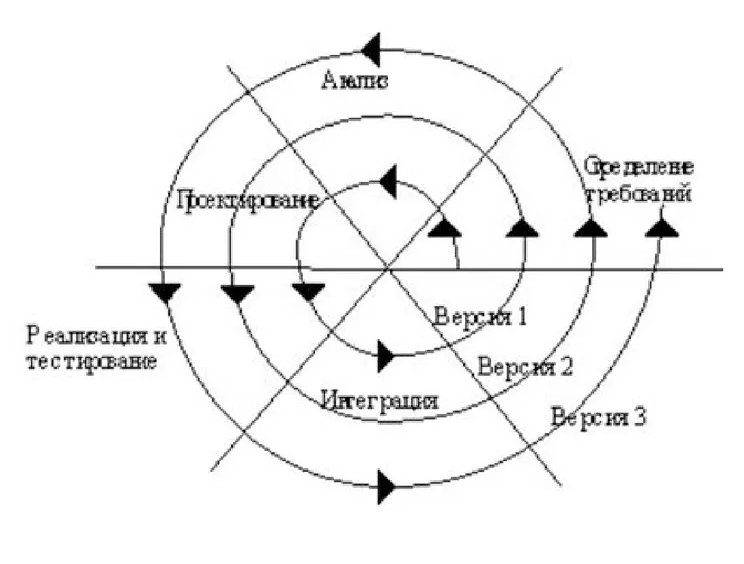


Рис.3. Спиральная модель

## 2.4. V-модель.

V-модель — это расширение каскадного подхода, в котором особое внимание уделяется параллельному планированию этапов тестирования. Каждой стадии разработки соответствует свой этап валидации и верификации, формируя структуру в виде буквы "V". Такой подход обеспечивает высокую степень контроля над качеством и надёжностью продукта.

Преимуществом этой модели является её чёткая ориентированность на проверку каждого результата разработки. Благодаря тому, что тестирование планируется заранее и идёт параллельно с реализацией, можно заблаговременно учитывать риски и повысить надёжность конечного продукта. V-модель хорошо подходит для проектов, где критично качество и стабильность, таких как банковские системы или медицинское ПО.

Однако она остаётся достаточно жёсткой и не приспособлена к частым изменениям требований. В случае пересмотра требований после начала разработки внесение изменений может оказаться затруднительным. Кроме того, при использовании этой модели пользователи также получают результат только в финале, что ограничивает их участие в процессе и делает внедрение изменений затратным. [7]

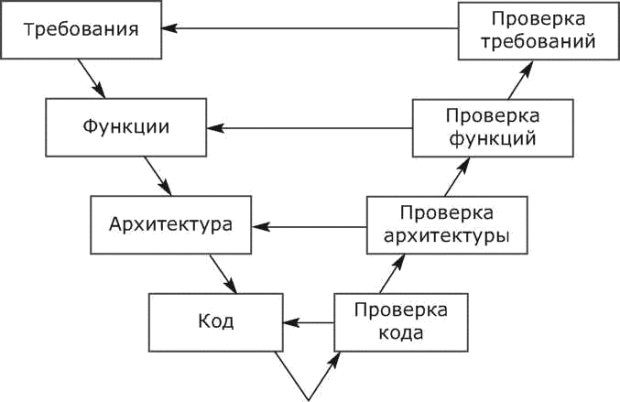


Рис.4. V-образная модель

## 2.5. Итеративная модель.

Итеративная модель основывается на повторении циклов разработки, в которых каждый следующий этап уточняет, дополняет и улучшает предыдущий. Вместо того чтобы разрабатывать всю систему сразу, команда постепенно создаёт рабочие версии продукта, улучшая их с каждой новой итерацией. Такой подход позволяет адаптироваться к изменениям и развивать проект на основе обратной связи.

Сильной стороной итеративной модели является её гибкость: проект не нужно полностью определять заранее, можно корректировать требования по мере появления новых данных. Кроме того, пользователи вовлекаются в процесс уже на ранних стадиях, получают промежуточные версии продукта и дают обратную связь. Это способствует созданию более качественного решения, максимально соответствующего потребностям заказчика.

Тем не менее, итеративный подход требует хорошего управления и постоянного взаимодействия с заказчиком. Без чёткой стратегии развития можно попасть в "ловушку бесконечной доработки", когда проект затягивается из-за постоянных изменений. Также важно учитывать, что каждая итерация требует ресурсов, а неправильная оценка сроков может привести к перерасходу бюджета. [7]

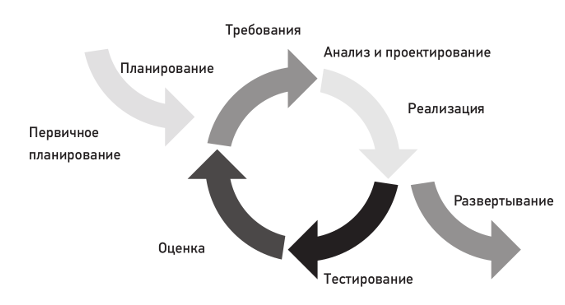


Рис.5. Итеративная модель

## 2.6. Инкрементная модель.

Инкрементная модель предполагает поэтапное наращивание функциональности системы, когда каждая новая версия добавляет к уже существующей новые возможности. При этом каждый инкремент представляет собой законченный блок, который можно протестировать и внедрить независимо. Разработка ведётся постепенно, и пользователи могут использовать часть системы ещё до завершения всего проекта.

К её достоинствам относится возможность быстрого запуска базовой версии и постепенного её расширения. Это даёт заказчику возможность начать использовать продукт и получать пользу ещё до полного завершения разработки. Также такой подход позволяет лучше управлять рисками и контролировать бюджет, ведь каждый инкремент — это самостоятельный этап с чёткими задачами.

С другой стороны, важно внимательно планировать структуру системы, чтобы каждый инкремент логично дополнял предыдущие и не возникало проблем с интеграцией. Неудачная реализация одного блока может повлиять на всю систему. Кроме того, проект может затянуться, если будет слишком много этапов, или если заказчик начнёт постоянно менять требования на основе промежуточных версий. [7]

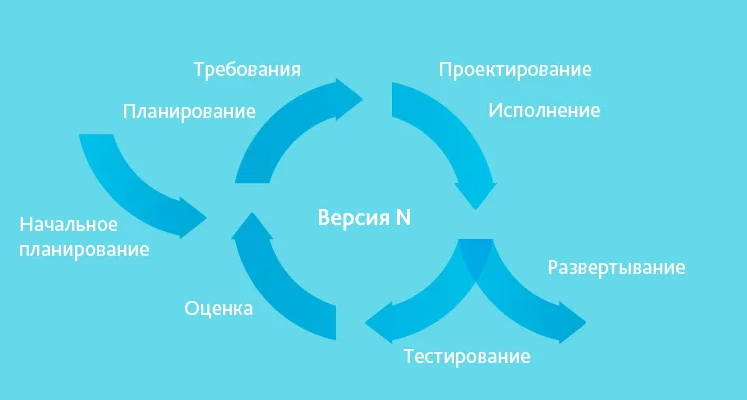


Рис.6. Инкрементальная модель

## 2.7. Agile.

Agile — это гибкая методология, в которой процесс разработки разбит на короткие циклы, направленные на быстрое создание рабочих версий продукта. Основная идея Agile — это постоянное сотрудничество с заказчиком, отзывчивость к изменениям и упор на создаваемую ценность. Команда работает небольшими итерациями, демонстрируя результат по завершении каждой из них.

Одним из главных плюсов Agile является его приспособляемость. Изменения требований, новые идеи или корректировки от заказчика могут быть быстро учтены без необходимости переписывать весь проект. Пользователь получает рабочий продукт на ранних стадиях и может вносить предложения, что повышает удовлетворённость и снижает риск ошибок. Команда при этом становится более автономной, вовлечённой и мотивированной.

Однако Agile может быть неэффективен без достаточного уровня самоорганизации и зрелости команды. Без чёткой стратегии проект может пойти по неверному пути, особенно если заказчик сам не до конца понимает, чего хочет. Также такая методология требует постоянной вовлечённости всех участников, что не всегда возможно в условиях жёстких корпоративных структур. [7]

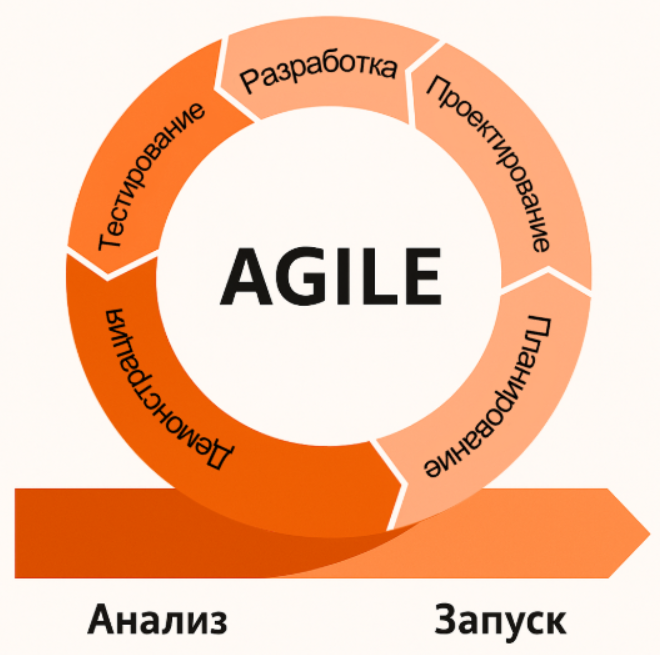


Рис.7. Методология разработки Agile

## 2.8. SCRUM.

SCRUM — это одна из самых популярных реализаций Agile, построенная на чётко структурированных ролях (Scrum Master, Product Owner и команда разработки) и повторяющихся спринтах продолжительностью 1–4 недели. В каждом спринте команда создаёт работающий фрагмент продукта, который затем оценивается и дорабатывается.

Преимуществом SCRUM является высокая управляемость и прозрачность процессов. Команда регулярно оценивает свой прогресс, устраняет препятствия и получает обратную связь. Такая структура способствует постоянному улучшению продукта и повышает вовлечённость участников. Кроме того, SCRUM помогает быстрее выявлять неэффективные практики и заменять их более результативными.

Однако SCRUM требует дисциплины и понимания принципов методологии от всех участников. При неправильной постановке задач или неэффективной роли Scrum-мастера процесс может утратить свою эффективность. Также для сложных и долгосрочных проектов SCRUM может оказаться слишком фрагментированным, особенно если задачи зависят друг от друга и требуют комплексной координации. [7]



Рис.8. Инкрементальная модель

## 2.9. RAD.

RAD — это методология, ориентированная на максимально быструю разработку программного обеспечения с помощью прототипирования, использования готовых компонентов и активного взаимодействия с пользователем. Процесс разработки делится на фазы, включающие быструю сборку, обратную связь и доработку.

Основное достоинство RAD — это высокая скорость. Продукт разрабатывается быстро, и заказчик может сразу видеть, как работает система. Такой подход особенно полезен в условиях ограниченного времени, когда важна не столько полнота, сколько результат, пригодный для практического применения. Регулярные прототипы позволяют быстро проверять идеи и снижать риски.

Однако RAD не подходит для проектов с высокой технической сложностью или необходимостью строгой архитектуры. Быстрая разработка может привести к техническому долгу, особенно если акцент делается только на скорость. Также такая модель требует постоянного участия заказчика, что не всегда возможно в условиях корпоративной среды. [7]

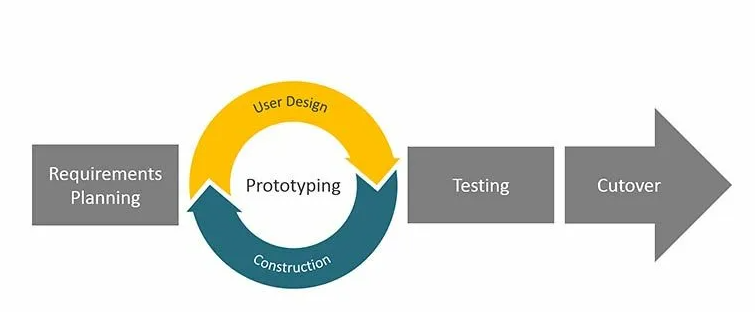


Рис.9. RAD

## 2.10. DevOps.

DevOps — это методология, направленная на объединение разработки (Development) и эксплуатации (Operations) в единый процесс. Её цель — ускорить цикл разработки, повысить стабильность, автоматизировать доставку и обеспечить непрерывное внедрение новых версий с минимальными рисками.

Преимуществом DevOps является высокая скорость и надёжность выпуска обновлений. Благодаря автоматизации тестирования, сборки, развёртывания и мониторинга достигается стабильная и масштабируемая инфраструктура. Кроме того, тесное взаимодействие команд разработки и эксплуатации сокращает количество ошибок и упрощает обслуживание.

Однако внедрение DevOps требует серьёзной перестройки процессов и культуры компании. Необходим высокий уровень автоматизации, зрелая инфраструктура и инвестиции в инструменты. Также возможны трудности с обучением персонала и сопротивлением изменениям, особенно в традиционно структурированных компаниях. Тем не менее, при правильной реализации DevOps значительно повышает эффективность всей ИТ-системы. [7]



Рис.10. Модель DevOps

## Выбор методологии Agile

Для курсового проекта была выбрана методология Agile в качестве модели жизненного цикла, так как она позволяет быстро адаптироваться к изменениям, обеспечивая гибкость и постоянное взаимодействие с заказчиком, потому что заказчик административное подразделение Управление информационных технологий Череповецкого государственного университета может изменять требования в процессе работы. Работа будет вестить итерационно, где каждая итерация длится 2–3 недели. [5]

Это особенно важно для современных IT-проектов, где требования могут изменяться в процессе разработки. Agile обеспечивает высокое качество продукта за счет непрерывной интеграции, тестирования и обратной связи. В условиях быстроменяющейся среды, Agile позволяет минимизировать риски, повышать эффективность команды и ускорять процесс разработки программного обеспечения [5].

# 3. Этапы жизненного цикла информационной системы «Стипендии ЧГУ»: серверная часть.

Выбранная методология разработки программного обеспечения (Agile) имеет пять основных этапов (см.рис.11):

1. Планирование;
2. Проектирование;
3. Разработка;
4. Тестирование;
5. Демонстрация.

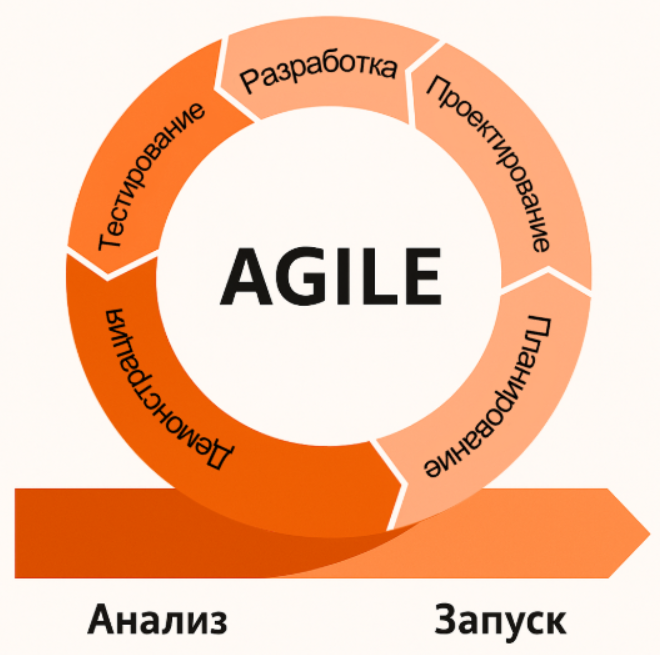


Рис. 11. Выбранная методология

## 3.1. Планирование

На данном этапе команда определяет основные потребности и ожидания заказчика с помощью бесед, интервью, опросов и других методов.

Деятельность проектировщика на данном этапе заключается в определение ключевых пользователей, в нашем случае это: студент, проверяющий, жюри, администратор. Это необходимо для дальнейшего проектирования ИС, а также анализа бизнес-процессов, таких как подачи и проверки заявлений, и формирование списка функций, которые требуется сделать, согласно техническому заданию.

Основные объекты нашей предметной области (повышенные стипендии Череповецкого государственного университета):

* Заявление (ФИО, группа, достижения, статус и др. Данные заявления);
* Пользователь (роль);
* Документы (подтверждающие файлы).

Выходной результат этапа:

* сформирован перечень функциональных требований;
* определен стек технологий: Django в качестве веб-фреймворка и SQLite как система управления базами данных.

## 3.2. Проектирование

На данном этапе команда разрабатывает архитектуру решения, создаёт макеты и прототипы, учитывая все собранные требования и ожидания от продукта.

Деятельность проектировщика заключается в проектировании архитектуры системы, анализ CASE-средств и выбор: для данного проекта была выбрана программа ERwin для моделирования базы данных, потому что она:

* позволяет создать ER-диаграмму базы данных;
* обеспечивает автоматическую генерацию SQL-кода;
* упрощает анализ зависимостей между сущностями;
* достаточно популярна и удобна в использовании.

Итоговые сущности, полученные в результате нормализации до 3 нормальной формы, которые необходимо включить при проектировании базы данных:

* Статус (STATUS) (требуется для сохранения статусов всех пользователей);
* Заявления (STATMENT) (требуется для сохранения данных о заявлениях);
* Период (PERIOD) (требуется для сохранения сроков в которые разрешено подавать заявления на повышенную стипендию);
* Пользователи (USER) (требуется для сохранения данных пользователя);
* История действий (LOG) (требуется для отслеживания ошибок);
* Роли (ROLE) (требуется для ролей пользователей).

Спроектированная структура базы данных с помощью программы ERwin (см. рис. 11).

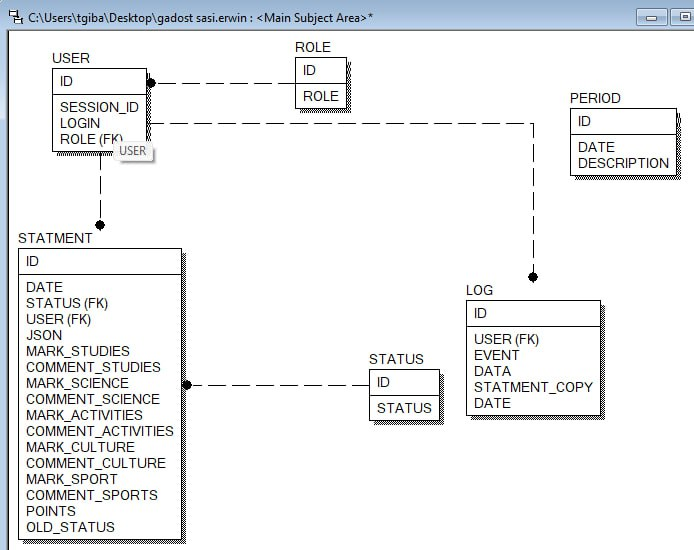


Рис. 12. Спроектированная база данных в программе ERwin

Описание полей каждой сущности приведено в таблицах 1-6.

Таблица 1

Сущность User

|  |  |  |  |
| --- | --- | --- | --- |
| ID (PK) | SESSION\_ID | LOGIN | ROLE(FK) |
| Integer | String | String | Integer |
| 1 | Asd&yg6b | Student | 1 |
| 2 | Hjgyg65gl | Admin | 2 |
| 3 | Lkijvgy65hjcg705 | VasiaRe | 3 |

Таблица 2

Сущность Statment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID(PK) | DATE | STATUS (FK) | USER (FK) | JSON | MARK\_STUDIES |
| Integer | Date | Integer | Integer | String | Integer |
| 1 | 04.05.2003 | 0 | 1 | {type:1,docum… | 3 |
| 2 | 25.08.2002 | 3 | 3 | {type:2, files[… | 5 |
| 3 | 13.04.2001 | 1 | 2 | {type:1,docum… | 7 |

Продолжение табл. 2

|  |  |  |  |
| --- | --- | --- | --- |
| COMMENT\_STUDIES | MARK\_SCIENCE | COMMENT\_ SCIENCE | MARK\_ACTIVITIES |
| String | Integer | String | Integer |
| Недостаточно данных | 1 | Всё хорошо | 32 |
| Неверные файлы | 2 | Отлично | 34 |
| Всё хорошо | 10 | Ошибка в 3 столбце | 3 |

Продолжение табл. 2

|  |  |  |  |
| --- | --- | --- | --- |
| COMMENT\_ACTIVITIES | MARK\_CULTURE | COMMENT\_CULTURE | MARK\_SPORT |
| String | Integer | String | Integer |
| Неверные файлы | 12 | Устаревшие файлы | 1 |
| Ошибочный файл | 1 | Нет комментария | 21 |
| Все нормально | 7 | Не открывается файл | 3 |

Продолжение табл. 2

|  |  |  |
| --- | --- | --- |
| COMMENT\_SPORT | POINTS | OLD\_STATUS |
| Integer | Integer | Bool |
| 1 | 49 | False |
| 2 | 43 | False |
| 3 | 30 | True |

Таблица 3

Сущность Period

|  |  |  |
| --- | --- | --- |
| ID(PK) | DATE | DESCRIPTION |
| Integer | Date | String |
| 1 | 19.01.2025 | Начало подачи... |
| 2 | 28.01.2025 | Конец подачи... |

Таблица 4

Сущность Role

|  |  |
| --- | --- |
| ID(PK) | ROLE |
| Integer | String |
| 1 | STUDENT |
| 2 | ADMIN |
| 3 | JURY |
| 0 | INSPECTOR |

Таблица 5

Сущность Status

|  |  |
| --- | --- |
| ID(PK) | STATUS |
| Integer | String |
| 0 | process |
| 1 | error |
| 2 | verified |
| 3 | conflict |
| 4 | confirm |
| 5 | deny |

Таблица 6

Сущность Log

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID(PK) | USER (FK) | EVENT | DATA | STATEMENT\_COPY | DATE |
| Integer | Integer | String | String | String | Date |
| 1 | 0 | Отправка | заявления н.. | {type:1,docum… | 04.05.2003 |
| 2 | 1 | Отклонение | статуса з… | {type:2, files[… | 04.05.2002 |
| 3 | 51 | Удаление | пользователя | {type:1,docum… | 04.05.2001 |

Выходной результат этапа:

* ER-модель спроектирована в ERwin;
* Схема базы данных.

## 3.3. Разработка

На этапе разработки команда приступает к написанию кода. Интегрирование базы данных, а также написание функциональной части, которая взаимодействует с базой данных и основного функционала программы.

Основные шаги на данном этапе:

1. Внедрение базы данных.
2. Написание моделей для взаимодействия с базой данных (см. рис.12).

class Role(models.Model):  
 name = models.TextField("Роль", default="")  
  
 @classmethod  
 def create\_default\_records(cls):  
 # Данные для записи  
 default\_records = [  
 {'id': 0, 'name': "Student"},  
 {'id': 1, 'name': "Administrator"},  
 {'id': 2, 'name': "Jury"},  
 {'id': 3, 'name': "Inspector studies"},  
 {'id': 4, 'name': "Inspector science"},  
 {'id': 5, 'name': "Inspector activities"},  
 {'id': 6, 'name': "Inspector culture"},  
 {'id': 7, 'name': "Inspector sport"},  
 ]  
 for record in default\_records:  
 cls.objects.get\_or\_create(\*\*record)

Рис.13. Пример кода модели на основе Role

1. Написание основного кода программы

Программный код, который был написан в процессе данных шагов, изображен в прил.3.

Программа разработана на языке высокого уровня Python с использованием фреймворка Django и предоставляет удобный интерфейс для студентов и администраторов. Она позволяет студентам заполнять заявления на получение повышенной стипендии, а администраторам – рассматривать, одобрять или отклонять поданные заявления. [6]

Преимущества системы:

* автоматизация процесса подачи и рассмотрения заявлений;
* удобный интерфейс для студентов и администраторов;
* гибкость и расширяемость благодаря использованию Django;
* защищенная аутентификация пользователей.

Приложение имеет клиент-серверную архитектуру, где вся логика приложения находится в модуле «App Logic», она уже в свою очередь взаимодействует с моделями из модуля «Model». Модели постоянно взаимодействуют с базой данных, выгружая, записывая, удаляя и редактируя данные в базе данных. [6]

Схема архитектуры изображена на рис. 13.

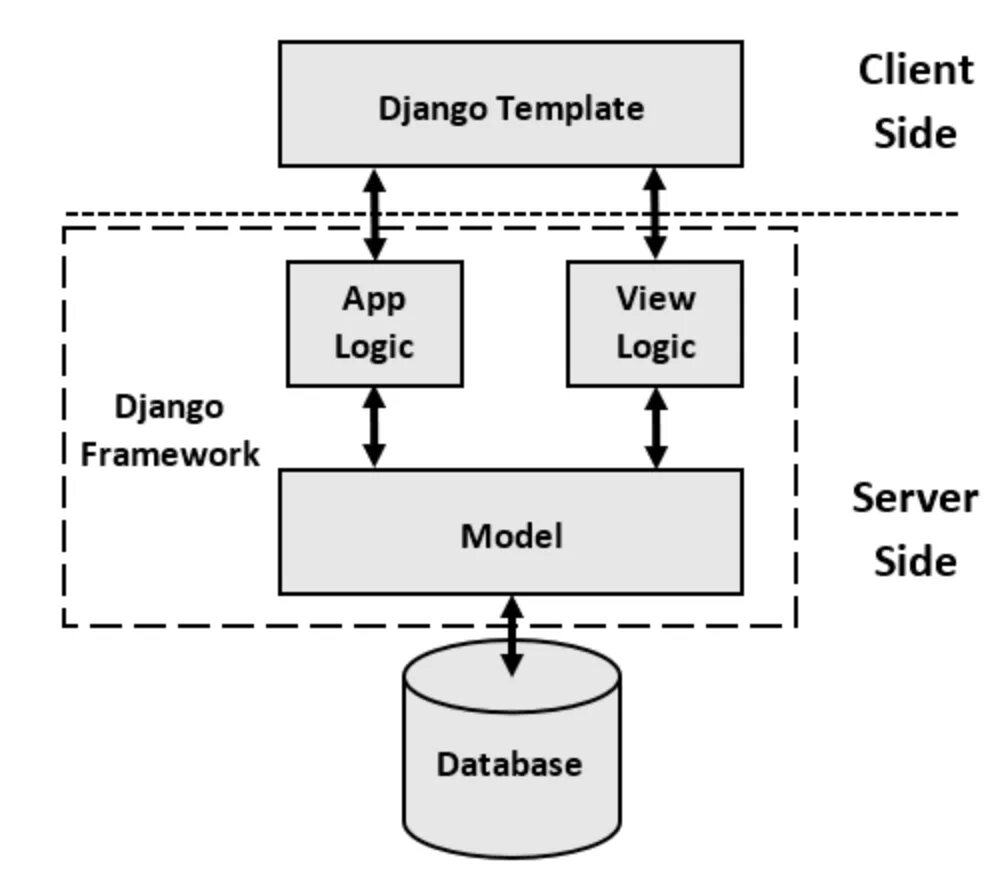


Рис.13. Архитектура Django

## 3.4. Тестирование

На данном этапе команда направляет все силы на поиск и устранение при наличии недостатков программы, сбоев при некорректных данных, ошибки.

Для успешного выполнения этого этапа команде необходимо составить тестовые наборы данных, которые загружаются в актуальную версию программы и проверяются результаты работы программы. При наличии сбоя, команда находит место возникновения сбоя и устраняет его. [5]

Ключевым методом на текущем этапе является ручное тестирование — оно применяется в первую очередь, так как система активно дорабатывается, и важна живая обратная связь. Ручное тестирование позволяет на практике проверить каждый запрос к серверу, каждое взаимодействие с базой данных глазами конечного пользователя, выявляя не только технические ошибки, но и недочеты в логике.

Особенности тестирования, проведенного нашей командой:

* Исключения и валидация: проверялась реакция системы на ошибочные или неполные данные: отсутствие обязательных полей, ввод заведомо некорректных значений, загрузка файлов недопустимых форматов. Это позволило протестировать обработку исключений и убедиться в корректной работе системы в нестандартных ситуациях.
* Структурное тестирование: осуществлялся переход по всем страницам и модулям веб-приложения — с точки зрения как студентов, так и проверяющих, жюри и администратора. Проверялась связность между модулями, соответствие URL-маршрутов, а также корректность переходов и возвратов с сохранением состояния.
* Проверка базы данных: вводимые данные отслеживались в базе — фиксировалось, сохраняются ли данные корректно, как они отображаются при повторном входе и как обрабатываются запросы со стороны сервера. Также проводилась проверка соответствия данных требованиям 3НФ, чтобы убедиться в отсутствии избыточности и дублирования.
* Связь между модулями: особое внимание уделялось взаимодействию различных компонентов, например, как информация из таблицы пользователей попадает в раздел персонала или как статус заявления влияет на его отображение в других разделах.
* Тестирование на стороне заказчика: на раннем этапе была проведена работоспособность и базовое тестирование на компьютере заказчика, что позволило получить прямую обратную связь от конечного пользователя и убедиться в корректной работе системы.

## 3.5. Демонстрация

После завершения каждой итерации (или спринта) продукт демонстрируется заказчику (подразделении Череповецкого государственного университета УИТ). Это позволяет получить обратную связь, узнать, удовлетворяет ли продукт его потребностям, и внести необходимые корректировки.

В нашем случае демонстрация велась в несколько этапов:

* Ранние версии проекта были продемонстрированы еще на начальных стадиях в онлайн режиме. Несмотря на сырую реализацию, это позволило сразу получить конструктивную критику и понять, какие элементы интерфейса и логики нужно адаптировать под реальные сценарии работы заказчика, например заказчик высказал недовольство интерфейсом и отсутствием некоторых функциональных решений, которые были прописаны в ТЗ, а также смог подсказать на счет алгоритмов распределения стипендий, который мы на месте и смогли продумать.
* Доработка и адаптация. После получения обратной связи команда оперативно дорабатывала продукт: упрощала подачу информации, корректировала визуальную структуру интерфейса и адаптировала форматы хранения данных.
* Тестирование на стороне заказчика**:** Финальная демонстрация проходила очно — непосредственно на рабочих местах заказчика.Команда выехала с презентацией в административное подразделение Управления информационных технологий, где приложение было развернуто на компьютере самого заказчика, а также на ПК сотрудников, работающих с системой заявок. В ходе демонстрации проводилось тестирование всех ключевых сценариев: заполнение заявлений, смена ролей, отслеживание логики обработки данных и взаимодействия с базой. Это позволило не только подтвердить корректность функционирования системы, но и наглядно показатьеё стабильность и готовность к реальному использованию.

Заключение

В рамках данного курсового проекта была спроектирована база данных для серверной части программного обеспечения, предназначенного для автоматизации обработки заявлений на повышенную стипендию, а также разработана часть функционала приложения для корректной работы с базой данных.

Основное внимание уделялось проектированию ключевых элементов архитектуры, выбору оптимальных методологий разработки.

На первоначальном этапе жизненного цикла Agile была проведена аналитическая работа, включающая исследование поставленных задач и подбор, наиболее подходящих для нее решений. Этот анализ позволил определить основные требования к системе: удобство использования, гибкость настройки, интеграцию с существующими системами аутентификации и поддержку различных форматов данных и подобрать соответствующий стек технологий.

Для выполнения поставленных задач были выбраны современные инструменты, а для проектирования базы данных использовался ERwin.

Созданное программное обеспечение не только решает поставленные в техническом задании задачи, но и открывает перспективы для дальнейшего развития — от расширения функционала до внедрения аналитических модулей и дополнительных сервисов поддержки принятия решений.

В целом проект можно считать успешным шагом на пути цифровизации образовательных процессов и повышения эффективности административной работы в вузе.

# Источники

1. Е.В. Ершов, Л.Н. Виноградова, В.В. Селивановских, О.Л. Селяничев Методика и организация самостоятельной работы студентов: Учеб. пособие. – ФГБОУ ВПО «Череповецкий государственный университет», 2012 г.
2. Общие требования о проектировании баз данных [Электронный ресурс] <https://cde.osu.ru/courses2/course10/theory/theme4_5.html> адрес. Дата обращения: 24.02.24;
3. Обработка информации с помощью СУБД [Электронный ресурс] <https://infopedia.su/5x19e7.html> адрес. Дата обращения: 24.02.24.
4. Основные команды работы с sql [Электронный ресурс] https://tproger.ru/translations/sql-recap адрес. Дата обращения: 24.02.24;
5. Agile: что это такое и где используется, принципы методологии [Электронный ресурс] https://practicum.yandex.ru/blog/metodology-agile адрес. Дата обращения: 24.02.25;
6. Django Documentation: Models [Электронный ресурс] https://docs.djangoproject.com/en/5.1/topics/db/models/ адрес. Дата обращения: 24.02.25;
7. Д. Б. БЕРГ Е. А. УЛЬЯНОВА П. В. ДОБРЯК МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА Учеб. Пособие. - Издательство Уральского университета, 2014 г.;

# Приложение 1. Техническое задание

МИНОБРАНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ »

Институт информационных технологий

Математическое и программное обеспечение ЭВМ

Администрирование баз данных

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ

д. т.н., профессор Ершов Е.В.

« » \_\_\_\_\_\_\_\_\_\_\_2024 г.

Разработка программного обеспечения «Стипендии ЧГУ»: серверная часть.

Техническое задание на курсовой проект

Листов 4

Руководитель: доцент Селяничев О.Л.

Исполнитель: студент гр. 1ПИб-01-2оп-21

Ульянов Александр Сергеевич

2025 г.

Введение

В рамках курсового проекта планируется разработать программное обеспечение, которое ускорит процесс подачи и обработки заявлений на повышенную стипендию. Цель проекта — ускорить выполнение задач и сократить объем ручной работы, повышая общую эффективность и автоматизируя ключевые этапы процедуры.

1. Основание для разработки

Основанием для разработки является задание на курсовой проект, выданное на кафедре МПО ЭВМ «Череповецкий Государственный Университет» по заказу УИТ ЧГУ.

Дата утверждения: 10.02.2025 года.

Наименование темы разработки: Разработка программного обеспечения «Стипендии ЧГУ»: серверная часть.

1. Назначение разработки

Разрабатываемое программное обеспечение предназначено для автоматизации процесса подачи и обработки заявлений на повышенную стипендию. Система позволит сократить время обработки заявок, минимизировать количество ошибок при ручной обработке и обеспечить упрощенное взаимодействие между студентами и администрацией учебного заведения. Автоматизация ключевых этапов подачи заявлений позволит улучшить контроль за процессом и повысить оперативность принятия решений.

1. Требования к программе
   1. Требования к функциональным характеристикам

Программное обеспечение должно выполнять следующие функции, описанные ниже.

1. Авторизация пользователя и применение роли: студент, проверяющий, жюри, администратор.
2. Предоставлять доступ для заполнения заявления, в котором необходимо заполнить данные по направлениям: общие сведение, спорт, учеба, культура и творчество, наука, общественная деятельность. Ниже приведены поля для каждого направления:

1. Общие сведения

* ФИО (студента)
* Группа
* Контактный телефон
* Электронная почта
* Направление подготовки
* Курс
* Уровень подготовки (бакалавриат/магистратура/специалитет)
* Средний балл за предыдущую сессию
* Дата подачи заявления

2. Поля для направления "Учеба":

* Название мероприятия
* Дата проведения
* Организатор
* Форма участия (очная/заочная)
* Вид мероприятия (олимпиада, конкурс и т. д.)
* Достижения (место, награда)
* Подтверждающий документ (возможность прикрепления файла)

3. Поля для направления "Наука":

* Наименование научного мероприятия
* Дата и место проведения
* Тип мероприятия (конференция, конкурс научных работ)
* Уровень (международный, всероссийский, региональный и т. д.)
* Наименование работы/проекта
* Результат участия (доклад, публикация, награда)
* Подтверждающий документ

4. Поля для направления "Общественная деятельность":

* Вид деятельности (организация мероприятий, участие в волонтерских акциях и т. д.)
* Название проекта/мероприятия
* Дата участия
* Роль в мероприятии (организатор, участник и т. д.)
* Результат (благодарственное письмо, сертификат, диплом)
* Подтверждающий документ

5. Поля для направления "Культура и творчество":

* Название творческого мероприятия
* Дата и место проведения
* Тип участия (индивидуальное/групповое)
* Достижение (награда, участие, выступление)
* Подтверждающий документ

6. Поля для направления "Спорт":

* Вид спорта;
* Название спортивного мероприятия;
* Дата и место проведения
* Достижение (место, награда, участие)
* Уровень соревнований (международный, всероссийский и т. д.)
* Подтверждающий документ.

1. Заявление должно обладать одним из статусов: на проверке, ошибочно, проверено, одобрено, отклонено, конфликтное. Предусмотреть возможность изменять его в зависимости от этапа проверки.
2. В зависимости от роли пользователя, система должна предоставлять следующие функциональные возможности:

* Студент:
  + просмотр своих ранее поданных заявлений и отслеживание их статуса;
  + заполнить/редактировать (до определенной даты) заявление на повышенную стипендию с возможностью прикрепления файлов и заполнения полей с информацией о себе и пройденных мероприятиях.
* Проверяющий:
  + поставить оценку части заявления, выбранного из списка согласно своему направлению;
  + посмотреть список заявлений на повышенную стипендию со статусом «на проверке», «проверено», «ошибочно»;
  + посмотреть часть выбранного из списка заявления, согласно своему направлению.
* Жюри:
  + одобрить/отклонить заявление выбранного из списка;
  + просмотреть список заявлений на повышенную стипендию со статусом «проверено», «одобрено», «конфликтное»;
  + просмотреть заявление, выбранное из списка.
* Администратор:
* просмотреть список заявлений;
* просмотреть список устаревших заявлений;
* администрирование студентов (просмотреть список пользователей, просмотреть студента выбранного из списка);
* администрирование жюри (Снять/назначить на должность жюри, просмотреть список жюри, просмотреть жюри выбранного из списка);
* администрирование проверяющих (Снять/назначить на должность проверяющего, просмотреть список проверяющих, просмотреть проверяющего, выбранного из списка);
* возможность пометить все заявления как устаревшие;
* просмотр истории действий, а также возможность ее очистить;
* задать дату начала и конца подачи заявления на повышенную стипендию.

1. Предусмотреть возможность разграничения заявлений текущего периода от заявлений прошедших периодов.
2. Предусмотреть возможность сохранения истории действий для того, чтобы отслеживать через нее ошибочные действия для своевременного устранения.
3. Доступ к данным студентов будет осуществляться по логину и паролю через API предоставленные заказчиком.
4. Возможность отправки сообщений на электронную почту студенту при изменении статуса заявления, которое им подано.

Серверная часть программного обеспечения должна выполнять следующие функции:

* работа с базой данных:
  + чтение данных;
  + изменение данных;
  + удаление данных;
  + сохранение после каждой проведенной операции.
* делать запросы к API с данными студентов и работников ЧГУ и обрабатывать полученные ответы;
* делать запросы к Gmail API для отправки электронных писем;
* отвечать на запросы клиентов, в зависимости от их доступа, выполняя соответствующие функции, указанные в П.1 пункт 3.1.4;
* обрабатывать исключительные ситуации;
* при наступлении заданной администратором даты окончания подачи заявлений выполнить изменение статусов не устаревших заявлений, так чтобы разделить заявления на 3 группы по статусам:
  + отклоненное;
  + одобренное;
  + конфликтное.

3.2 Входные данные

Входные данные, которые требует программа для корректной работы:

* данные для доступа к API c данными студентов и работников ЧГУ;
* данные для доступа к Gmail API (Почта, пароль, ключ);
* локальная база данных;
* запросы клиентов.

3.3 Выходные данные

* Запросы к Gmail API;
* Ответы сервера клиентам на их запросы;
* Изменение локальной базы данных;
* Файлы, прикрепленные клиентами и отправленные через запросы.

3.3 Требования к надежности

Программное обеспечение должно обеспечить сохранность информации при наступлении следующих событий:

* Неверно заполненное заявление;
* Некорректно указанные данные в заявлении;
* Некорректные запросы к серверу;
* Ошибки в сессии клиента;
* Отсутствие необходимых данных в базе данных;
* Отсутствие связи с сервисами через API;
* Аварийное завершение работы компьютера, в данных ситуациях желательно иметь бесперебойный источник питания.

Для устранения и отслеживания непредвиденных ошибок предусмотрена история действий.

3.4 Условия эксплуатации

Для оптимальной работы с приложением должны быть соблюдены следующие условия:

1. Компьютер, предназначенный для работы в закрытом отапливаемом помещении при следующих условиях окружающей среды:

* температура окружающего воздуха от +10°C до +35°C;
* относительная влажность воздуха не более 80%;
* запыленность воздуха не более 0,75 мг/м³.

1. Для работы с веб-приложением необходим веб-браузер.

3.4 Требования к составу и параметрам технических средств

Программа должна корректно работать при соблюдении минимальных системных требований:

* процессор с тактовой частотой не менее 2.5 ГГц и 4 ядрами;
* оперативная память: не менее 8 Гб;
* свободное место на жестком диске: 150 Мб и более;
* наличие устройств ввода: клавиатура, мышь.
* стабильный интернет;

3.5 Требования к информационной и программной совместимости

* Система должна быть совместима с Windows 10;
* На системе должен быть установлен Python версии 3.12;
* У системы должен быть доступ к Gmail API;
* У системы должен быть доступ к API c данными студентов ЧГУ;
* Клиенты должны использовать браузеры Google Chrome, Opera и Mozilla FireFox.

4. Требование к программной документации

* 1. Содержание расчётно-пояснительной записки

Программная документация должна содержать расчётно-пояснительную записку и приложения, такие как:

* Техническое задание;
* Текст программы;
* Руководство пользователя.

4.2. Требования к оформлению

Требования к оформлению, установленные ЕСПД ГОСТ и пособии по самостоятельной работе, должны быть выполнены на протяжении всей работы без каких-либо изменений (табл. П1.2).

Таблица П1.2.

Требования к оформлению

|  |  |
| --- | --- |
| Документ | Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б. Файлы предъявляются на компакт-диске: РПЗ с ТЗ; программный код. Листы и диск в конверте вложены в пластиковую папку скоросшивателя. |
| Страницы | Ориентация – книжная; отдельные страницы, при необходимости, альбомная. Поля: верхнее, нижнее – по 2 см, левое – 3 см, правое – 1 см. |
| Абзацы | Межстрочный интервал – 1,5, перед и после абзаца – 0. |
| Шрифты | Кегль – 14. В таблицах шрифт 11. Шрифт листинга – 8 (возможно в 2 колонки). |
| Рисунки | Подписывается под ним по центру: Рис.Х. Название В приложениях: Рис.П1.3. Название |
| Таблицы | Подписывается: над таблицей, выравнивание по правому: «Таблица Х».  В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) – по центру. В теле таблицы (записи) текстовые значения – выровнены по левому краю, числа, даты – по-правому. |

5. Стадии и этапы разработки

В данном пункте продемонстрированы стадии и этапы разработки, которые должны быть пройдены во время работы с программой (табл. П1.3).

Таблица П1.3.

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки | Сроки разработки | Результаты  выполнения | Отметки о  выполнении |
| Разработка ТЗ | 01.02.2025 | Оформленное ТЗ |  |
| Анализ и сбор требований | 05.02.2025 | Проведен анализ и сбор требований |  |
| Проектирование системы | 05.02.2025 | Спроектирована структура системы |  |
| Программирование системы | 10.02.2025 | Создана программа |  |
| Тестирование системы | 15.02.2025 | Протестирована система |  |
| Написание РПЗ | 30.02.2025 | Оформлена РПЗ |  |

6. Порядок контроля и приемки

Порядок контроля и приемки описан в табл. П1.4.

Таблица П1.4.

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения выпускной квалификационной работы | Сроки разработки | Результаты  выполнения | Отметка о приемке результата контрольного этапа |
| Сдача технического задания | 12.02.2025 | Документ «Техническое задание» проверен |  |
| Демонстрация работы программы | 20.02.2025 | Работа программы проверена |  |
| Поиск ошибок в программе | 23.02.2025 | Все найденные ошибки исправлены |  |
| Защита курсового проекта | 31.02.2025 | Курсовой проект защищен |  |

# Приложение 2. Руководство пользователя

1.Инструкция по работе

Для запуска программы необходимо запустить файл manage.py консольной командой python manage.py -runserver (рис.П2.1).

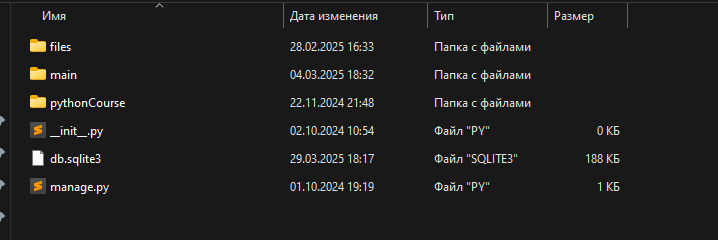


Рис.П2.1. Файлы программы

При запуске программы нас встречает окно открытия базы данных (рис.П2.2).

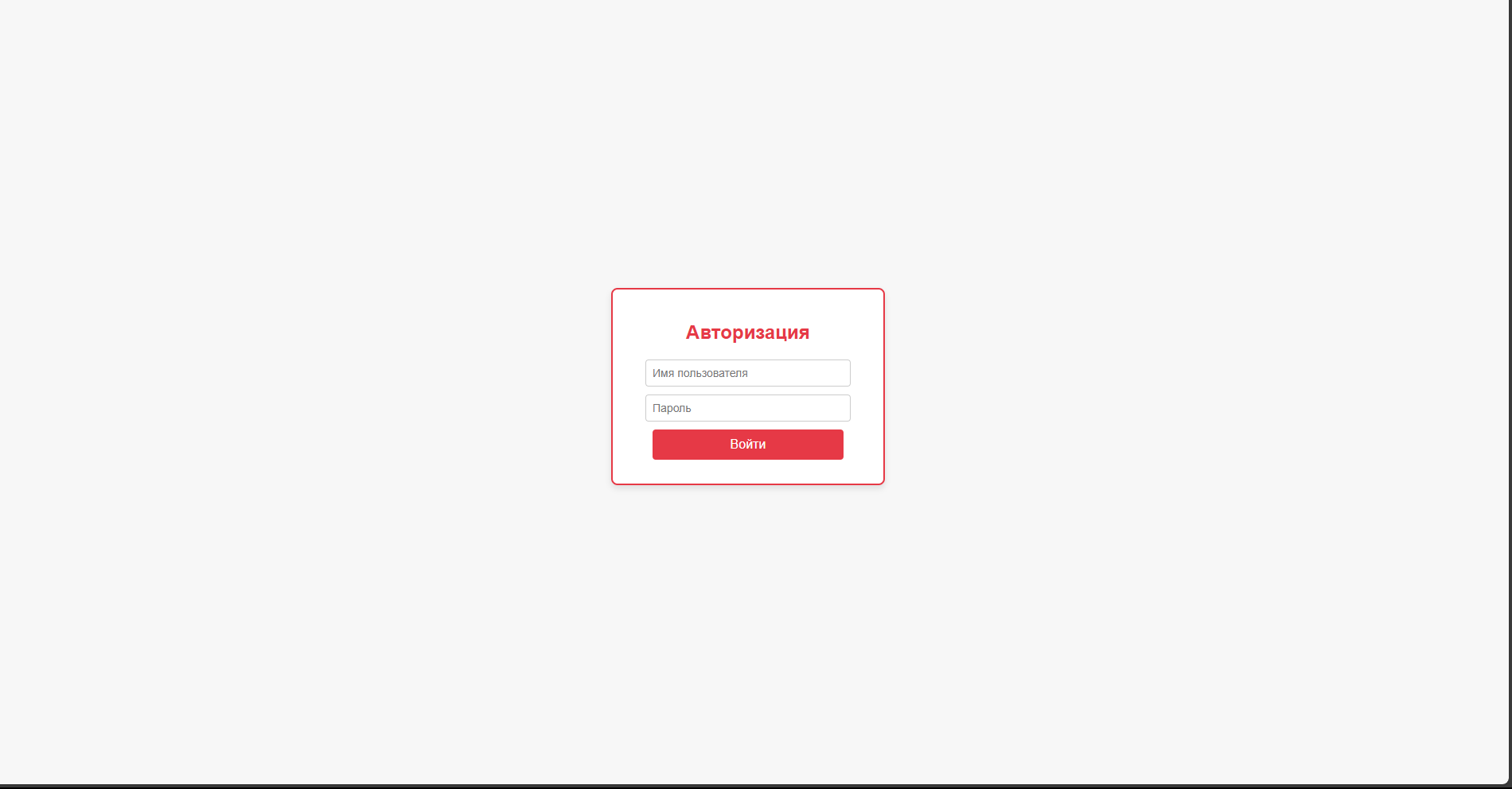


Рис.П2.2. Окно открытия

Рассмотрим возможности роли студента, для этого авторизуемся и перед нами откроется панель управления с возможностями «Заполнить заявление» и «Мои заявления» (рис.П2.3).

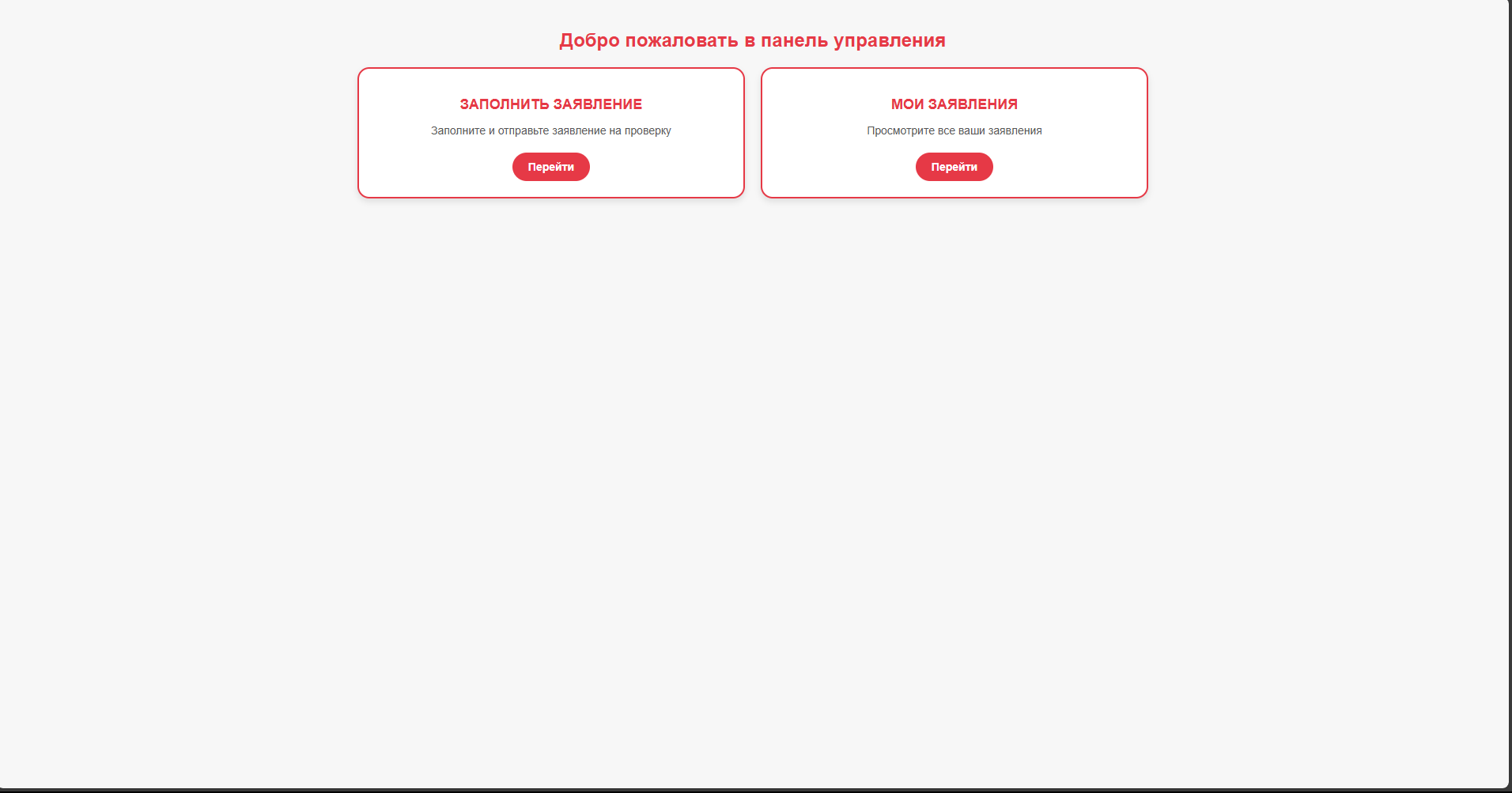


Рис.П2.3. Страница «Панель управления» для студента

При нажатии на кнопку «Перейти» в окне «Заполнить заявление» нас перекинет на соответствующую страницу с полями для заполнения (рис.П2.4).

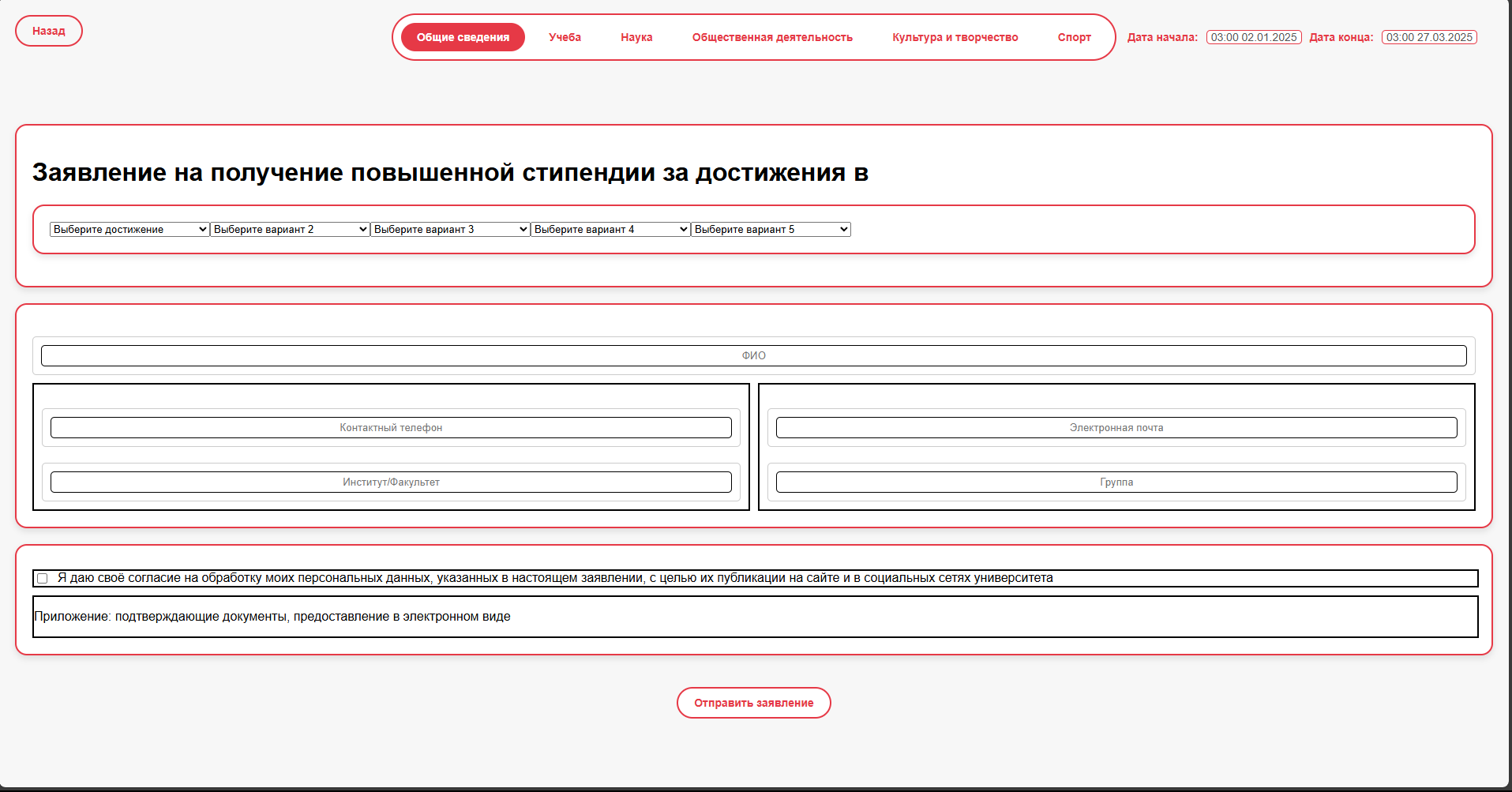


Рис.П2.4. Страница с заполнением заявления

Кнопки, которые находятся сверху позволяют переключаться между разделами заявления (рис.П2.5 и рис.П2.6).

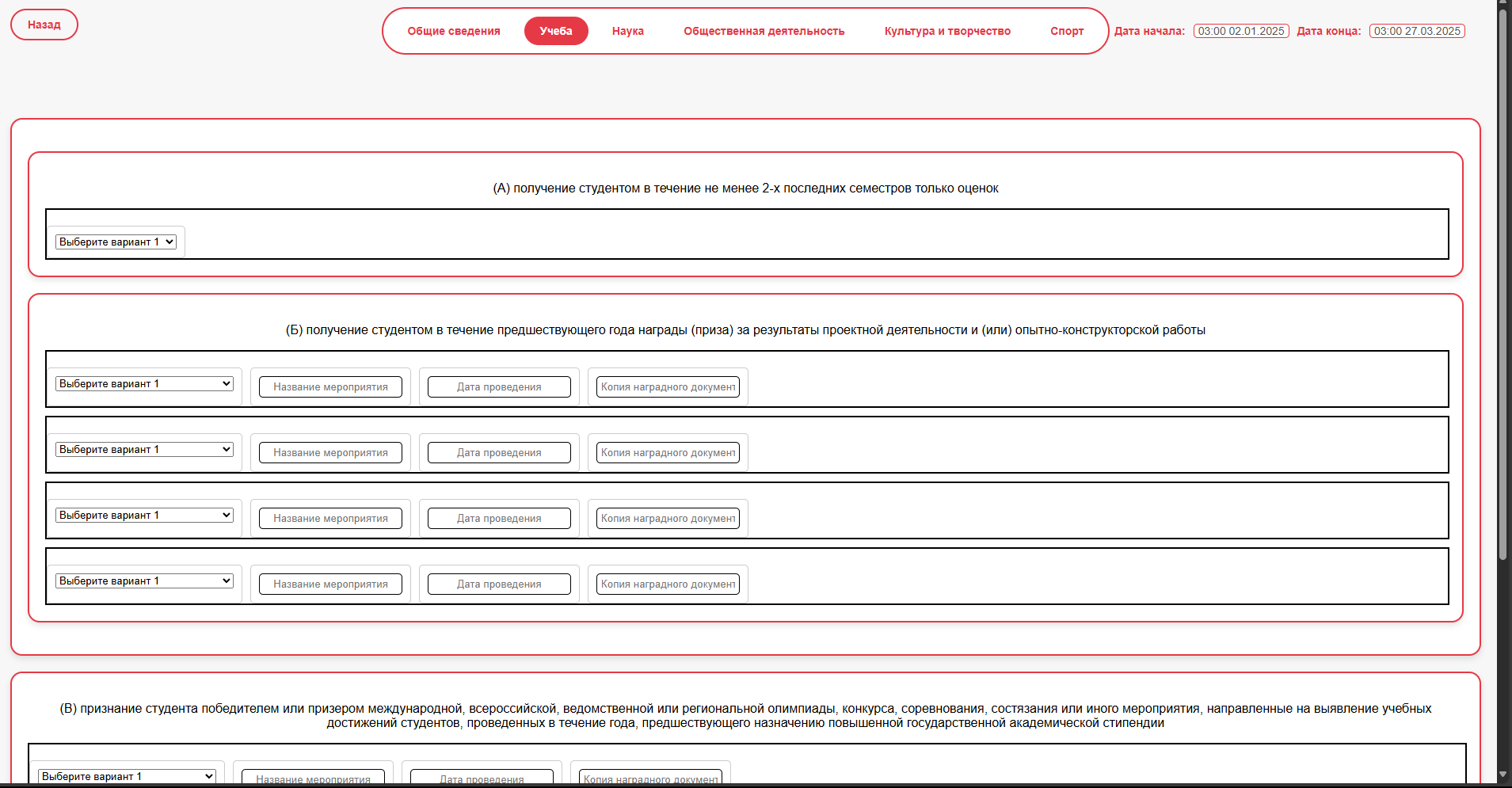


Рис.П2.5. Главное окно приложения

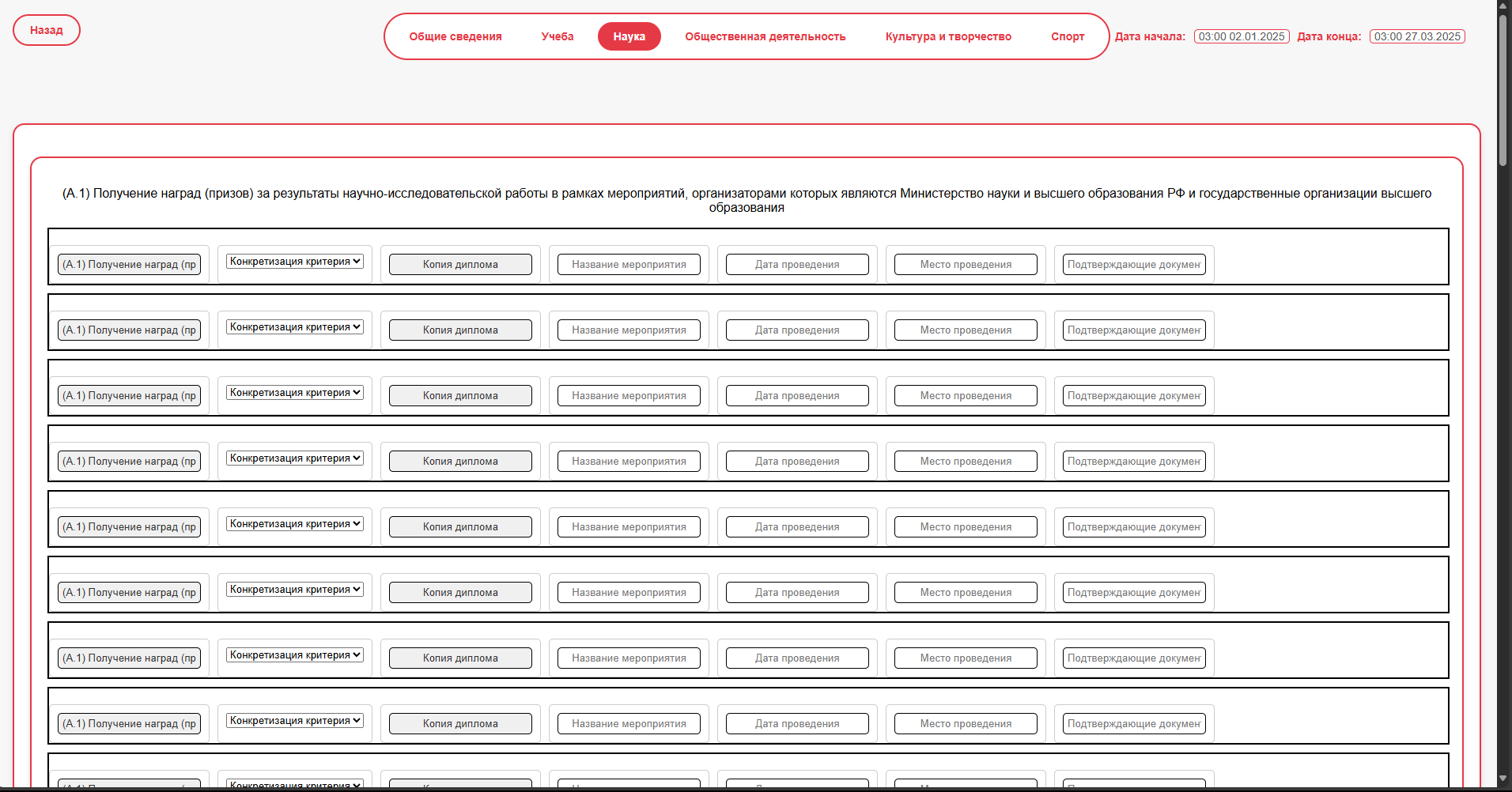


Рис.П2.6. Главное окно приложения

Также рассмотрим панель управления администратора, тут возможностей уже больше: «Просмотреть список пользователей», «Задать дату начала и конца подачи заявлений», «Просмотреть список заявлений», «Просмотреть список устаревших заявлений» и «Просмотреть историю действий» (рис.П2.7).

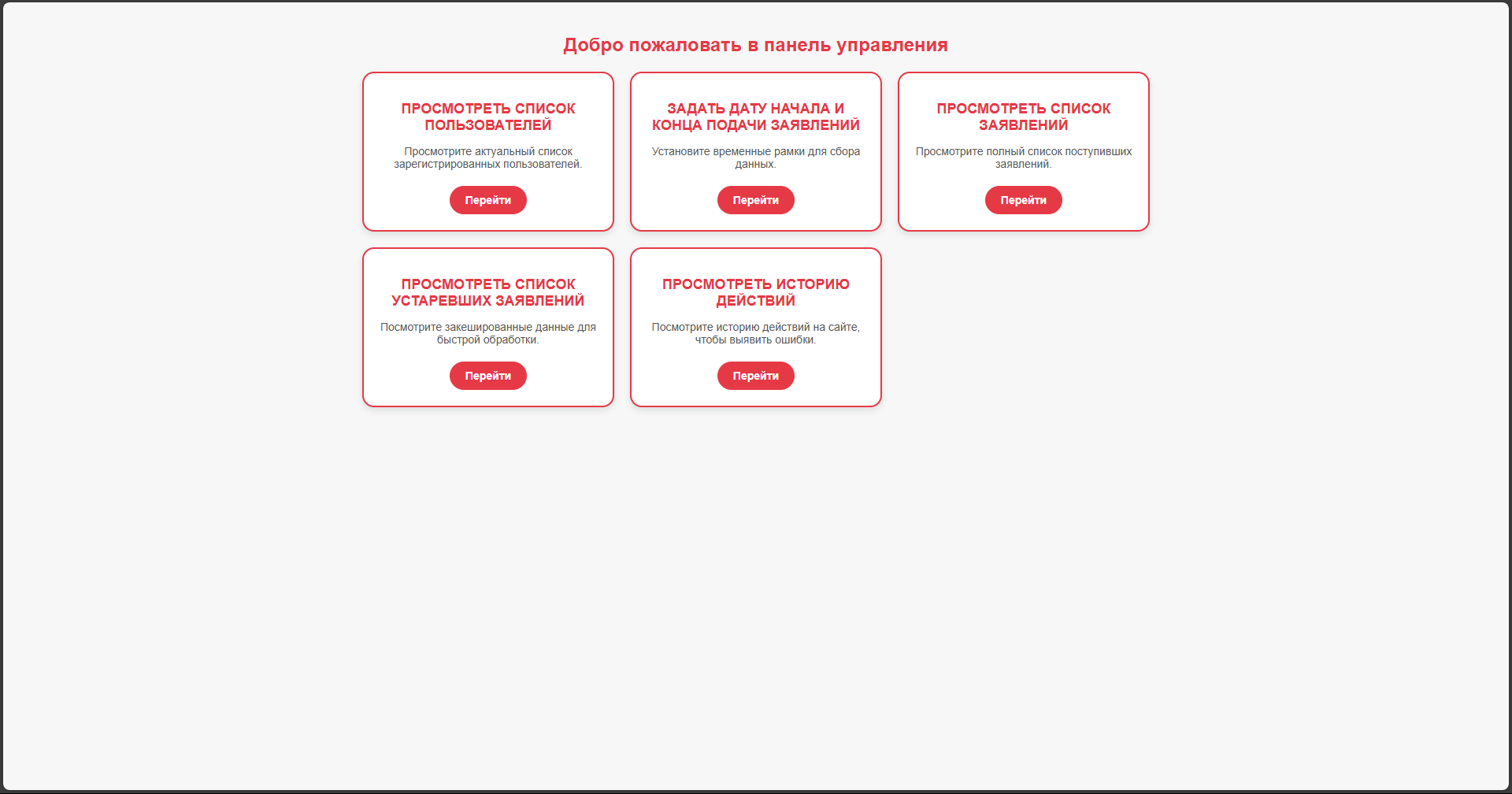


Рис.П2.7. Страница «Панель управления» для администратора

Просмотр списка пользователей администратором (рис.П2.8).

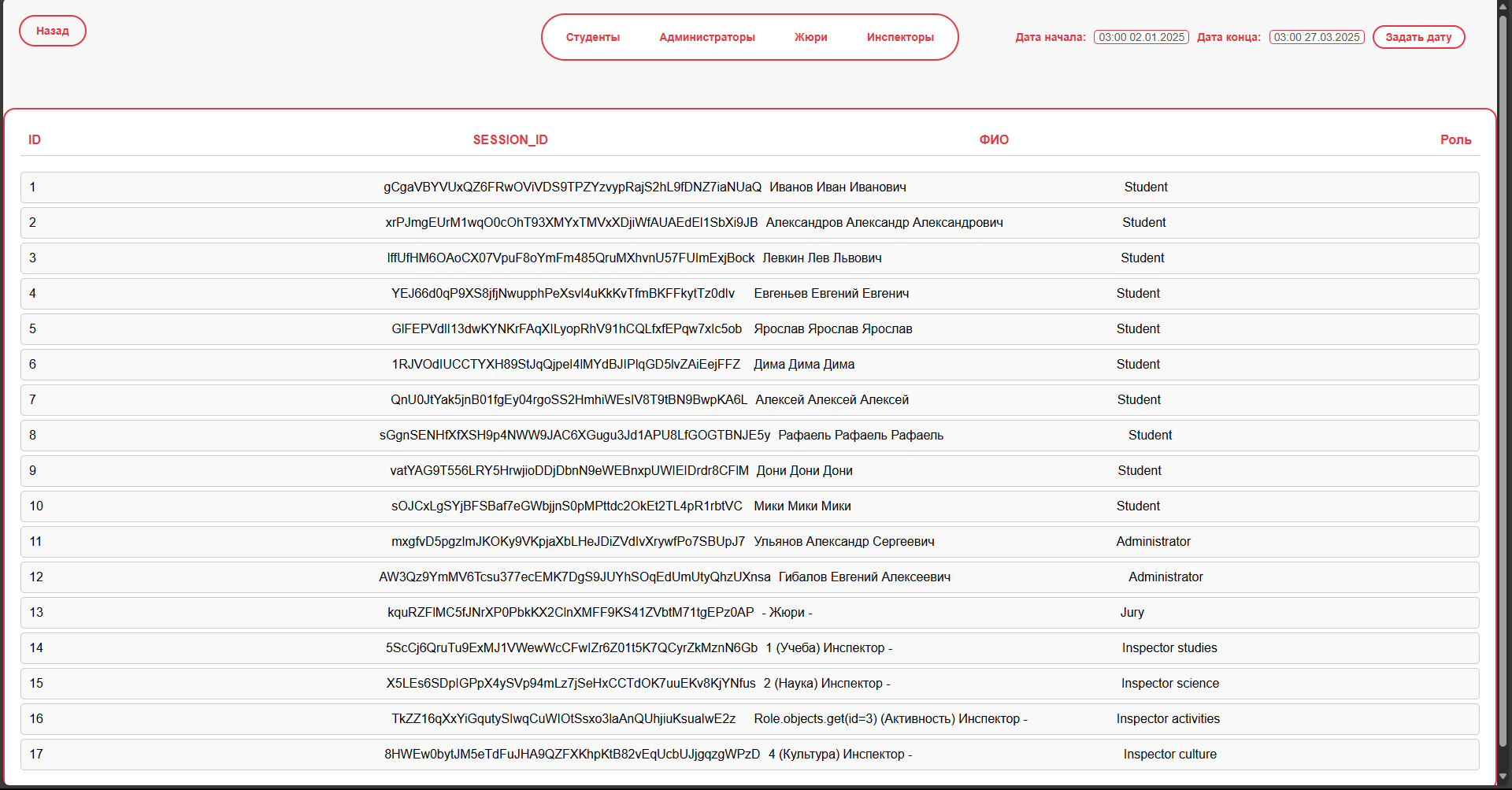


Рис.П2.8. Страница «Список пользователей»

При нажатии на кнопку «Задать дату начала и конца подачи заявлений» осуществится переход на соответствующую страницу (рис.П2.9).

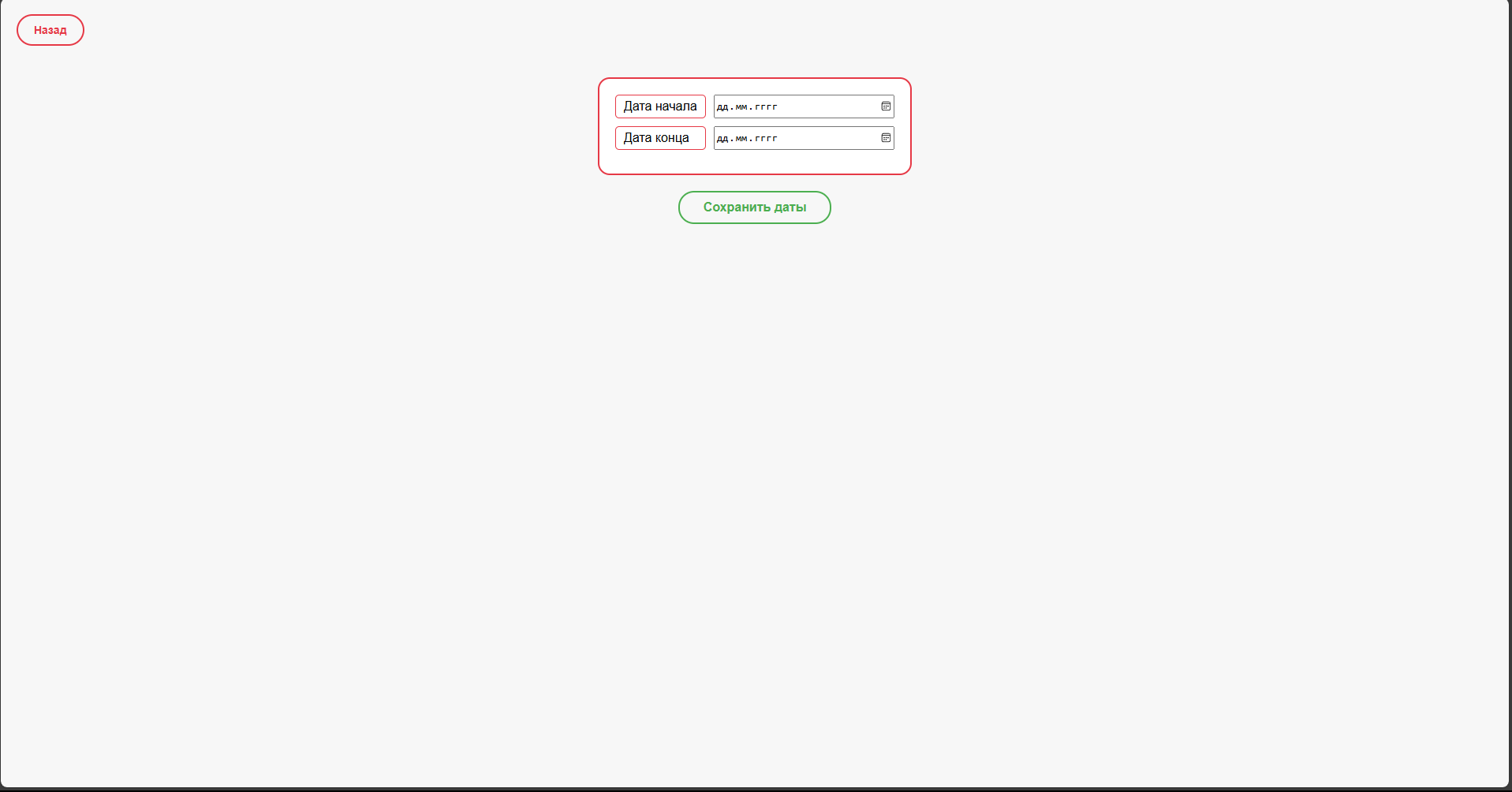


Рис.П2.9. Страница «Изменение даты подачи заявлений»

Попробуем изменить даты (рис.П2.10).

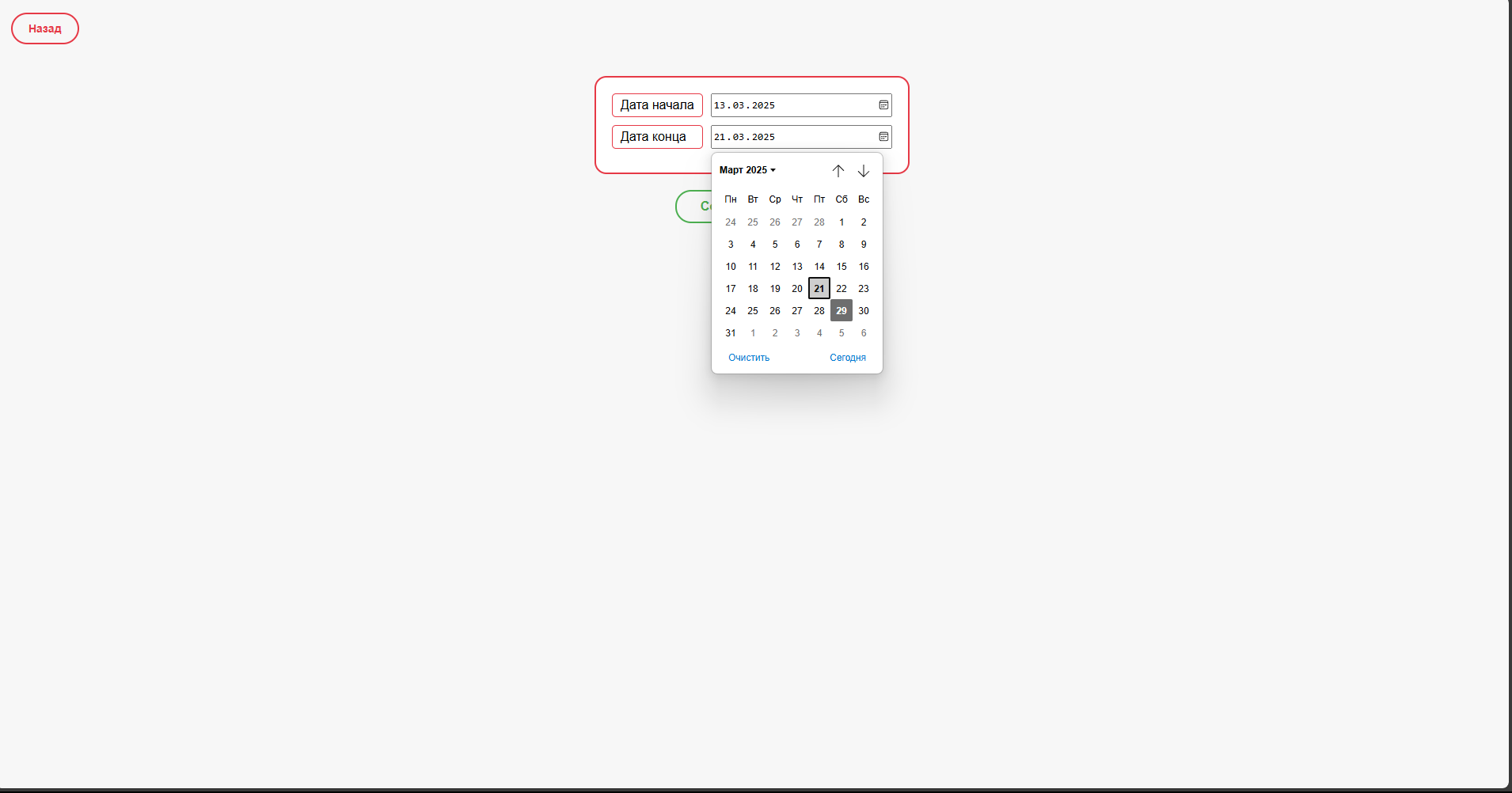


Рис.П2.10. Изменяем дату

Как видим, даты успешно изменены, это можно заметить на других страницах, где она отображается (рис.П2.11 и рис.П2.12).

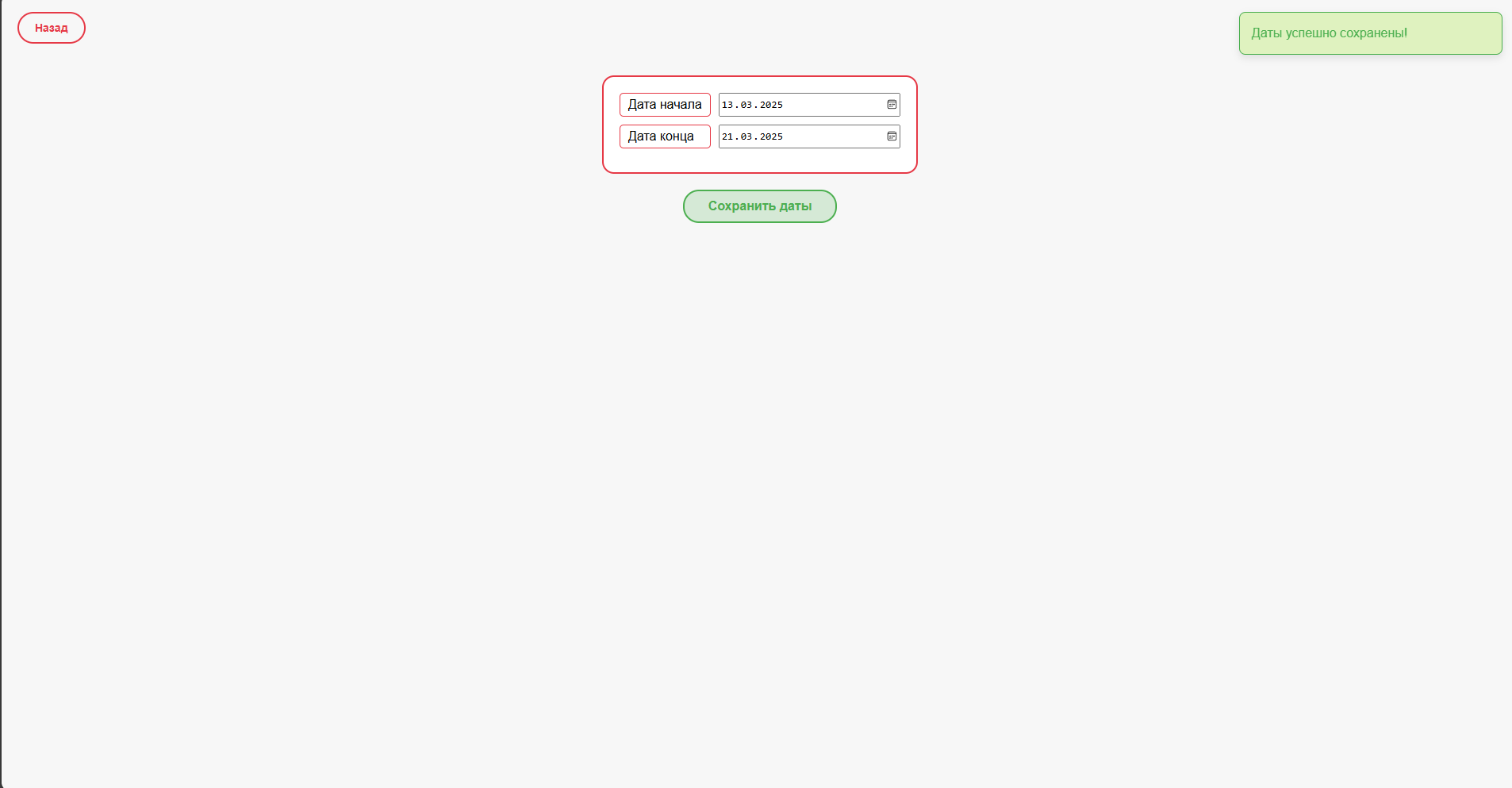


Рис.П2.11. Успешное изменение даты

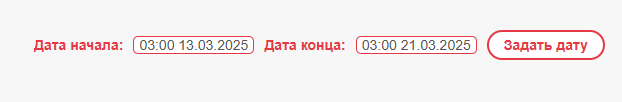


Рис.П2.12. Отображение измененной даты на другой странице

Также при нажатии на кнопку «История действий» мы перейдем на соответствую страницу, где можем ознакомится с историей действий в нашей программе (рис.П2.13).

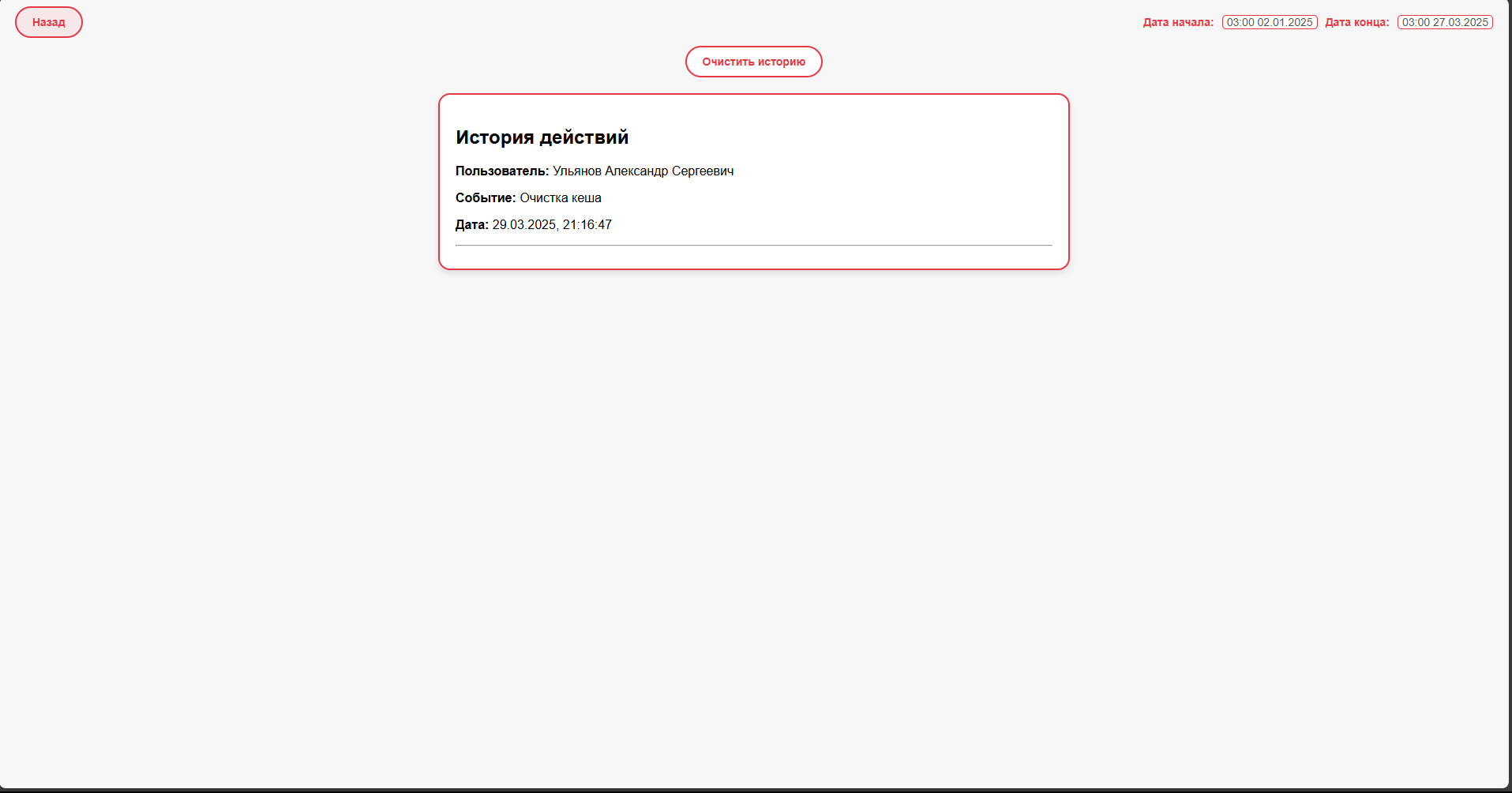


Рис.П2.13. Страница «История действий»

# Приложение 3. Текст программы

**Models.py**

from datetime import datetime, timezone  
from email.policy import default  
  
from django.db import models  
from django.template.defaultfilters import random  
from django.core.files.storage import FileSystemStorage  
import re  
# Create your models here.  
  
  
class Status(models.Model):  
 name = models.TextField("Статус", default="")  
  
 @classmethod  
 def create\_default\_records(cls):  
 # Данные для записи  
 default\_records = [  
 {'id': 0, 'name': "process"},  
 {'id': 1, 'name': "error"},  
 {'id': 2, 'name': "verified"},  
 {'id': 3, 'name': "conflict"},  
 {'id': 4, 'name': "confirm"},  
 {'id': 5, 'name': "deny"}  
 ]  
 for record in default\_records:  
 cls.objects.get\_or\_create(\*\*record)  
  
  
class Role(models.Model):  
 name = models.TextField("Роль", default="")  
  
 @classmethod  
 def create\_default\_records(cls):  
 # Данные для записи  
 default\_records = [  
 {'id': 0, 'name': "Student"},  
 {'id': 1, 'name': "Administrator"},  
 {'id': 2, 'name': "Jury"},  
 {'id': 3, 'name': "Inspector studies"},  
 {'id': 4, 'name': "Inspector science"},  
 {'id': 5, 'name': "Inspector activities"},  
 {'id': 6, 'name': "Inspector culture"},  
 {'id': 7, 'name': "Inspector sport"},  
 ]  
 for record in default\_records:  
 cls.objects.get\_or\_create(\*\*record)  
  
  
class User(models.Model):  
  
 id = models.TextField("Идентификатор", unique=True, editable=False, blank=True, primary\_key=True)  
  
 session = models.TextField("ID Сессии", default="", blank=True)  
  
 login = models.TextField("Логин",default="")  
 password = models.TextField("Пароль")  
  
 role = models.ForeignKey(Role, on\_delete=models.CASCADE, related\_name='users', default=0)  
  
 surname = models.TextField("Фамилия", blank=True)  
 name = models.TextField("Имя")  
 middlename = models.TextField("Отчество", blank=True)  
  
 email = models.TextField("Адрес", blank=True)  
 phone = models.TextField("Номер телефона", blank=True)  
  
 def \_\_str\_\_(self):  
 return str(self.surname) + " " + str(self.name) + " " + str(self.middlename)  
  
 def save(self, \*args, \*\*kwargs):  
 if not self.id: # Генерировать ID только если он ещё не установлен  
 self.id = User.generate\_id()  
 super().save(\*args, \*\*kwargs)  
  
 def get\_data(self):  
 return {"surname" : self.surname,  
 "name" : self.name,  
 "middlename": self.middlename,  
 "role": self.role.name,  
 "user-id": self.id  
 }  
  
 def get\_data\_full(self):  
 return {"session" : self.session,  
 "surname" : self.surname,  
 "login" : self.login,  
 "password" : self.password,  
 "name" : self.name,  
 "middlename": self.middlename,  
 "role": self.role.name,  
 "user-id": self.id,  
 "email": self.email,  
 "phone": self.phone  
 }  
  
 def generate\_session(self):  
 key = ""  
 for i in range(50):  
 key += random("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789")  
 self.session = key  
 self.save()  
 return key  
  
 def is\_student(self):  
 return self.role and self.role.id == 0  
  
 def is\_admin(self):  
 return self.role and self.role.id == 1  
  
 def is\_jury(self):  
 return self.role and self.role.id == 2  
  
 def is\_inspector(self):  
 return self.role and self.role.id in [3, 4, 5, 6, 7]  
  
  
 def get\_role(self):  
 return self.role.name  
  
  
 def change\_role(self, new\_role):  
 new\_role = int(new\_role)  
 self.role = Role.objects.filter(id = new\_role).last()  
 self.save()  
  
 # Получаем объект по session  
 @staticmethod  
 def get\_by\_session(session):  
 users = User.objects.filter(session=session)  
 if users.exists():  
 return users.first()  
 else:  
 return None  
  
 # Получаем объект по id  
 @staticmethod  
 def get\_by\_id(id\_):  
 users = User.objects.filter(id=id\_)  
 if users.exists():  
 return users.first()  
 else:  
 return None  
  
  
 # Получаем запись по логину и паролю  
 @staticmethod  
 def get\_by\_login\_password(login, password):  
 users = User.objects.filter(login=login, password=password)  
 if users.exists():  
 return users.first()  
 else:  
 return None

def get\_statements(self):  
 statements = Statement.objects.filter(user\_id=self.id)  
 if statements.exists():  
 return statements  
 else:  
 return None  
  
 def get\_id(self):  
 return self.id  
  
 @staticmethod  
 def add(login, password, role, name, surname):  
 user = User.objects.create(id=User.generate\_id(),login=login, password=password, role=Role.objects.get(id=role), name=name, surname=surname)  
 user.save()  
  
 @staticmethod  
 def get\_users\_info():  
 users\_info = {}  
 for i in range(7):  
 users\_info[i] = []  
 for s in User.objects.filter(role\_\_id = i):  
 users\_info[i].append(s.get\_data())  
  
 return users\_info  
  
 @staticmethod  
 def generate\_id():  
 while True:  
 key = ""  
 for i in range(50):  
 key += random("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789")  
 if not User.objects.filter(id=key).exists(): # Проверка на уникальность  
 return key  
  
 @classmethod  
 def create\_default\_records(cls):  
 # Данные для записи  
 default\_records = [  
 {'login': "admin", 'password': "admin", 'role': Role.objects.get(id=1), 'surname': "Ульянов", 'name': "Александр", 'middlename': "Сергеевич", 'email': "herobrinechekplay@gmail.com", 'phone': "89005442577"},  
 {'login': "admin2", 'password': "admin2", 'role': Role.objects.get(id=1), 'surname': "Гибалов", 'name': "Евгений", 'middlename': "Алексеевич", 'email': "process", 'phone': "process"},  
  
 {'login': "inspector1", 'password': "inspector1", 'role': Role.objects.get(id=3), 'surname': "1 (Учеба)", 'name': "Инспектор", 'middlename': "-", 'email': "-", 'phone': "-"},  
 {'login': "inspector2", 'password': "inspector2", 'role': Role.objects.get(id=4), 'surname': "2 (Наука)", 'name': "Инспектор", 'middlename': "-", 'email': "-", 'phone': "-"},  
 {'login': "inspector3", 'password': "inspector3", 'role': Role.objects.get(id=5), 'surname': "Role.objects.get(id=3) (Активность)", 'name': "Инспектор", 'middlename': "-", 'email': "-", 'phone': "-"},  
 {'login': "inspector4", 'password': "inspector4", 'role': Role.objects.get(id=6), 'surname': "4 (Культура)", 'name': "Инспектор", 'middlename': "-", 'email': "-", 'phone': "-"},  
 {'login': "inspector5", 'password': "inspector5", 'role': Role.objects.get(id=7), 'surname': "5 (Спорт)", 'name': "Инспектор", 'middlename': "-", 'email': "-", 'phone': "-"},  
  
 {'login': "jury", 'password': "jury", 'role': Role.objects.get(id=2), 'surname': "-", 'name': "Жюри", 'middlename': "-", 'email': "-", 'phone': "-"},  
  
 {'login': "student", 'password': "student", 'role': Role.objects.get(id=0), 'surname': "Иванов", 'name': "Иван",  
 'middlename': "Иванович", 'email': "ivan@gmail.com", 'phone': "909090909"},  
 {'login': "student1", 'password': "student1", 'role': Role.objects.get(id=0), 'surname': "Александров", 'name': "Александр",  
 'middlename': "Александрович", 'email': "alex@gmail.com", 'phone': "909090909"},  
 {'login': "student2", 'password': "student2", 'role': Role.objects.get(id=0), 'surname': "Левкин", 'name': "Лев",  
 'middlename': "Львович", 'email': "lev@gmail.com", 'phone': "909090909"},  
 {'login': "student3", 'password': "student3", 'role': Role.objects.get(id=0), 'surname': "Евгеньев", 'name': "Евгений",  
 'middlename': "Евгенич", 'email': "evgen@gmail.com", 'phone': "909090909"},  
 {'login': "student4", 'password': "student4", 'role': Role.objects.get(id=0), 'surname': "Ярослав", 'name': "Ярослав",  
 'middlename': "Ярослав", 'email': "yar@gmail.com", 'phone': "909090909"},  
 {'login': "student5", 'password': "student5", 'role': Role.objects.get(id=0), 'surname': "Дима", 'name': "Дима",  
 'middlename': "Дима", 'email': "dima@gmail.com", 'phone': "909090909"},  
 {'login': "student6", 'password': "student6", 'role': Role.objects.get(id=0), 'surname': "Алексей", 'name': "Алексей",  
 'middlename': "Алексей", 'email': "alexey@gmail.com", 'phone': "909090909"},  
 {'login': "student7", 'password': "student7", 'role': Role.objects.get(id=0), 'surname': "Рафаель", 'name': "Рафаель",  
 'middlename': "Рафаель", 'email': "raf@gmail.com", 'phone': "909090909"},  
 {'login': "student8", 'password': "student8", 'role': Role.objects.get(id=0), 'surname': "Дони", 'name': "Дони",  
 'middlename': "Дони", 'email': "doni@gmail.com", 'phone': "909090909"},  
 {'login': "student9", 'password': "student9", 'role': Role.objects.get(id=0), 'surname': "Мики", 'name': "Мики",  
 'middlename': "Мики", 'email': "miki@gmail.com", 'phone': "909090909"}  
 ]  
 for record in default\_records:  
 cls.objects.get\_or\_create(\*\*record)  
  
class Statement(models.Model):  
  
 id = models.TextField("Идентификатор", unique=True, editable=False, blank=True, primary\_key=True)  
  
 user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name="statements", default=None)  
  
 status = models.ForeignKey(Status, on\_delete=models.CASCADE, related\_name='statements', blank=True, default=0)  
  
 json = models.JSONField("Json", default=list, blank=True)  
 urls = models.JSONField("Json", default=list, blank=True)  
  
 mark\_studies = models.IntegerField("Оценка учеба", default=False, blank=True)  
 comment\_studies = models.TextField("Комментарий учеба", blank=True)  
  
 mark\_science = models.IntegerField("Оценка наука", default=False, blank=True)  
 comment\_science = models.TextField("Комментарий наука", blank=True)  
  
 mark\_activities = models.IntegerField("Оценка мероприятия", default=False, blank=True)  
 comment\_activities = models.TextField("Комментарий мероприятия", blank=True)  
  
 mark\_culture = models.IntegerField("Оценка культура", default=False, blank=True) # СДЕЛАТЬ МИГРАТЕ  
 comment\_culture = models.TextField("Комментарий культура", blank=True)  
  
 mark\_sport = models.IntegerField("Оценка спорт", default=False, blank=True)  
 comment\_sport = models.TextField("Комментарий спорт", blank=True)  
  
 date = models.DateTimeField("Дата", default=datetime.now(), blank=True)  
  
 points = models.IntegerField("Баллы", default=0, blank=True)  
  
 old\_status = models.BooleanField("Кеш-статус", blank=True, default=False)  
  
  
 def get\_data(self):  
 return {"user" : str(self.user),  
 "status" : self.status.name,  
 "points": self.points,  
 "date": self.date.strftime("%H:%M %d.%m.%Y"),  
 "statement-id": self.id,  
 "old-status": self.old\_status  
 }  
  
 def get\_json\_data(self):  
 return self.json  
  
  
 def mark(self, user, value, comment):  
 if user.role.id == 3:  
 self.set\_mark\_studies(value, comment)  
 elif user.role.id == 4:  
 self.set\_mark\_science(value, comment)  
 elif user.role.id == 5:  
 self.set\_mark\_activities(value, comment)  
 elif user.role.id == 6:  
 self.set\_mark\_culture(value, comment)  
 elif user.role.id == 7:  
 self.set\_mark\_sport(value, comment)  
  
 def set\_mark\_studies(self, value, comment):  
 self.mark\_studies = value  
 if not value:  
 self.comment\_studies = comment  
 else:  
 self.check\_all\_marks()  
  
 def set\_mark\_science(self, value, comment):  
 self.mark\_science = value  
 if not value:  
 self.comment\_science = comment  
 else:  
 self.check\_all\_marks()  
  
 def set\_mark\_activities(self, value, comment):  
 self.mark\_activities = value  
 if not value:  
 self.comment\_activities = comment  
 else:  
 self.check\_all\_marks()  
  
 def set\_mark\_culture(self, value, comment):  
 self.mark\_culture = value  
 if not value:  
 self.comment\_culture = comment  
 else:  
 self.check\_all\_marks()  
  
 def set\_mark\_sport(self, value, comment):  
 self.mark\_sport = value  
 if not value:  
 self.comment\_sport = comment  
 else:  
 self.check\_all\_marks()  
  
  
 def check\_all\_marks(self):  
 if self.mark\_studies and self.mark\_science and self.mark\_activities and self.mark\_culture and self.mark\_sport and self.status == 0:  
 self.status = 2  
  
 def set\_status(self, id\_):  
 self.status.id = Status.objects.get(id=id\_)  
  
 def set\_status\_up(self):  
 if self.status.id == 1:  
 self.status = Status.objects.get(id=0)  
 elif self.status.id == 0:  
 self.status = Status.objects.get(id=2)  
  
 def set\_status\_down(self):  
 if self.status.id == 0:  
 self.status = Status.objects.get(id = 1)  
 elif self.status.id == 2:  
 self.status = Status.objects.get(id = 0)  
  
 def remove\_files(self):  
 if self.urls is not [] or self.urls is not None:  
 for file in self.urls:  
 fs = FileSystemStorage(location="files")  
 if fs.exists(file):  
 fs.delete(file)  
 self.urls = {}  
 self.save()  
  
 def delete(self, \*args, \*\*kwargs):  
 self.remove\_files()  
 super().delete(\*args, \*\*kwargs)  
  
 @staticmethod  
 def upload(user, json, files):  
 date = datetime.now(timezone.utc)  
 if Period.is\_require(date):  
 statement\_temp = Statement.get\_by\_user(user)  
 if statement\_temp is not None and statement\_temp.old\_status is not True:  
 statement\_temp.remove\_files()  
 json\_change = Statement.replace\_at\_values\_with\_links(json, files)  
 statement\_temp.json = json\_change[0]  
 statement\_temp.urls = json\_change[1]  
 statement\_temp.date = datetime.now()  
 else:  
 json\_change = Statement.replace\_at\_values\_with\_links(json, files)  
 statement\_temp = Statement.objects.create(id=Statement.generate\_id(), user=user, json=json\_change[0], date=date, urls = json\_change[1])  
 # statement\_temp = Statement.objects.create(id=Statement.generate\_id(), user=user, json=json\_change[0], date=date, urls = json\_change[1])  
  
 statement\_temp.save()  
  
 return True  
 else:  
 return False  
  
  
 @staticmethod  
 def system\_checkout(counts):  
 statements = Statement.objects.filter(status\_\_id=2).order\_by('-points')  
  
 part\_of\_statements = {}  
 prev\_points = 0  
  
 # пробегаем и формируем списки с одинаковыми баллами  
 for i in statements:  
 if prev\_points == i.point:  
 part\_of\_statements[i.point].append(i)  
 else:  
 part\_of\_statements[i.point] = [i]  
  
  
 for i in part\_of\_statements:  
 if len(i) > counts:  
 for s in i:  
 s.status = Status.objects.get(id=4)  
 counts -= len(i)  
 elif len(i) < counts and counts > 0:  
 for s in i:  
 s.status = Status.objects.get(id=3)  
 counts -= len(i)  
 else:  
 for s in i:  
 s.status = Status.objects.get(id=5)  
  
 @staticmethod  
 def get\_by\_user(user):  
 statements = user.statements.all()  
 if statements.exists():  
 return statements.last()  
 else:  
 return None  
  
  
 @staticmethod  
 def get\_by\_id(statement\_id):  
 statements = Statement.objects.filter(id=statement\_id)  
 if statements.exists():  
 return statements.last()  
 else:  
 return None  
  
 @staticmethod  
 def get\_statements\_by\_statuses(statuses):  
 statements\_info = {}  
 for i in statuses:  
 statements\_info[i] = []  
 for s in Statement.objects.filter(status\_\_name = i, old\_status = False):  
 statements\_info[i].append(s.get\_data())  
  
 return statements\_info  
  
 @staticmethod  
 def get\_statements\_old():  
 statements\_info = []  
 for s in Statement.objects.filter(old\_status = True):  
 statements\_info.append(s.get\_data())  
  
 return statements\_info  
  
 @staticmethod  
 def get\_part\_statement(user, statement\_id):  
 statement = Statement.get\_by\_id(statement\_id)  
 statement = statement.get\_json\_data()  
 if user.role.id == 3:  
 return statement["studies"]  
 elif user.role.id == 4:  
 return statement["science"]  
 elif user.role.id == 5:  
 return statement["activities"]  
 elif user.role.id == 6:  
 return statement["culture"]  
 elif user.role.id == 7:  
 return statement["sport"]  
 return None  
  
  
 @staticmethod  
 def generate\_id():  
 while True:  
 key = ""  
 for i in range(50):  
 key += random("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789")  
 if not Statement.objects.filter(id=key).exists(): # Проверка на уникальность  
 return key  
  
 @staticmethod  
 def set\_old():  
 statements = Statement.objects.filter(old\_status = False)  
 for s in statements:  
 s.old\_status = True  
 s.save()  
  
  
 # Функция для замены значений "@n" на ссылки  
 @staticmethod  
 def replace\_at\_values\_with\_links(data, files):  
 urls = []  
 def replace\_values(item, files\_):  
 nonlocal urls  
 if isinstance(item, dict):  
 for key, value in item.items():  
 item[key] = replace\_values(value, files\_)  
 elif isinstance(item, list):  
 item = [replace\_values(elem, files\_) for elem in item]  
 elif isinstance(item, str): # Проверка, что item - строка  
 match = re.fullmatch(r"@(\d+)", item)  
 if match:  
 index = int(match.group(1)) # Получаем число после @  
 if index < len(files\_):  
 fs = FileSystemStorage(location="files")  
 filename = fs.save(files\_[index].name, files\_[index])  
 item = fs.url(filename)  
 urls.append(filename)  
 else:  
 raise ValueError(f"Файл с индексом {index} отсутствует в переданном массиве файлов.")  
 return item  
  
 # Создаем копию данных и передаем её для обработки  
 values = replace\_values(data, files.getlist("files[]"))  
 return [values, urls]  
  
class Period(models.Model):  
 date = models.DateTimeField("Дата")  
 description = models.TextField("Описание")  
  
 @staticmethod  
 def get\_period():  
 return [Period.objects.get(id = 0).date.strftime("%H:%M %d.%m.%Y"), Period.objects.get(id = 1).date.strftime("%H:%M %d.%m.%Y")]  
  
 @staticmethod  
 def is\_require(date):  
 period\_start = Period.objects.get(id=0).date  
 period\_end = Period.objects.get(id=1).date  
  
 return period\_start < date < period\_end  
  
 @staticmethod  
 def set\_start\_and\_end(datestr\_start, datestr\_end):  
 period\_temp = Period.objects.get(id = 0)  
 period\_temp.date = datetime.strptime(datestr\_start, "%H:%M %d.%m.%Y")  
 period\_temp.save()  
  
 period\_temp = Period.objects.get(id = 1)  
 period\_temp.date = datetime.strptime(datestr\_end, "%H:%M %d.%m.%Y")  
 period\_temp.save()  
  
 @classmethod  
 def create\_default\_records(cls):  
 # Данные для записи  
 default\_records = [  
 {'id': 0, 'date': datetime(2020, 1, 1, 0, 0, 0), 'description': "Дата, с которой можно подавать заявления."},  
 {'id': 1, 'date': datetime(2020, 1, 1, 0, 0, 0), 'description': "Дата, до которой включительно можно подавать заявления."}  
 ]  
 for record in default\_records:  
 cls.objects.get\_or\_create(\*\*record)  
  
class Log(models.Model):  
 event = models.TextField("Событие", default="")  
 statement\_copy = models.JSONField("Контент", blank=True)  
 user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name="logs")  
 date = models.DateTimeField("Дата", default=datetime.now(), blank=True)  
 data = models.TextField("Примечание", default="")  
  
 def get\_data(self):  
 return {"user" : str(self.user),  
 "statement" : self.statement\_copy,  
 "event": self.event,  
 "date": self.date,  
 "data":self.data  
 }  
  
 @staticmethod  
 def add(user, event, data, json):  
 statement\_temp = Log.objects.create(user=user, event=event, data=data, statement\_copy=json, date=datetime.now())  
 statement\_temp.save()  
  
 @staticmethod  
 def get\_list():  
 logs = []  
 for l in Log.objects.all():  
 logs.append(l.get\_data())  
  
 return logs  
  
  
 @staticmethod  
 def reset():  
 for s in Log.objects.all():  
 s.delete()