СОДЕРЖАНИЕ

ВВЕДЕНИЕ 2

1 Постановка задачи 4

2 Разработка структуры базы данных 5

2.1 Описание предметной области 5

2.2 Проектирование системы с помощью UML-диаграмм 11

2.3 Функциональные требования к системе 22

2.4 Пользователи системы и их привилегии 24

3 Разработка программной системы 26

3.1 Структура программы 26

3.2 Схема данных 27

3.3 Тестирование программы 32

3.4 Руководство пользователя 40

ЗАКЛЮЧЕНИЕ 42

ПРИЛОЖЕНИЕ А 44

ПРИЛОЖЕНИЕ Б 97

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 102

# ВВЕДЕНИЕ

Лаборатории играют ключевую роль в научных исследованиях, обеспечивая условия для проведения экспериментов, анализа данных и генерации новых идей. Однако без современных инструментов для управления внутренними процессами эффективность работы сотрудников может значительно снижаться. Взаимодействие между коллегами, контроль задач, а также ведение документации часто усложняются из-за отсутствия централизованных цифровых решений.

Создание веб-приложения для сотрудников лаборатории является актуальной задачей, так как оно способно объединить основные рабочие процессы в единую цифровую платформу. Такое приложение предоставит сотрудникам возможность общения в чатах, совместного написания научных статей, отслеживания задач и управления их выполнением. Администраторы, в свою очередь, смогут регистрировать новых пользователей, редактировать данные сотрудников, сортировать и фильтровать списки, добавлять задачи, а также экспортировать данные в Excel. Дополнительной функциональностью станет возможность настройки цветовой темы интерфейса.

Разработка данного решения основана на применении современных технологий: клиентская часть реализуется с использованием ReactJS, серверная логика разделена между двумя микросервисами на ExpressJS и GO (с использованием фреймворка Gin), а для хранения данных используется MySQL Workbench. Такое техническое решение предоставляет следующие преимущества:

* структурированное хранение данных о сотрудниках, задачах и статьях;
* удобный интерфейс для работы с программой;
* минимизация времени на поиск и обработку информации;
* автоматизация рутинных операций, таких как учёт и добавление задач, оформление научных статей, учёт информации о сотрудниках;
* повышение эффективности командной работы и контроля задач;
* гибкость настройки интерфейса под предпочтения пользователей.

Целью учебной практики является создание веб-приложения с базой данных, которая будет содержать информацию о сотрудниках лаборатории, их задачах, чатах, научных статьях, а также предоставлять возможность управления данными о пользователях, их регистрацией и взаимодействием с системой.

Для достижения этой цели необходимо решить следующие задачи:

* спроектировать диаграммы IDEF0 и DFD для анализа предметной области и выделения основных информационных потоков данных;
* спроектировать диаграмму IDEF1X с описанием структуры таблиц БД;
* спроектировать диаграмму Use case, описывающая основные функциональные требования к программе;
* спроектировать диаграмму деятельности, иллюстрирующая исходы вариантов тестирования функционала проектируемой ИС;
* спроектировать диаграмму классов;
* спроектировать диаграмму компонентов;
* сформулировать спецификацию на разработку программного продукта в виде фрагмента технического задания, в котором описываются разделы:
  1. назначение программы;
  2. требования к функциональным характеристикам;
  3. требования к надежности;
  4. требования к составу и параметрам технических средств;
  5. требования к информационной и программной совместимости;
  6. требования к программной документации;
* заполнить базу данных в соответствии с семантикой её полей;
* создать главную форму приложения с навигацией по разделам, формы регистрации и авторизации пользователей, формы, соответствующие таблицам базы данных, с возможностью просмотра, добавления, удаления, обновления, поиска, фильтрации записей в таблицах базы данных. Ограничить доступ для разных ролей пользователей к формам. Создать как минимум одну форму для выполнения вычисляемой функции;
* произвести тестирование ИС и отладить её.

# 1 Постановка задачи

Создание программного обеспечения для веб-приложения сотрудников лаборатории включает следующие ключевые аспекты:

Основной фактор – приложение предназначено для эффективной организации рабочих процессов, включая управление задачами, коммуникацию между сотрудниками и взаимодействия с научными статьи.

Основные функции:

* учет данных о сотрудниках, их задачах, чатах, научных статьях и операциях в системе;
* хранение информации о пользователях (ФИО, права доступа, имя пользователя, пароль, должность, зарплата, пол, дату рождения, семейный статус, наличие детей, учёная степень, отдел, опыт работы, номер телефона, адрес электронной почты), задачах (описание, прогресс выполнения) и чатах;
* поддержка фильтрации, поиска и сортировки данных по различным критериям;
* реализация функционала для администраторов, включая изменение данных сотрудников, создание регистрационных кодов, экспорт данных в Excel, а также увольнение сотрудников;
* возможность изменения цветовой темы приложения для улучшения пользовательского опыта.

Эти функции и особенности позволяют создать программное обеспечение, ориентированное на удовлетворение потребностей лаборатории и обеспечение ее эффективного функционирования.

# 2 Разработка структуры базы данных

# 2.1 Описание предметной области

Предметная область − часть реального мира, исследуемая в конкретном проекте, которая описывается с помощью инструментария системы управления базами данных.

Описание предметной области: Вы работаете в лаборатории и занимаетесь организацией рабочих процессов и взаимодействием между сотрудниками. В вашем распоряжении имеются данные о сотрудниках (ФИО, права доступа, имя пользователя, пароль, должность, зарплата, пол, дату рождения, семейный статус, наличие детей, учёная степень, отдел, опыт работы, номер телефона, адрес электронной почты), задачах (описание, прогресс выполнения), а также научных статьях, созданных сотрудниками. Администраторы могут управлять данными пользователей, назначать задачи, регистрировать новых сотрудников через специальные коды, экспортировать данные и редактировать информацию. Сотрудники имеют возможность вести переписку в чатах, следить за своими задачами и писать научные статьи.

Целью приложения является помощь в управлении сотрудниками лаборатории, упрощение выполнения задач и ведения документации, а также обеспечение удобной платформы для взаимодействия между сотрудниками и администраторами.

Задачи проектирования приложения для лаборатории могут быть сформулированы следующим образом:

1. Обеспечение платформы для управления сотрудниками и задачами: Создание среды, в которой пользователи (сотрудники и администраторы) могут взаимодействовать с системой для организации рабочих процессов лаборатории.
2. Поиск, фильтрация и сортировка: Разработка системы поиска, фильтрации и сортировки сотрудников и задач по различным параметрам (ФИО, должность, статус задачи, сроки выполнения), что позволяет пользователям быстро находить нужную информацию.
3. Учет операций: Реализация учета операций, связанных с созданием и выполнением задач, ведением научных статей и взаимодействием сотрудников в чатах.
4. Формирование отчетов: Предоставление возможности формирования отчётов с информации о сотрудниках, включая экспорт данных в формат Excel.
5. Удобство взаимодействия: Создание интуитивного интерфейса, который обеспечивает легкость использования системы для всех категорий пользователей. Это включает доступ к функциям создания чатов, отслеживания задач, работы с научными статьями и управления данными.
6. Хранение и защита данных: Обеспечение надежного хранения информации о сотрудниках, задачах, чатах и научных статьях, а также защита данных от утраты или несанкционированного доступа;
7. Персонализация интерфейса: Внедрение возможности изменения цветовой темы приложения, что позволит пользователям настроить внешний вид интерфейса под свои предпочтения.

Из приведенного описания можно сделать вывод, что база данных для веб-приложения лаборатории должна включать следующие сущности:

* отделы (ID отдела, название);
* пользователи (ID пользователя, логин, пароль, права доступа, личная информация, должность, ученая степень, отдел, стаж, зарплата, контакты, статус блокировки, ссылка на профильную фотографию);
* ключи для регистрации (ID задачи, ID сотрудника, описание задачи, прогресс выполнения);
* задачи (ID задачи, текст задачи, ID сотрудника);
* статьи (ID статьи, название, ссылка на HTML, статус завершения, ID автора, дата создания, привязка к тематике: биология, химия, ИТ, физика);
* комментарии к статьям (ID комментария, ID статьи, ID автора комментария, текст комментария);
* чаты (ID чата, название, статус приватности);
* пользователи чатов (ID записи, ID пользователя, ID чата);
* сообщения в чатах (ID сообщения, ID чата, ID отправителя, текст сообщения, дата отправки, статус закрепления);
* медиафайлы сообщений (ID медиафайла, ID сообщения, ссылка на файл);
* избранные статьи (ID записи, ID статьи, ID пользователя).

Они предназначены для хранения систематизированной информации о сотрудниках, задачах, чатах, научных статьях, сообщениях и файлах сообщений. Администраторам должны быть предоставлены удобные средства для добавления, редактирования и управления данными, обеспечивая эффективную организацию работы лаборатории.

IDEF0 (Integration Definition for Function Modeling) – методология функционального моделирования для описания функций предприятия, предлагающая язык функционального моделирования для анализа, разработки, реинжиниринга и интеграции информационных систем бизнес процессов; или анализа инженерии разработки ПО (or software engineering analysis).

Методология IDEF0 является развитием метода структурного анализа и проектирования SADT (Structured Analysis and Design Technique).

IDEF0 – Integration DEFinition language 0 – основан на SADT и в своей исходной форме включает одновременно: определение языка графического моделирования (синтаксис и семантику) и описание полной (comprehensive) методологии разработки моделей.

IDEF0 используется для создания функциональной модели, то есть результатом применения методологии IDEF0 к системе есть функциональная модель IDEF0.

Функциональная модель – это структурное представление функций, деятельности или процессов в пределах моделируемой системы или предметной области.

Перед началом рассмотрения синтаксиса DFD следует отдельно отметить, что в отличие от SADT (IDEF0) DFD методологией не является. Другими словами DFD – это всего лишь набор общепринятых обозначений без жестких ограничений к способам моделирования и применения полученных моделей.

Хорошо спроектированные информационные системы многократно улучшают эффективность работы почти любого предприятия, в том числе и научной лаборатории.

Сначала была создана контекстная диаграмма «Взаимодействие с научными статьями» в методологии IDEF0.

В управление вошли:

* правила оформления статей.

Механизмами осуществления процесса являются:

* пользователь.

Входными данными являются:

* контент статьи;
* текст комментария;
* нажатие на кнопку «добавить в избранное».

Выходные данные:

* опубликованная статья;
* комментарий;
* список избранных статей.

На рисунке 2.1 изображена контекстная диаграмма «Взаимодействие с научными статьями».

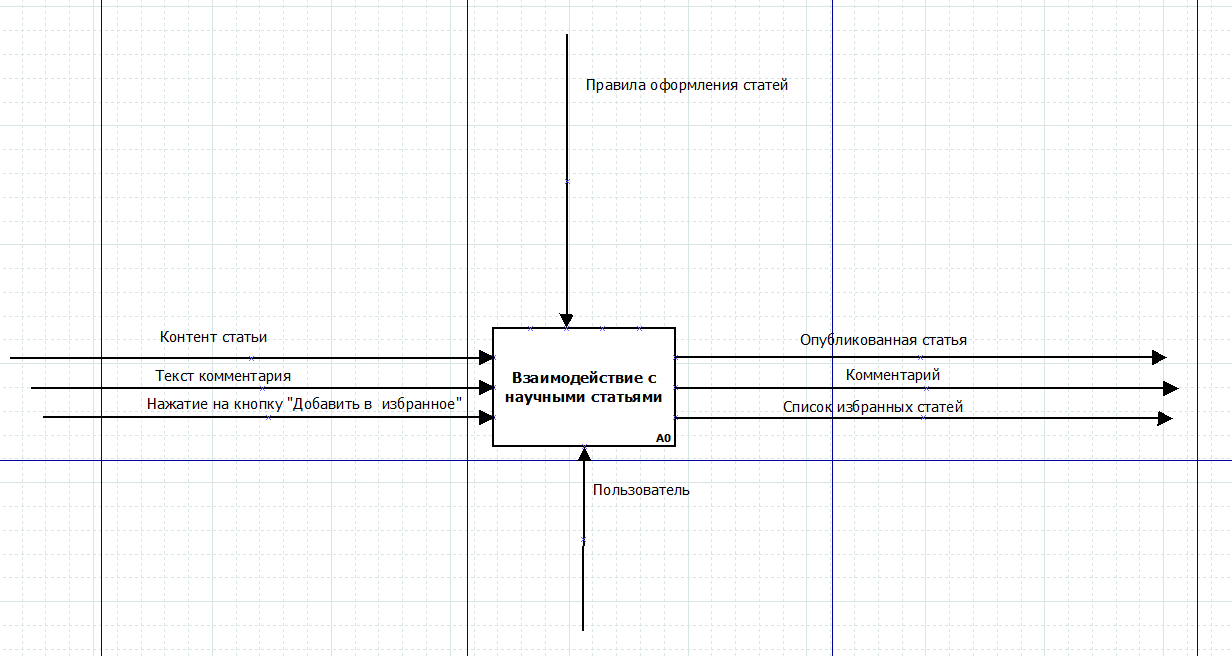


Рисунок 2.1 — Контекстная диаграмма «Взаимодействие с научными статьями»

Декомпозиция позволяет постепенно и структурированно представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой.

В процессе декомпозиции функциональный блок, который в контекстной диаграмме отображает систему как единое целое, подвергается детализации на другой диаграмме. Получившаяся диаграмма второго уровня содержит функциональные блоки, отображающие главные подфункции функционального блока контекстной диаграммы и называется дочерней по отношению к нему (каждый из функциональных блоков, принадлежащих дочерней диаграмме, соответственно называется дочерним блоком). В свою очередь, функциональный блок - предок называется родительским блоком по отношению к дочерней диаграмме, а диаграмма, к которой он принадлежит – родительской диаграммой. Каждая из подфункций дочерней диаграммы может быть далее детализирована путем аналогичной декомпозиции соответствующего ей функционального блока. Важно отметить, что в каждом случае декомпозиции функционального блока все интерфейсные дуги, входящие в данный блок, или исходящие из него фиксируются на дочерней диаграмме.

Проведем декомпозицию контекстной диаграммы, описав последовательность взаимодействия с научной статьёй:

1. написание статьи;
2. добавление тегов и названия;
3. публикация статьи;
4. написание комментария;
5. добавление в избранное.

В начале использования приложения сотрудники должны пройти регистрацию по коду, который создаст для них администратор. После этого они могут создавать чаты, писать и редактировать научные статьи, а также отслеживать свои задачи. В процессе работы они могут просматривать чужие статьи, добавлять их в избранное или оставлять комментарии к ним.

На рисунке 2.2 изображена декомпозиция диаграммы «Взаимодействие с научными статьями».

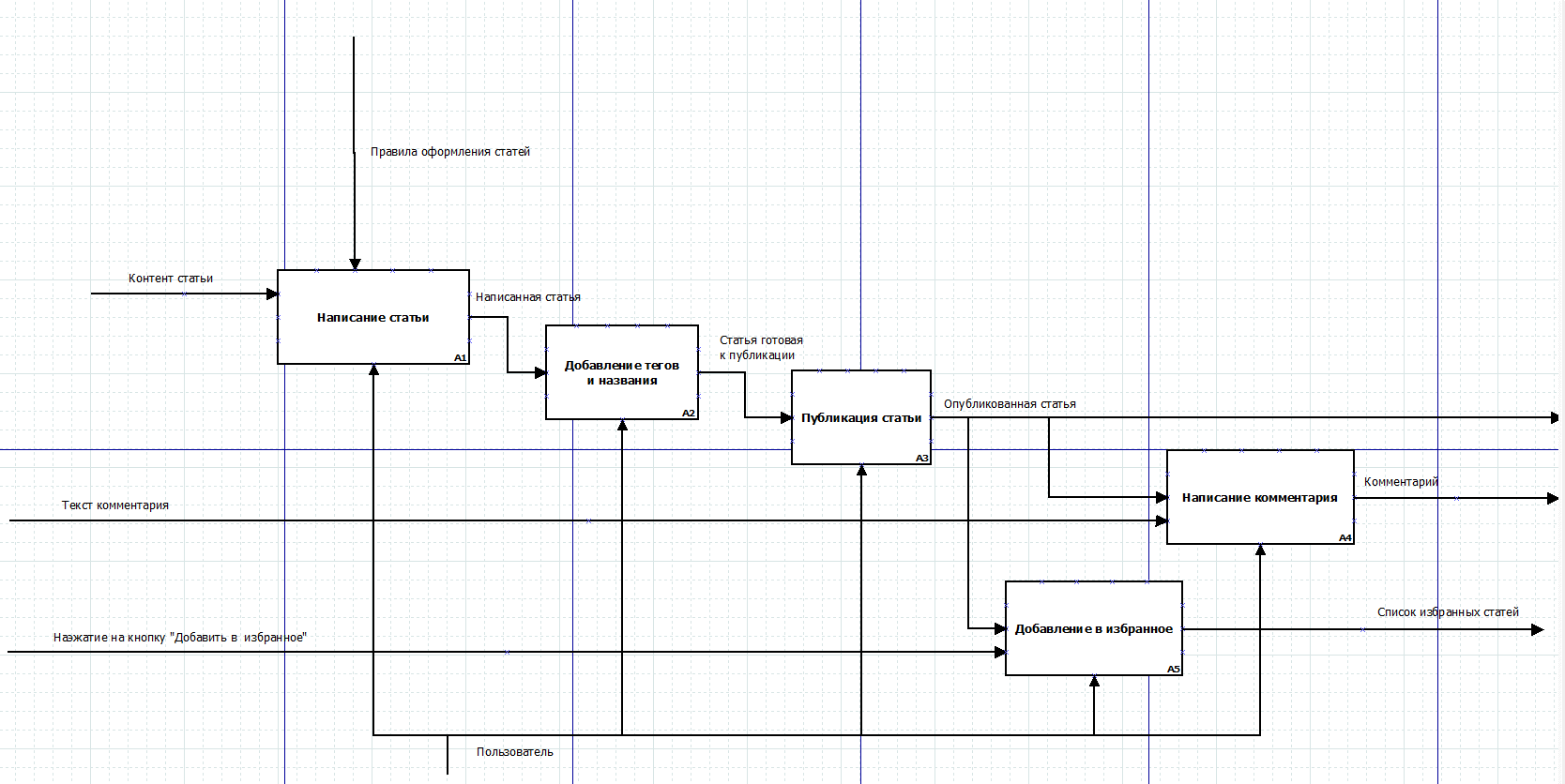


Рисунок 2.2 — Декомпозиция диаграммы «Взаимодействие с научными статьями»

Диаграммы DFD можно использовать как дополнение к диаграммам IDEF0 для описания документооборота и обработки информации. Рассмотрим работу «Взаимодействие с научными статьями».

Внешней сущностью будут пользователи, определим потоки данных, которыми они обмениваются с системой. Получим диаграмму, изображённую на рисунке 2.3.

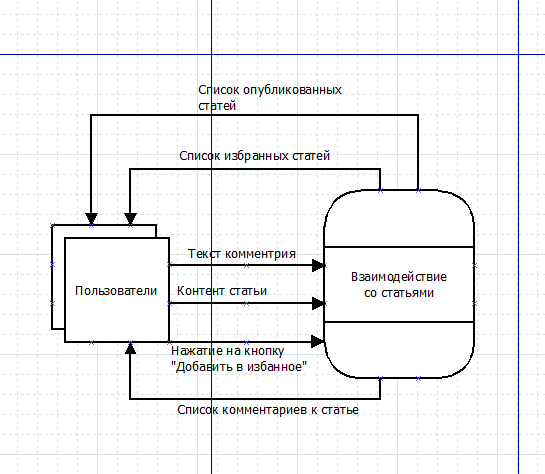


Рисунок 2.3 — DFD-диаграмма работы «Взаимодействие с научными статьями»

На рисунке 2.4 изображена DFD-декомпозиция диаграммы «Взаимодействие с научными статьями».

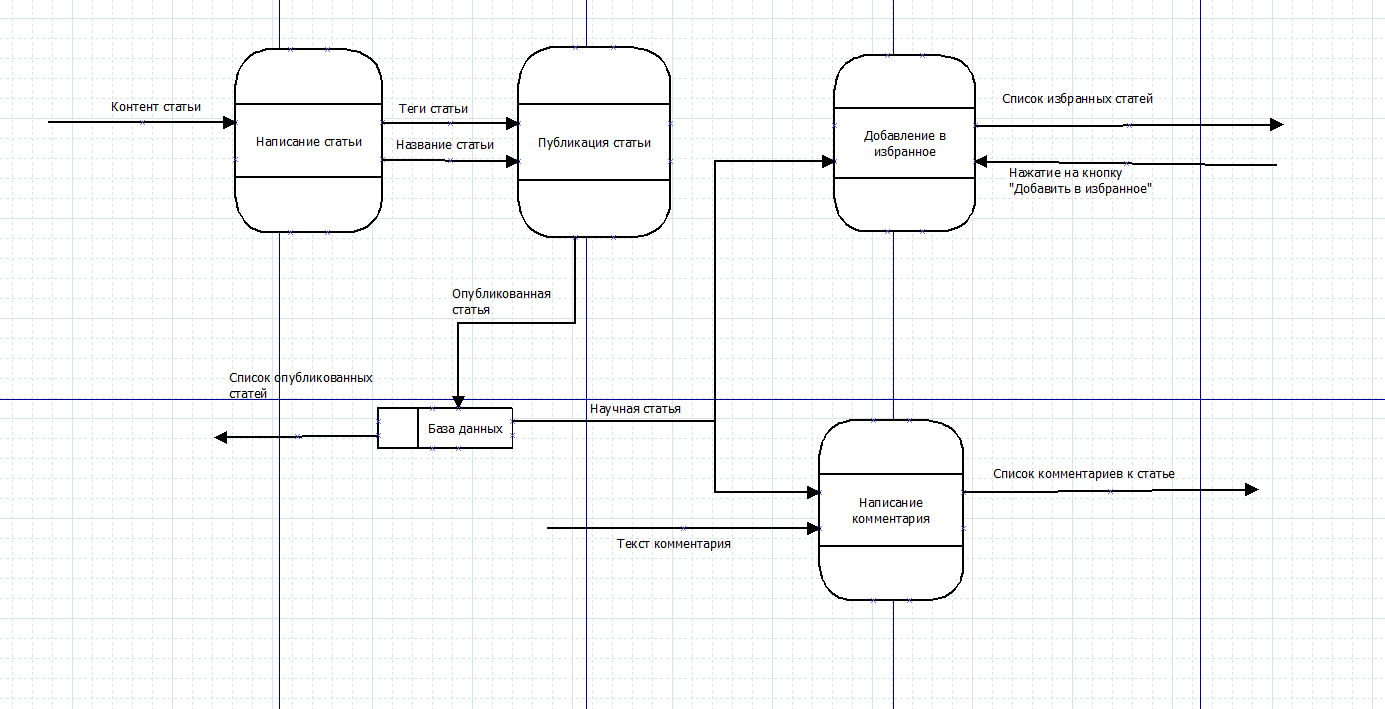


Рисунок 2.4 — DFD-декомпозиция диаграммы «диаграммы «Взаимодействие с научными статьями»

# 2.2 Проектирование системы с помощью UML-диаграмм

На рисунке 2.5 изображена диаграмма Use case (Вариантов использования), описывающая основные функциональные требования к программе. Вариант использования представляет собой типичное взаимодействие пользователя и проектируемой системы.

В простейшем случае вариант использования создается в процессе обсуждения с пользователями тех вещей, которые они хотели бы получить от системы. При этом каждой отдельной функции, которую они хотели бы реализовать, присваивается некоторое имя и записывается ее краткое текстовое описание.

Основными элементами диаграммы вариантов использования являются действующие лица, варианты использования и отношения между ними. Действующее лицо — это роль, которую пользователь играет по отношению к системе. Действующие лица могут играть различные роли по отношению к варианту использования. Они могут применять его результаты или сами непосредственно в нем участвовать.

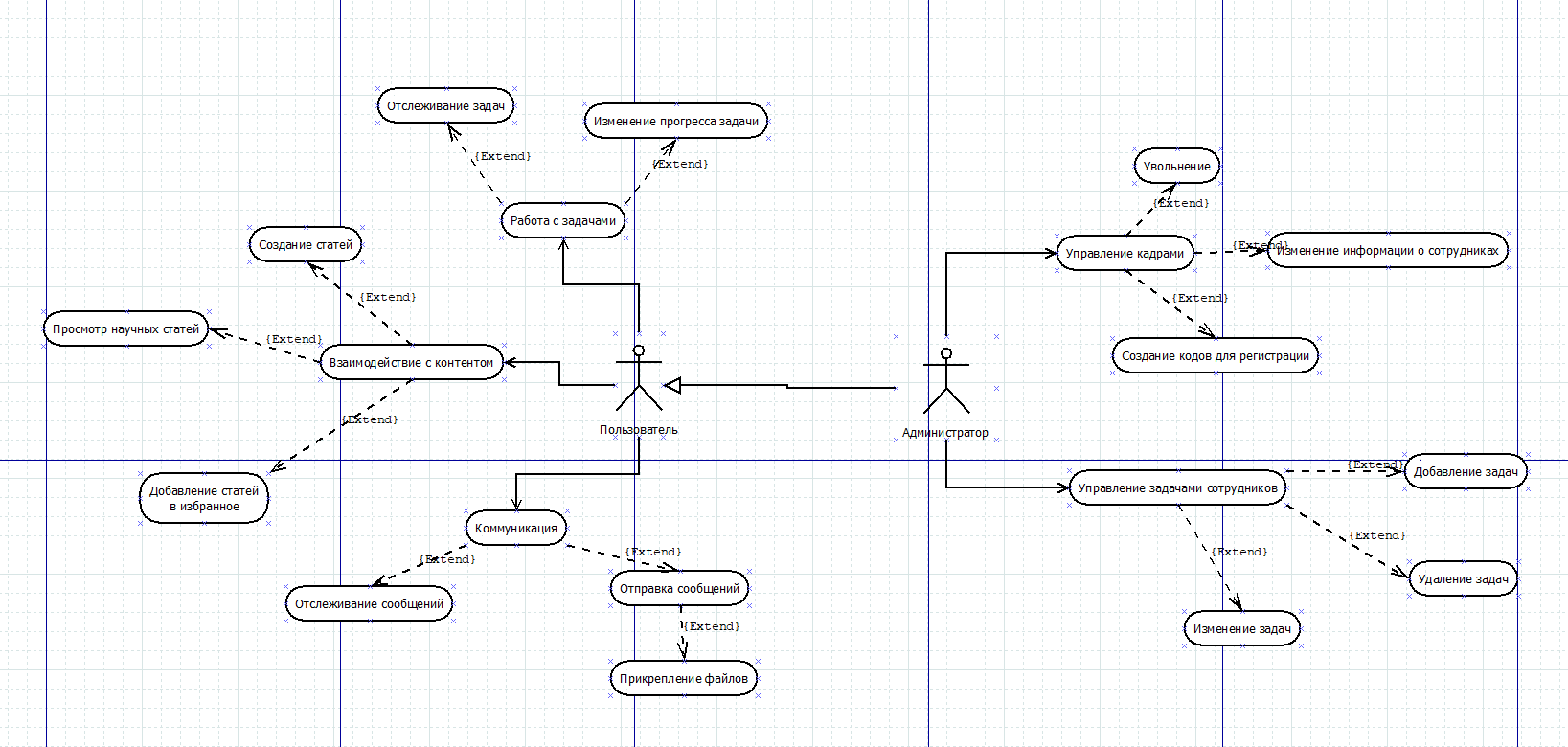


Рисунок 2.5 — Диаграмма Use case

На рисунке 2.6 изображена диаграмма деятельности, иллюстрирующая исходы вариантов тестирования функционала проектируемой ИС. Диаграмма деятельности принадлежит к логической модели. Поток управления запускается завершением действий (или действий) в системе. Поток может быть последовательным, параллельным или разветвленным, что обозначено формами, такими как дорожки, разветвления и соединения.

Диаграммы деятельности могут показать, как события в варианте использования связаны друг с другом или как набор вариантов использования координируется для представления бизнес-процесса.

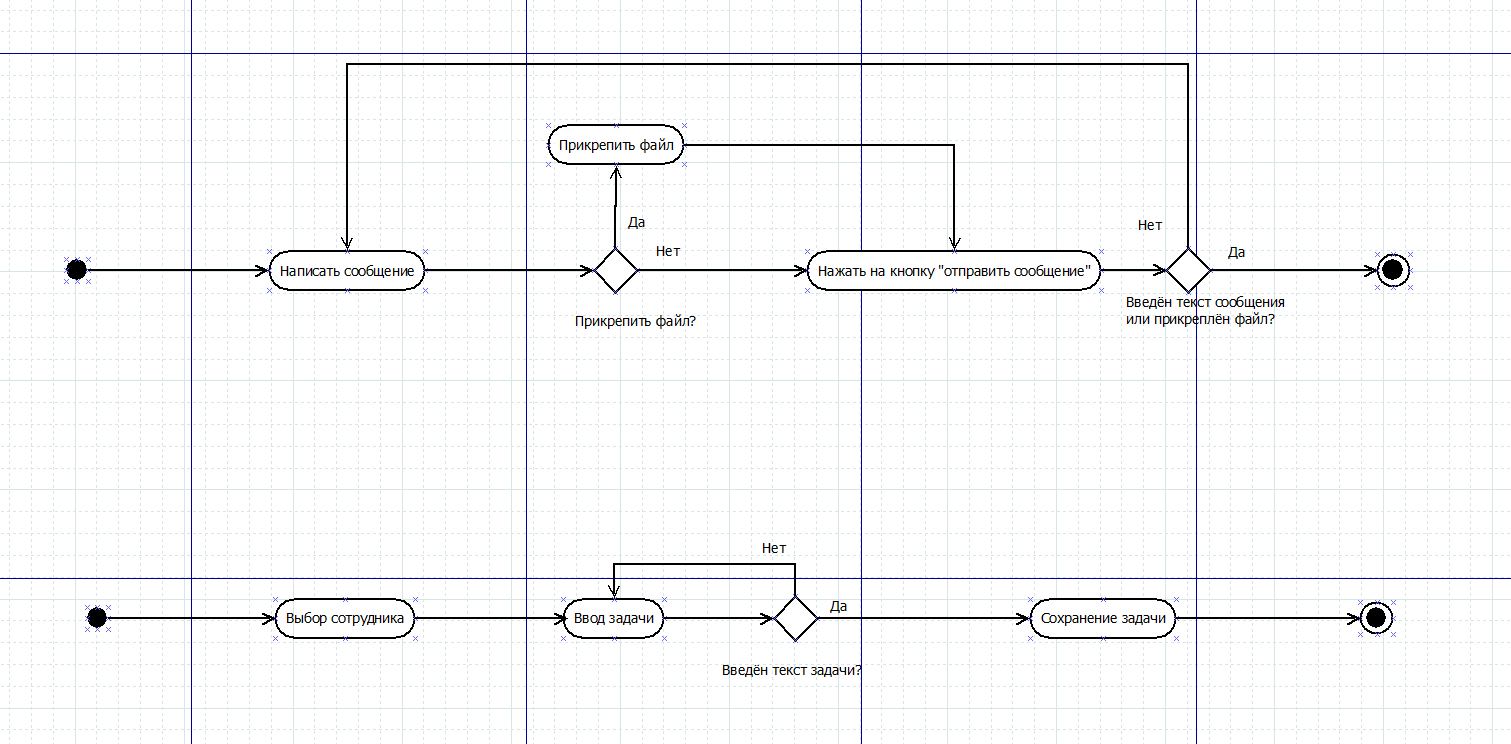


Рисунок 2.6 — Диаграмма деятельности

На рисунке 2.7 изображена диаграмма классов. Диаграммы классов являются центральным звеном методологии объектно-ориентированных анализа и проектирования.

Диаграмма классов показывает классы и их отношения, тем самым представляя логический аспект проекта. Отдельная диаграмма классов представляет определенный ракурс структуры классов. На стадии анализа диаграммы классов используются, чтобы выделить общие роли и обязанности сущностей, обеспечивающих требуемое поведение системы. На стадии проектирования диаграммы классов используются, чтобы передать структуру классов, формирующих архитектуру системы.

Каждый класс должен иметь имя; если имя слишком длинно, его можно сократить или увеличить сам значок на диаграмме. Имя каждого класса должно быть уникально в содержащем его проекте.

На диаграммах классов изображаются также атрибуты классов, операции и ограничения, которые накладываются на связи между объектами.

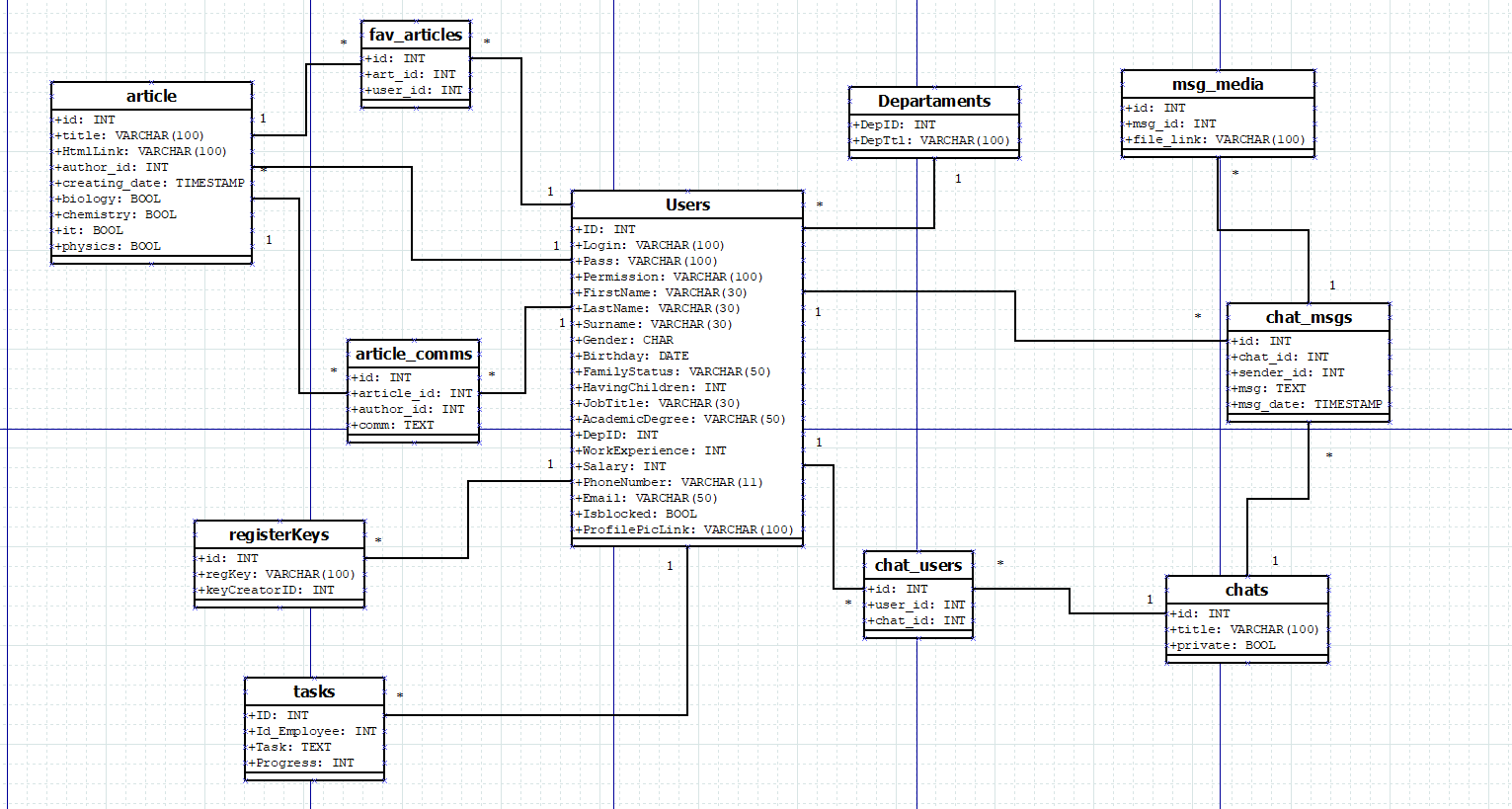


Рисунок 2.7 — Диаграмма классов

В таблицах 2.1-2.12 представлено описание классов и их атрибутов.

Таблица 2.1 - Класс Departaments

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Departaments | Класс, представляющий собой отдел лаборатории. |
| Атрибуты | DepID (INT): уникальный идентификатор отдела.  DepTtl (VARCHAR): название отдела. |
| Операции |  |
| Таблица 2.2 - Класс Users | |
| **Параметр** | **Значение** |
| Users | Класс, представляющий собой пользователя. |

Продолжение таблицы 2.2

|  |  |
| --- | --- |
| Атрибуты | ID (INT): Уникальный идентификатор пользователя.  Login (VARCHAR): Имя учётной записи пользователя.  Pass (VARCHAR): Пароль от учётной записи пользователя.  Permission (VARCHAR): Права доступа пользователя.  FirstName (VARCHAR): Имя пользователя.  LastName (VARCHAR): Фамилия пользователя.  Surname (VARCHAR): Отчество пользователя.  Gender (CHAR): Пол пользователя.  Birthday (DATE): День рождения пользователя.  FamilyStatus (VARCHAR): Семейное положение пользователя.  HavingChildren (INT): Количество детей пользователя.  JobTitle (VARCHAR): Должность пользователя.  AcademicDegree (VARCHAR): Учёная степень пользователя.  DepID (INT): Идентификатор отдела.  WorkExperience (INT): Опыт работы пользователя.  Salary (INT): Заработная плата пользоавтеля. |
| Продолжение таблицы 2.2 | |
| Атрибуты | PhoneNumber (VARCHAR) : Номер телефона пользователя.  Email (VARCHAR): Адрес электронной почты пользователя.  Isblocked (BOOL): Заблокирован пользователь или нет.  ProfilePicLink (VARCHAR): Ссылка на файл с фотографией профиля пользователя. |
| Операции |  |
| Таблица 2.3 - Класс article | |
| **Параметр** | **Значение** |
| article | Класс, представляющий научную статью |
| Атрибуты | id (INT): Уникальный идентификатор статьи.  title (VARCHAR): Название статьи.  HtmlLink (VARCHAR): Ссылка на файл с содержимым статьи.  author\_id (INT): Идентификатор автора статьи.  creating\_date (TIMESTAMP): Дата и время создания статьи.  biology (BOOL): Статья по биологии или нет.  chemistry (BOOL): Статья по химии или нет. |

Продолжение таблицы 2.3

|  |  |
| --- | --- |
| Атрибуты | it (BOOL): Статья по информационным технологиям или нет.  physics (BOOL): Статья по физике или нет. |
| Операции |  |
| Таблица 2.4 - Класс chats | |
| **Параметр** | **Значение** |
| chats | Класс, представляющий собой чат. |
| Атрибуты | id (INT): Уникальный идентификатор чата.  title (VARCHAR): Название чата.  private (BOOL): Личный это чат или групповой. |
| Операции |  |
| Таблица 2.5 - Класс registerKeys | |
| **Параметр** | **Значение** |
| Комментарий | Класс, представляющий собой ключи регистрации. |
| Атрибуты | id (INT): Уникальный идентификатор ключа регистрации.  regKey (VARCHAR): Ключ регистрации.  keyCreatorID (INT): Идентификатор создателя ключа |
| Операции |  |

|  |  |
| --- | --- |
| Таблица 2.6 - Класс tasks | |
| **Параметр** | **Значение** |
| tasks | Класс, представляющий собой класс задач пользователей. |
| Атрибуты | ID (INT): Уникальный идентификатор задачи.  Id\_Employee (integer): Уникальный идентификатор сотрудника.  Progress (INT): Прогресс выполнения задачи. |
| Операции |  |

|  |  |
| --- | --- |
| Таблица 2.7 - Класс chat\_msgs | |
| **Параметр** | **Значение** |
| chat\_msgs | Класс, представляющий собой сообщения в чате. |
| Атрибуты | id (INT): Уникальный идентификатор сообщения.  chat\_id (INT): Идентификатор чата.  sender\_id (INT): Идентификтор отправителя.  msg (TEXT): Текст сообщения.  msg\_date (TIMESTAMP): Дата и время отправки сообщения. |
| Операции |  |

|  |  |
| --- | --- |
| Таблица 2.8 - Класс msg\_media | |
| **Параметр** | **Значение** |
| msg\_media | Класс, представляющий собой файлы сообщения. |

Продолжение таблицы 2.8

|  |  |
| --- | --- |
| Атрибуты | id (INT): Уникальный идентификатор файла.  msg\_id (INT): Идентификатор сообщения.  file\_link (VARCHAR) : Ссылка на файл |
| Операции |  |
| Таблица 2.9 - Класс article\_comms | |
| **Параметр** | **Значение** |
| article\_comms | Класс, представляющий собой комментарии к статье. |
| Атрибуты | id (INT): Уникальный идентификатор комментария.  article\_id (INT): Идентификатор статьи.  author\_id (INT): Идентификатор автора комментария.  comm (TEXT): Текст комментария. |
| Операции |  |

|  |  |
| --- | --- |
| Таблица 2.10 - Класс fav\_articles | |
| **Параметр** | **Значение** |
| fav\_articles | Класс, представляющий собой избранные статьи пользователя. |
| Атрибуты | id (INT): Уникальный идентификатор записи об избранной статье.  art\_id (INT): Идентификатор статьи.  user\_id (INT): Идентификатор пользователя. |

|  |  |  |  |
| --- | --- | --- | --- |
| Продолжение таблицы 2.10   |  |  | | --- | --- | | Операции |  |   Таблица 2.11 - Класс chat\_users | |
| **Параметр** | **Значение** |
| chat\_users | Класс, представляющий собой связь пользователя и чата. |
| Атрибуты | id (INT): Идентификатор связи.  user\_id (INT): Идентификатор пользователя.  chat\_id (INT): Идентификатор чата. |
| Операции |  |

На рисунке 2.8 изображена диаграмма компонентов. Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ.

Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

* Визуализации общей структуры исходного кода программной системы.
* Спецификации исполнимого варианта программной системы.
* Обеспечения многократного использования отдельных фрагментов программного кода.
* Представления концептуальной и физической схем баз данных.

Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие — на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов.

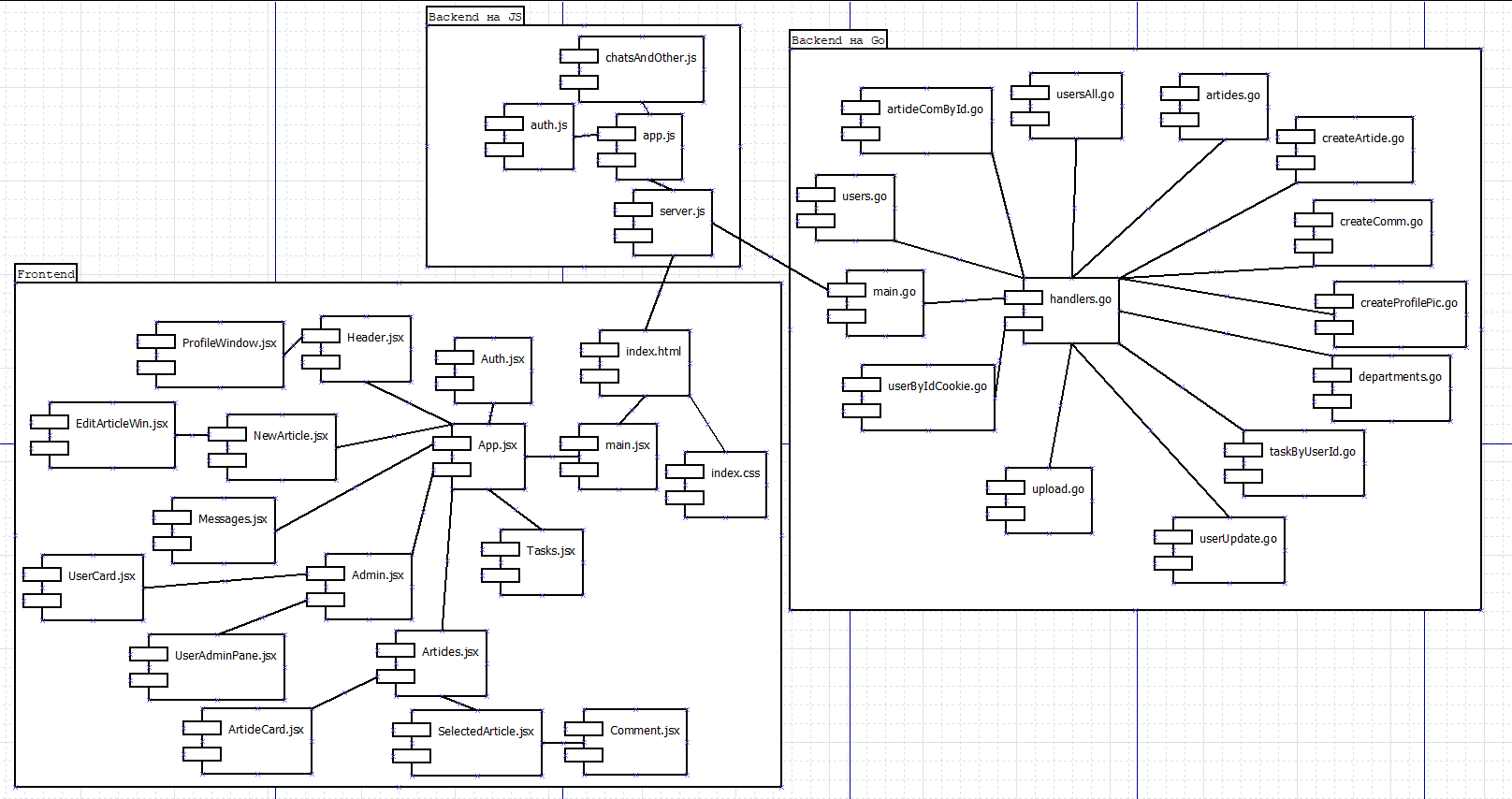


Рисунок 2.8 — Диаграмма компонентов

В таблице 2.13 представлено описание набора компонентов системного продукта по теме «Приложение для взаимодействия работников лаборатории».

Таблица 2.13 – Описание набора компонентов

|  |  |
| --- | --- |
| **Название** | **Описание** |
| Frontend | Компоненты клиентской части приложения |
| Backend на JS | Компоненты серверной части приложения на языке JavaScript |
| Backend на Go | Компоненты серверной части приложения на языке Go |

Продолжение таблицы 2.13

|  |  |
| --- | --- |
| app.js | Объединяет файлы серверной части на языке JavaScript |
| auth.js | Отвечает за авторизацию пользователя |
| chatsAndOther.js | Отвечает за коммуникацию |
| index.html | Страница сайта, которая отображается у пользователя и загружает файл main.jsx |
| index.css | Отвечает за визуальное оформление программы |
| main.jsx | Входная точка программы |
| App.jsx | Объединяет файлы клиентской части |
| Header.jsx | Шапка сайта |
| Auth.jsx | Обрабатывает авторизацию пользователя |
| EditArticleElWin.jsx | Окно сохранения статьи |
| ProfileWindow.jsx | Меню пользователя |
| Admin.jsx | Список сотрудников |
| Articles.jsx | Окно с научными статьями |
| Messages.jsx | Окно коммуникации |
| NewArticle.jsx | Окно создания статьи |
| Tasks.jsx | Окно задач пользователя |
| ArticleCard.jsx | Миниатюра статьи в окне статей |
| Comment.jsx | Комментарий к статье |
| SelectedArticle.jsx | Содержание вобранной статьи |
| UserAdminPane.jsx | Окно управления пользователем |
| UserCard.jsx | Миниатюра пользователя в окне сотрудников |
| main.go | Точка запуска сервиса на Go |
| handlers.go | Обрабатывает все запросы |

Продолжение таблицы 2.13

|  |  |
| --- | --- |
| articleComByld.go | Показывает комментарии по идентификатору статьи |
| articles.go | Запросы со статьями по идентификатору |
| articlesAll.go | Показывает все статьи |
| createArticle.go | Создаёт статью |
| createComm.go | Создаёт комментарий |
| createProfilePic.go | Создаёт аватар пользователя |
| departments.go | Создаёт отделы |
| projects.go | Создаёт проект |
| taskByUserId.go | Показывает задачи по идентификатору пользователя |
| tasks.go | Показывает все задачи |
| upload.go | Загружает файлы в редактор статей |
| userByIdCookie.go | Загружает информацию пользователей из Cookie |
| users.go | Удаляет пользователя |
| usersAll.go | Показывает всех пользователей |
| userUpdate.go | Изменяет информацию о пользователе |

# 2.3 Функциональные требования к системе

1. Общие сведения

* Наименование проекта: Разработка программного модуля для взаимодействия работников лаборатории
* Заказчик: Митрошенкова Е.А.
* Исполнитель: Демидов Д.А.

1. Требования к функциональным характеристикам
   1. Возможность регистрация и авторизация пользователей:

* возможность регистрации пользователей с проверкой пароля на соответствие требованиям:
  1. от 4 до 16 символов;
  2. запрещенные символы: \*, &, {, }, |, +;
  3. наличие заглавных букв и цифр;
* авторизация по логину и паролю;
* хранение данных о пользователях в базе данных.
  1. Возможность организации ролей пользователей:
* пользователь: просмотр, написание, комментирование и добавление в избранное научных статей, создание чатов, отправка сообщений, прикрепление файлов к сообщениям, просмотр и изменение прогресса задач, просмотр информации о сотрудниках;
* администратор: генерация кодов регистрации для новых пользователей, изменение данных о сотрудниках, добавление задач сотрудникам, увольнение сотрудников.
  1. Возможность управление данными:
* поиск и фильтрация статей;
* поиск чатов;
* поиск и фильтрация сотрудников;
  1. Возможность генерирования отчетности:
* генерация и экспорт данных о сотрудниках в формате Excel.

1. Нефункциональные требования
2. Требования к надежности:

* ограничение доступа к административным функциям;
* стабильность.
  1. Требования к составу и параметрам технических средств:
* подключение к интернету;
* операционная система: Windows 7 и выше, macOS, Linux, Android, iOS;
* минимальные требования: процессор 1 ГГц, 2 ГБ оперативной памяти, 300 МБ свободного пространства на диске;
  1. Требования к информационной и программной совместимости:
* совместимость с базой данных;
* интеграция с внешними приложениями (Excel);
* совместимость операционных систем.
  1. Требования к программной документации:
* техническое задание на разработку программного модуля;
* руководство пользователя.

# 2.4 Пользователи системы и их привилегии

Программный модуль для пользователей и администраторов системы взаимодействия с научными статьями предназначен для обеспечения функциональности, удобства использования и эффективности работы с научным контентом. Вот несколько ключевых задач и доступных привилегий, которые такой модуль мог бы выполнять и обеспечивать:

Для пользователей (сотрудников):

1. Пользователи могут писать научные статьи, публиковать их, просматривать чужие статьи, писать комментарии к статьям, добавлять статьи в избранное.
2. Возможность мониторинга поставленных задач и изменения их прогресса.
3. Возможность создавать и редактировать чаты, писать сообщения, прикреплять файлы к сообщениям, добавлять новых участников.
4. Возможность просматривать информацию о других сотрудниках, экспортировать данные в Excel.
5. Персонализация цветовой темы приложения.

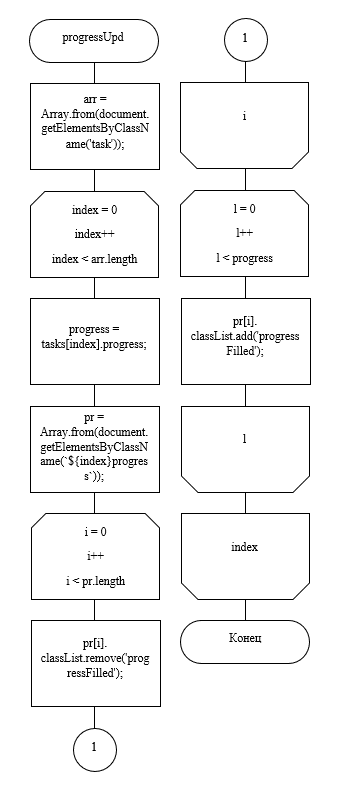
Для администраторов:

1. Возможность создания кодов регистрации для новых пользователей.
2. Изменение информации о зарегистрированных сотрудниках.
3. Увольнение сотрудников.

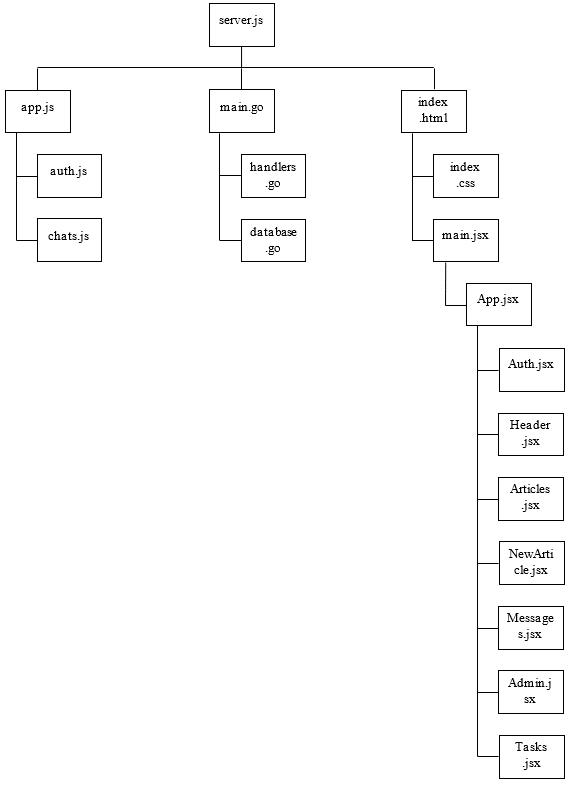
Таким образом, программный модуль предназначен для удобного и функционального взаимодействия пользователей с системой статей, обеспечения эффективного контроля задач и взаимодействия со статьями и пользователями, а также для ведения учёта информации о сотрудниках и назначения задач сотрудникам для администраторов.

# 3 Схемы алгоритма программы и процедуры

3.1 Схема алгоритма функции «progressUpd»



# 3.2 Схема программы



# 3.2 Схема данных

Структура таблицы определяется составом столбцов, типом данных и размерами столбцов, ключами таблицы.

К базовым понятиями модели БД «сущность – связь» относятся: сущности, связи между ними и их атрибуты (свойства).

Сущность – любой конкретный или абстрактный объект в рассматриваемой предметной области. Сущности – это базовые типы информации, которые хранятся в БД (в реляционной БД каждой сущности назначается таблица). К сущностям могут относиться: пользователи, статьи, отделы и т.д. Экземпляр сущности и тип сущности — это разные понятия. Понятие тип сущности относится к набору однородных личностей, предметов или событий, выступающих как целое (например, статья, задача и т.д.). Экземпляр сущности относится, например, к конкретной личности в наборе. Типом сущности может быть сотрудник, а экземпляром –Александр, Николай и т. д.

Атрибут – это свойство сущности в предметной области. Его наименование должно быть уникальным для конкретного типа сущности. Например, для сущности сотрудник могут быть использованы следующие атрибуты: фамилия, имя, отчество, дата рождения, зарплата, отдел, учёная степерь и т.д. В реляционной БД атрибуты хранятся в полях таблиц.

В таблицах 3.1 – 3.12 описаны все сущности, необходимые для создания базы данных.

Таблица 3.1 – Сущность Users

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| ID | INT | - |
| Login | VARCHAR | 100 |
| Pass | VARCHAR | 100 |
| Permission | VARCHAR | 100 |
| FirstName | VARCHAR | 30 |
| LastName | VARCHAR | 30 |
| Surname | VARCHAR | 30 |
| Gender | CHAR | 1 |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Birthday | DATE | - |
| FamilyStatus | VARCHAR | 100 |
| HavingChildren | INT | - |
| JobTitle | VARCHAR | 30 |
| AcademicDegree | VARCHAR | 50 |
| DepID | INT | - |
| WorkExperience | INT | - |
| Salary | INT | - |
| PhoneNumber | VARCHAR | 11 |
| Email | VARCHAR | 50 |
| Isblocked | TINYINT | 1 |
| ProfilePicLink | VARCHAR | 100 |

Таблица 3.2 – Сущность tasks

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| ID | INT | - |
| Id\_Employee | INT | - |
| Task | TEXT | - |
| Progress | INT | - |

Таблица 3.3 – Сущность Departaments

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| DepID | INT | - |
| DepTtl | VARCHAR | 100 |
| ProjID | INT | - |

Таблица 3.4 – Сущность Chats

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| Id | INT | - |
| Title | VARCHAR | 100 |
| Private | TINYINT | 1 |

Таблица 3.5 – Сущность chat\_users

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| Id | INT | - |
| User\_id | INT | - |
| Chat\_id | INT | - |

Таблица 3.6 – Сущность chat\_msgs

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| Id | INT | - |
| Chat\_id | INT | - |
| Sender\_id | INT | - |
| Msg | TEXT | - |
| Msg\_date | TIMESTAMP | - |

Таблица 3.7 – Сущность msg\_media

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| Id | INT | - |
| Msg\_id | INT | - |
| File\_link | VARCHAR | 100 |

Таблица 3.8 – Сущность article

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| Id | INT | - |

Продолжение таблицы 3.8

|  |  |  |
| --- | --- | --- |
| Title | VARCHAR | 100 |
| HtmlLink | VARCHAR | 100 |
| Completed | TINYINT | 1 |
| Author\_id | INT | - |
| Creating\_date | TIMESTAMP | - |

Таблица 3.9 – Сущность fav\_articles

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| Id | INT | - |
| Art\_id | INT | - |
| User\_id | INT | - |

Таблица 3.10 – Сущность article\_comms

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| Id | INT | - |
| Article\_id | INT | - |
| Author\_id | INT | - |
| Comm | TEXT | - |

Таблица 3.11 – Сущность registerKeys

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Тип данных** | **Ограничение** |
| Id | INT | - |
| keyCreator | INT | - |
| regKey | VARCHAR | 100 |

На рисунке 3.1 представлена диаграмма IDEF1X.

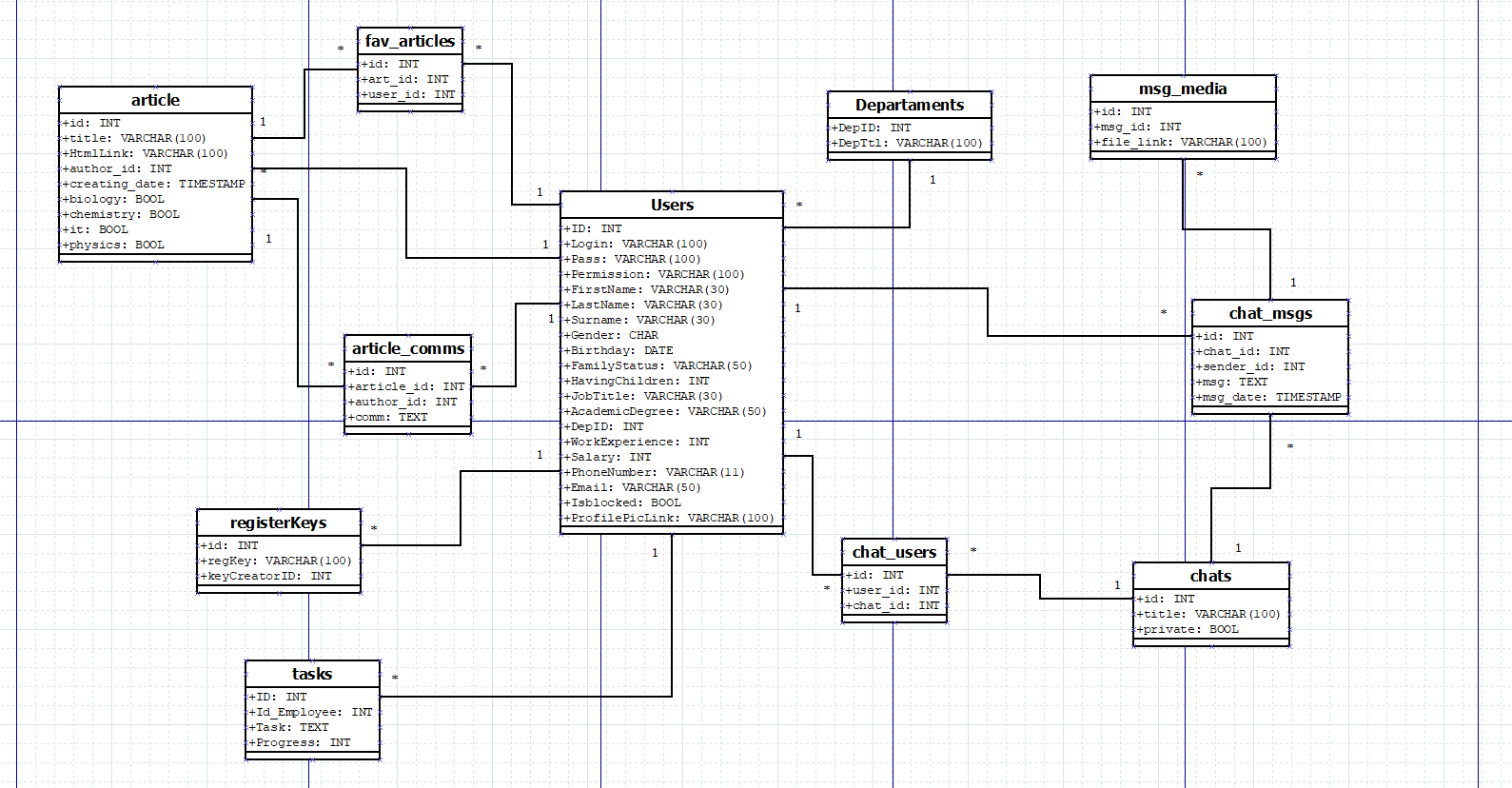


Рисунок 3.1 − Диаграмма IDEF1X

# 3.3 Тестирование программы

Тестирование — это набор процедур и действий, предназначенных для демонстрации правильности работы программы в заданных режимах и внешних условиях. Цель тестирования — выявить наличие ошибок или убедительно продемонстрировать их отсутствие.

Таблица 3.13 – Негативный тест добавления информации в базу данных

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Входим в учётную запись администратора. | * Форма открыта; * Доступен каталог книг; * Видно поле поиска книг; * Видна шапка с различными функциями. | Пройдено |
| 2. Нажимаем на кнопку «База данных» в верхнем меню.  Выбираем таблицу «Автор» в выпадающем списке. | * Форма базы данных открылась; * Видна информация об авторе: * ID * FIO | Пройдено |
| 3. Добавляем информацию об авторе:   * Нажимаем на кнопку «Добавить».   Вводим данные:   * ID: w * Автор: Author | * Всплывающее окно ввода информации открылось. | Пройдено |
| 4. Создаем автора:   * Нажимаем на кнопку «ОК». | * Всплывает уведомление о неправильно введенном ID. | Пройдено |
|  | | |

Таблица 3.14 – Позитивный тест добавления информации в базу данных

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Входим в учётную запись администратора. | * Форма открыта; * Доступен каталог книг; * Видно поле поиска книг; * Видна шапка с различными функциями. | Пройдено |
| 2. Нажимаем на кнопку «База данных» в верхнем меню.  Выбираем таблицу «Автор» в выпадающем списке. | * Форма базы данных открылась; * Видна информация об авторе: * ID * FIO | Пройдено |
| 3. Добавляем информацию об авторе:   * Нажимаем на кнопку «Добавить».   Вводим данные:   * ID: 14 * Автор: Author 14 | * Всплывающее окно ввода информации открылось. | Пройдено |
| 4. Создаем автора:   * Нажимаем на кнопку «ОК». | * Автор добавляется в базу данных. | Пройдено |
| Изображение выглядит как текст, Шрифт, число, снимок экрана  Автоматически созданное описание | | |

Таблица 3.15 –Негативный тест входа

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Открываем форму входа. | * Форма открыта; * Доступны 2 поля ввода: * Логин * Пароль * На полях видны подсказки. | Пройдено |
| 2. Вводим данные:   * Логин: User * Пароль: 123421 | * Поля заполнены. | Пройдено |
| 3. Входим:   * Нажимаем на кнопку «Войти». | * Выводится уведомление о неправильно введенных данных. | Пройдено |
|  | | |

Таблица 3.16 – Позитивный тест входа

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Открываем форму входа. | * Форма открыта; * Доступны 2 поля ввода: * Логин * Пароль * На полях видны подсказки. | Пройдено |
| 2. Вводим данные:   * Логин: login * Пароль: Parol123 | * Поля заполнены. | Пройдено |

Продолжение таблицы 3.16

|  |  |  |
| --- | --- | --- |
| 3. Входим:   * Нажимаем на кнопку «Войти». | * Вход в аккаунт произведен успешно, об этом говорит переход на главную страницу; * В левом верхнем углу шапки появилась приветственная надпись и кнопка выхода из аккаунта. | Пройдено |
|  | | |

Таблица 3.17 – Позитивный тест поиска книги в каталоге

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Открываем главную страницу. | * Форма открыта; * Доступен каталог книг; * Видно поле поиска книг; * Видна шапка с различными функциями. | Пройдено |
| 2. Осуществляем поиск книги:   * Нажимаем на поле поиска и вводим «1». | * Подходящие книги показываются под полем поиска. | Пройдено |

Продолжение таблицы 3.17

|  |
| --- |
|  |

Таблица 3.18 – Негативный тест регистрации

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Открываем форму входа:   * Кликаем на кнопку «Регистрация». | * Форма открыта; * Доступны 4 поля ввода: * ФИО * Логин * Пароль * Повторите пароль * На полях видны подсказки. | Пройдено |
| 2. Вводим данные   * ФИО: Reader Ten * Логин: qwerty1234 * Пароль: Reader4329\* * Повторите пароль: Reader4329\* | * Поля заполнены. | Пройдено |
| 3. Регистрируемся:   * Нажимаем на кнопку «Зарегистрироваться». | * Выводится уведомление о неправильно введенном пароле. | Пройдено |

Продолжение таблицы 3.18

|  |
| --- |
|  |

Таблица 3.19 – Позитивный тест регистрации

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Открываем форму входа:   * Кликаем на кнопку «Регистрация». | * Форма открыта; * Доступны 4 поля ввода: * ФИО * Логин * Пароль * Повторите пароль * На полях видны подсказки. | Пройдено |
| 2. Вводим данные   * ФИО: Reader Ten * Логин: qwerty1234 * Пароль: Reader4329 * Повторите пароль: Reader4329 | * Поля заполнены; | Пройдено |

Продолжение таблицы 3.19

|  |  |  |
| --- | --- | --- |
| 3. Регистрируемся:   * Нажимаем на кнопку «Зарегистрироваться». | * Регистрация аккаунта произведена успешно, об этом говорит переход на главную страницу и всплывающее уведомление; * В левом верхнем углу шапки появилась приветственная надпись и кнопка «Выйти из учётной записи». | Пройдено |
|  | | |

Таблица 3.20 – Позитивный тест изменения информации в базе данных

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Входим в учётную запись администратора. | * Форма открыта; * Доступен каталог книг; * Видно поле поиска книг; * Видна шапка с различными функциями. | Пройдено |
| 2. Нажимаем на кнопку «База данных» в верхнем меню.  Выбираем таблицу «Автор» в выпадающем списке. | * Форма базы данных открылась; * Видна информация об авторе: * ID * FIO | Пройдено |

Продолжение таблицы 3.20

|  |  |  |
| --- | --- | --- |
| 3. Изменяем информацию об авторе:   * Нажимаем на кнопку «Изменить».   Вводим данные:   * ID: 11 * Автор: А.С. Пушкин | * Видим измененные данные; * Данные изменяются в базе данных. | Пройдено |
|  | | |

Таблица 3.21 – Негативный тест изменения информации в базе данных

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Входим в учётную запись администратора. | * Форма открыта; * Доступен каталог книг; * Видно поле поиска книг; * Видна шапка с различными функциями. | Пройдено |
| 2. Нажимаем на кнопку «База данных» в верхнем меню.  Выбираем таблицу «Автор» в выпадающем списке. | * Форма базы данных открылась; * Видна информация об авторе: * ID * FIO | Пройдено |

Продолжение таблицы 3.21

|  |  |  |
| --- | --- | --- |
| 3. Изменяем информацию об авторе:   * Нажимаем на кнопку «Изменить».   Вводим данные:   * ID: 10 * Автор: А.С. Пушкин | * Выводится предупреждение о не уникальном ID. | Пройдено |
|  | | |

Таблица 3.22 – Позитивный тест функции расчёта

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Результат** |
| 1. Входим в учётную запись администратора. | * Форма открыта; * Доступен каталог книг; * Видно поле поиска книг; * Видна шапка с различными функциями. | Пройдено |
| 2. Нажимаем на кнопку «Отчёты» в верхнем меню. | * Форма отчётов открылась. | Пройдено |
| 3. Выбираем диапазон дат:   * От: 15.01.2020 * До: 15.01.2025   Нажимаем на кнопку «Вычислить» | * Выводится информация о закупках в промежуток этого времени. | Пройдено |

Продолжение таблицы 3.22

|  |
| --- |
|  |

# 3.4 Руководство пользователя

* + - 1. Назначение программы:

Программный модуль предназначен для автоматизации работы сотрудников лаборатории. Пользователи приложения, такие как сотрудники и администраторы, могут взаимодействовать друг с другом, создавать чаты, писать научные статьи и отслеживать свои задачи, смотреть информацию о других сотрудниках, сортировать и выводить данные о сотрудниках в Excel. Администраторы имеют расширенные возможности, включая создание кодов для регистрации новых пользователей, управление данными сотрудников, добавление задач сотрудникам. Приложение также позволяет изменять цветовую тему для удобства работы. Клиентская часть разработана на ReactJS, серверная часть реализована в виде двух микросервисов: один на ExpressJS, второй на Go с использованием фреймворка Gin. В качестве СУБД используется MySQL Workbench.

* + - 1. Требования к выполнению программы:
* поддерживаемая платформа: любая платформа, которая имеет доступ в интернет и где установлен браузер;
* минимальные требования: 1 ГГц, 2 ГБ оперативной памяти, 300 МБ свободного пространства на диске;
* требования к запуску программы: для получения доступа к программен, необходимо запустить браузер и ввести в поисковой строке адрес программы.
  + - 1. Входные данные:
    - данные регистрируемых пользователей;
    - контент научных статей;
    - текст сообщений.

Данные для анализа:

* + - информация о сотрудниках;
    - контент научных статей.
      1. Выполнение программы:
* запустите браузер;
* введите в поисковой строке адрес программы;
* войдите в систему, используя свои учётные данные;
* работайте с программой.

1. Выполнение задач.

Для сотрудников:

* отслеживайте список назначенных задач;
* пишите и публикуйте научные статьи;
* просматривайте, комментируйте и добавляйте в избранное чужие научные статьи;
* создавайте чаты с другими сотрудниками;
* пишите сообщения в чатах и прикрепляйте к ним изображения;
* персонализируйте интерфейс программы при помощи выбора цветовой темы;
* просматривайте данные о других сотрудниках.

Для администраторов:

* изменяйте данные о сотрудниках;
* добавляйте задачи сотрудникам;
* cоздавайте коды для регистрации новых пользователей.

1. Завершение работы.

Закройте вкладку с программой.

1. Сообщения:

Уведомления об ошибках, например, при попытке добавления в чат сотрудника, который уже в нём находится, при пустом содержимом текста задачи при добавлении, при незаполненных данных при регистарции, при ошибках во время авторизации.

1. Выходные данные:

Научные статьи, написанные сотрудниками, отчёты с информацией о выбранных сотрудниках в Excel.

# ЗАКЛЮЧЕНИЕ

В данной работе была спроектирована и реализована база данных для автоматизации процессов лаборатории, обеспечивающая удобное взаимодействие сотрудников научной лаборатории. В рамках проекта были решены следующие задачи:

* составлены диаграммы IDEF0 и DFD для анализа предметной области и выделения основных информационных потоков данных;
* составлена диаграмма IDEF1X с описанием структуры таблиц БД;
* составлена диаграмма Use case, описывающая основные функциональные требования к программе;
* составлена диаграмма деятельности, иллюстрирующая исходы вариантов тестирования функционала проектируемой ИС;
* составлена диаграмма классов;
* составлена диаграмма компонентов;
* сформулирована спецификация на разработку программного продукта в виде фрагмента технического задания, в котором описываются разделы:
  1. назначение программы;
  2. требования к функциональным характеристикам;
  3. требования к надежности;
  4. требования к составу и параметрам технических средств;
  5. требования к информационной и программной совместимости;
  6. требования к программной документации;
* заполнена база данных в соответствии с семантикой её полей;
* создана главная форма приложения с навигацией по разделам, формы регистрации и авторизации пользователей, формы, соответствующие таблицам базы данных, с возможностью просмотра, добавления, удаления, обновления, поиска, фильтрации записей в таблицах базы данных. Ограничен доступ для разных ролей пользователей к формам. Создана как минимум одну форму для выполнения вычисляемой функции;
* произвести тестирование ИС и отладить её.

Полученная база данных соответствует всем требованиям целостности, удобна в использовании и готова к дальнейшему развитию. Система пригодна для внедрения в лабораторные учреждения, а также может быть масштабирована и адаптирована под другие научные и производственные организации.

Разработка базы данных позволяет более эффективно управлять процессами лаборатории, минимизируя риски ошибок и обеспечивая высокую производительность при работе с данными. Итоговое приложение предоставляет сотрудникам интуитивно понятный интерфейс для общения, выполнения задач и работы с научными статьями, а администраторам – удобный инструмент для управления данными и задачами сотрудников.

ПРИЛОЖЕНИЕ А

Листинг модуля server.js

const express = require('express');

const { createProxyMiddleware } = require('http-proxy-middleware');

const app = express();

const clientApp = express();

const cors = require('cors');

const path = require('path');

const corsOptions = {

origin: ['http://localhost:5173', 'http://localhost:5500', 'http://localhost:3000','http://localhost:3001', 'http://localhost:3002'], // Массив разрешенных источников

credentials: true,

};

app.use(cors(corsOptions));

clientApp.use(cors(corsOptions));

clientApp.use(express.static('../frontend/dist'));

app.use((req, res, next) => {

res.header('Access-Control-Allow-Origin', req.headers.origin || corsOptions.origin[0]);

res.header('Access-Control-Allow-Credentials', 'true');

res.header('Access-Control-Allow-Methods', 'GET,POST,PUT,DELETE,OPTIONS');

res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, Authorization');

next();

});

const JS\_SERVICE\_URL = 'http://localhost:3001'; // Микросервис на JS

const GO\_SERVICE\_URL = 'http://localhost:3002'; // Микросервис на Go

app.use('/js-service', createProxyMiddleware({

target: JS\_SERVICE\_URL,

changeOrigin: true,

pathRewrite: {

'^/js-service': '',

},

}));

app.use('/rest-api-service', createProxyMiddleware({

target: GO\_SERVICE\_URL,

changeOrigin: true,

pathRewrite: {

'^/rest-api-service': '',

},

}));

const PORT = 3000;

app.listen(PORT, () => {

console.log(`API Gateway запущен на порту ${PORT}`);

});

clientApp.get('\*', (req, res) => {

res.sendFile(path.join(\_\_dirname, 'dist', 'index.html'));

});

Листинг модуля app.js

const express = require('express');

const cookieParser = require('cookie-parser');

const cors = require('cors');

const authApp = require('./js/auth')

const sanyaApp = require('./js/chatsAndOther')

app = express();

app.use(express.json());

app.use(cookieParser());

app.use('/auth', authApp);

app.use('/sanya', sanyaApp);

const PORT = 3001;

const start = async () => {

try {

app.listen(PORT, () => {

console.log(`Node.js service is running on http://localhost:${PORT}`);

});

} catch (e) {

console.log(e);

}

}

start();

Листинг модуля chatsAndOther.js

const cors = require('cors');

const express = require('express');

const mysql = require("mysql");

const cookieParser = require('cookie-parser');

const sanyaApp = express();

sanyaApp.use(express.json());

const db = mysql.createConnection({

host: "localhost",

user: "root",

password: "spiritbinge",

database: "lab",

});

sanyaApp.get("/chats", (req, res) => {

const user\_id = req.cookies.userid;

console.log(user\_id);

const getChats = `

SELECT

c.id AS chat\_id,

c.title,

c.author\_id,

c.private,

GROUP\_CONCAT(DISTINCT CONCAT(u2.FirstName) SEPARATOR ', ') AS participants

FROM

chats c

LEFT JOIN chat\_users cu1 ON c.id = cu1.chat\_id

LEFT JOIN chat\_users cu2 ON c.id = cu2.chat\_id

LEFT JOIN users u2 ON cu2.user\_id = u2.id

WHERE

cu1.user\_id = ?

GROUP BY

c.id, c.title;

`

db.query(getChats, [user\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

res.status(200).json(rs);

}

});

});

sanyaApp.get("/chatmsgs/:chatid", (req, res) => {

const chatid = req.params.chatid;

let user\_id = req.cookies.userid;

console.log(req.params);

console.log(chatid + " " + user\_id);

const inChat = `

SELECT \* FROM chat\_users WHERE chat\_id = ? AND user\_id = ?;

`

db.query(inChat, [chatid, user\_id], (err, rsIn) => {

console.log(rsIn);

if (err) {

console.log(err);

} else if (rsIn.length >= 1) {

const MsgView = `

SELECT

cm.id,

cm.sender\_id,

cm.msg,

DATE\_FORMAT(cm.msg\_date, '%Y-%m-%d %H:%i:%s') AS msg\_date,

cm.pinned,

u.FirstName,

u.LastName,

u.Surname,

u.ProfilePicLink,

GROUP\_CONCAT(mm.file\_link) AS files

FROM

chat\_msgs AS cm

JOIN

Users AS u ON cm.sender\_id = u.ID

LEFT JOIN

msg\_media AS mm ON cm.id = mm.msg\_id

WHERE

cm.chat\_id = ?

GROUP BY

cm.id, cm.sender\_id, cm.msg, cm.msg\_date, cm.pinned, u.FirstName, u.LastName, u.Surname, u.ProfilePicLink

ORDER BY

cm.msg\_date ASC;

`;

db.query(MsgView, [chatid], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

res.status(200).json({rs, user\_id})

}

});

} else {

res.status(401).json({message: "User is not in this chat!"});

}

});

});

sanyaApp.get("/chatusers/:chat\_id", (req, res) => {

const chat\_id = req.params.chat\_id;

console.log(chat\_id);

const ChatUserView = `

SELECT

cu.chat\_id,

c.title AS chat\_title,

u.ID AS user\_id,

u.FirstName,

u.LastName,

u.Surname,

u.ProfilePicLink

FROM

chat\_users cu

JOIN

Users u ON cu.user\_id = u.ID

JOIN

chats c ON cu.chat\_id = c.id

WHERE

cu.chat\_id = ?;

`;

db.query(ChatUserView, [chat\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

res.status(200).json(rs)

}

});

});

sanyaApp.post("/chatcreate", (req, res) => {

const{title} = req.body;

console.log(req.cookies);

const ID = req.cookies.userid;

console.log(title + " " + ID);

const ChatCreate = `

INSERT INTO chats (title, author\_id)

VALUES

(?, ?);

`;

db.query(ChatCreate, [title, ID], (err, rsChat) => {

console.log(rsChat);

if (err) {

console.log(err);

} else {

const chatID = rsChat.insertId;

const UserChat = `

INSERT INTO chat\_users (user\_id, chat\_id, private)

VALUES

(?, ?, false);

`;

db.query(UserChat, [ID, chatID], (err, rsUserChat) => {

console.log(rsUserChat);

if (err) {

console.log(err);

} else {

console.log("New Chat Created!");

res.status(200).json(rsUserChat);

}

});

}

});

});

sanyaApp.post("/privatechat", (req, res) => {

const ID = req.cookies.userid;

const{user\_id} = req.body;

console.log(ID + " " + user\_id);

const chatCreated = `

SELECT c.id,

cu1.user\_id,

cu2.user\_id

FROM chats c

JOIN chat\_users cu1 ON cu1.chat\_id = c.id

JOIN chat\_users cu2 ON cu2.chat\_id = c.id

WHERE c.private = true

AND cu1.user\_id = ?

AND cu2.user\_id = ?

GROUP BY c.id

`

db.query(chatCreated, [ID, user\_id], (err, rsChat) => {

console.log(rsChat);

if (err) {

console.log(err);

} else if (rsChat.length >= 1) {

console.log(rsChat);

res.status(200).json(rsChat);

} else {

const createChat = `

INSERT INTO chats (private) VALUES (true);

`

db.query(createChat, [], (err, rsCreate) => {

console.log(rsCreate);

if (err) {

console.log(err);

} else {

const chat\_id = rsCreate.insertId;

const addUsers = `

INSERT INTO chat\_users (user\_id, chat\_id) VALUES (?, ?), (?, ?);

`

db.query(addUsers, [ID, chat\_id, user\_id, chat\_id], (err, rsAdd) => {

console.log(rsAdd);

if (err) {

console.log(err);

} else {

console.log("New private chat created!");

res.status(200).json({rsAdd, rsCreate});

}

});

}

});

}

});

});

sanyaApp.post("/chatadduser", (req, res) => {

const{chat\_id, self} = req.body;

let user\_id;

if (self) {

user\_id = req.cookies.userid;

} else {

user\_id = req.body.user\_id;

};

console.log(user\_id + " " + chat\_id);

const inChat = `

SELECT \* FROM chat\_users WHERE user\_id = ? AND chat\_id = ?;

`

db.query(inChat, [user\_id, chat\_id], (err, rsIn) => {

console.log(rsIn);

if (err) {

console.log(err);

} else if (rsIn.length >= 1) {

res.status(409).json({messege: "User already is in chat!"});

} else {

const addUser = `

INSERT INTO chat\_users (user\_id, chat\_id)

VALUES

(?, ?);

`

db.query(addUser, [user\_id, chat\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

console.log("New user added to chat!");

res.status(200).json(rs)

}

});

}

});

});

sanyaApp.delete("/chatdeluser", (req, res) => {

const{chat\_id, user\_id} = req.body;

console.log(user\_id + " " + chat\_id);

const inChat = `

SELECT \* FROM chat\_users WHERE user\_id = ? AND chat\_id = ?;

`

db.query(inChat, [user\_id, chat\_id], (err, rsIn) => {

console.log(rsIn);

if (err) {

console.log(err);

} else if (rsIn.length <= 0) {

res.status(409).json({messege: "User not in chat!"});

} else {

const delUser = `

DELETE FROM chat\_users WHERE user\_id = ? AND chat\_id = ?;

`

db.query(delUser, [user\_id, chat\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

console.log("User deleted from chat!");

res.status(200).json(rs)

}

});

}

});

});

sanyaApp.delete("/chatdelete", (req, res) => {

const{chat\_id} = req.body;

const user\_id = req.cookies.userid;

console.log(user\_id + " " + chat\_id);

const inChat = `

SELECT \* FROM chat\_users WHERE user\_id = ? AND chat\_id = ?;

`

db.query(inChat, [user\_id, chat\_id], (err, rsIn) => {

console.log(rsIn);

if (err) {

console.log(err);

} else if (rsIn.length <= 0) {

res.status(409).json({messege: "User not in chat!"});

} else {

const delChat = `

DELETE FROM chats WHERE id = ?;

`

db.query(delChat, [chat\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

console.log("Chat deleted");

res.status(200).json(rs)

}

});

}

});

});

sanyaApp.post("/msgsend", (req, res) => {

const{chat\_id, msg, file\_link} = req.body;

let sender\_id = req.cookies.userid;

console.log("sender\_id: "+sender\_id);

console.log("chat\_id: "+chat\_id);

console.log("file\_link: "+file\_link);

const inChat = `

SELECT \* FROM chat\_users WHERE chat\_id = ? AND user\_id = ?;

`;

db.query(inChat, [chat\_id, sender\_id], (err, rsIn) => {

console.log(rsIn);

if (err) {

console.log(err);

} else if (rsIn.length >= 1) {

console.log(chat\_id + " " + sender\_id + " " + msg);

const sendMsg = `

INSERT INTO chat\_msgs (chat\_id, sender\_id, msg)

VALUES

(?, ?, ?);

`;

db.query(sendMsg, [chat\_id, sender\_id, msg], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

const msg\_id = rs.insertId;

if (file\_link && file\_link.length > 0) {

for (let i = 0; i < file\_link.length; i++) {

const addFile = `

INSERT INTO msg\_media (msg\_id, file\_link)

VALUES

(?, ?);

`;

db.query(addFile, [msg\_id, file\_link[i]], (err, rsAdd) => {

console.log(rsAdd);

if (err) {

console.log(err);

} else if (i === file\_link.length - 1) {

console.log("Files linked!");

res.status(200).json({rs, message: "Files linked!"});

};

});

};

} else {

console.log("Message sended!");

res.status(200).json(rs);

};

};

});

} else {

res.status(401).json({message: "User is not in this chat!"});

};

});

});

sanyaApp.patch("/taskupdate", (req, res) => {

const{Progress, task\_id} = req.body;

console.log(Progress + " " + task\_id);

const taskUpdate = `

UPDATE tasks SET Progress = ? WHERE ID = ?;

`;

db.query(taskUpdate, [Progress, task\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

console.log("Task updated!");

res.status(200).json(rs);

};

});

});

sanyaApp.patch("/chatupdate", (req, res) => {

const{chat\_id, title} = req.body;

console.log(chat\_id + " " + title);

const chatUpdate = `

UPDATE chats SET title = ? WHERE id = ?;

`;

db.query(chatUpdate, [title, chat\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

console.log("Chat updated!");

res.status(200).json(rs);

};

});

});

sanyaApp.patch("/task/complete", (req, res) => {

const{Progress, task\_id} = req.body;

console.log(Progress + " " + task\_id);

const taskUpdate = `

UPDATE tasks SET Completed = true WHERE ID = ?;

`;

db.query(taskUpdate, [task\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

console.log("Task completed!");

res.status(200).json(rs);

};

});

});

module.exports = sanyaApp;

Листинг модуля auth.js

const express = require('express');

const cors = require('cors');

const cookieParser = require('cookie-parser');

const mysql = require('mysql');

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const JWT\_SECRET = 'abobaPenisDickHYUzalupka';

const authApp = express();

authApp.use(express.json());

authApp.use(cookieParser());

const corsOptions = {

origin: ['http://localhost:5173', 'http://localhost:5500', 'http://localhost:3000','http://localhost:3001', 'http://localhost:3002'], // Массив разрешенных источников

credentials: true,

};

authApp.use(cors(corsOptions));

authApp.use((req, res, next) => {

res.header('Access-Control-Allow-Origin', req.headers.origin || corsOptions.origin);

res.header('Access-Control-Allow-Credentials', 'true');

res.header('Access-Control-Allow-Methods', 'GET,POST,PUT,DELETE,OPTIONS');

res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, Authorization');

next();

});

const db = mysql.createConnection({

host: "localhost",

user: "root",

password: "spiritbinge",

database: "lab",

});

authApp.get('/cookieclear', async (req, res) => {

res.cookie('token', null, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

});

res.cookie('userid', null, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

});

res.cookie('admin', null, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

});

res.json({ message: 'cookie с токеном очищен' })

})

authApp.post('/register', async (req, res) => {

let access = false;

const {regKey} = req.cookies;

console.log(regKey);

const keyQuery = `

SELECT \* FROM registerKeys WHERE regKey = ?;

`

try {

db.query(keyQuery,

[regKey],

(err, result) => {

if (err) {

console.error('Ошибка при проверке ключа:', err);

return res.status(500).json({ message: "Ошибка сервера" });

} else if (result.length >= 1) {

access = true;

console.log('Заебумба')

if (access) {

const { login, password,

email, phone, firstname,

lastname, surname,

birthday, academdeg,

workexp, gender, childrencount,

familstat } = req.body;

console.log(req.body);

const queryUser = `

SELECT \* FROM Users WHERE Login = ?;

`

db.query(queryUser, [login], async (err, result) => {

if (err) {

console.error('Ошибка сервера');

return res.status(500).json({ message: "Ошибка сервера" })

} else if (result.length >= 1) {

console.log('Пользователь уже существует');

return res.status(409).json({ message: "Пользователь уже зарегистрирован" });

} else {

const queryReg = `

INSERT INTO Users

(Login,

Pass,

Gender,

Birthday,

Email,

AcademicDegree,

HavingChildren,

WorkExperience,

PhoneNumber,

FirstName,

LastName,

Surname,

FamilyStatus,

DepID,

Salary,

Permission)

VALUES

(?,

?,

?,

?,

?,

?,

?,

?,

?,

?,

?,

?,

?,

13,

0,

'user')

`;

try {

const hashedPassword = await bcrypt.hash(password, 10);

db.query(queryReg,

[

login,

hashedPassword,

gender,

birthday,

email,

academdeg,

req.body.havingChildren || 0, // Добавьте обработку для `HavingChildren`

workexp,

phone,

firstname,

lastname,

surname,

familstat,

],

(err, result) => {

if (err) {

console.error('Ошибка при регистрации:', err);

return res.status(500).json({ message: "Ошибка сервера" });

}

console.log('Пользователь создан')

const keyDeleteQuery = `

DELETE FROM registerKeys WHERE regKey = ?;

`

db.query(keyDeleteQuery, [regKey], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

console.log("Ключ удалён и пользователь зарегистрирован");

const query = `

SELECT \* FROM Users WHERE Login = ?

`;

db.query(query, [login], async (err, result) => {

if (err) {

console.error('Ошибка при логине:', err);

return res.status(500).json({ message: "Ошибка сервера" });

}

const user = result[0];

const isPasswordValid = await bcrypt.compare(password, user.Pass);

if (!isPasswordValid) {

return res.status(400).json({ message: "Неправильный пароль" });

}

console.log(user);

if (user.Isblocked) {

return res.status(400).json({message: "Вы уволены"});

}

const token = jwt.sign({ id: user.ID, permission: user.Permission }, JWT\_SECRET, { expiresIn: '1h' });

res.cookie('token', token, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

// maxAge: 100

maxAge: 10 \* 24 \* 60 \* 60 \* 1000, // Срок жизни 30 дней

});

res.cookie('userid', user.ID, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

// maxAge: 100

maxAge: 10 \* 24 \* 60 \* 60 \* 1000, // Срок жизни 30 дней

});

res.cookie('admin', user.Permission, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

// maxAge: 100

maxAge: 10 \* 24 \* 60 \* 60 \* 1000, // Срок жизни 30 дней

});

res.json({ token, message: 'Пользователь зарегистрирован и авторизован', userid: user.ID, permission: user.Permission });

});

}

});

});

} catch (err) {

console.error('Ошибка при хэшировании пароля:', err);

res.status(500).json({ message: "Ошибка сервера" });

}

}

})

}

} else {

access = false

res.status(403).json({access: false, message: "Неправильный код регистрации"});

}

});

} catch (err) {

console.error('Ошибка: ', err);

res.status(500).json({ message: "Ошибка сервера" });

}

});

authApp.post('/login', (req, res) => {

const { login, password } = req.body;

const query = `

SELECT \* FROM Users WHERE Login = ?

`;

console.log('login: ' + login);

console.log('password: ' + password);

db.query(query, [login], async (err, result) => {

if (err) {

console.error('Ошибка при логине:', err);

return res.status(500).json({ message: "Ошибка сервера" });

}

if (result.length === 0) {

return res.status(400).json({ message: "Пользователь не найден" });

}

const user = result[0];

const isPasswordValid = await bcrypt.compare(password, user.Pass);

if (!isPasswordValid) {

return res.status(400).json({ message: "Неправильный пароль" });

}

console.log(user);

if (user.Isblocked) {

return res.status(400).json({message: "Вы уволены"});

}

const token = jwt.sign({ id: user.ID, permission: user.Permission }, JWT\_SECRET, { expiresIn: '1h' });

res.cookie('token', token, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

// maxAge: 100

maxAge: 10 \* 24 \* 60 \* 60 \* 1000, // Срок жизни 30 дней

});

res.cookie('userid', user.ID, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

// maxAge: 100

maxAge: 10 \* 24 \* 60 \* 60 \* 1000, // Срок жизни 30 дней

});

res.cookie('admin', user.Permission, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

// maxAge: 100

maxAge: 10 \* 24 \* 60 \* 60 \* 1000, // Срок жизни 30 дней

});

res.json({ token, message: 'Пользователь авторизован', userid: user.ID, permission: user.Permission });

});

});

authApp.get('/cookiecheck', (req, res)=>{

const userid = req.cookies.userid;

const admin = req.cookies.admin;

res.json({ userid: userid, permission: admin });

})

authApp.get("/keys", (req, res) => {

const isAdmin = req.cookies.admin == "admin";

console.log(isAdmin);

const getKeys = `

SELECT \* FROM registerKeys;

`

if (isAdmin) {

db.query(getKeys, [], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

res.status(200).json(rs);

}

});

} else {

res.status(403).json({ message: "Отказано в доступе" })

}

});

authApp.post('/keycheck', (req, res)=>{

const {key} = req.body;

const keyQuery = `

SELECT \* FROM registerKeys WHERE regKey = ?;

`

try {

db.query(keyQuery,

[key],

(err, result) => {

if (err) {

console.error('Ошибка при проверке ключа:', err);

return res.status(500).json({ message: "Ошибка сервера" });

} else if (result.length >= 1) {

res.cookie('regKey', key, {

httpOnly: true,

secure: false,

sameSite: 'Lax',

maxAge: 2 \* 60 \* 60 \* 1000, // Срок жизни 2 дней

});

res.status(201).json({ access: true, message: "Вы можете зарегистрироваться" });

} else {

res.status(403).json({access: false, message: "Неправильный код регистрации"});

}

});

} catch (err) {

console.error('Ошибка: ', err);

res.status(500).json({ message: "Ошибка сервера" });

}

})

authApp.post('/addkey', (req, res)=>{

const {key} = req.body;

const {user\_id} = req.cookies.userid;

const addKey = `

INSERT INTO registerKeys (regKey, keyCreatorID) VALUES (?, ?);

`;

db.query(addKey, [key, user\_id], (err, rs) => {

console.log(rs);

if (err) {

console.log(err);

} else {

console.log("Key created");

res.status(200).json(rs);

};

});

})

authApp.post('/protected', (req, res) => {

const token = req.cookies.token;

const user\_id = req.cookies.userid;

let haveAccess = true;

let accessMsg = "";

if (!token) {

return res.status(401).json({ message: "Доступ запрещён" });

}

const query = `

SELECT \* FROM Users WHERE ID = ?

`;

db.query(query, [user\_id], async (err, result) => {

if (err) {

console.log(err)

} else if (result.length >= 1) {

console.log(result);

const user = result[0];

console.log(user.Isblocked);

if (user.Isblocked) {

haveAccess = false;

accessMsg = 'Вы уволены';

}

console.log(haveAccess);

if (haveAccess) {

jwt.verify(token, JWT\_SECRET, (err, user) => {

if (err) {

return res.status(403).json({ message: "Неправильный токен", access: false });

}

res.json({ message: `Привет пользователь с ID: ${user.id}`, access: true });

});

} else {

res.json({ message: accessMsg, access: false });

}

} else {

res.status(401).json({ message: "Ваш профиль удалён", access: false });

}

})

});

module.exports = authApp;

Листинг модуля main.go

package main

import (

"fmt"

"log"

"rest-api/internal/handler/http"

connect "rest-api/pkg/db"

"rest-api/pkg/requests"

"github.com/gin-gonic/gin"

\_ "github.com/go-sql-driver/mysql"

"github.com/joho/godotenv"

)

const (

allUsers = "/users"

userById = "/users/:id"

article = "/articles"

articleById = "/articles/:id"

commentsByID = "/articles/:id/comments"

addComm = "/articles/:id/comments"

addToFavorite = "/articles/:id/favorites"

articlesByAuthorId = "/articles/author/:id"

tasksByUserId = "/users/:id/tasks"

CreateDepartments = "/departments"

CreateProject = "/projects"

uploadFilePath = "/upload"

staticUploadFile = "/uploads"

uploadHtmlPath = "/upload-html"

getUserByIDCookie = "/self"

getTaskByID = "/tasks/selftasks"

createPFP = "/users/:id/avatar"

userFired = "/users/:id/fired"

)

func main() {

// Загружаем данные из .env файла

err := godotenv.Load()

if err != nil {

log.Fatalf("Ошибка при загрузке .env файла: %v", err)

}

// Подключение к БД

db, err := connect.ConToDatabase()

if err != nil {

log.Fatalf("Ошибка при подключении к базе данных: %v", err)

}

defer db.Close()

fmt.Println("Подключение к базе данных успешно установлено")

r := gin.Default()

//запросы уровня кондиций

r.GET(allUsers, http.GetAllUsersHandler(db))

r.GET(userById, http.GetUserByIdHandler(db))

r.GET(article, http.GetAllArticlesHandler(db))

r.GET(articleById, http.GetArticlesByIdHandler(db))

r.GET(tasksByUserId, http.GetTasksByUserIdHandler(db))

r.GET(commentsByID, http.GetCommentByIdHandler(db))

r.GET(getUserByIDCookie, http.GetUserBySelfHandler(db))

r.GET(getTaskByID, http.GetTaskByUserIdHandler(db))

r.GET(articlesByAuthorId, http.GetArticleByAuthorIdHandler(db))

r.GET(CreateDepartments, http.GetAllDepartaments(db))

r.POST(CreateDepartments, http.CreateDepartmentHandler(db))

r.POST(CreateProject, http.CreateProjectHandler(db))

r.POST(addComm, http.CreateComHandler(db))

r.POST(addToFavorite, http.AddToVadoritesHandler(db))

r.POST(tasksByUserId, http.CreateTaskByUserIdHandler(db))

r.PATCH(userById, http.UserUpdateHandler(db))

r.PATCH(createPFP, http.CreatePFPHandler(db))

r.PATCH(userFired, http.BlockUserHandler(db))

r.POST(uploadFilePath, requests.UploadFile)

r.Static(staticUploadFile, "./uploads")

r.POST(uploadHtmlPath, http.CreateArticleHandler(db))

r.Run(":3002")

}

Листинг модуля handlers.js

package http

import (

"database/sql"

"fmt"

"net/http"

"path/filepath"

"rest-api/pkg/requests"

"strconv"

"time"

"github.com/gin-gonic/gin"

)

func GetAllUsersHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

users, err := requests.GetAllUsers(db)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

c.JSON(200, users)

}

}

func GetUserByIdHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

idStr := c.Param("id")

id, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid user ID"})

return

}

user, err := requests.GetUser(db, id)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

if user == nil {

c.JSON(404, gin.H{"error": "User not found"})

return

}

c.JSON(200, user)

}

}

func GetAllArticlesHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

// Извлекаем ID пользователя из куки

cookie, err := c.Cookie("userid")

if err != nil {

c.JSON(401, gin.H{"error": "Unauthorized: userid not found in cookie"})

return

}

userId, err := strconv.Atoi(cookie)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid userid in cookie"})

return

}

// Получаем статьи

articles, err := requests.GetAllArticles(db, userId)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

// Возвращаем статьи клиенту

c.JSON(200, articles)

}

}

func GetArticlesByIdHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

// Получаем ID статьи из параметров

idStr := c.Param("id")

articleID, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid article ID"})

return

}

// Получаем ID пользователя из куки

cookie, err := c.Cookie("userid")

if err != nil {

c.JSON(401, gin.H{"error": "Unauthorized: missing userid in cookie"})

return

}

userID, err := strconv.Atoi(cookie)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid userid in cookie"})

return

}

// Получаем статью

article, err := requests.GetArticleById(db, articleID, userID)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

if article == nil {

c.JSON(404, gin.H{"error": "Article not found"})

return

}

c.JSON(200, article)

}

}

func GetTasksByUserIdHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

//получаю айдишник из юрл

idStr := c.Param("id")

userId, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid user ID"})

return

}

//получаю таски

tasks, err := requests.GetTaskByID(db, userId)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

c.JSON(200, tasks)

}

}

func CreateDepartmentHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

var newDepartment requests.Dep

if err := c.ShouldBindJSON(&newDepartment); err != nil {

c.JSON(400, gin.H{"error": err.Error()})

return

}

depID, err := requests.CreateDepartment(db, newDepartment)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

c.JSON(201, gin.H{"message": "Department created", "department\_id": depID})

}

}

func CreateProjectHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

var newProject requests.Project

if err := c.ShouldBindJSON(&newProject); err != nil {

c.JSON(400, gin.H{"error": err.Error()})

return

}

// Добавляем новый проект в базу данных

projID, err := requests.CreateProject(db, newProject)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

// Возвращаем ответ с ID нового проекта

c.JSON(201, gin.H{"message": "Project created", "project\_id": projID})

}

}

func GetCommentByIdHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

idStr := c.Param("id")

id, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid comment ID"})

return

}

comment, err := requests.GetCommentsByArticleId(db, id)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

if comment == nil {

c.JSON(404, gin.H{"error": "comment not found"})

return

}

c.JSON(200, comment)

}

}

func CreateComHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

cookie, err := c.Cookie("userid")

if err != nil {

c.JSON(401, gin.H{"error": "Unauthorized, userid is not found in cookie"})

return

}

authorID, err := strconv.Atoi(cookie)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid userid in cookie"})

}

idStr := c.Param("id")

articleID, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid article ID"})

return

}

var newComment requests.CreateComment

if err := c.ShouldBindJSON(&newComment); err != nil {

c.JSON(400, gin.H{"error": err.Error()})

return

}

//вставляю комент в бдшку

err = requests.AddComment(db, articleID, authorID, newComment.Comm)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

c.JSON(201, gin.H{"message": "Comment added successfully"})

}

}

func UserUpdateHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

idStr := c.Param("id")

userId, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid user ID"})

return

}

var updateUser requests.UserUdpate

if err := c.ShouldBindJSON(&updateUser); err != nil {

c.JSON(400, gin.H{"error": err.Error()})

return

}

err = requests.UserUpdate(db, userId, updateUser)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

c.JSON(204, gin.H{"message": "User updated successfully!"})

}

}

func CreateArticleHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

requests.CreateArticle(c, db)

}

}

func GetUserBySelfHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

cookie, err := c.Cookie("userid")

if err != nil {

c.JSON(401, gin.H{"error": "Unauthorized, no userid cookie found"})

return

}

userId, err := strconv.Atoi(cookie)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid user ID"})

return

}

user, err := requests.GetSelf(db, userId)

if err != nil {

c.JSON(400, gin.H{"error": err.Error()})

return

}

if user == nil {

c.JSON(400, gin.H{"error": "User not found"})

return

}

c.JSON(200, user)

}

}

func GetTaskByUserIdHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

cookie, err := c.Cookie("userid")

if err != nil {

c.JSON(401, gin.H{"error": "Unauthorized, userid is not found in cookie"})

return

}

userId, err := strconv.Atoi(cookie)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid userid in cookie"})

return

}

tasks, err := requests.GetTaskByID(db, userId)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

c.JSON(200, tasks)

}

}

func AddToVadoritesHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

cookie, err := c.Cookie("userid")

if err != nil {

c.JSON(401, gin.H{"error": "Unauthorized, userid is not found in cookie"})

return

}

userId, err := strconv.Atoi(cookie)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid userid in cookie"})

return

}

articleIDStr := c.Param("id")

articleID, err := strconv.Atoi(articleIDStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid article ID"})

return

}

err = requests.AddToVadoriteArticles(db, userId, articleID)

if err != nil {

c.JSON(500, gin.H{"error": "Failied to add article to favorite"})

return

}

c.JSON(201, gin.H{"message": "Article added to favorite successfully"})

}

}

func GetArticleByAuthorIdHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

idStr := c.Param("id")

authorId, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid author id"})

return

}

articles, err := requests.GetArticlesByAuthorID(db, authorId)

if err != nil {

c.JSON(500, gin.H{"error": "Failed to get articles"})

return

}

if len(articles) == 0 {

c.JSON(404, gin.H{"error": "No articles found for this author"})

return

}

c.JSON(200, articles)

}

}

func CreateTaskByUserIdHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

var newTask requests.TaskByID

idStr := c.Param("id")

employeeId, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid employee ID"})

return

}

if err := c.ShouldBindJSON(&newTask); err != nil {

c.JSON(400, gin.H{"error": "Invalid input"})

return

}

err = requests.CreateTaskByEmployeeId(db, employeeId, newTask)

if err != nil {

c.JSON(400, gin.H{"error": "Failed to add task"})

return

}

c.JSON(201, gin.H{"message": "Task added successfully!"})

}

}

func CreatePFPHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

// Получаем ID пользователя из параметров URL

userIDStr := c.Param("id")

userID, err := strconv.Atoi(userIDStr)

if err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid user ID"})

return

}

// Получаем файл из запроса

file, err := c.FormFile("avatar")

if err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": "Failed to upload avatar"})

return

}

// Генерируем уникальное имя файла на основе таймстемпа UNIX

uniqueFilename := fmt.Sprintf("%d\_%s", time.Now().Unix(), file.Filename)

// Сохраняем файл в папку uploads/pics

path := filepath.Join("uploads", "pics", uniqueFilename)

if err := c.SaveUploadedFile(file, path); err != nil {

c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to save avatar"})

return

}

// Формируем URL для файла

avatarURL := fmt.Sprintf("http://localhost:3002/uploads/pics/%s", uniqueFilename)

// Обновляем ссылку на аватар в базе данных

err = requests.CreatePFP(db, userID, avatarURL)

if err != nil {

c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update avatar in database"})

return

}

// Возвращаем успешный ответ с URL аватарки

c.JSON(http.StatusOK, gin.H{

"message": "Avatar updated successfully",

"avatar\_url": avatarURL,

})

}

}

func BlockUserHandler(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

idStr := c.Param("id")

userID, err := strconv.Atoi(idStr)

if err != nil {

c.JSON(400, gin.H{"error": "Invalid user ID"})

return

}

err = requests.FireUser(db, userID)

if err != nil {

c.JSON(500, gin.H{"error": "Failed to block user"})

return

}

c.JSON(204, gin.H{"message": "User blocked successfully!"})

}

}

func GetAllDepartaments(db \*sql.DB) gin.HandlerFunc {

return func(c \*gin.Context) {

departments, err := requests.GetAllDepartments(db)

if err != nil {

c.JSON(500, gin.H{"error": err.Error()})

return

}

c.JSON(200, departments)

}

}

Листинг модуля main.jsx

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App.jsx';

import './index.css';

import { BrowserRouter as Router, Route, Routes, NavLink, Navigate, useNavigate } from 'react-router-dom';

ReactDOM.createRoot(document.getElementById('root')).render(

<Router>

<App />

</Router>,

)

Листинг модуля App.jsx

import { useState, useEffect } from 'react';

import { BrowserRouter as Router, Route, Routes, NavLink, Navigate, useNavigate } from 'react-router-dom';

import Auth from './modules/Auth.jsx';

import Header from './modules/Header.jsx';

import NewArticle from './modules/Pages/NewArticle.jsx';

import Messages from './modules/Pages/Messages.jsx';

import Admin from './modules/Pages/Admin.jsx';

import Articles from './modules/Pages/Articles.jsx';

import Tasks from './modules/Pages/Tasks.jsx';

import SelectedArticle from './modules/Cards/SelectedArticle.jsx';

function App() {

const [logged, setLogged] = useState(undefined);

const [selectedPage, setSelectedPage] = useState('none');

const [firstLogin, setFirstLogin] = useState(true);

const [showArticleEditor, setShowArticleEditor] = useState(false);

const [usrInf, setUsrInf] = useState({});

const [usrId, setUsrId] = useState();

const [showHint, setShowHint] = useState(false);

const [permission, setPermission] = useState('user');

const navigate = useNavigate();

const AuthTry = async () => {

try {

const response = await fetch("http://localhost:3000/js-service/auth/protected", {

method: 'POST',

credentials: 'include',

});

const responseData = await response.json();

console.log(responseData);

if (response.status === 401 || !responseData.access) {

setLogged(false);

navigate('/login'); // Перенаправление на страницу логина

return;

}

setLogged(true);

// navigate('/articles');

const userResponse = await fetch("http://localhost:3000/js-service/auth/cookiecheck", {

method: 'GET',

credentials: 'include',

});

const userData = await userResponse.json();

setUsrId(userData.userid);

setPermission(userData.permission);

} catch (error) {

console.error("Ошибка:", error);

}

};

const cookieClear = async () => {

const response = await fetch("http://localhost:3000/js-service/auth/cookieclear", {

method: 'GET',

credentials: 'include',

withCredentials: true,

});

}

const loadUsrInfo = async () => {

// console.log(usrId);

try {

const response = await fetch(`http://localhost:3000/rest-api-service/self`, {

method: 'GET',

credentials: 'include',

withCredentials: true,

});

const userData = await response.json();

setUsrInf(userData);

console.log(userData);

} catch (error) {

console.error("Ошибка:", error);

}

}

useEffect(() => {

if (firstLogin) {

AuthTry();

}

const root = document.getElementById('root');

root.classList.add(localStorage.getItem('theme') || 'dark');

}, []);

useEffect(() => {

if (logged === undefined) {

AuthTry();

}

}, [logged]);

useEffect(() => {

if (logged) {

console.log(usrId);

loadUsrInfo(usrId);

}

}, [usrId]);

return (

// <Router>

<>

{/\* {(logged == false) &&

<Auth permission={(permission)=>{setPermission(permission); console.log(permission)}} userId={(userId)=>{setUsrId(userId)}} logged={()=>{{setLogged(true); }}}

/>} \*/}

{(logged == true) &&

<Header permission={permission} userInfo={usrInf}

showArticleEditor={(hide) => { !hide? setShowArticleEditor(!showArticleEditor) : setShowArticleEditor(false) }}

logout={ async () => { navigate("/login", { replace: true }); await cookieClear(); setLogged(false); setShowArticleEditor(false); setSelectedPage('none'); }}

selectedFunc={(selectedId)=>{setSelectedPage(selectedId)}}

/>}

<Routes>

<Route path="/login" element={!logged &&

<Auth

permission={(permission) => { setPermission(permission); console.log(permission); }}

userId={(userId) => { setUsrId(userId);}}

logged={() => { loadUsrInfo(); setLogged(true); navigate('/articles')}}

/>} />

<Route path="/" element={<Navigate to="/articles" replace />} />

<Route path="/employee" element={<Admin permission={permission} userInfo={usrInf}/>}>

<Route path=":userid" element={<SelectedArticle />} />

</Route>

<Route path="/newarticle" element={<NewArticle />} />

<Route path="/messages" element={<Messages userInfo={usrInf}/>} >

<Route path=":chatid" element={<></>} />

</Route>

<Route path="/tasks" element={<Tasks />} />

<Route path="/articles" element={<Articles />}>

<Route path=":articleId" element={<SelectedArticle />} />

</Route>

<Route path="\*" element={<Navigate to="/articles" replace />} />

</Routes>

</>

// </Router>

)

}

export default App

Листинг модуля database.go

package connect

import (

"database/sql"

"fmt"

"os"

\_ "github.com/go-sql-driver/mysql"

)

func ConToDatabase() (\*sql.DB, error) {

root := os.Getenv("DB\_USER")

password := os.Getenv("DB\_PASSWORD")

host := os.Getenv("DB\_HOST")

port := os.Getenv("DB\_PORT")

dbName := os.Getenv("DB\_NAME")

dsn := fmt.Sprintf("%s:%s@tcp(%s:%s)/%s", root, password, host, port, dbName)

db, err := sql.Open("mysql", dsn)

if err != nil {

return nil, err

}

return db, nil

}

ПРИЛОЖЕНИЕ Б

Результаты выполнения программы

На рисунке Б.1 представлена форма авторизации. Здесь пользователь может авторизироваться под своими данными, заполнив поля авторизации и нажав кнопку «войти» или перейти к регистрации, нажав на «Нет аккаунта», если у него ещё нет аккаунта.

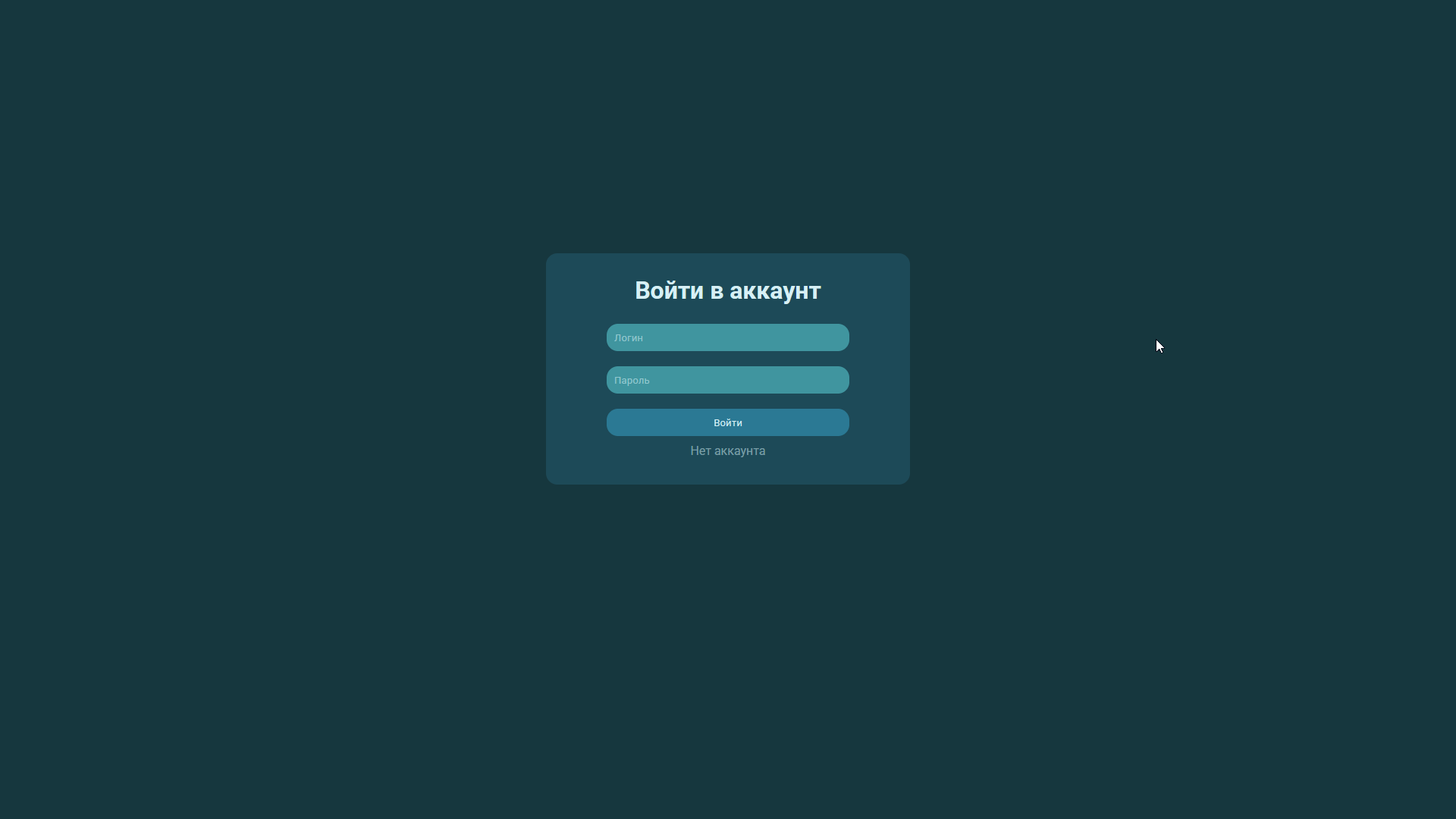


Рисунок Б.1 – Форма авторизации

При переходе к регистрации, пользователю нужно будет ввести код регистрации (рисунок Б.2).

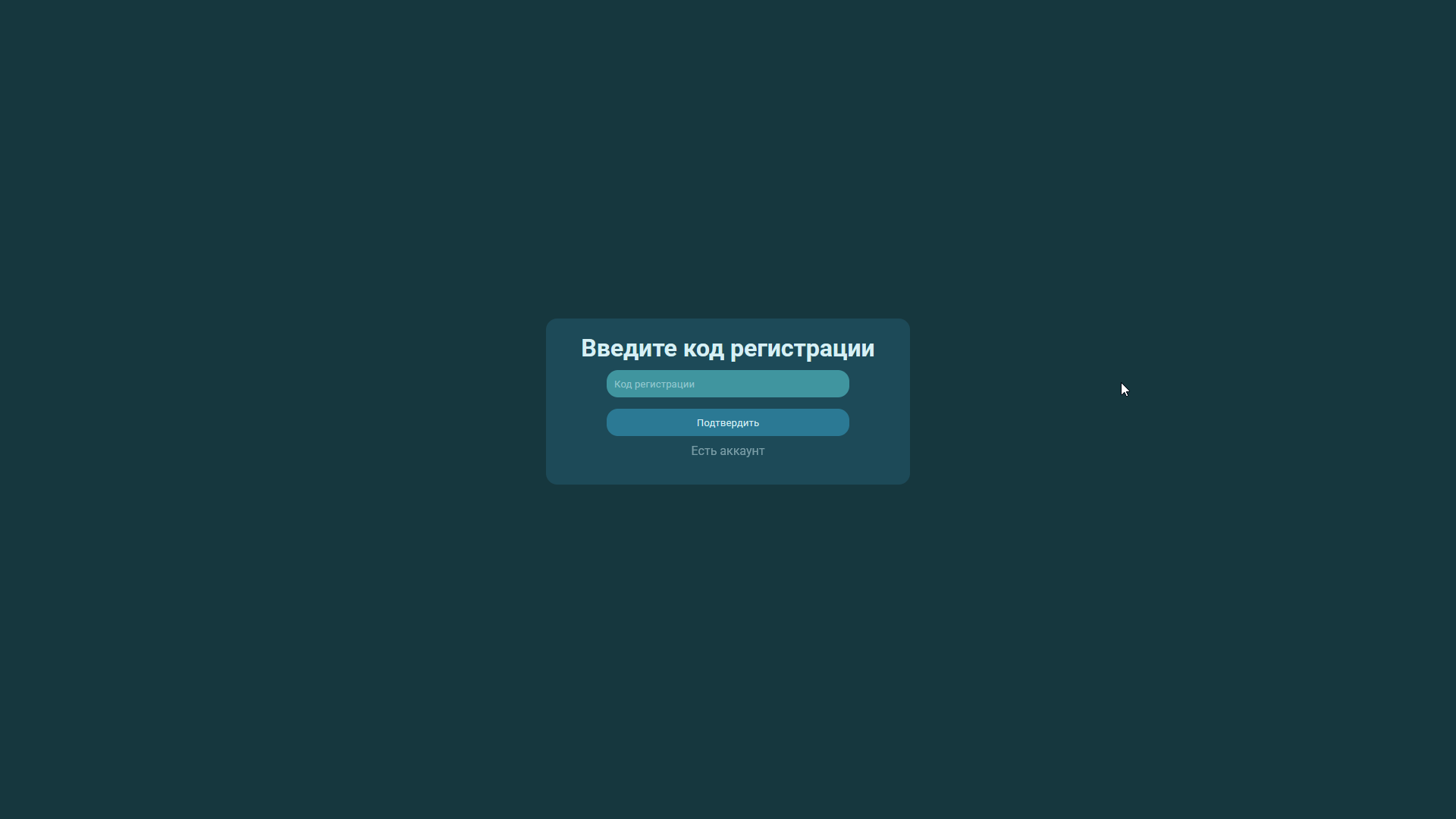


Рисунок Б.2 – Форма кода регистрации

На рисунке Б.3 представлена форма регистрации. Здесь пользователь может вернуться к форме авторизации, нажав на «Есть аккаунт», или зарегистрироваться, заполнив все поля и нажав на кнопку «Зарегистрироваться».

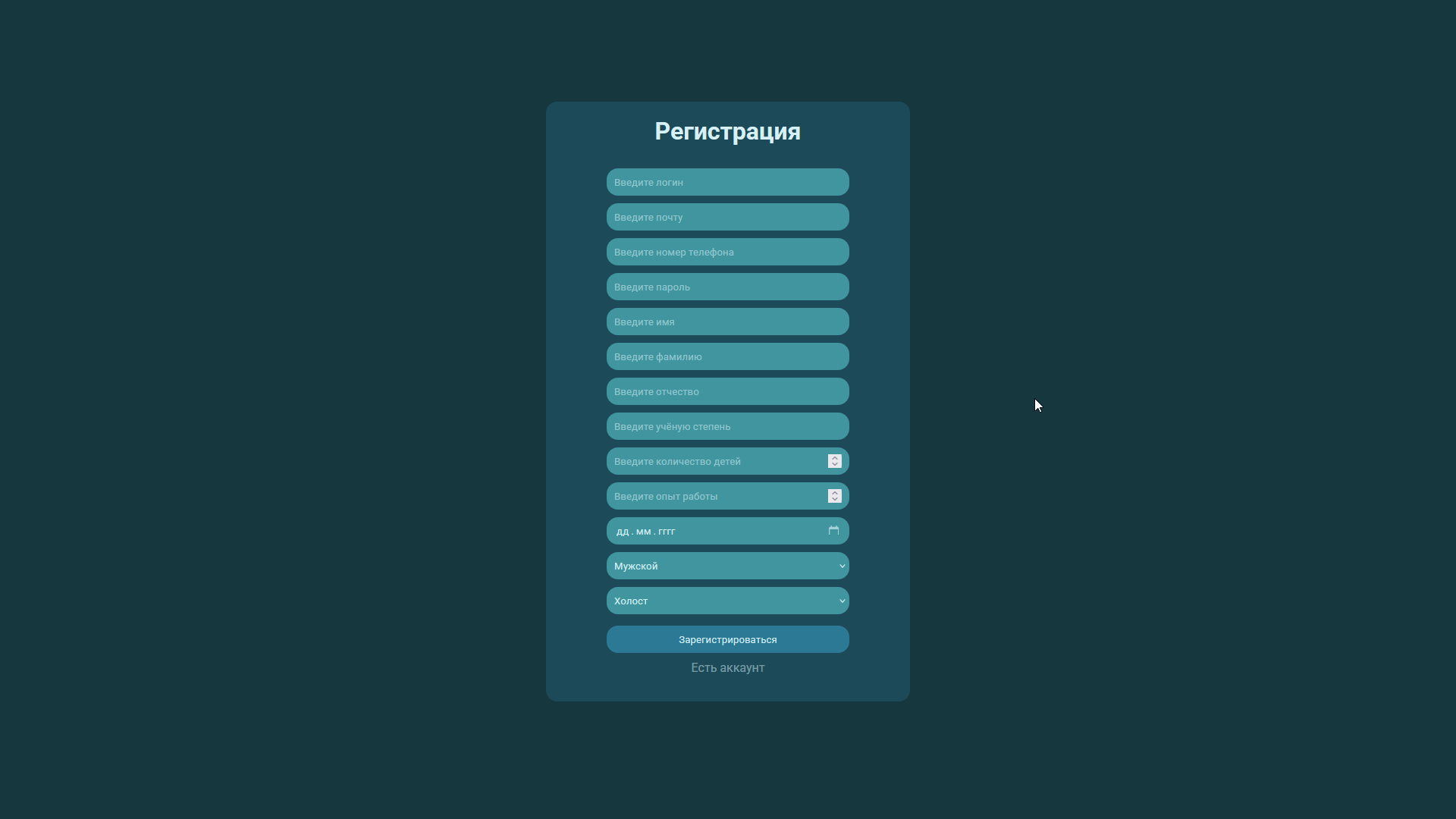


Рисунок Б.3 – Форма регистрации

После регистрации или авторизации, пользователь попадает на форму со списком всех научных статей (рисунок Б.4). Здесь пользователь может выбрать и просмотреть любую научную статью, найти статью по названию или тегам, посмотреть список своих избранных статей или написать свою статью.

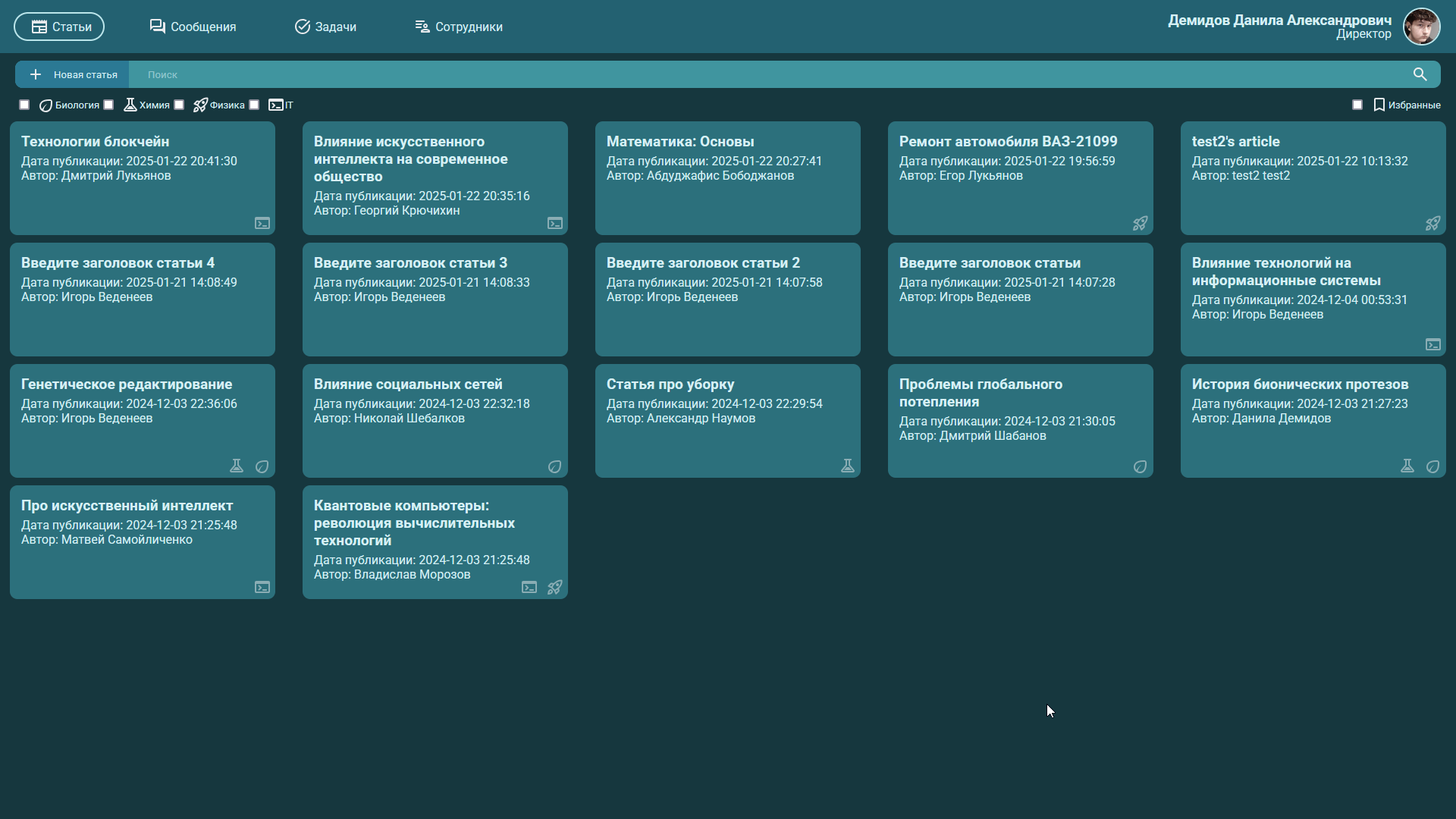


Рисунок Б.4 – Форма со статьями

На рисунке Б.5 представлена форма с просмотром содержимого выбранной статьи. Здесь пользователь может просмотреть содержимое выбранной статьи, найти и выбрать другую статью, добавить выбранную статью в избранное и оставить комментарий.

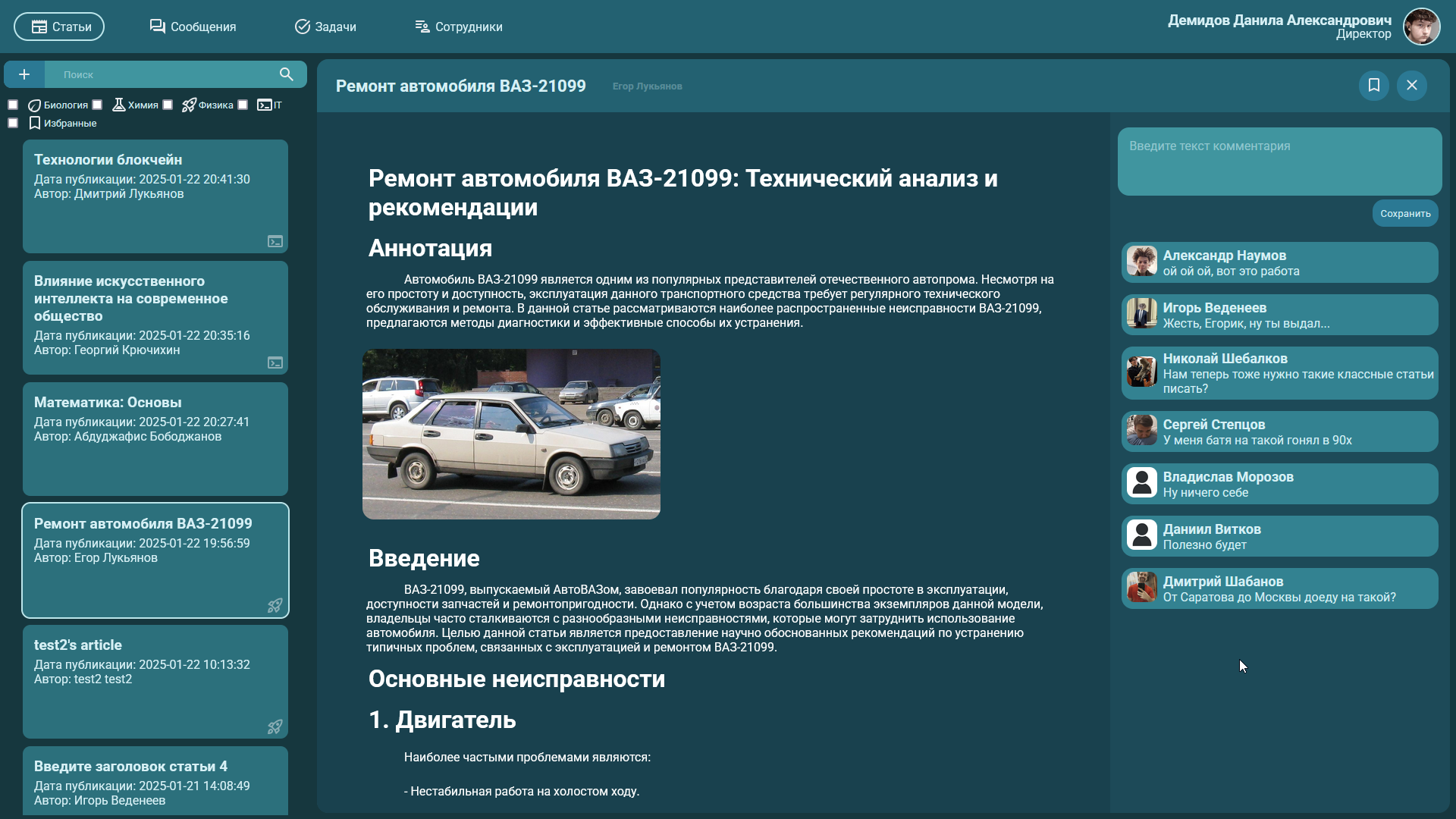


Рисунок Б.5 – Форма выбранной статьи

На рисунке Б.6 представлена форма создания статьи. Здесь пользователь может добавлять различные блоки, из которых будет состоять статья, такие как заголовок, параграф, изображение, аудио или видео, может перейти к публикации, нажав на кнопку «Сохранить» и может вернуться на форму со всеми статьями.

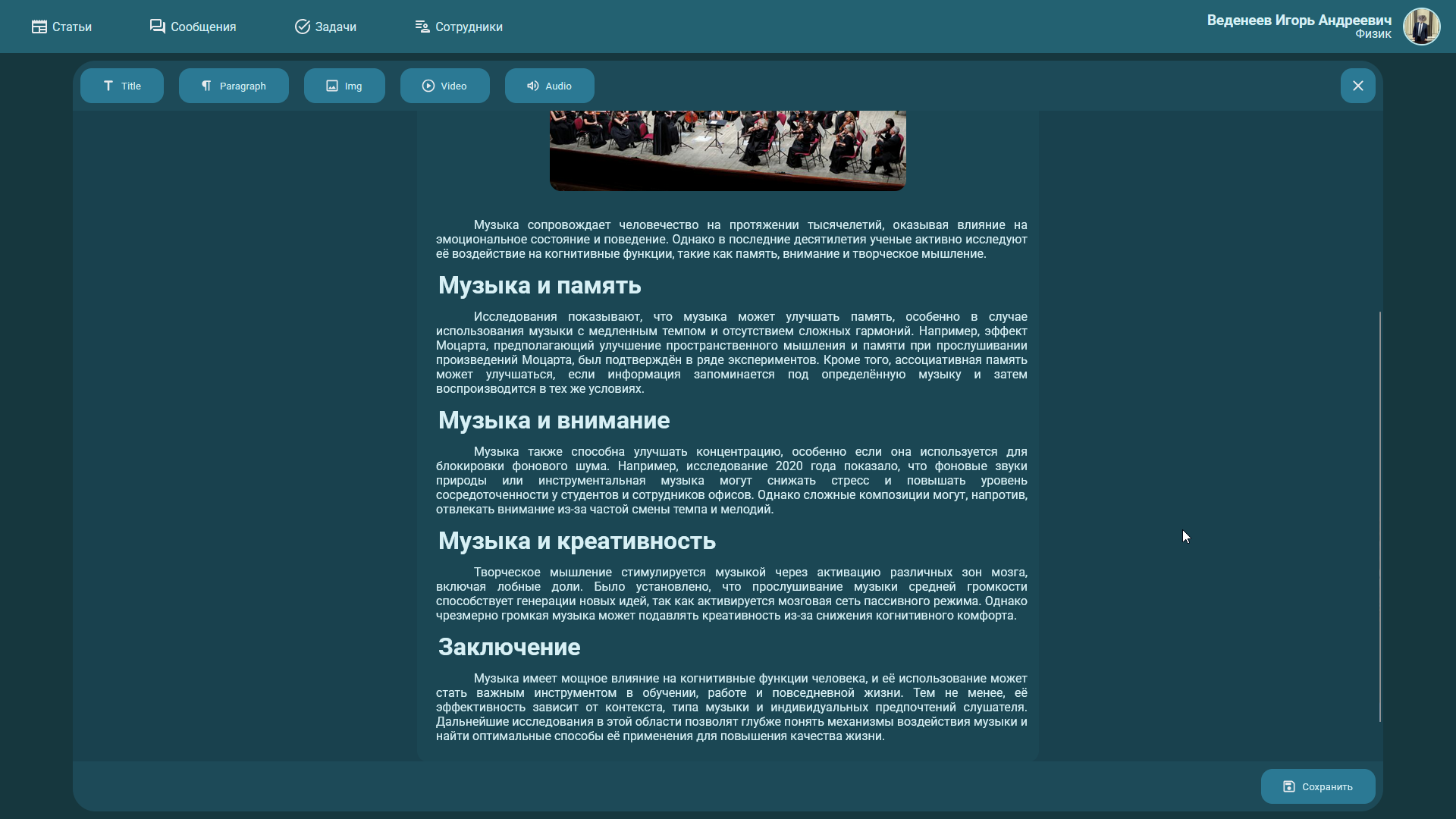


Рисунок Б.6 – Главная страница для администратора

На рисунке Б.7 представлена форма публикации статьи. Здесь пользователь должен ввести название для статьи, может выбрать теги и опубликовать статью.

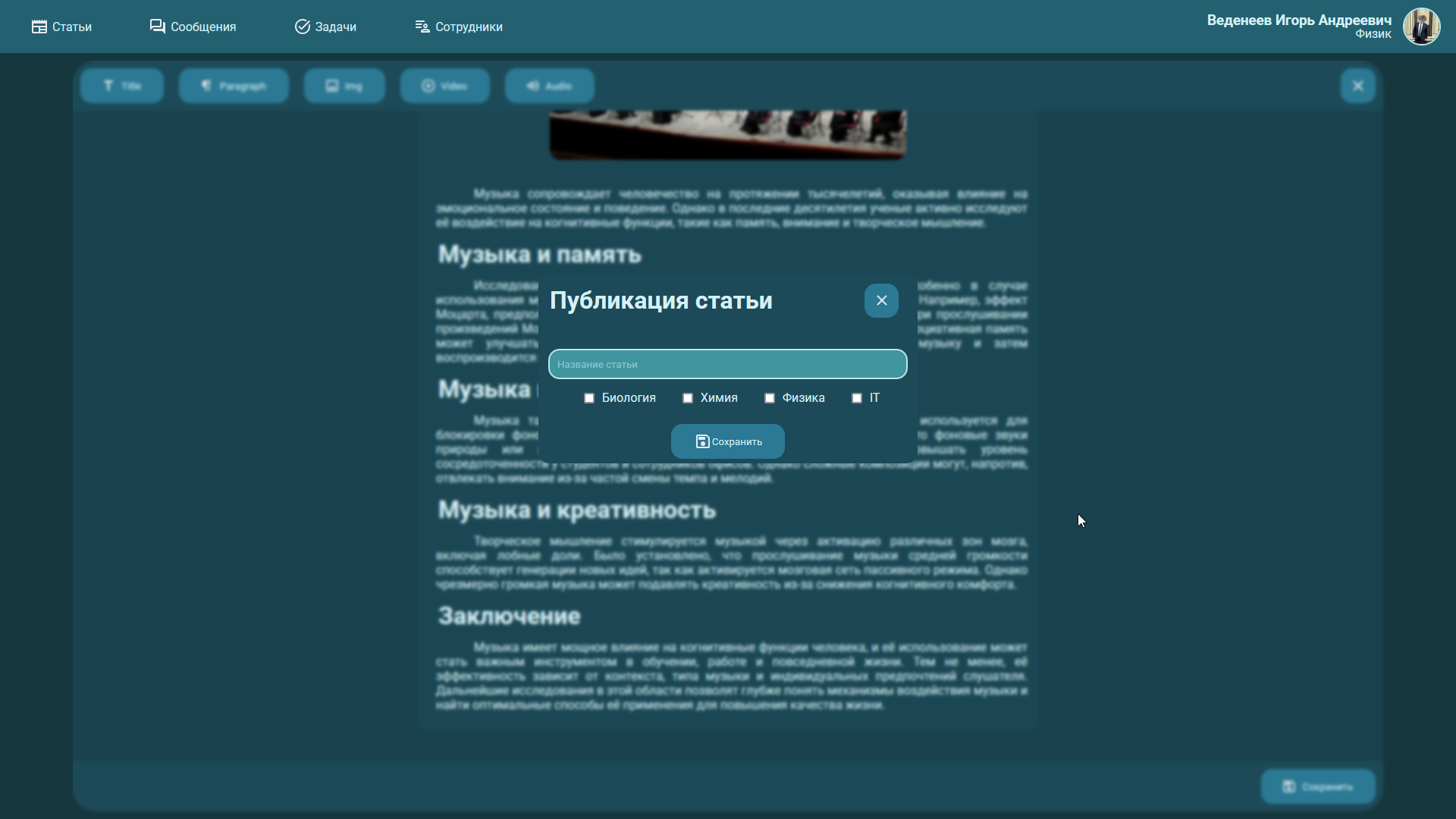


Рисунок Б.7 – Форма публикации статьи

На рисунке Б.8 представлена форма чатов. Здесь пользователь может выбрать чат, написать в него сообщение, открыть настройки чата или создать новый чат.

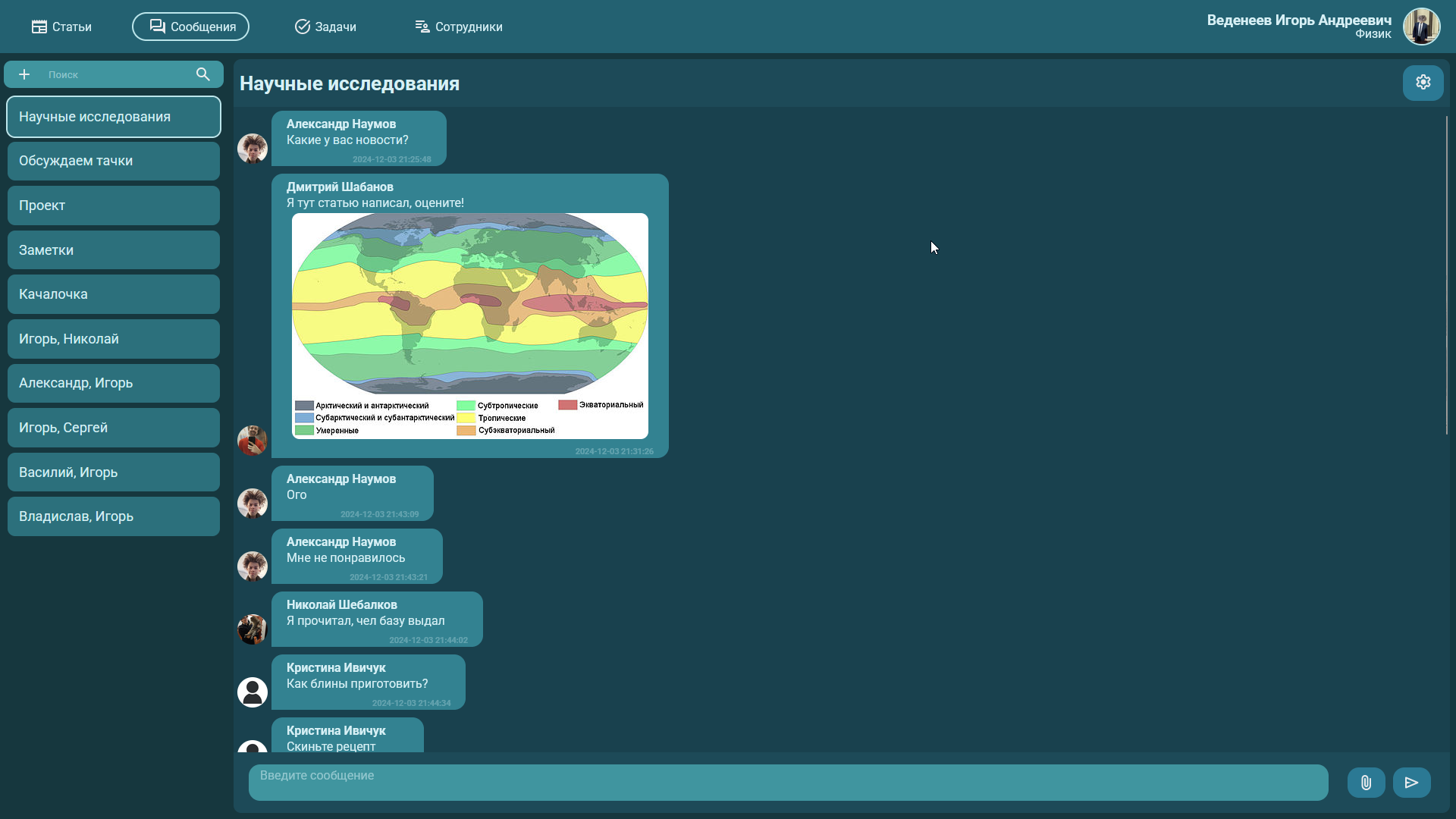


Рисунок Б.8 – Форма чатов

На рисунке Б.9 представлена форма создания чатов. Здесь пользователь должен ввести название чата и нажать кнопку «Добавить чат», после чего новый чат появится в списке чатов.

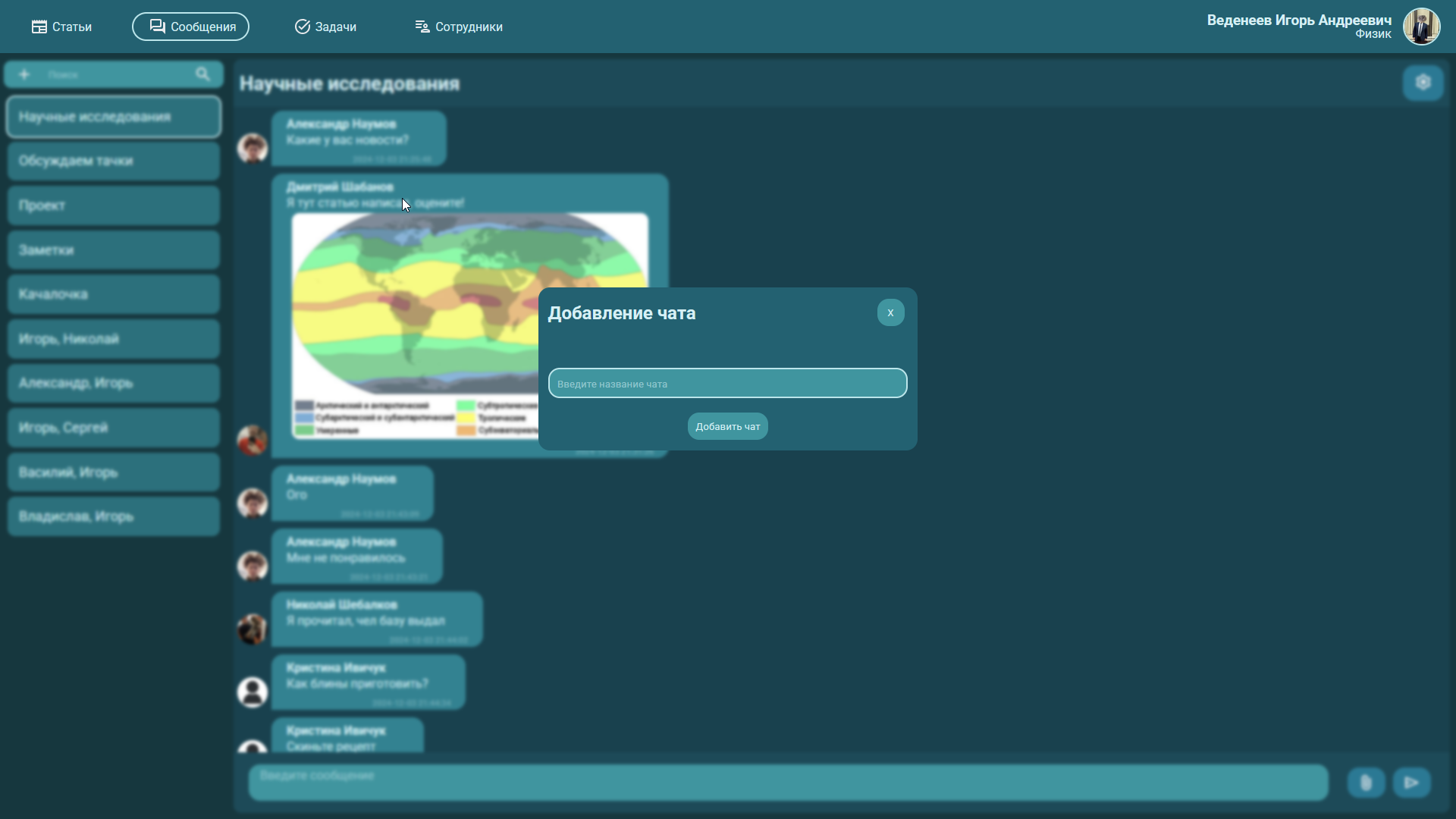


Рисунок Б.9 – Форма базы данных

На рисунке Б.10 представлена форма настроек чата. Здесь пользователь может добавить нового участника в этот чат, а также, если он создатель этого чата, он может изменить ему название или удалить чат.

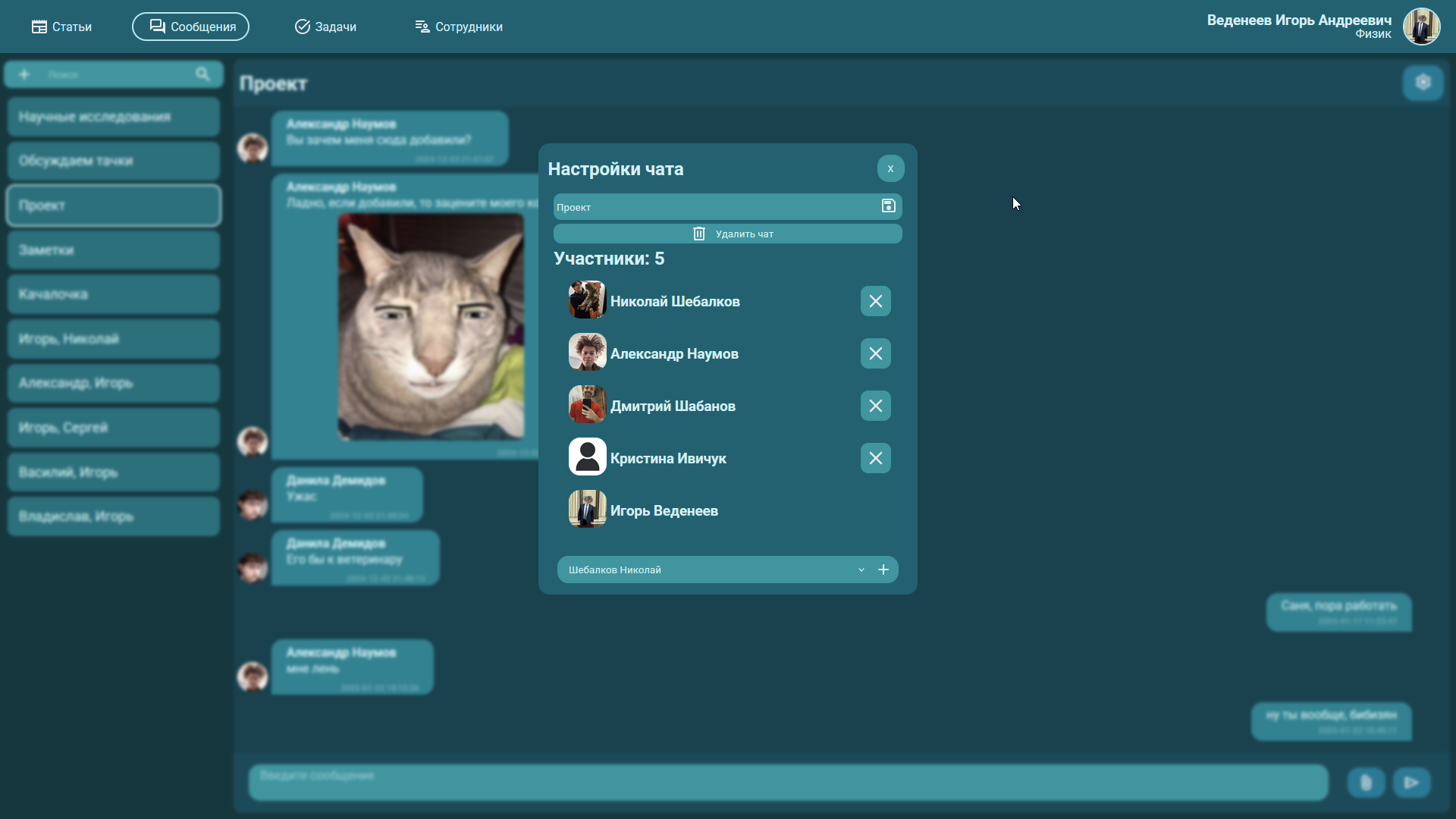


Рисунок Б.10 – Форма настроек чата

На рисунке Б.8 представлена главная страница для читателя библиотеки. В верхнем меню появляется вкладка взятых книг, которая хранит в себе историю выданных книг этого пользователя.

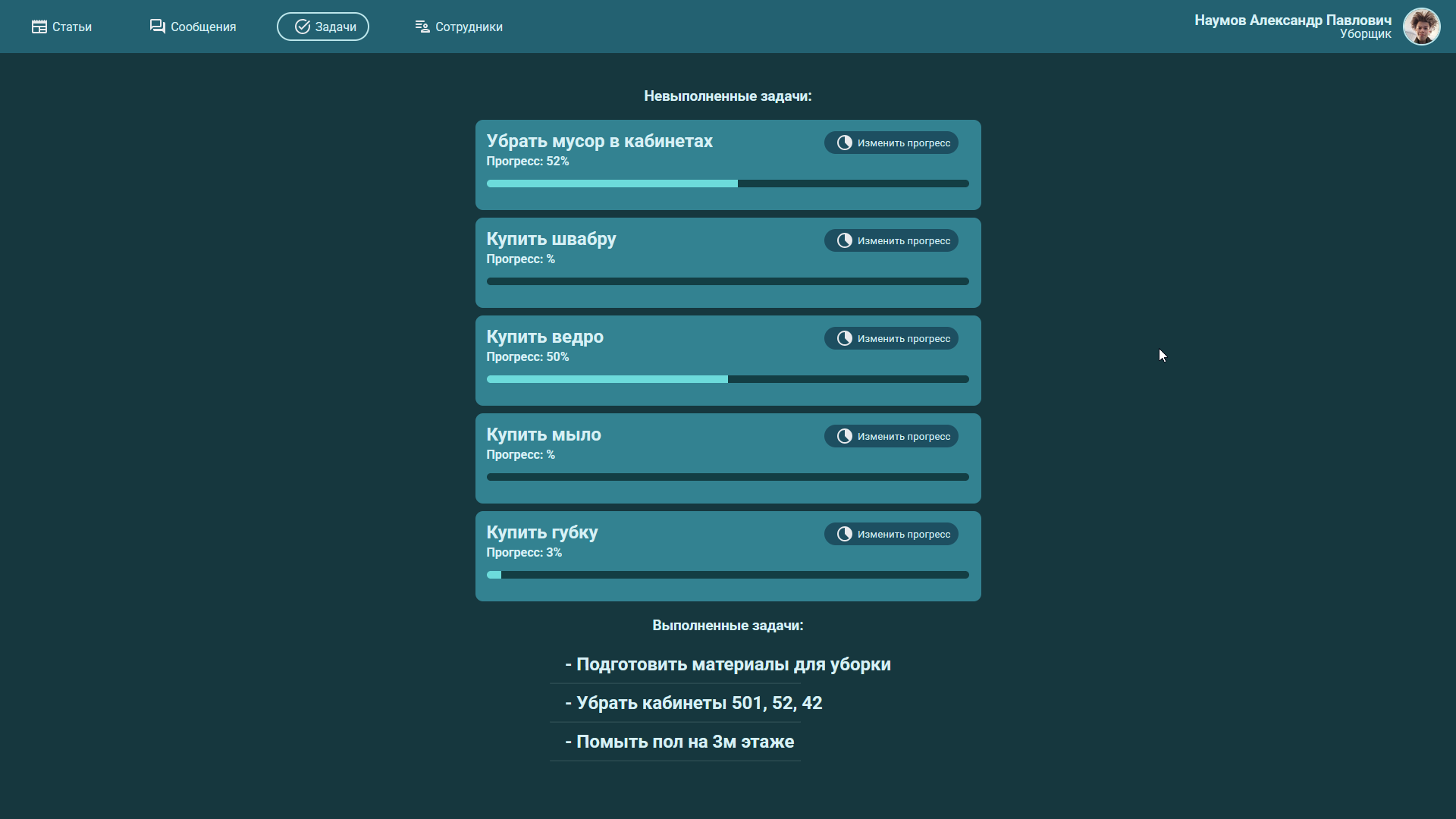


Рисунок Б.8 – Главная форма для читателя библиотеки

На рисунке Б.9 представлена форма взятых книг читателем.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок Б.9 – Форма взятых книг читателем

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Проектирование реляционной базы данных: Метод. указания к домашнему заданию по курсу "Базы данных" / Московский институт электроники и математики им. А.Н. Тихонова НИУ ВШЭ; Сост.: И.П. Карпова. - М., 2017 - 29 с.
2. Илюшечкин, В.М. Основы использования и проектирования баз данных / В.М. Илюшечкин. - М.: Юрайт, 2015. - 516 с.
3. Форта, Бен. SQL за 10 минут, 4-е изд.: Пер. с англ.- М .: ООО "И.Д. Вильямс", 2014 - 288 с
4. Стружкин, Н. П. Базы данных. Проектирование. Учебник / Н.П. Стружкин, В.В. Годин. - М.: Юрайт, 2016. - 478 с.
5. Фуфаев, Э. В. Базы данных / Э.В. Фуфаев, Д.Э. Фуфаев. - М.: Академия, 2016. - 320 с
6. Латыпова, Р. Р. Базы данных. Курс лекций / Р.Р. Латыпова. - Москва: Высшая школа, 2016. - 177 с.
7. Альтернативы для замены SQL Developer Data Modeler [Электронный ресурc] — Режим доступа: <https://ruprogi.ru/software/sql> (<https://ruprogi.ru/software/sql-)-> developer-data-modeler
8. Проектирование баз данных [Электронный ресурс] - Режим доступа: https: // [helpiks.org/5-40515.html](http://helpiks.org/5-40515.html)
9. Юрий Магда, Разработка приложений Microsoft Office 2007 в Delphi, 2009 -160 с.