|  |  |  |
| --- | --- | --- |
| Институт (факультет) | | Институт информационных технологий |
| Кафедра | | Математическое и программное обеспечение ЭВМ |
|  | | |
|  | | |
| КУРСОВАЯ РАБОТА | | |
|  | | |
| по дисциплине | С#-программирование | |
| на тему | Разработка программного обеспечения на языке C# | |
|  |  | |

|  |  |
| --- | --- |
| Выполнил студент группы | |
| 1ПИб-02-1оп-22 | *группа* |
| направления подготовки (специальности) | |
| 09.03.04 Программная инженерия | |
| *шифр, наименование* | |
| Микуцких Григорий Андреевич | |
| *фамилия, имя, отчество* | |
|  | |
| Руководитель | |
| Шаханов Н.И. | |
| *фамилия, имя, отчество* | |
| доцент | |
| *должность* | |
|  | |
| Дата представления работы | |
| «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г. | |
|  | |
| Заключение о допуске к защите | |
|  | |
|  | |
|  | |
|  | |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
| количество баллов | |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |

|  |  |
| --- | --- |
| Череповец, | 2025 |
|  | *год* |

Аннотация

Курсовая работа посвящена разработке приложения «Составитель курсовой работы/отчёта по требованиям кафедры МПО ЭВМ». В ходе работы были организована работа в команде, написана программа на языке высокого уровня (ЯВУ) «C#» с использованием фреймворков «ASP.NET Core», «Blazor» и «Entity Framework Core», составлена диаграмма вариантов использования.

Были реализованы аутентификация (но не авторизация), обработка исключений на своём уровне Middleware, обработка http-запросов, хранение данных в базе данных PostgreSQL, простой и понятный пользовательский интерфейс. Архитектуры приложения: MVC и MVVM.

В работе присутствуют введение в предметную область, описание этапов разработки программы, текст итоговой программы, сопровождение графическим материалом.

Оглавление

[Введение 4](#_Toc200404534)

[1. Теоретическая часть 5](#_Toc200404535)

[1.1. Анализ предметной области 5](#_Toc200404536)

[1.2. Обоснование выбора стека 9](#_Toc200404537)

[1.3. Архитектуры MVC и MVVM 9](#_Toc200404538)

[1.4. Blazor и ASP.NET Core 10](#_Toc200404539)

[2. Практическая часть 13](#_Toc200404540)

[2.1. Разработка представления (клиентская часть и интерфейс) 13](#_Toc200404541)

[2.2. Разработка контроллера (модели представления и серверной части) 15](#_Toc200404542)

[2.3. Разработка модели (база данных и логика) 16](#_Toc200404543)

[2.4. Основная роль в команде 20](#_Toc200404544)

[Заключение 23](#_Toc200404545)

[Список литературы 24](#_Toc200404546)

[Приложение 1 25](#_Toc200404547)

[Приложение 2 31](#_Toc200404548)

[Приложение 3 59](#_Toc200404549)

# Введение

Написание курсовой работы или отчёта помимо содержания включает в себя оформление, которое может меняться во времени или по требованию преподавателя. Чтобы не приходилось вручную настраивать оформление и сосредоточиться только на содержании, необходимо иметь программу, которая возьмёт на себя ответственность за оформление и по окончании работы соберёт итоговый файл в формате DOCX.

Таким образом, необходимо написать приложение, решающее данную проблему, на ЯВУ «C#».

# Теоретическая часть

## Анализ предметной области

Для того, чтобы определиться с функционалом приложения, была составлена диаграмма вариантов использования (рис. 1).

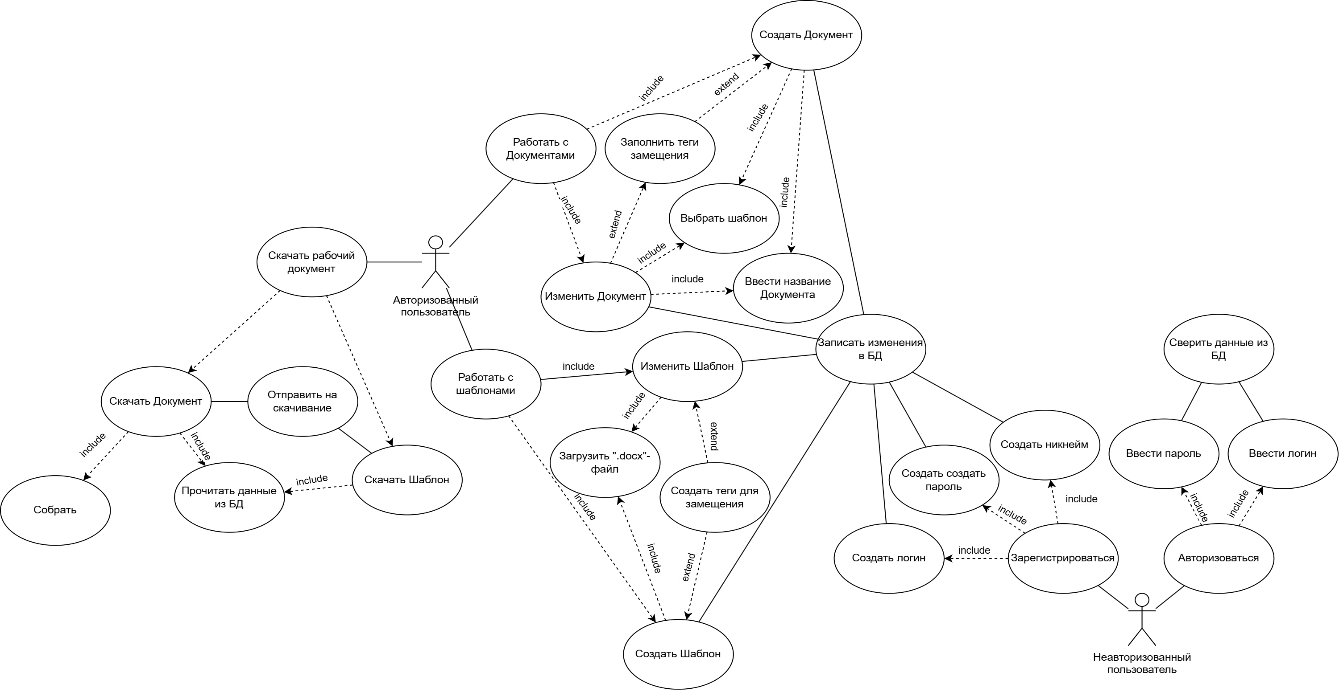


Рис. 1. Диаграмма вариантов использования приложения

Создание документов в формате DOCX в приложении будет опираться на определённые шаблоны, которые можно будет свободно менять, загружать или скачивать. При завершении работы над документом (ввода содержания тегов), итоговый файл можно будет собрать и скачать. При таком подходе содержимое можно будет перемещать по разным шаблонам «одним щелчком».

Ввод пользовательского текста в шаблон будет осуществляться заменой специальных тегов, которые можно определить при создании шаблона.

Приложения могут разрабатываться по следующим архитектурным стилям:

1. монолитная – система как единое целое, где все части тесно связаны [8];
2. микросервисная – система разбивается на небольшие независимые сервисы, которые взаимодействуют друг с другом [8];
3. клиент-серверная –приложение делится на клиентскую и серверную части;
4. событийно-ориентированная – компоненты взаимодействуют через события [8];
5. слоистая – программа организована в виде слоёв, например, презентационный слой, бизнес-логика и база данных [8].

Приложения могут иметь следующие типы:

1. нативные – устанавливаются на конкретную платформу, используя инструменты и языки программирования, специфичные для неё;
2. web-приложения – приложения, не требующие установки и работающие в браузере устройства;
3. гибридные – запускаются на разных платформах с единой кодовой базой с помощью специальных программ-контейнеров.

Язык программирования C# можно использовать для создания консольных приложений. Однако гораздо чаще язык используют для создания приложений, имеющих доступ к множеству технологий, связанных с .NET. Приложения на языке C# можно разрабатывать с помощью следующих технологий:

1. Windows Forms – это платформа пользовательского интерфейса для создания классических приложений Windows, которые можно разрабатывать графически сложные приложения, которые просто развёртывать, обновлять, и с которыми удобно работать как в автономном режиме, так и в сети [2];
2. Windows Presentation Foundation (WPF) – это платформа пользовательского интерфейса, которая не зависит от разрешения и использует векторный механизм визуализации, предоставляет комплексный набор функций разработки приложений, включающий в себя язык XAML, элементы управления, привязку к данным, макет, двумерную и трёхмерную графику, анимацию, стили, шаблоны, документы, мультимедиа, текст и типографические функции [3];
3. ASP.NET Core – это кроссплатформенная, высокопроизводительная среда с открытым исходным кодом для создания современных облачных приложений, подключённых к Интернету, которая включает в себя Blazor для создания богатого интерактивного веб-интерфейса и интегрируется с другими популярными JavaScript фреймворками, такими как Angular, React, Vue и Bootstrap [4].

Данные обычно хранят в базах данных. Примеры существующих систем управления базами данных (СУБД) описаны ниже.

* Oracle — ведущая коммерческая мульти-модельная система управления реляционными базами данных, поддерживающая документное хранение, графовые базы данных, RDF-хранилища, пространственные данные и векторные функции. Широко используется в корпоративных приложениях благодаря своей надёжности и масштабируемости. Когда использовать: крупные корпорации, банки, государственные учреждения [5].
* MySQL – это открытая реляционная СУБД, также поддерживающая документное хранение и пространственные данные. Популярна среди web-разработчиков и используется в многочисленных web-приложениях и онлайн-сервисах. Когда использовать: web-сайты, CMS, электронная коммерция, где важна скорость и простота [5].
* Microsoft SQL Server – это реляционная мульти-модельная СУБД от Microsoft, включающая поддержку документарного хранения, графовых баз данных и пространственных данных. Когда использовать: корпоративные приложения, требующие сложной логики, аналитики и интеграции с «.NET» [5].
* PostgreSQL – это открытая реляционная мульти-модельная СУБД, поддерживающая документное хранение, графовые базы данных, пространственные данные и векторные функции. Поддерживает полнотекстовый поиск, репликацию, хранимые процедуры, JSON и многое другое. Когда использовать: сложные бизнес-приложения, требующие надёжности, гибкости и масштабируемости [5].
* MongoDB – это документно-ориентированная мульти-модельная СУБД, поддерживающая хранение пространственных данных, функции поиска, временные ряды и векторные данные. Предназначена для работы с неструктурированными данными и широко используется в приложениях, требующих гибкости в структуре данных. Когда использовать: быстро меняющиеся схемы, хранение неструктурированных данных [5].

Для работы с базой данных необходимо выбрать фреймворк, который позволит абстрагироваться от выбранного типа базы данных и оставить большую часть кода без изменений при смене СУБД. Существуют такие инструменты:

1. Entity Framework Core — платформа объектно-реляционного отображения (ORM), где, благодаря поддержке LINQ и автоматической миграции схемы отношений, упрощается работа с БД в приложениях на C# [7];
2. Dapper — это легковесный и высокопроизводительный микрофреймворк ORM, созданный для разработчиков, которые предпочитают необработанные SQL-запросы сопоставлениям ORM, и предоставляющий простой и эффективный способ сопоставления результатов работы с базой данных с объектами C# без дополнительных затрат на полноценную ORM-платформу [7];
3. SQLite — это автономный, бессерверный и не требующий настройки механизм работы с SQL-базами данных, который идеально подходит для мобильных приложений, небольших проектов или сценариев без полноценного сервера [7].

Для того, чтобы работать с файлами формата «.docx» можно использовать библиотеку «DocumentFormat.OpenXml.Framework». «Open XML SDK» предоставляет инструменты для работы с документами Office Word, Excel и PowerPoint. Он поддерживает следующие сценарии: создание текстовых документов, электронных таблиц и презентаций, изменение документа (например добавление, обновление и удаление содержимого и метаданных), поиск и замена содержимого с использованием регулярных выражений, разделение файла на несколько файлов и объединение нескольких в один и прочее [9].

## Обоснование выбора стека

В рамках выполнения задачи было принято решение разрабатывать web-приложение с использованием базы данных PostgreSQL и фреймворков «ASP.NET Core», «Blazor» и «EF Core» по следующим причинам:

1. клиент-серверная архитектура позволит отделить сервер и клиент так, что к одному серверу можно будет написать несколько разных клиентов;
2. серверная и клиентская части создаются на одном языке программирования – C# –, что упростит разработку в небольшой команде;
3. PostgreSQL – это одна из самых популярных и востребованных СУБД;

Итоговое приложение не будет требовать от пользователя обновляться вручную и позволит поддерживать шаблоны в актуальном состоянии.

## Архитектуры MVC и MVVM

Для разработки приложения будут использоваться архитектуры MVC и MVVM для разработки серверной и клиентской частей соответственно.

1. Model-View-Controller (MVC, «Модель-Представление-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента так, что модификация каждого компонента может быть независима:

* Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;
* Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели;
* Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений [7].

1. Model-View-ViewModel (MVVM) — шаблон проектирования архитектуры приложения, ориентированный на современные платформы разработки, такие как Windows Presentation Foundation, Silverlight от компании Microsoft, ZK framework [7]. Разделяется на следующие компоненты:

* Model (Модель) – бизнес-логика;
* View (Представление) – пользовательский интерфейс;
* ViewModel (Модель представления) – логика представления и посредник между Model и View, содержит свойства, к которым View привязывается напрямую.

## Blazor и ASP.NET Core

«ASP.NET Core» будет использоваться для разработки серверной части, так как фреймворк поддерживает удобные инструменты для разработки приложений по архитектуре MVC. Тип приложения: «Web-API ASP.NET Core».

«Blazor» будет использоваться для разработки клиентской части по архитектуре MVVM. В качестве ViewModel будут использоваться объекты передачи данных (Data Transfer Object, DTO), расположенные в библиотеке классов, к которой будут иметь доступ и клиентское, и серверное приложение. Тип приложения: «Изолированное приложение Blazor WebAssembly».

В качестве среды разработки выбрана Visual Studio — комплексная интегрированная среда разработки (IDE), которую можно использовать для записи, редактирования, отладки и сборки кода.

1. Rest API

REST – это архитектурный стиль, который описывает как web-приложения и сервисы должны обмениваться данными. Состоит их следующих принципов:

1. клиент-серверная архитектура – чёткое разделение между клиентом и сервером, где клиент запрашивает данные, а сервер их предоставляет [6];
2. отсутствие состояния (stateless) – каждый запрос от клиента к серверу обрабатывается независимо от предыдущих [6];
3. кеширование – ответы сервера могут кешироваться, чтобы снизить нагрузку на сервер и ускорить загрузку данных [6];
4. единообразие интерфейса – все ресурсы должны иметь чёткую структуру и единообразные URL-адреса, запросы к API выполняются с использованием стандартных методов HTTP, а данные передаются в предсказуемых форматах, таких как JSON или XML [6];
5. система уровней (Layered System) – приложение может включать несколько уровней, таких как балансировщики нагрузки, прокси-серверы и системы аутентификации, где каждый уровень выполняет свою функцию [6];
6. возможность выполнения кода по требованию – приложение может поддерживать загрузку и выполнение кода на стороне клиента, например в виде скриптов или небольших программ [6].

Rest API – это программная интерфейс, который следует принципам REST и позволяет клиентам взаимодействовать с сервером с помощью HTTP-протокола. Он использует стандартные HTTP-методы:

* GET – получить данные;
* POST – создать новые данные;
* PUT – обновить существующие данные;
* DELETE – удалить данные;
* PATCH – частично обновить данных.

1. Entity Framework

«EF Core» позволит абстрагироваться от SQL-запросов, специфичной для каждой СУБД. Таким образом, получится повысить степень повторного использования кода, что упростит переход на другие СУБД при необходимости, сохранив существующую схему отношений и базу данных.

1. PostrgreSQL

В качестве таблиц для базы данных выделены три основные сущности:

1. Пользователи (Users);
2. Шаблоны (Templates);
3. Документы (Documents).

В таблицах будут храниться данные клиентов (логин, захешированный пароль и прочее), данные шаблонов (word-документ, теги для замещения и прочее) и данные документов для экспорта (ссылка на шаблон, наполнение тегов шаблона, принадлежность пользователю и прочее) соответственно.

Для администрирования БД будет использоваться программа «pgAdmin 4».

# Практическая часть

## Разработка представления (клиентская часть и интерфейс)

Была разработана клиентская часть, предоставляющая следующие рабочие окна (рис. 2-7).

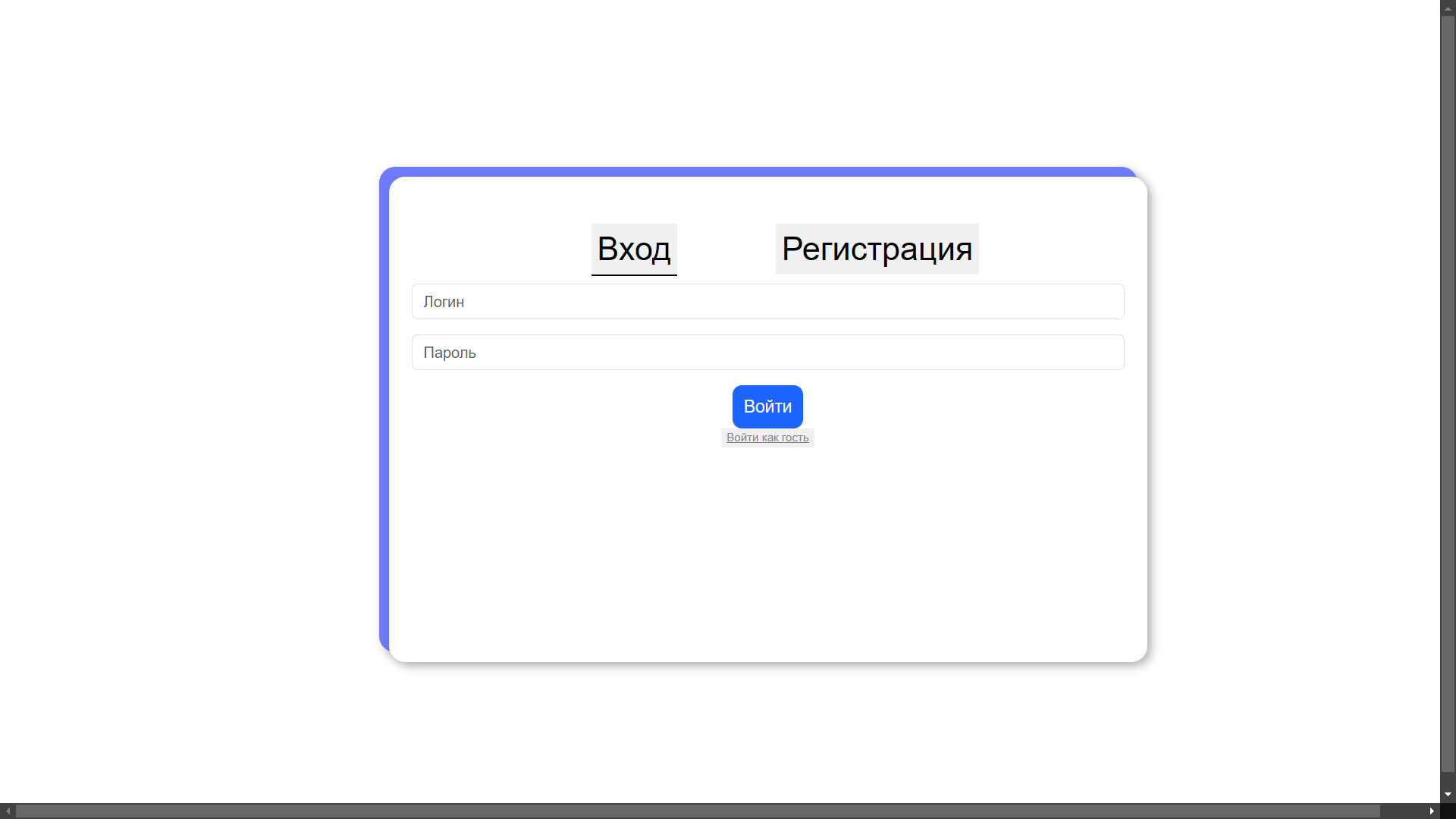


Рис. 2. Пример окна аутентификации

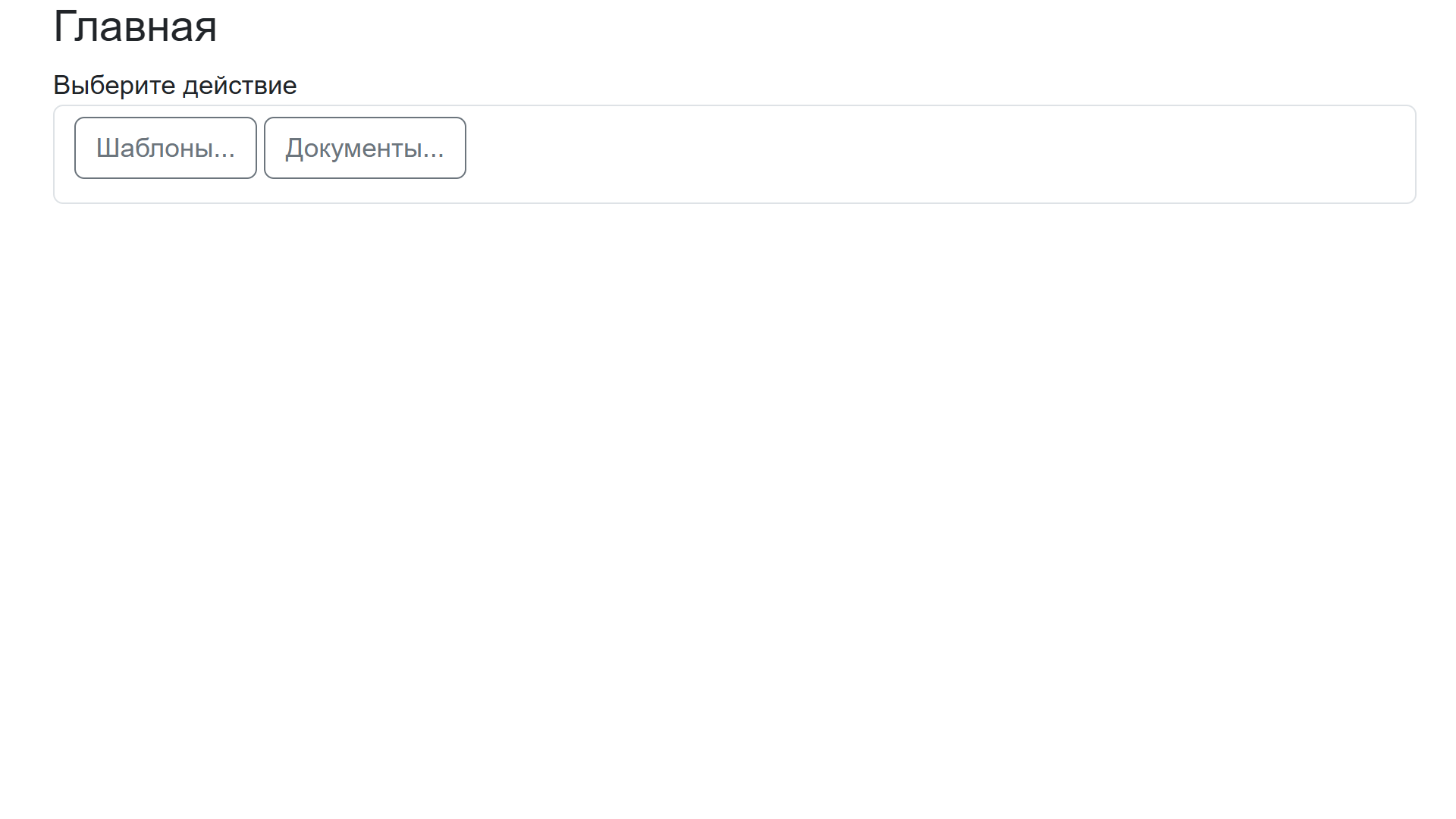


Рис. 3. Пример главной страницы

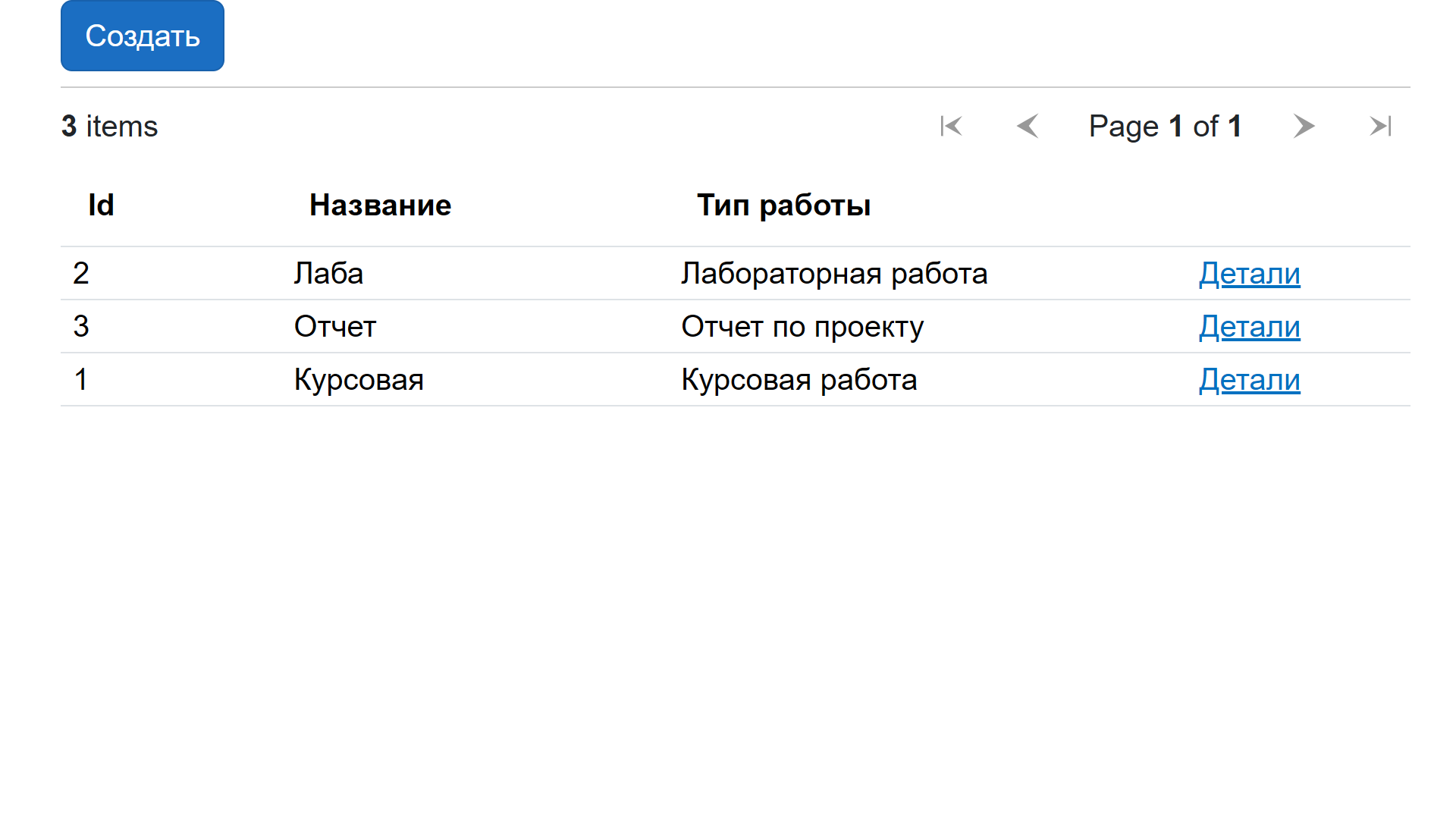


Рис. 4. Пример окна выбора шаблона

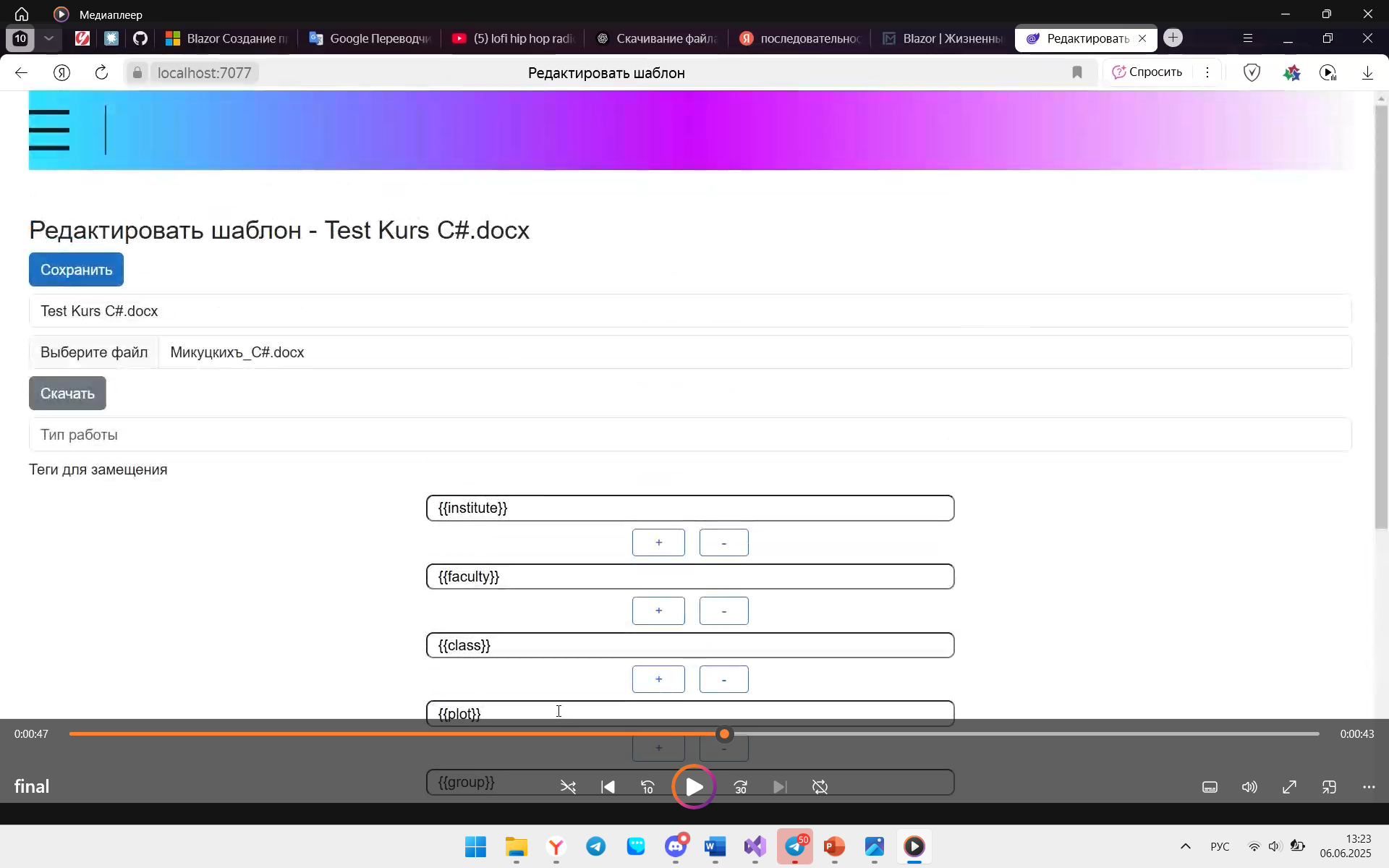


Рис. 5. Пример окна редактирования шаблона

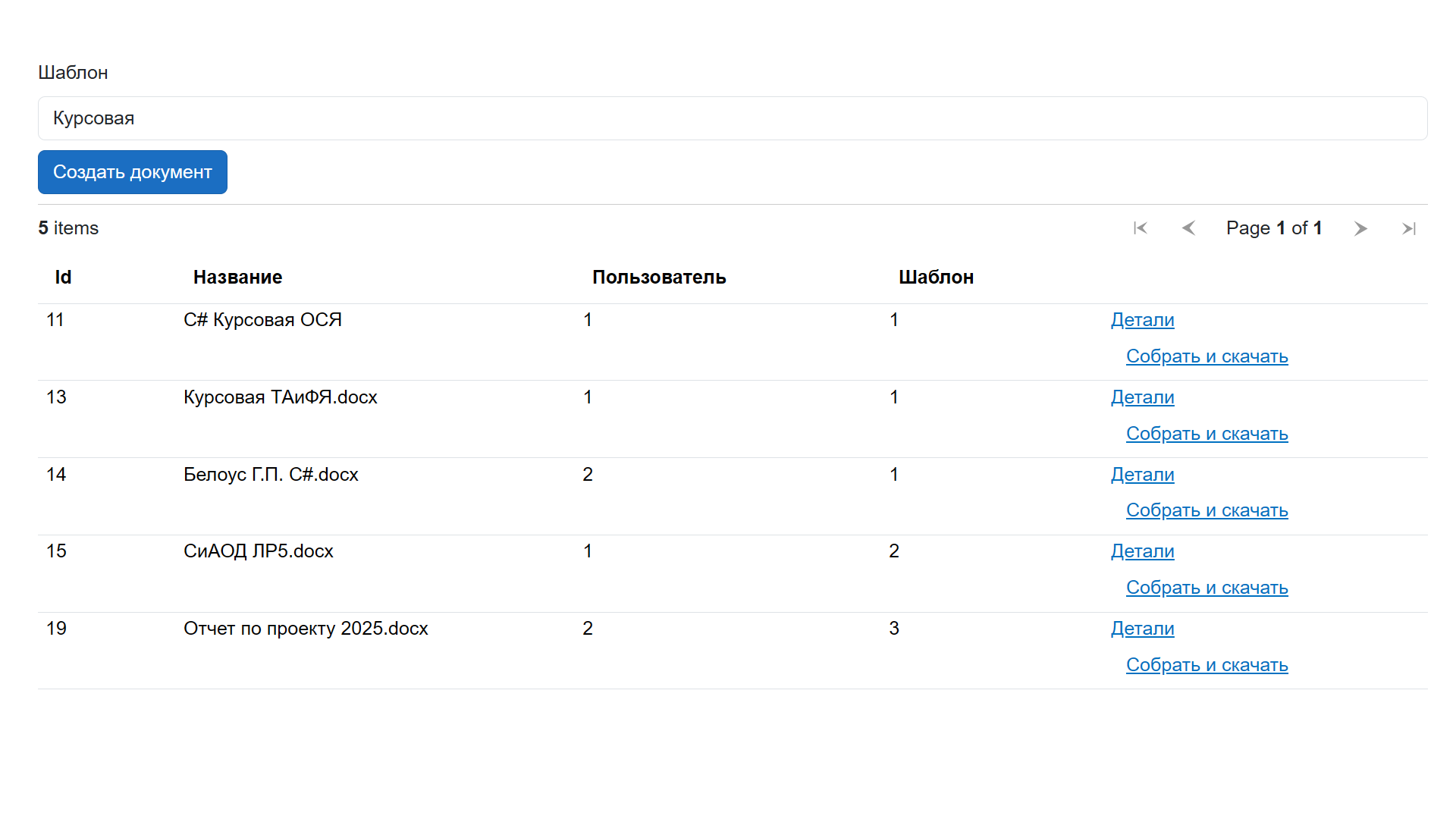


Рис. 6. Пример окна выбора документа

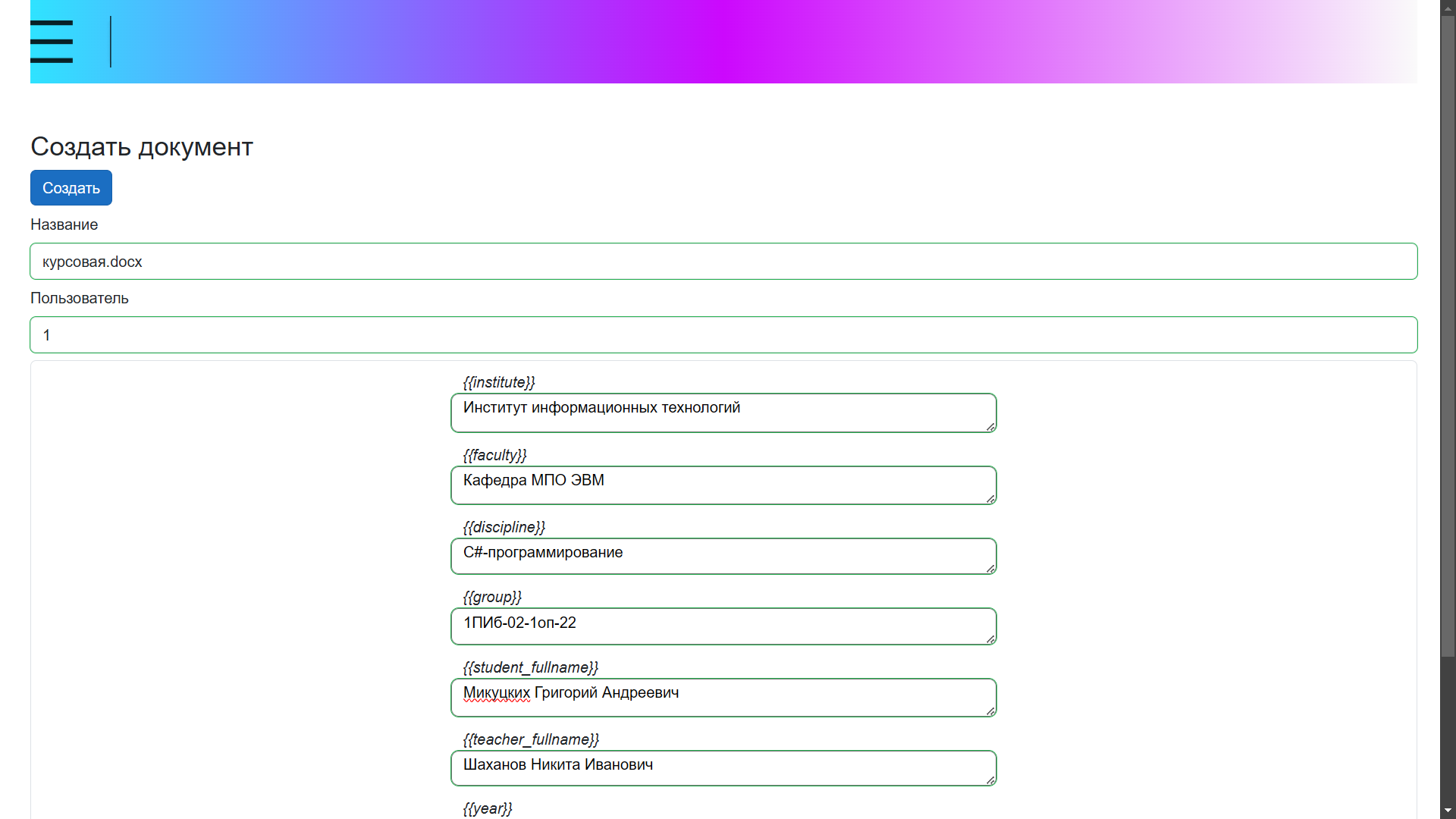


Рис. 7. Пример окна редактирования документа

Ниже представлены примеры содержимого базы данных через интерфейс программы «pgAdmin 4» (рис. 8-9).

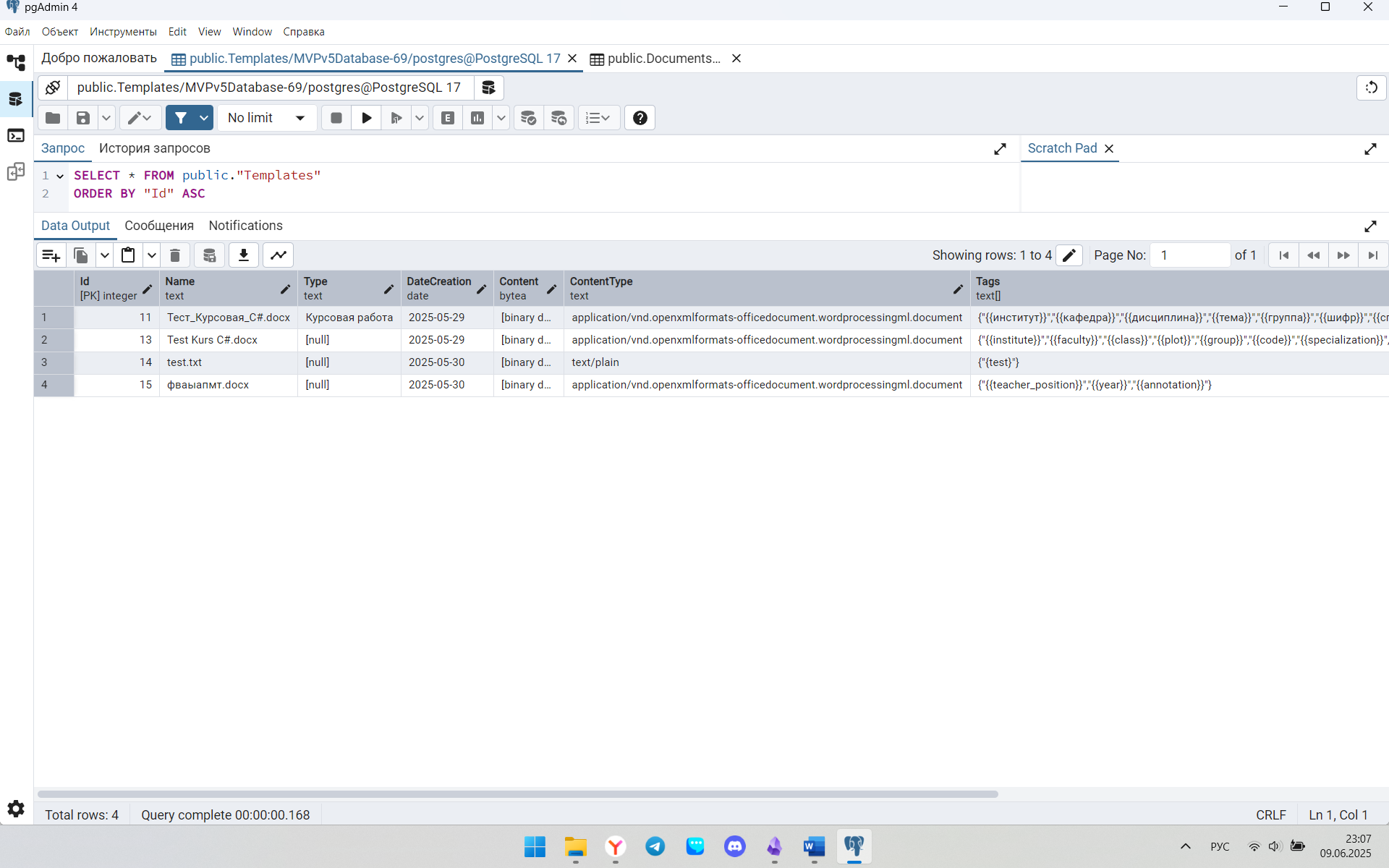


Рис. 8. Пример вывода информации записей таблицы «Шаблоны»

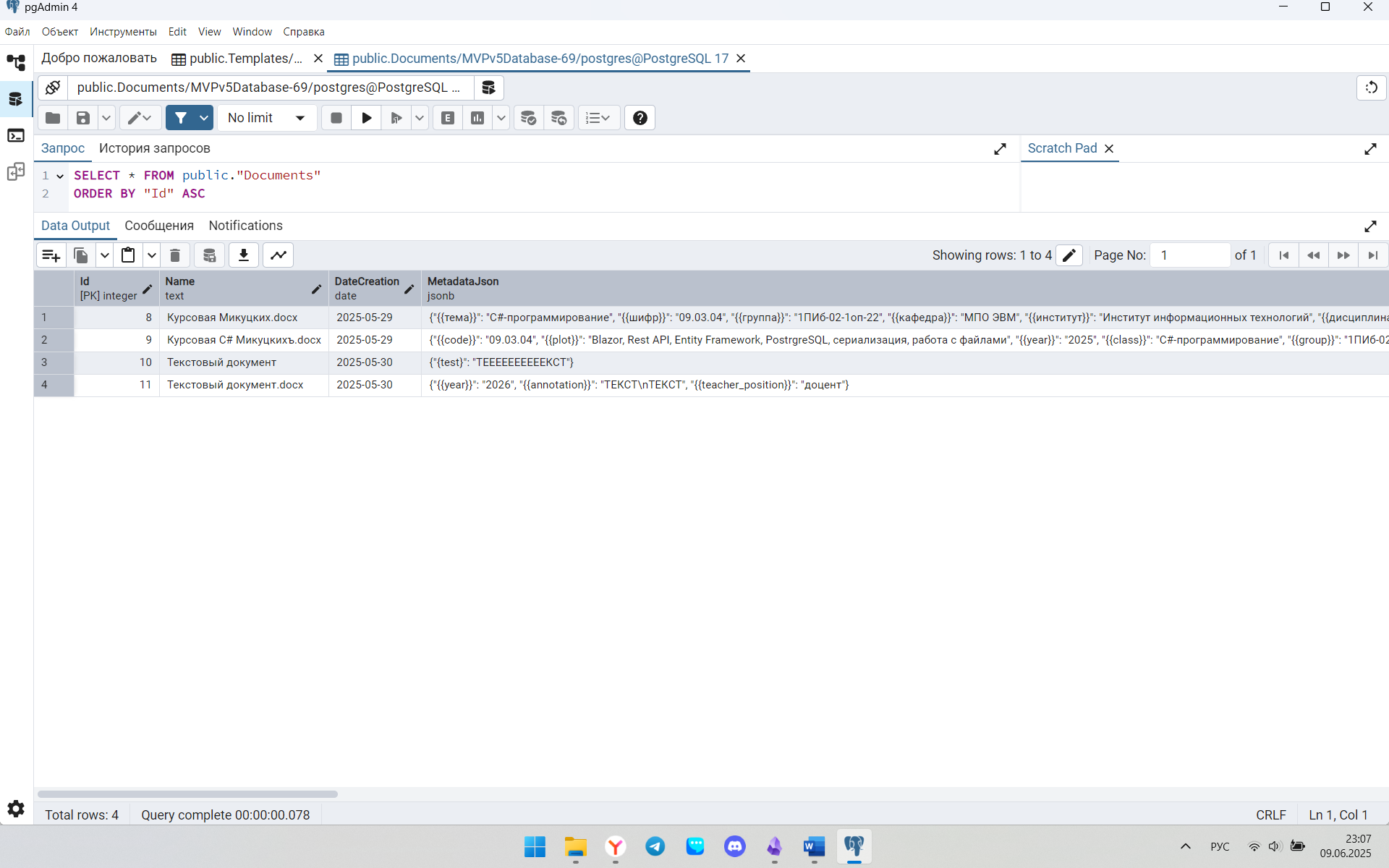


Рис. 9. Пример вывода информации записей таблицы «Документы»

## Разработка контроллера (модели представления и серверной части)

Были разработаны контроллеры, предоставляющие возможность работать с данными пользователей, шаблонов и документов. Авторизация (проверка прав доступа пользователя к ресурсам после успешной аутентификации) реализована не была, поэтому приложение не безопасно от посещения страниц неавторизованными пользователями.

Разработаны следующие контроллеры:

* UserController.cs
* TemplateController.cs
* DocumentController.cs

## Разработка модели (база данных и логика)

Для соблюдения архитектур MVC И MVVM разработаны следующие типы объектов:

1. контракты (Contracts) – DTO-классы в качестве ViewModel, хранятся в отдельной библиотеке классов, доступной серверной и клиентской части;
2. модели (Models) – классы для передачи данных между средствами работы с базой данных и контроллером;
3. сущности (Entities) – классы для проектирования базы данных методом «сначала код» («Code first»).

Разработаны следующие сущности:

* UserEntity;
* TemplateEntity;
* DocumentEntity.

Разработаны следующие модели:

* UserModel;
* TemplateModel;
* DocumentModel.

Разработаны следующие контракты:

* UserCreateRequest;
* UserLoginRequest;
* UserPatchPasswordRequest;
* UserReadResponse;
* TemplateCreateRequest;
* TemplatePatchRequest;
* TemplateDownloadRequest;
* DocumentCreateRequest;
* DocumentPatchRequest;
* DocumentReadResponse.

Разработаны следующие классы-Service для валидации входных данных и вызовов методов для взаимодействия с базой данных:

* UserService;
* TemplateService;
* DocumentService.

Разработаны следующие классы-Repository для работы с базой данных:

* UserRepository;
* TemplateRepository;
* DocumentRepository.

Были выделены следующие интерфейсы из классов сервисов и репозиториев:

* IUserService;
* ITemplateService;
* IDocumentService;
* IUserRepository;
* ITemplateRepository;
* IDocumentRepository.

Для того, чтобы абстрагироваться от реализации классов, использован паттерн «Внедрение зависимостей» (Dependency Injection, DI). Пример реализации внедрения зависимостей языковыми конструкциями ASP.NET Core представлен ниже.

|  |
| --- |
| builder.Services.AddDbContextFactory<MVPv5DbContext>(options =>  options.UseNpgsql(builder.Configuration.GetConnectionString(nameof(MVPv5DbContext))  ?? throw new InvalidOperationException($"'{nameof(MVPv5DbContext)}' not found.")));  builder.Services.AddScoped<IUserService, UserService>();  builder.Services.AddScoped<IUserRepository, UserRepository>();  builder.Services.AddScoped<TemplateController>();  builder.Services.AddScoped<ITemplateService, TemplateService>();  builder.Services.AddScoped<ITemplateRepository, TemplateRepository>();  builder.Services.AddScoped<IDocumentService, DocumentService>();  builder.Services.AddScoped<IDocumentRepository, DocumentRepository>(); |

Чтобы вынести обработку исключений за пределы контроллеров и не дублировать код, был написан свой слой Middleware в файле «ExMiddleware.cs».

Пример структуры итогового проекта представлен на рис. 10.

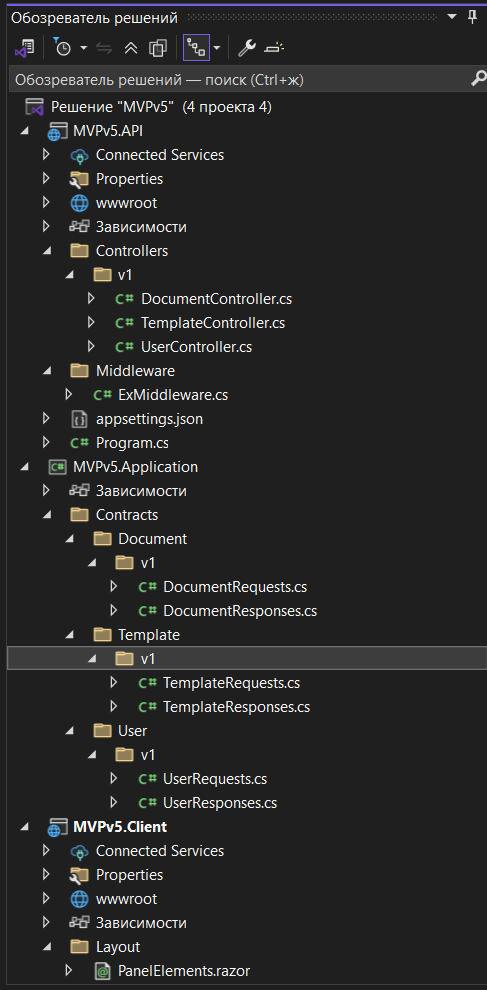
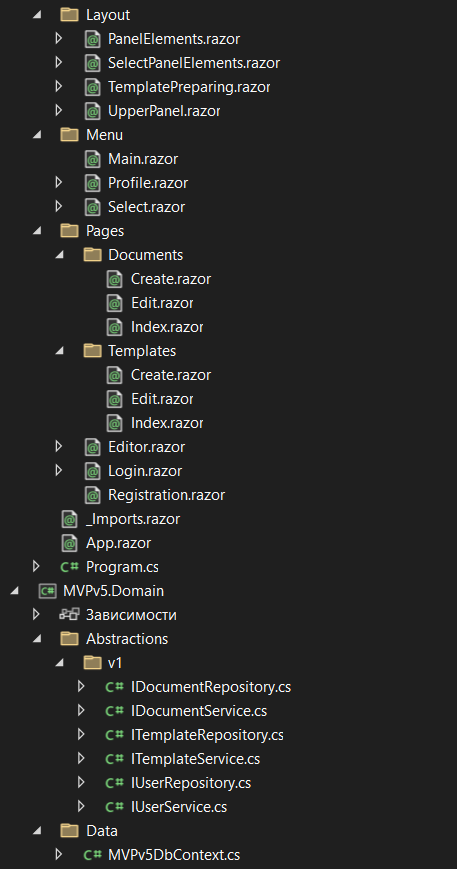
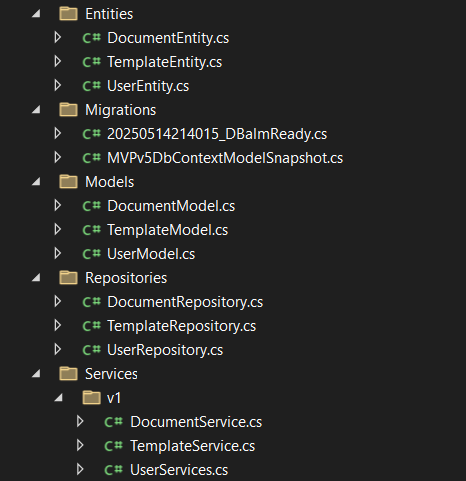


Рис. 10. Структура решения итогового проекта

Продолжение рис. 10



Продолжение рис. 10



Код программы представлен в «Приложение 2. Текст программы». Подробнее про работу программы в «Приложение 3. Руководство пользователя».

## Основная роль в команде

Автор расчётно-пояснительной записки являлся тимлидом (лидером команды) и главным разработчиком в команде из семи человек. Работа была организована с помощью системы контроля версий Git. В качестве хостинга выбран GitHub. Пример графа коммитов представлен на рис. 11.

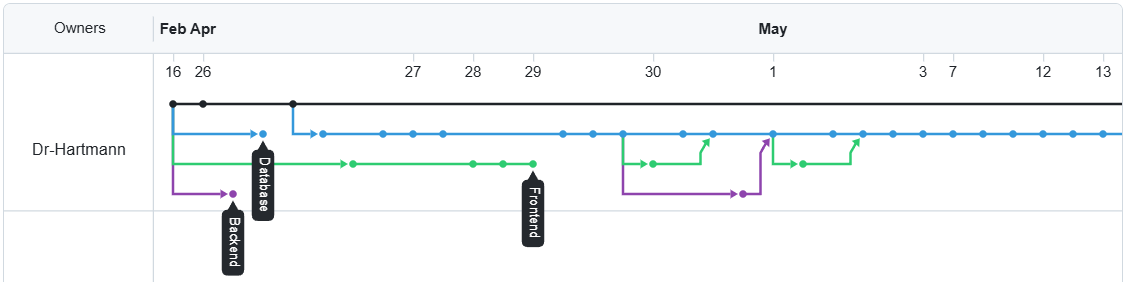


Рис. 11. Пример графа коммитов

Основными каналами для коммуникации выступали программы «Telegram» (общение и постановка задач) и «Discord» (видео-встречи) (рис. 12-13).

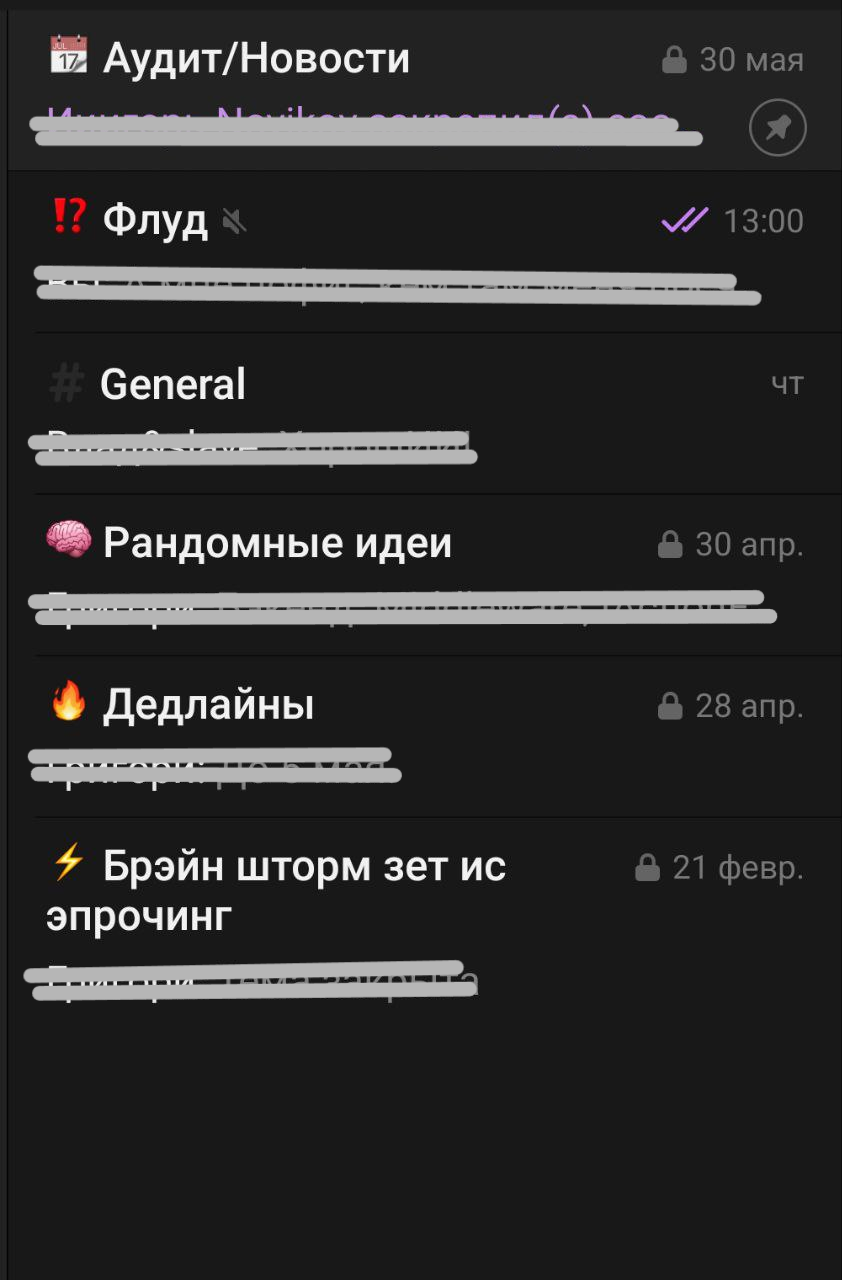


Рис. 12. Пример рабочего чата в «Telegram»

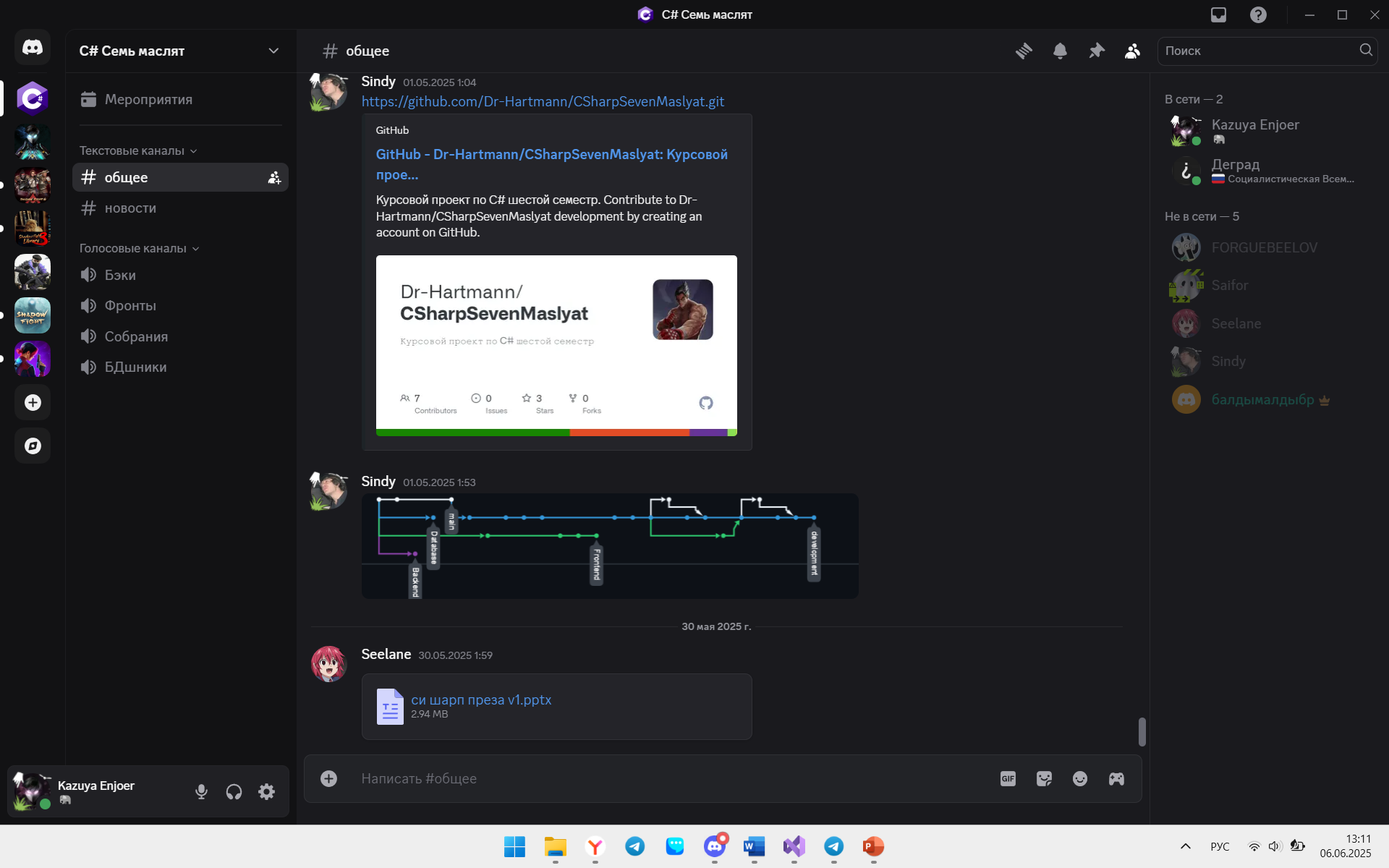


Рис. 13. Пример рабочего чата в «Discord»

В ходе работы возникали различные трудности:

1. недостаточно ответственный подход к работе от некоторых участников;
2. сокращение команды – трое участников потенциально могли покинуть проект из-за трудностей в учебном процессе и возможности отчисления;
3. низкий уровень навыков, а именно hard-skills у некоторых участников и soft-skills у тимлида;
4. некорректная постановка задач и сроков их выполнения.

Итогом является слабая заинтересованность всех участников команды в реализации приложения, из-за чего был реализован лишь основной функционал практически без средств обеспечения безопасности, оптимизации кода и полностью отсутствует тестирование.

# Заключение

В ходе работы был написан основной функционал клиент-серверного приложения «Составитель курсовой работы/отчёта по требованиям кафедры МПО ЭВМ» на ЯВУ «C#» с использованием фреймворков «ASP.NET Core», «Blazor» и «EF Core».

Были реализованы аутентификация, обработка исключений на своём уровне Middleware, обработка http-запросов, асинхронная работа методов работы с базой данных PostgreSQL, простой и понятный пользовательский интерфейс.

# Список литературы

1. Методика и организация самостоятельной работы студентов / Е.В. Ершов, д-р техн. наук, проф.; Л.Н. Виноградова, канд. техн. наук; В.В. Селивановских, канд. техн. наук, доцент; О.Л. Селяничев, канд. техн. наук, доцент; О.В. Юдина; О.С. Сальникова; Н.Е. Сивков; вед. ред. Г.В. Иванова, вед. тех. редактор М.Н. Авдюхова Е.В. – ФГБОУ ВПО «Череповецкий государственный университет» 162600 г. Череповец, пр. Луначарского, 5., 2015. – 243 c.
2. Что такое Windows Forms - Windows Forms | Microsoft Learn– Электронный ресурс. – URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/ winforms/overview/?view=netdesktop-9.0 (05.06.25).
3. Что такое Windows Presentation Foundation - WPF | Microsoft Learn – Электронный ресурс. – URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/ overview/?view=netdesktop-9.0&viewFallbackFrom=netdesktop-5.0 (05.06.25).
4. Общие сведения об ASP.NET Core | Microsoft Learn – Электронный ресурс. – URL: https://learn.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core? view=aspnetcore-9.0 (05.06.25).
5. Самые популярные СУБД в начале 2025 года. | Python.Study – Электронный ресурс. – URL: https://db-engines.com/en/ (18.05.25).
6. REST, что же ты такое? Понятное введение в технологию для ИТ-аналитиков / Хабр – Электронный ресурс. – URL: https://habr.com/ru/articles/590679/ (06.06.25).
7. Архитектура приложений и интеграции: гайд по основным понятиям простыми словами / Хабр – Электронный ресурс. – URL: https://habr.com/ru/companies/itq\_group/articles/705598/ (06.06.25).
8. Архитектура программного обеспечения: виды, для чего нужна - Ycla Coding – Электронный ресурс. – URL: https://ycla-coding.com/blog/arhitektura-programmnogo-obespecheniya-vidy-dlya-chego-nuzhna (06.06.25).
9. NuGet Gallery | DocumentFormat.OpenXml.Framework 3.3.0 – Эл. р-с. – URL: https://www.nuget.org/packages/DocumentFormat.OpenXml.Framework (09.06.25).

# Приложение 1

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий

наименование института (факультета)

Математическое и программное обеспечение ЭВМ

наименование кафедры

С#-программирование

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ,

д.т.н., профессор \_\_\_\_\_\_\_\_\_\_ Ершов Е.В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ЯЗЫКЕ C#

Техническое задание на курсовую работу

Листов 7

|  |  |
| --- | --- |
| Руководитель | Шаханов Н.И. |
|  | Ф.И.О преподавателя |
| Исполнитель |  |
| студент | 1ПИб-02-1оп-22 |
|  | группа |
|  | Микуцких Г.А. |
|  | Фамилия, имя, отчество |

2025 год

Введение

Написание курсовой работы или отчёта по лабораторной работе помимо содержания включает в себя оформление, которое может меняться во времени или по требованию преподавателя. Чтобы не приходилось вручную настраивать оформление и сосредоточиться только на содержании, необходимо иметь текстовый редактор, который возьмёт на себя ответственность за оформление и по окончании работы соберёт итоговый файл в формате DOCX.

Целью курсовой работы является разработка приложения «Редактор программной документации» для студентов ЧГУ кафедры МПО ЭВМ.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине «С#-программирование», выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения: 21 февраля 2025 года.

Наименование темы разработки: «Разработка программного обеспечения на языке C#».

1. Назначение разработки

Освоение на практике материала, полученного в ходе изучения дисциплины «С#-программирование».

1. Требования к приложению
2. Требования к функциональным характеристикам

Приложение должно обладать следующими основными качествами:

1. возможность загружать свои шаблоны;
2. возможность создавать и редактировать содержание документа;
3. сборка и скачивание итогового документа.
4. примерный вид интерфейса и расположение основных его элементов отображено на макетах на рис. П1.1-П1.2.

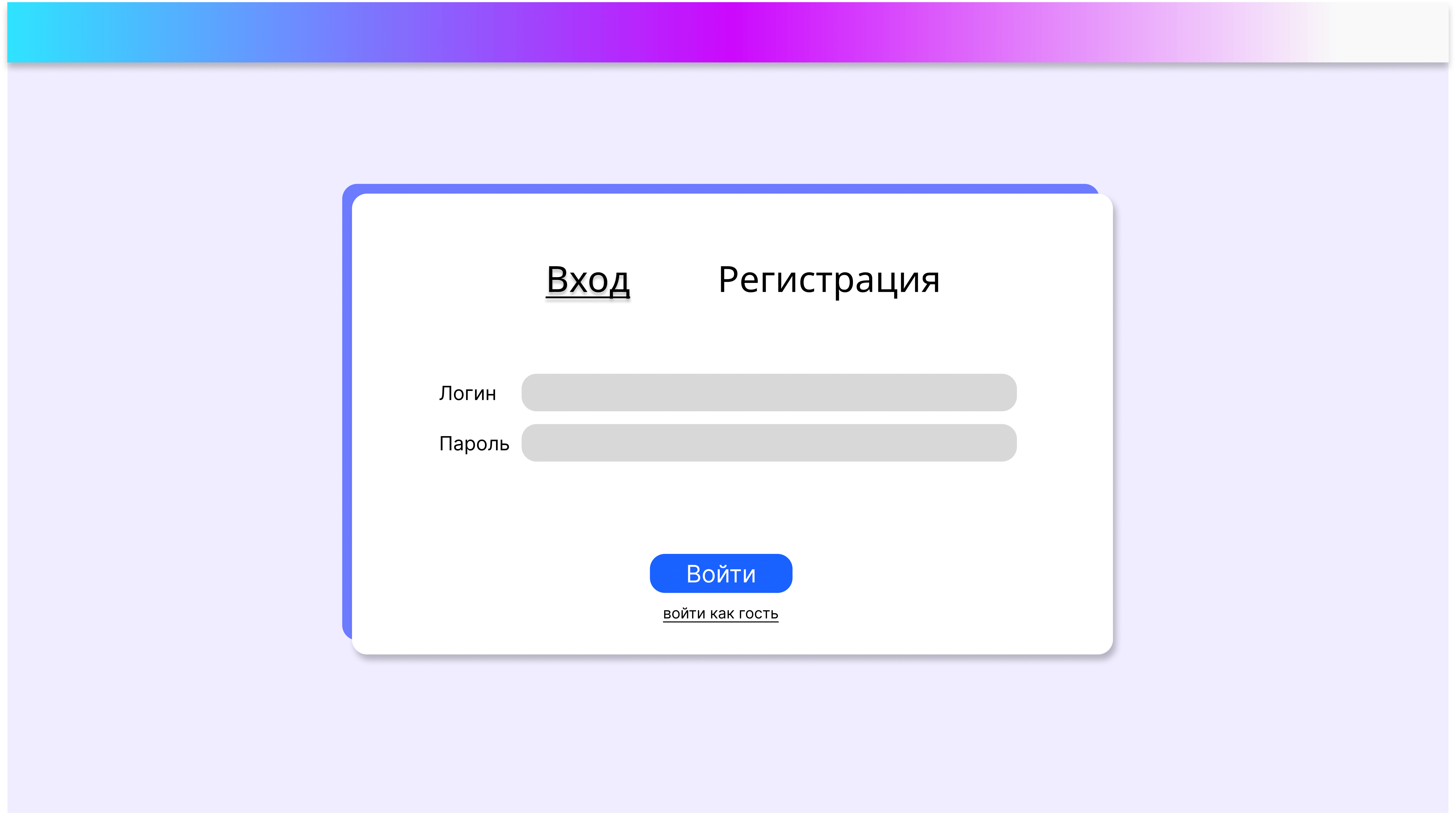


Рис. П1.1. Примерный макет окна регистрации

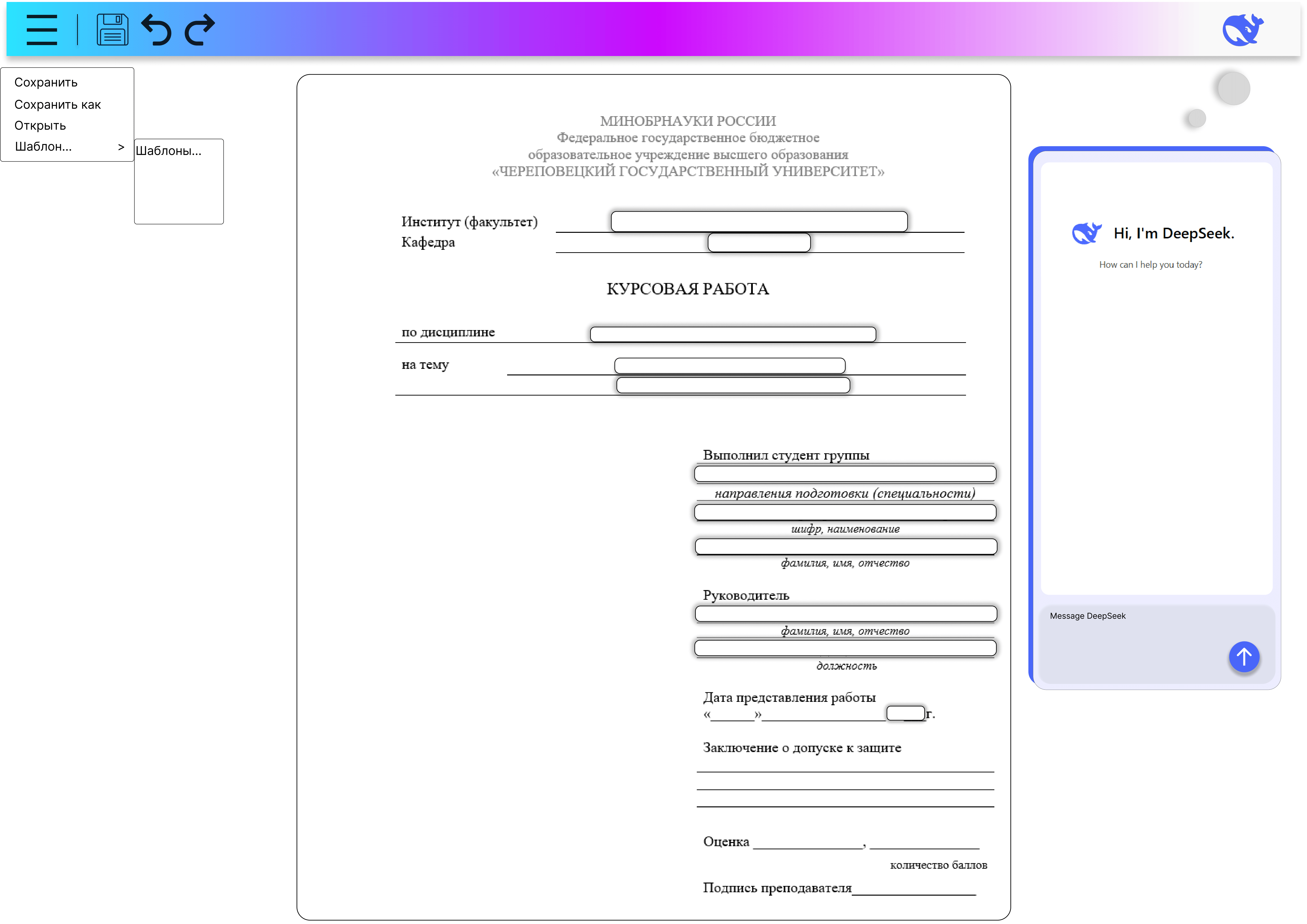


Рис. П1.2. Примерный макет окна редактирования документа

1. Требования к надёжности

Предъявляются следующие требования надёжности:

1. должна быть обеспечена защита от неправильных загружаемых данных пользователем и реализована обработка соответствующих исключений;
2. данные пользователей должны быть защищены от несанкционированного доступа.
3. Условия эксплуатации

Соответствуют с условиями эксплуатации пользователя ПК.

1. Требования к составу и параметрам технических средств

ЭВМ должна соответствовать следующим параметрам:

* 64-разрядный (x64) процессор с тактовой частотой 1 ГГц или выше;
* 2 ГБ ОЗУ, 128 ГБ свободного места на накопителе (HDD, SSD);
* графическое устройство DirectX 9 с драйвером WDDM 1.0 или новее;
* поддержка клавиатуры, компьютерной мыши, монитора разрешением 1920×1080 пикселей.

1. Требования к маркировке и упаковке

Приложение должно корректно работать на ОС Windows 10 и выше.

1. Требования к транспортированию и хранению

Всем пользователям запрещается осуществлять действия под видом оригинального ПО, нарушающие статью 273 «Создание, использование и распространение вредоносных компьютерных программ».

1. Специальные требования

Для эффективной работы с приложением требуется наличие опыта работы в текстовых редакторах, программах пакета «Microsoft Office», в частности «Word», и «сёрфинга» в Интернете.

1. Требование к программной документации
2. Содержание расчётно-пояснительной записки

Программная документация должна содержать расчётно-пояснительную записку, содержание которой:

Титульный лист

Аннотация

Оглавление

Введение

Теоретическая часть

Анализ предметной области

Обоснование выбора стека

Практическая часть

Заключение

Список литературы

Приложения

1. Требования к оформлению

Элементы курсовой работы оформлены в соответствии с табл. П1.1.

Таблица П1.1

Требования к оформлению

|  |  |
| --- | --- |
| Элемент | Требования |
| Документ | 1. Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б.  2. Файлы предъявляются на компакт-диске: РПЗ с ТЗ; программный код.  3. Листы и диск в конверте вложены в пластиковую папку скоросшивателя. |
| Страницы | 1. Ориентация – книжная; отдельные страницы, при необходимости, альбомная.  2. Поля: верхнее, нижнее – по 2 см, левое – 3 см, правое – 1 см. |
| Абзацы | Межстрочный интервал – 1.5, перед и после абзаца – 0. |
| Шрифты | Кегль – 14. В таблицах шрифт 12. Шрифт программного кода – 8 (возможно в 2 колонки). |
| Рисунки | Подписывается под ним по центру: Рис. Х. Название  В приложениях: Рис. ПX.X. Название |
| Таблицы | 1. Подписывается: над таблицей, выравнивание по правому: «Таблица Х».  2. В следующей строке по центру Название  3. Надписи в «шапке» (имена столбцов, полей) – по центру.  4. В теле таблицы (записи) текстовые значения – выровнены по левому краю, числа, даты – по правому. |

1. Стадии и этапы разработки

Курсовая работа и программа будут разрабатываться в течение следующих этапов (табл. П1.2).

Таблица П1.2

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование  этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Оформление технического задания | 01.03.25 - 06.03.25 | Составлено техническое задание |  |
| Изучение предметной области | 08.03.25 – 22.03.25 | Предметная область изучена, опре-делены основные структуры данных |  |
| Кодирование | 23.03.25 –  20.05.25 | Приложение написано |  |
| Тестирование | 29.04.25 – 29.05.25 | Приложение отлажено |  |
| Оформление расчётно-пояснительной записки | 08.05.25 – 17.06.25 | Оформлена расчётно-пояснительная записка |  |

1. Порядок контроля и приёмки

Курсовая работа будет приниматься в течение этапов в табл. П1.3.

Таблица П1.3

Порядок контроля и приёмки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения курсовой работы | Сроки  контроля | Результат выполнения | Отметка о приёмке результата контрольного этапа |
| Сдача технического задания |  | Техническое задание принято |  |
| Исправление ошибок |  | Ошибки приложения исправлены |  |
| Демонстрация приложения |  | Приложение принято |  |
| Сдача расчётно-пояснительной записки |  | Расчётно-пояснительная записка принята |  |
| Защита курсовой работы |  | Курсовая работа защищена |  |

# Приложение 2

Текст программы.

Файл «UserRequests.cs»

using System.ComponentModel.DataAnnotations;

namespace MVPv5.Application.Contracts.User.v1;

public class UserCreateRequest

{

[Required, StringLength(20, MinimumLength = 2)]

public string Nickname { get; set; }

[Required, StringLength(50, MinimumLength = 10, ErrorMessage = "Не короче 5 и не больше 30 символов"),

RegularExpression("^[A-Za-z0-9\_-]+@chsu.ru$", ErrorMessage = "Должен быть почтой ЧГУ")]

public string Login { get; set; }

[Required, StringLength(30, MinimumLength = 5, ErrorMessage = "Не короче 5 и не больше 30 символов")]

public string Password { get; set; }

[Compare("Password", ErrorMessage = "Пароли не совпадают")]

public string PasswordConfirm { get; set; }

}

public class UserLoginRequest

{

[Required, StringLength(50, MinimumLength = 10, ErrorMessage = "Не короче 5 и не больше 30 символов"),

RegularExpression("^[A-Za-z0-9\_-]+@chsu.ru$", ErrorMessage = "Должен быть почтой ЧГУ")]

public string Login { get; set; }

[Required, StringLength(30, MinimumLength = 5, ErrorMessage = "Не короче 5 и не больше 30 символов")]

public string Password { get; set; }

}

public class UserPatchPasswordRequest

{

[Required, StringLength(50, MinimumLength = 10)]

public required string Login { get; set; }

[StringLength(30, MinimumLength = 5)]

public required string Password { get; set; }

[Compare("Password", ErrorMessage = "Пароли не совпадают")]

public required string PasswordConfirm { get; set; }

}

Файл «UserResponse.cs»

namespace MVPv5.Application.Contracts.User.v1;

public record UserReadResponse(

int Id,

string Nickname,

string Login,

string Password,

byte AccessRule,

DateOnly DateCreation);

Файл «TemplateRequests.cs»

using System.ComponentModel.DataAnnotations;

namespace MVPv5.Application.Contracts.Template.v1;

public class TemplateCreateRequest

{

[Required, MaxLength(100)]

public string? Name { get; set; }

public string? Type { get; set; }

[Required]

public byte[]? Content { get; set; }

[Required]

public string? ContentType { get; set; }

public IEnumerable<string>? Tags { get; set; }

}

public class TemplatePatchRequest

{

public int Id { get; set; }

[Required, MaxLength(100)]

public string? Name { get; set; }

public string? Type { get; set; }

[Required]

public byte[]? Content { get; set; }

[Required]

public string? ContentType { get; set; }

public IEnumerable<string>? Tags { get; set; }

}

public record TemplateDownloadRequest(

[Required] string Name,

[Required] byte[] Content,

[Required] string ContentType);

Файл «TemplateResponse.cs»

namespace MVPv5.Application.Contracts.Template.v1;

public record TemplateReadResponse(

int Id,

string? Name,

string? Type,

DateOnly? DateCreation,

byte[]? Content,

string? ContentType,

IEnumerable<string>? Tags);

Файл «DocumentRequests.cs»

using System.ComponentModel.DataAnnotations;

namespace MVPv5.Application.Contracts.Document.v1;

public class DocumentCreateRequest

{

[Required]

public string? Name { get; set; }

public IDictionary<string, string>? Data { get; set; }

[Required]

public int TemplateId { get; set; }

[Required]

public int UserId { get; set; }

}

public class DocumentPatchRequest

{

[Required]

public required int Id { get; set; }

[Required]

public string? Name { get; set; }

public IDictionary<string, string>? Data { get; set; }}

Файл «DocumentResponses.cs»

namespace MVPv5.Application.Contracts.Document.v1;

public record DocumentReadResponse(

int Id,

string? Name,

DateOnly? DateCreation,

IDictionary<string, string>? Data,

int TemplateId,

int UserId);

Файл «DocumentEntity.cs»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Text.Json;

namespace MVPv5.Domain.Entities;

public class DocumentEntity : IDisposable

{

[Key]

public int Id { get; set; }

[Required]

public required string Name { get; set; }

[Required]

public required DateOnly DateCreation { get; set; }

[Column(TypeName = "jsonb")]

public JsonDocument? MetadataJson { get; set; }

[ForeignKey(nameof(TemplateEntity))]

public required int TemplateId { get; set; }

public TemplateEntity? Template { get; set; }

[ForeignKey(nameof(UserEntity))]

public required int UserId { get; set; }

public UserEntity? User { get; set; }

public void Dispose() => MetadataJson?.Dispose();

}

Файл «DocumentModel.cs»

using System.ComponentModel.DataAnnotations;

using System.Text.Json;

using MVPv5.Domain.Entities;

namespace MVPv5.Domain.Models;

public class DocumentModel

{

private DocumentModel(DocumentEntity entity)

{

Id = entity.Id;

Name = entity.Name;

TemplateId = entity.TemplateId;

UserId = entity.UserId;

DateCreation = entity.DateCreation;

Dictionary = entity.MetadataJson?.Deserialize<IDictionary<string, string>>();

}

private DocumentModel(string? name, DateOnly? dateCreation, IDictionary<string, string>? dictionary, int? templateId, int? userId)

{

Name = name ?? string.Empty;

DateCreation = dateCreation ?? DateOnly.FromDateTime(DateTime.Now);

Dictionary = dictionary;

TemplateId = templateId ?? default;

UserId = userId ?? default;

}

public int Id { get; set; }

public string Name { get; set; }

public DateOnly DateCreation { get; set; }

public IDictionary<string, string>? Dictionary { get; set; }

public int TemplateId { get; set; }

public int UserId { get; set; }

public static (DocumentModel Document, string Error) Create(DocumentEntity? entity)

{

if (entity is null)

{

throw new ValidationException("Передана пустая сущность");

}

var error = string.Empty;

// TODO

return (new DocumentModel(entity), error);

}

public static DocumentModel Create(string? name, DateOnly? dateCreation, IDictionary<string, string>? dictionary, int? templateId, int? userId)

{

if (string.IsNullOrEmpty(name))

{

throw new ValidationException("Некорректные данные");

}

// TODO

return new DocumentModel(name, dateCreation, dictionary, templateId, userId);

}}

Файл «DocumentRepository.cs»

using Microsoft.EntityFrameworkCore;

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Data;

using MVPv5.Domain.Entities;

using MVPv5.Domain.Models;

using System.ComponentModel.DataAnnotations;

using System.Text.Json;

namespace MVPv5.Domain.Repositories;

public class DocumentRepository(MVPv5DbContext dbContext) : IDocumentRepository

{

public async Task AddAsync(DocumentModel model, CancellationToken token)

{

if (await dbContext.Documents.AnyAsync(u => u.Name == model.Name, token))

{

throw new ValidationException("Такой документ уже существует");

}

await dbContext!.Documents.AddAsync(new DocumentEntity

{

Name = model.Name,

DateCreation = model.DateCreation!,

MetadataJson = JsonDocument.Parse(JsonSerializer.Serialize(model.Dictionary)),

TemplateId = model.TemplateId,

UserId = model.UserId,

}, token);

await dbContext.SaveChangesAsync(token);

}

public async Task<(DocumentModel Document, string Error)> GetByIdAsync(int id, CancellationToken token)

{

return GetDocument(await dbContext.Documents

.AsNoTracking()

.FirstOrDefaultAsync(d => d.Id == id, token));

}

public async Task<IEnumerable<(DocumentModel Document, string Error)>> GetAllAsync(CancellationToken token)

{

return GetListOfDocuments(await dbContext.Documents

.AsNoTracking()

.ToListAsync(token));

}

public async Task UpdateNameAsync(int id, string name, CancellationToken token)

{

await dbContext.Documents

.Where(document => document.Id == id)

.ExecuteUpdateAsync(document => document

.SetProperty(u => u.Name, name),

token);

await dbContext.SaveChangesAsync(token);

}

public async Task UpdateMetaDataAsync(int id, IDictionary<string, string> metadataJson, CancellationToken token)

{

var a = JsonDocument.Parse(JsonSerializer.Serialize(metadataJson));

await dbContext.Documents

.Where(document => document.Id == id)

.ExecuteUpdateAsync(document => document

.SetProperty(u => u.MetadataJson, a),

token);

await dbContext.SaveChangesAsync(token);

}

public async Task UpdateAsync(int id, DocumentModel model, CancellationToken token)

{

var document = await dbContext.Documents.FirstOrDefaultAsync(d => d.Id == model.Id, token);

if (document == null)

{

throw new Exception("Документ не найден");

}

if (document.Name != model.Name && await dbContext.Documents.AnyAsync(d => d.Name == model.Name, token))

{

throw new Exception("Документ с таким именем уже существует");

}

document.Name = model.Name;

document.MetadataJson = JsonDocument.Parse(JsonSerializer.Serialize(model.Dictionary));

document.TemplateId = model.TemplateId;

document.UserId = model.UserId;

await dbContext.SaveChangesAsync(token);

}

public async Task DeleteById(int id, CancellationToken token)

{

var count = await dbContext.Documents

.Where(document => document.Id == id)

.ExecuteDeleteAsync(token);

if (count != 1)

{

throw new DbUpdateException($"Удалено {count} документов вместо 1");

}

}

private (DocumentModel Document, string Error) GetDocument(DocumentEntity? response)

{

return DocumentModel.Create(response);

}

private IEnumerable<(DocumentModel Document, string Error)> GetListOfDocuments(IEnumerable<DocumentEntity>? response)

{

if (response == null) throw new Exception("Пустой лист в ответе");

return response.Select(GetDocument);

}}

Файл «DocumentService.cs»

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Models;

namespace MVPv5.Domain.Services.v1;

public class DocumentService(IDocumentRepository repository) : IDocumentService

{

public async Task CreateAsync(DocumentModel model, CancellationToken token)

{

await repository.AddAsync(model, token);

}

public async Task<DocumentModel> GetByIdAsync(int id, CancellationToken token)

{

var response = await repository.GetByIdAsync(id, token);

if (response.Error != string.Empty)

{

throw new KeyNotFoundException($"Ошибка: {response.Error}");

}

return response.Document;

}

public async Task<IEnumerable<DocumentModel>> GetAllAsync(CancellationToken token)

{

var response = await repository.GetAllAsync(token);

var errors = response.Where(resp => !string.IsNullOrWhiteSpace(resp.Error)).ToList();

if (errors.Any())

{

var errMessages = string.Join(" | ", errors.Select(e => e.Error));

throw new ApplicationException($"Обнаружены ошибки при получении документов: {errMessages}");

}

return response.Select(l => l.Document);

}

public async Task UpdateMetaDataById(int id, IDictionary<string, string>? metadataJson, CancellationToken token)

{

if (metadataJson is null)

{

throw new Exception("Данные тегов пусты");

}

await repository.UpdateMetaDataAsync(id, metadataJson, token);

}

public async Task UpdateNameAsync(int id, string? name, CancellationToken token)

{

if (string.IsNullOrEmpty(name))

{

throw new Exception("Имя пусто");

}

await repository.UpdateNameAsync(id, name, token);

}

public async Task UpdateAsync(int id, DocumentModel model, CancellationToken token)

{

await repository.UpdateAsync(id, model, token);

}

public async Task DeleteByIdAsync(int id, CancellationToken token)

{

await repository.DeleteById(id, token);

}}

Файл «IDocumentRepository.cs»

using MVPv5.Domain.Models;

namespace MVPv5.Domain.Abstractions.v1;

public interface IDocumentRepository

{

Task AddAsync(DocumentModel model, CancellationToken token);

Task<(DocumentModel Document, string Error)> GetByIdAsync(int id, CancellationToken token);

Task<IEnumerable<(DocumentModel Document, string Error)>> GetAllAsync(CancellationToken token);

Task UpdateNameAsync(int id, string name, CancellationToken token);

Task UpdateMetaDataAsync(int id, IDictionary<string, string> metadataJson, CancellationToken token);

Task UpdateAsync(int id, DocumentModel model, CancellationToken token);

Task DeleteById(int id, CancellationToken token);}

Файл «IDocumentService.cs»

using MVPv5.Domain.Models;

namespace MVPv5.Domain.Abstractions.v1;

public interface IDocumentService

{

Task CreateAsync(DocumentModel model, CancellationToken token);

Task<DocumentModel> GetByIdAsync(int id, CancellationToken token);

Task<IEnumerable<DocumentModel>> GetAllAsync(CancellationToken token);

Task UpdateNameAsync(int id, string? name, CancellationToken token);

Task UpdateMetaDataById(int id, IDictionary<string, string>? metadataJson, CancellationToken token);

Task UpdateAsync(int id, DocumentModel model, CancellationToken token);

Task DeleteByIdAsync(int id, CancellationToken token);}

Файл «TemplateEntity.cs»

using System.ComponentModel.DataAnnotations;

namespace MVPv5.Domain.Entities;

public class TemplateEntity

{

[Key]

public int Id { get; set; }

[Required]

public required string Name { get; set; }

public string? Type { get; set; }

public DateOnly DateCreation { get; set; }

[Required]

public required byte[] Content { get; set; }

[Required]

public required string ContentType { get; set; }

[Required]

public required string[] Tags { get; set; }

}

Файл «TemplateModel.cs»

using MVPv5.Domain.Entities;

namespace MVPv5.Domain.Models;

public class TemplateModel

{

private TemplateModel(TemplateEntity entity)

{

Id = entity.Id;

Name = entity.Name;

Type = entity.Type;

DateCreation = entity.DateCreation;

Content = entity.Content;

ContentType = entity.ContentType;

Tags = entity.Tags;

}

private TemplateModel(string name, string? type, DateOnly? dateCreation, byte[] content,

string contentType, IEnumerable<string>? tags)

{

Name = name;

Type = type;

DateCreation = dateCreation ?? DateOnly.FromDateTime(DateTime.Now);

Content = content;

ContentType = contentType;

Tags = tags;

}

public int Id { get; }

public string Name { get; }

public string? Type { get; }

public DateOnly DateCreation { get; }

public byte[] Content { get; }

public string ContentType { get; }

public IEnumerable<string>? Tags { get; set; }

public static (TemplateModel Template, string Error) Create(TemplateEntity? entity)

{

if(entity is null)

{

throw new Exception("Передана пустая сущность");

}

var error = string.Empty;

// TODO

return (new TemplateModel(entity), error);

}

public static TemplateModel Create(string name, string? type, DateOnly? dateCreation,

byte[] content, string contentType, IEnumerable<string>? tags)

{

// TODO

return new TemplateModel(name, type, dateCreation, content, contentType, tags);

}}

Файл «TemplateRepository.cs»

using Microsoft.EntityFrameworkCore;

using MVPv5.Domain.Data;

using MVPv5.Domain.Entities;

using MVPv5.Domain.Models;

using System.ComponentModel.DataAnnotations;

namespace MVPv5.Domain.Repositories;

public class TemplateRepository(MVPv5DbContext dbContext) : ITemplateRepository

{

public async Task AddAsync(TemplateModel model, CancellationToken token)

{

if (await dbContext.Templates.AnyAsync(u => u.Name == model.Name, token))

{

throw new ValidationException("Такой шаблон уже существует");

}

await dbContext!.Templates.AddAsync(new TemplateEntity

{

Name = model.Name,

Type = model.Type,

DateCreation = model.DateCreation!,

Content = model.Content,

ContentType = model.ContentType,

Tags = [.. model.Tags ?? Array.Empty<string>()]

}, token);

await dbContext.SaveChangesAsync(token);

}

public async Task<(TemplateModel Template, string Error)> GetByIdAsync(int id, CancellationToken token)

{

return GetTemplate(await dbContext.Templates

.AsNoTracking()

.FirstOrDefaultAsync(t => t.Id == id, token));

}

public async Task<IEnumerable<(TemplateModel Template, string Error)>> GetAllAsync(CancellationToken token)

{

return GetListOfTemplates(await dbContext.Templates

.AsNoTracking()

.ToListAsync(token));

}

public async Task UpdateNameAsync(int id, string name, CancellationToken token)

{

await dbContext.Templates

.Where(template => template.Id == id)

.ExecuteUpdateAsync(template => template

.SetProperty(t => t.Name, name),

token);

await dbContext.SaveChangesAsync(token);

}

public async Task UpdateTypeAsync(int id, string type, CancellationToken token)

{

await dbContext.Templates

.Where(template => template.Id == id)

.ExecuteUpdateAsync(template => template

.SetProperty(t => t.Type, type),

token);

await dbContext.SaveChangesAsync(token);

}

public async Task UpdateContentAndContentTypeAsync(int id, byte[] content, string contentType, CancellationToken token)

{

await dbContext.Templates

.Where(template => template.Id == id)

.ExecuteUpdateAsync(template => template

.SetProperty(t => t.Content, content)

.SetProperty(t=> t.ContentType, contentType),

token);

await dbContext.SaveChangesAsync(token);

}

public async Task UpdateTagsAsync(int id, IEnumerable<string> tags, CancellationToken token)

{

await dbContext.Templates

.Where(template => template.Id == id)

.ExecuteUpdateAsync(template => template

.SetProperty(t => t.Tags, tags),

token);

await dbContext.SaveChangesAsync(token);

}

//public async Task UpdateAsync(int id, TemplateModel model, CancellationToken token)

//{

// var template = await dbContext.Templates.FirstOrDefaultAsync(t => t.Id == id, token);

// if (template == null)

// {

// throw new KeyNotFoundException($"Шаблон не найден (ID = {id})");

// }

// if (await dbContext.Templates.AnyAsync(t => t.Id != id && t.Name == model.Name, token))

// {

// throw new ValidationException("Шаблон с таким именем уже существует.");

// }

// template.Name = model.Name;

// template.Type = model.Type;

// template.Content = model.Content;

// template.ContentType = model.ContentType;

// template.Tags = [.. model.Tags];

// await dbContext.SaveChangesAsync(token);

//}

public async Task DeleteByIdAsync(int id, CancellationToken token)

{

var count = await dbContext.Templates

.Where(template => template.Id == id)

.ExecuteDeleteAsync(token);

if (count != 1)

{

throw new DbUpdateException($"Удалено {count} шаблонов вместо 1");

}

}

private (TemplateModel Template, string Error) GetTemplate(TemplateEntity? response)

{

return TemplateModel.Create(response);

}

private IEnumerable<(TemplateModel Template, string Error)> GetListOfTemplates(IEnumerable<TemplateEntity>? response)

{

if (response is null || !response.Any()) throw new Exception("Пустой лист в ответе");

return response.Select(GetTemplate);

}

}

Файл «TemplateService.cs»

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Models;

using MVPv5.Domain.Repositories;

namespace MVPv5.Domain.Services.v1;

public class TemplateService(ITemplateRepository repository) : ITemplateService

{

public async Task AddAsync(TemplateModel model, CancellationToken token)

{

await repository.AddAsync(model, token);

}

public async Task<TemplateModel> GetByIdAsync(int id, CancellationToken token)

{

var response = await repository.GetByIdAsync(id, token);

if (!string.IsNullOrWhiteSpace(response.Error))

{

throw new KeyNotFoundException($"Шаблон не найден (ID = {id}): {response.Error}");

}

return response.Template;

}

public async Task<IEnumerable<TemplateModel>> GetAllAsync(CancellationToken token)

{

var response = await repository.GetAllAsync(token);

var errors = response.Where(resp => !string.IsNullOrWhiteSpace(resp.Error)).ToList();

if (errors.Any())

{

var errMessages = string.Join(" | ", errors.Select(e => e.Error));

throw new ApplicationException($"Обнаружены ошибки при получении шаблонов: {errMessages}");

}

return response.Select(resp => resp.Template);

}

public async Task UpdateNameAsync(int id, string? name, CancellationToken token)

{

if (name is null) return;

await repository.UpdateNameAsync(id, name, token);

}

public async Task UpdateTypeAsync(int id, string? type, CancellationToken token)

{

if (type is null) return;

await repository.UpdateTypeAsync(id, type, token);

}

public async Task UpdateContentAndContentTypeAsync(int id, byte[]? content, string? contentType, CancellationToken token)

{

if (content is null || contentType is null) return;

await repository.UpdateContentAndContentTypeAsync(id, content, contentType, token);

}

public async Task UpdateTagsAsync(int id, IEnumerable<string>? tags, CancellationToken token)

{

if (tags is null) return;

await repository.UpdateTagsAsync(id, tags, token);

}

//public async Task UpdateAsync(int id, TemplateModel model, CancellationToken token)

//{

// await repository.UpdateAsync(id, model, token);

//}

public async Task DeleteByIdAsync(int id, CancellationToken token)

{

await repository.DeleteByIdAsync(id, token);

}}

Файл «ITemplateRepository.cs»

using MVPv5.Domain.Models;

namespace MVPv5.Domain.Repositories;

public interface ITemplateRepository

{

Task AddAsync(TemplateModel model, CancellationToken token);

Task<(TemplateModel Template, string Error)> GetByIdAsync(int id, CancellationToken token);

Task<IEnumerable<(TemplateModel Template, string Error)>> GetAllAsync(CancellationToken token);

Task UpdateNameAsync(int id, string name, CancellationToken token);

Task UpdateTypeAsync(int id, string type, CancellationToken token);

Task UpdateContentAndContentTypeAsync(int id, byte[] content, string contentType, CancellationToken token);

Task UpdateTagsAsync(int id, IEnumerable<string> tags, CancellationToken token);

//Task UpdateAsync(int id, TemplateModel model, CancellationToken token);

Task DeleteByIdAsync(int id, CancellationToken token);}

Файл «ITemplateService.cs»

using MVPv5.Domain.Models;

namespace MVPv5.Domain.Abstractions.v1;

public interface ITemplateService

{

Task AddAsync(TemplateModel model, CancellationToken token);

Task<TemplateModel> GetByIdAsync(int id, CancellationToken token);

Task<IEnumerable<TemplateModel>> GetAllAsync(CancellationToken token);

Task UpdateNameAsync(int id, string? name, CancellationToken token);

Task UpdateTypeAsync(int id, string? type, CancellationToken token);

Task UpdateContentAndContentTypeAsync(int id, byte[]? content, string? contentType, CancellationToken token);

Task UpdateTagsAsync(int id, IEnumerable<string>? tags, CancellationToken token);

//Task UpdateAsync(int id, TemplateModel model, CancellationToken token);

Task DeleteByIdAsync(int id, CancellationToken token);}

Файл «UserEntity.cs»

using System.ComponentModel.DataAnnotations;

namespace MVPv5.Domain.Entities;

public class UserEntity

{

[Key]

public int Id { get; set; }

[Required]

public required string Nickname { get; set; }

[Required]

public required string Login { get; set; }

[Required]

public required string Password { get; set; }

[Required]

public byte AccessRule { get; set; }

public DateOnly DateCreation { get; set; }

}

Файл «UserModel.cs»

using System.ComponentModel.DataAnnotations;

using MVPv5.Domain.Entities;

namespace MVPv5.Domain.Models;

public class UserModel

{

private UserModel(UserEntity entity)

{

Id = entity.Id;

Nickname = entity.Nickname;

Login = entity.Login;

Password = entity.Password;

AccessRule = entity.AccessRule;

DateCreation = entity.DateCreation;

}

private UserModel(string nickname, string login, string password, byte accessRule, DateOnly? dateCreation)

{

Nickname = nickname;

Login = login;

Password = password;

AccessRule = accessRule;

DateCreation = dateCreation ?? DateOnly.FromDateTime(DateTime.Now);

}

public int Id { get; }

public string Nickname { get; }

public string Login { get; }

public string Password { get; }

public byte AccessRule { get; }

public DateOnly DateCreation { get; }

public static (UserModel User, string Error) Create(UserEntity? entity)

{

if (entity is null)

{

throw new KeyNotFoundException("Пользователь не найден");

}

var error = string.Empty;

if (string.IsNullOrEmpty(entity.Nickname) || string.IsNullOrEmpty(entity.Login) || string.IsNullOrEmpty(entity.Password))

{

error = "Никнейм, логин или пароль пусты";

}

return (new UserModel(entity), error);

}

public static UserModel Create(string nickname, string login, string password, byte accessRule, DateOnly? dateCreation)

{

if (string.IsNullOrEmpty(nickname) || string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password))

{

throw new ValidationException("Ошибка валидации");

}

return new UserModel(nickname, login, password, accessRule, dateCreation);

}

}

Файл «UserRepository.cs»

using Microsoft.EntityFrameworkCore;

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Data;

using MVPv5.Domain.Entities;

using MVPv5.Domain.Models;

using System.ComponentModel.DataAnnotations;

namespace MVPv5.Domain.Repositories;

public class UserRepository(MVPv5DbContext dbContext) : IUserRepository

{

public async Task AddAsync(UserModel model, CancellationToken token)

{

if (await dbContext.Users.AnyAsync(u => u.Login == model.Login, token))

{

throw new ValidationException("Такой пользователь уже существует");

}

await dbContext!.Users.AddAsync(new UserEntity

{

Login = model.Login,

Nickname = model.Nickname,

Password = model.Password,

AccessRule = model.AccessRule,

DateCreation = model.DateCreation,

}, token);

await dbContext.SaveChangesAsync(token);

}

public async Task UpdateNicknameByIdAsync(int id, string nickname, CancellationToken token)

{

await dbContext.Users

.Where(user => user.Id == id)

.ExecuteUpdateAsync(user => user

.SetProperty(u => u.Nickname, nickname),

token);

await dbContext.SaveChangesAsync(token);

}

public async Task UpdateLoginByIdAsync(int id, string login, CancellationToken token)

{

await dbContext.Users

.Where(user => user.Id == id)

.ExecuteUpdateAsync(user => user

.SetProperty(u => u.Login, login),

token);

await dbContext.SaveChangesAsync(token);

}

public async Task UpdatePasswordByLoginAsync(string login, string password, CancellationToken token)

{

await dbContext.Users

.Where(user => user.Login == login)

.ExecuteUpdateAsync(user => user

.SetProperty(u => u.Password, password),

token);

await dbContext.SaveChangesAsync(token);

}

public async Task<(UserModel User, string Error)> GetByIdAsync(int id, CancellationToken token)

{

return GetUser(await dbContext.Users

.AsNoTracking()

.FirstOrDefaultAsync(user => user.Id == id, token));

}

public async Task<(UserModel User, string Error)> GetByLoginAsync(string login, CancellationToken token)

{

return GetUser(await dbContext.Users

.AsNoTracking()

.FirstOrDefaultAsync(user => user.Login == login, token));

}

public async Task<(UserModel User, string Error)> GetByLoginAndPasswordAsync(string login, string password, CancellationToken token)

{

return GetUser(await dbContext.Users

.AsNoTracking()

.FirstOrDefaultAsync(user => user.Login == login && user.Password == password, token));

}

public async Task<IEnumerable<(UserModel User, string Error)>> GetAllAsync(CancellationToken token)

{

return GetListOfUsers(await dbContext.Users

.AsNoTracking()

.ToListAsync(token));

}

public async Task DeleteById(int id, CancellationToken token)

{

var count = await dbContext.Users

.Where(user => user.Id == id)

.ExecuteDeleteAsync(token);

await dbContext.SaveChangesAsync(token);

if (count != 1)

{

throw new Exception($"Удалено {count} пользователей вместо 1");

}

}

private (UserModel User, string Error) GetUser(UserEntity? response)

{

return UserModel.Create(response);

}

private IEnumerable<(UserModel User, string Error)> GetListOfUsers(IEnumerable<UserEntity>? response)

{

if (response == null) throw new Exception("Пустой лист в ответе");

return response.Select(GetUser);

}

}

Файл «UserService.cs»

using Microsoft.AspNetCore.Identity;

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Models;

namespace MVPv5.Domain.Services.v1;

public class UserService(IUserRepository repository) : IUserService

{

public async Task CreateAsync(string nickname, string login, string password, byte accessRule,

DateOnly dateCreation, CancellationToken token)

{

var hashedPassword = new PasswordHasher<string>().HashPassword(login, password);

await repository.AddAsync(UserModel.Create(nickname, login, hashedPassword, accessRule, dateCreation), token);

}

public async Task<UserModel> GetByIdAsync(int id, CancellationToken token)

{

var response = await repository.GetByIdAsync(id, token);

if (response.Error != string.Empty)

{

throw new KeyNotFoundException($"Ошибка: {response.Error}");

}

return response.User;

}

public async Task<UserModel> GetByLoginAsync(string login, CancellationToken token)

{

var response = await repository.GetByLoginAsync(login, token);

if (response.Error != string.Empty)

{

throw new KeyNotFoundException($"Ошибка: {response.Error}");

}

return response.User;

}

public async Task<UserModel> GetByLoginAndPasswordAsync(string login, string password, CancellationToken token)

{

var response = await repository.GetByLoginAsync(login, token);

var result = new PasswordHasher<string>().VerifyHashedPassword(login, response.User.Password, password);

if (result == PasswordVerificationResult.Failed)

{

throw new KeyNotFoundException($"Пароли не совпадают");

}

if (response.Error != string.Empty)

{

throw new KeyNotFoundException($"Ошибка: {response.Error}");

}

return response.User;

}

public async Task<IEnumerable<UserModel>> GetAllAsync(CancellationToken token)

{

var response = await repository.GetAllAsync(token);

var errors = response.Where(r => !string.IsNullOrEmpty(r.Error)).ToList();

if (errors.Any())

{

throw new Exception($"Ошибка: {string.Join(" | ", errors.Select(e => e.Error))}");

}

return response.Select(l => l.User);

}

public async Task UpdateNicknameAsync(int id, string nickname, string confirmNickname, CancellationToken token)

{

if (!string.Equals(nickname, confirmNickname))

{

throw new Exception($"Никнеймы не совпадают");

}

await repository.UpdateNicknameByIdAsync(id, nickname, token);

}

public async Task UpdateLoginAsync(int id, string login, string confirmLogin, CancellationToken token)

{

if (!string.Equals(login, confirmLogin))

{

throw new Exception($"Логины не совпадают");

}

await repository.UpdateLoginByIdAsync(id, login, token);

}

public async Task UpdatePasswordByLogin(string login, string password, string confirmPassword, CancellationToken token)

{

if (!string.Equals(password, confirmPassword))

{

throw new Exception($"Пароли не совпадают");

}

var hashedPassword = new PasswordHasher<string>().HashPassword(login, password);

await repository.UpdatePasswordByLoginAsync(login, password, token);

}

public async Task DeleteByIdAsync(int id, CancellationToken token)

{

await repository.DeleteById(id, token);

}

}

Файл «IUserRepository.cs»

using MVPv5.Domain.Models;

namespace MVPv5.Domain.Abstractions.v1;

public interface IUserRepository

{

Task AddAsync(UserModel entity, CancellationToken token);

Task UpdateNicknameByIdAsync(int id, string nickname, CancellationToken token);

Task UpdateLoginByIdAsync(int id, string login, CancellationToken token);

Task UpdatePasswordByLoginAsync(string login, string password, CancellationToken token);

Task<(UserModel User, string Error)> GetByIdAsync(int id, CancellationToken token);

Task<IEnumerable<(UserModel User, string Error)>> GetAllAsync(CancellationToken token);

Task DeleteById(int id, CancellationToken token);

Task<(UserModel User, string Error)> GetByLoginAsync(string login, CancellationToken token);

Task<(UserModel User, string Error)> GetByLoginAndPasswordAsync(string login, string password, CancellationToken token);

}

Файл «IUserService.cs»

using MVPv5.Domain.Models;

namespace MVPv5.Domain.Abstractions.v1;

public interface IUserService

{

Task CreateAsync(string nickname, string login, string password, byte accessRule,

DateOnly dateCreation, CancellationToken token);

Task<UserModel> GetByIdAsync(int id, CancellationToken token);

Task<UserModel> GetByLoginAsync(string login, CancellationToken token);

Task<IEnumerable<UserModel>> GetAllAsync(CancellationToken token);

Task UpdateNicknameAsync(int id, string nickname, string confirmNickname, CancellationToken token);

Task UpdateLoginAsync(int id, string login, string confirmLogin, CancellationToken token);

Task UpdatePasswordByLogin(string login, string password, string confirmPassword, CancellationToken token);

Task<UserModel> GetByLoginAndPasswordAsync(string login, string password, CancellationToken token);

Task DeleteByIdAsync(int id, CancellationToken token);

}

Файл «MVPv5DbContext.cs»

using Microsoft.EntityFrameworkCore;

using MVPv5.Domain.Entities;

namespace MVPv5.Domain.Data;

public class MVPv5DbContext : DbContext

{

public MVPv5DbContext(DbContextOptions<MVPv5DbContext> options)

: base(options)

{

Database.EnsureCreated();

}

public DbSet<UserEntity> Users { get; set; }

public DbSet<DocumentEntity> Documents { get; set; }

public DbSet<TemplateEntity> Templates { get; set; }

}

Файл «DocumentController.cs»

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using Microsoft.AspNetCore.Mvc;

using MVPv5.Application.Contracts.Document.v1;

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Models;

namespace MVPv5.API.Controllers.v1;

[ApiController]

[Area("v1")]

[Route("[controller]")]

public class DocumentController(IDocumentService service, ITemplateService templateService) : ControllerBase

{

[HttpPost("create")]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status409Conflict)]

public async Task<IActionResult> Add([FromBody] DocumentCreateRequest request, CancellationToken token)

{

await service.CreateAsync(DocumentModel.Create(request.Name, null, request.Data, request.TemplateId, request.UserId), token);

return Created();

}

[HttpGet("read/{id:int}")]

[ProducesResponseType(typeof(DocumentReadResponse), StatusCodes.Status200OK)]

public async Task<ActionResult<DocumentReadResponse>> GetById(int id, CancellationToken token)

{

var document = await service.GetByIdAsync(id, token);

return Ok(ModelToResponse(document));

}

[HttpGet("read-all")]

[ProducesResponseType(typeof(IEnumerable<DocumentReadResponse>), StatusCodes.Status200OK)]

public async Task<ActionResult<IEnumerable<DocumentReadResponse>>> GetAll(CancellationToken token)

{

var documents = await service.GetAllAsync(token);

return Ok(documents.Select(m => new DocumentReadResponse(

m.Id,

m.Name,

m.DateCreation,

m.Dictionary,

m.TemplateId,

m.UserId

)));

}

[HttpPatch("update")]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public async Task<IActionResult> Update([FromBody] DocumentPatchRequest request, CancellationToken token)

{

if (!ModelState.IsValid)

{

return ValidationProblem(ModelState);

}

if (!string.IsNullOrEmpty(request.Name))

{

await service.UpdateNameAsync(request.Id, request.Name, token);

}

if (request.Data is not null)

{

await service.UpdateMetaDataById(request.Id, request.Data, token);

}

return Ok();

}

[HttpDelete("delete/{id:int}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

public async Task<IActionResult> DeleteById(int id, CancellationToken token)

{

await service.DeleteByIdAsync(id, token);

return NoContent();

}

private Stream DocumentEditor(byte[] content, IDictionary<string, string> Data)

{

using var outputStream = new MemoryStream();

outputStream.Write(content, 0, content.Length);

outputStream.Position = 0;

using var doc = WordprocessingDocument.Open(outputStream, true);

var body = doc.MainDocumentPart?.Document.Body;

foreach (var text in body?.Descendants<Text>() ?? Enumerable.Empty<Text>())

{

foreach (var replacement in Data)

{

if (!text.InnerText.Contains(replacement.Key)) continue;

if (replacement.Value.Contains("\r\n"))

{

if (text.Parent is not Run run || run.Parent is not Paragraph paragraph) continue;

var pPr = paragraph.ParagraphProperties;

var rPr = run.RunProperties;

int index = body!.ToList().IndexOf(paragraph);

paragraph.Remove();

foreach (var newText in replacement.Value.Split("\r\n", StringSplitOptions.None))

{

var newPar = new Paragraph();

if (pPr != null) newPar.ParagraphProperties = (ParagraphProperties)pPr.CloneNode(true);

var newRun = new Run();

if (rPr != null) newRun.RunProperties = (RunProperties)rPr.CloneNode(true);

newRun.Append(new Text(newText));

newPar.Append(newRun);

body!.InsertAt(newPar, index++);

}

}

else

{

text.Text = text.Text.Replace(replacement.Key, replacement.Value);

}

}

doc.Save();

}

var result = new MemoryStream(outputStream.ToArray());

result.Position = 0;

return result;

}

[HttpGet("build-and-download/{id:int}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public async Task<IActionResult> DownloadByID(int id, CancellationToken token)

{

var response = await service.GetByIdAsync(id, token);

if (response is null)

{

return NotFound("Документ не найден");

}

var template = await templateService.GetByIdAsync(response.TemplateId, token);

if (template is null)

{

return NotFound("Шаблон не найден");

}

if (template.Content is null)

{

return NotFound("Файл шаблона не найден");

}

var stream = DocumentEditor(template.Content, response.Dictionary ?? new Dictionary<string, string>());

return File(stream, template.ContentType ?? "", response.Name, true);

}

private static DocumentReadResponse ModelToResponse(DocumentModel m) => new(

m.Id,

m.Name,

m.DateCreation,

m.Dictionary,

m.TemplateId,

m.UserId

);

}

Файл «TemplateController.cs»

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using MVPv5.Application.Contracts.Template.v1;

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Models;

namespace MVPv5.API.Controllers.v1;

[ApiController]

[Area("v1")]

[Route("[controller]")]

public class TemplateController(ITemplateService service) : ControllerBase

{

[HttpPost("create")]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status409Conflict)]

public async Task<IActionResult> Add([FromBody] TemplateCreateRequest request, CancellationToken token = default)

{

if (!ModelState.IsValid)

{

return ValidationProblem(ModelState);

}

await service.AddAsync(TemplateModel.Create(request.Name!, request.Type, DateOnly.FromDateTime(DateTime.Now),

request.Content!, request.ContentType!, request.Tags), token);

return Created();

}

[HttpGet("read/{id:int}")]

[ProducesResponseType(typeof(TemplateReadResponse), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public async Task<ActionResult<TemplateReadResponse>> Get(int id, CancellationToken token = default)

{

var response = await service.GetByIdAsync(id, token);

if (response is null)

{

return NotFound();

}

return Ok(ModelToResponse(response));

}

[HttpGet("read-all")]

[ProducesResponseType(typeof(IEnumerable<TemplateReadResponse>), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public async Task<ActionResult<IEnumerable<TemplateReadResponse>>> GetAll(CancellationToken token = default)

{

var result = await service.GetAllAsync(token);

if (result.IsNullOrEmpty())

{

return NotFound();

}

return Ok(result.Select(ModelToResponse));

}

[HttpPatch("update")]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public async Task<IActionResult> Patch([FromBody] TemplatePatchRequest request, CancellationToken token = default)

{

if (!ModelState.IsValid)

{

return ValidationProblem(ModelState);

}

await service.UpdateTagsAsync(request.Id, request.Tags, token);

await service.UpdateNameAsync(request.Id, request.Name, token);

await service.UpdateTypeAsync(request.Id, request.Type, token);

await service.UpdateContentAndContentTypeAsync(request.Id, request.Content, request.ContentType, token);

return Created();

}

[HttpDelete("delete")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

public async Task<IActionResult> Delete([FromBody] TemplateDeleteRequest request, CancellationToken token)

{

await service.DeleteByIdAsync(request.Id, token);

return NoContent();

}

[HttpDelete("delete/{id:int}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

public async Task<IActionResult> DeleteById(int id, CancellationToken token)

{

await service.DeleteByIdAsync(id, token);

return NoContent();

}

[HttpGet("download")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public async Task<IActionResult> DownloadByID([FromQuery] int id, CancellationToken token)

{

var template = await service.GetByIdAsync(id, token);

if (template.Content == null)

{

return NotFound("Файл не найден");

}

return File(template.Content, template.ContentType, template.Name, true);

}

private static TemplateReadResponse ModelToResponse(TemplateModel m) => new(

m.Id,

m.Name,

m.Type,

m.DateCreation,

m.Content,

m.ContentType,

m.Tags

);

}

Файл «UserController.cs»

using Microsoft.AspNetCore.Mvc;

using MVPv5.Application.Contracts.User.v1;

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Models;

using System.Security.Claims;

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authorization;

namespace MVPv5.API.Controllers.v1;

[ApiController]

[Area("v1")]

[Route("[controller]")]

public class UserController(IUserService service) : ControllerBase

{

[HttpPost("create")]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public async Task<ActionResult> Add([FromBody] UserCreateRequest user, CancellationToken token = default)

{

if (!ModelState.IsValid)

{

return ValidationProblem(ModelState);

}

await service.CreateAsync(user.Nickname, user.Login, user.Password, 30, DateOnly.FromDateTime(DateTime.Now), token);

return Created();

}

[HttpGet("read")]

[Authorize]

[ProducesResponseType(typeof(UserReadResponse), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public async Task<ActionResult<UserReadResponse>> Get(CancellationToken token = default)

{

var response = await service.GetByIdAsync(Convert.ToInt32(ClaimTypes.NameIdentifier), token);

if (response is null)

{

return NotFound();

}

return ModelToResponse(response);

}

[HttpPost("check")]

[ProducesResponseType(typeof(UserReadResponse), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status401Unauthorized)]

public async Task<ActionResult<UserReadResponse>> Get([FromBody] UserLoginRequest request, CancellationToken token = default)

{

var response = await service.GetByLoginAndPasswordAsync(request.Login, request.Password, token);

if (response is null)

{

return Unauthorized();

}

var claims = new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier, response.Id.ToString()),

new Claim(ClaimTypes.Email, response.Login),

new Claim(ClaimTypes.Name, response.Nickname),

new Claim(ClaimTypes.Role, response.AccessRule.ToString())

};

await HttpContext.SignInAsync("Cookies",

new(new ClaimsIdentity(claims, "Cookies")),

new AuthenticationProperties

{

IsPersistent = true,

ExpiresUtc = DateTimeOffset.UtcNow.AddDays(7)

});

return ModelToResponse(response);

}

[HttpPatch]

[Authorize]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public async Task<IActionResult> UpdatePassword([FromBody] UserPatchPasswordRequest user, CancellationToken token = default)

{

if (!ModelState.IsValid)

{

return ValidationProblem(ModelState);

}

await service.UpdatePasswordByLogin(user.Login, user.Password, user.PasswordConfirm, token);

return Created();

}

[HttpDelete("delete/{id:int}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

public async Task<IActionResult> Delete(int id, CancellationToken token = default)

{

await service.DeleteByIdAsync(id, token);

return NoContent();

}

private static UserReadResponse ModelToResponse(UserModel model)

{

return new(

model.Id,

model.Nickname,

model.Login,

model.Password,

model.AccessRule,

model.DateCreation

);

}

}

Файл «ExMiddleware.cs»

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.ComponentModel.DataAnnotations;

namespace MVPv5.API.Middleware;

public class ExMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger<ExMiddleware> \_logger;

public ExMiddleware(RequestDelegate next, ILogger<ExMiddleware> logger)

{

\_next = next;

\_logger = logger;

}

public async Task Invoke(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Произошла ошибка");

context.Response.ContentType = "application/json";

var statusCode = ex switch

{

KeyNotFoundException => StatusCodes.Status404NotFound,

ValidationException => StatusCodes.Status400BadRequest,

DbUpdateException => StatusCodes.Status409Conflict,

ArgumentException => StatusCodes.Status400BadRequest,

NotImplementedException => StatusCodes.Status501NotImplemented,

\_ => StatusCodes.Status500InternalServerError

};

context.Response.StatusCode = statusCode;

var problem = new ProblemDetails

{

Status = statusCode,

Title = GetTitleForStatus(statusCode),

Detail = ex.Message,

Instance = context.Request.Path

};

await context.Response.WriteAsJsonAsync(problem);

}

}

private static string GetTitleForStatus(int status) => status switch

{

StatusCodes.Status400BadRequest => "Ошибка валидации",

StatusCodes.Status404NotFound => "Контент не найден",

StatusCodes.Status409Conflict => "Проблема при работе с данными",

StatusCodes.Status500InternalServerError => "Внутренняя ошибка сервера",

StatusCodes.Status501NotImplemented => "Метод не реализован",

\_ => "Произошла ошибка"

};}

Файл «MVPv5.API/Program.cs»

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.EntityFrameworkCore;

using MVPv5.API.Controllers.v1;

using MVPv5.API.Middleware;

using MVPv5.Domain.Abstractions.v1;

using MVPv5.Domain.Data;

using MVPv5.Domain.Repositories;

using MVPv5.Domain.Services.v1;

namespace MVPv5.API;

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAntiforgery();

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddOpenApi();

builder.Services.AddAuthentication();

builder.Services.AddAuthorization();

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme);

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie(options =>

{

options.Cookie.Name = "cool\_user";

options.SlidingExpiration = true;

options.Cookie.IsEssential = true;

options.LoginPath = "/login";

options.LogoutPath = "/logout";

options.ExpireTimeSpan = TimeSpan.FromMinutes(10);

});

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowClient", policy =>

{

policy.WithOrigins("https://localhost:7077")

.AllowCredentials()

.AllowAnyHeader()

.AllowAnyMethod();

});

});

builder.Services.AddDbContextFactory<MVPv5DbContext>(options =>

options.UseNpgsql(builder.Configuration.GetConnectionString(nameof(MVPv5DbContext))

?? throw new InvalidOperationException($"'{nameof(MVPv5DbContext)}' not found.")));

builder.Services.AddScoped<IUserService, UserService>();

builder.Services.AddScoped<IUserRepository, UserRepository>();

builder.Services.AddScoped<TemplateController>();

builder.Services.AddScoped<ITemplateService, TemplateService>();

builder.Services.AddScoped<ITemplateRepository, TemplateRepository>();

builder.Services.AddScoped<IDocumentService, DocumentService>();

builder.Services.AddScoped<IDocumentRepository, DocumentRepository>();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.MapOpenApi();

app.UseSwaggerUI(options =>

{

options.SwaggerEndpoint("/openapi/v1.json", "v1");

options.IndexStream = ()

=> File.OpenRead(Path.Combine(app.Environment.WebRootPath!, "swagger", "swagger-index.html"));

});

app.UseMigrationsEndPoint();

app.UseHsts();

}

app.UseCors("AllowClient");

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseMiddleware<ExMiddleware>();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.UseAntiforgery();

app.Run();

}

}

Файл «MVPv5.Client/Program.cs»

using Microsoft.AspNetCore.Components.Web;

using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

namespace MVPv5.Client;

public class Program

{

public static async Task Main(string[] args)

{

var builder = WebAssemblyHostBuilder.CreateDefault(args);

builder.RootComponents.Add<App>("#app");

builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddScoped(sp => new HttpClient()

{

BaseAddress = new Uri(builder.Configuration["WebApiAddress"]!)

});

await builder.Build().RunAsync();

}

}

Файл «PanelElements.razor»

<div class="upper-panel p-0">

<button class="button-menu" @onclick="click"></button>

<img class="separator-icon" src="Separator.png" alt="Separator">

@if (ShowDiv)

{

<div class="PopUp-menu">

<button class="button-user-menu" @onclick="ShowModal"><img src="Profil.png" class="profil-img">Профиль</button>

<button class="button-user-menu" @onclick="DivShow"><img src="Shablon.png" class="profil-img">Выбор шаблона</button>

@if (ShowDiv1)

{

<**SelectPanelElements**></**SelectPanelElements**>

}

</div>

}

@if (ShowMenu)

{

<div class="modal-overlay">

<div class="modal-content">

<p class="profile-upper">

<button class="button-close" @onclick="CloseModal">X</button>

</p>

@using MVPv5.Client.Menu

<**Profile**></**Profile**>

</div>

</div>

}

</div>

@code {

public bool ShowMenu { get; set; } = false;

public bool ShowDiv1 { get; set; } = false;

public bool ShowDiv { get; set; } = false;

public void click()

{

ShowDiv = !ShowDiv;

if (!ShowDiv)

{

ShowDiv1 = false;

}

}

public void ShowModal()

{

ShowMenu = !ShowMenu;

click();

}

public void CloseModal()

{

ShowMenu = false;

}

public void DivShow()

{

ShowDiv1 = !ShowDiv1;

}

}

Файл «UpperPanel.razor»

@inherits LayoutComponentBase

<main>

<article class="content m-0 p-0">

@Body

</article>

</main>

Файл «Main.razor»

@inject NavigationManager NavigationManager

@page "/main"

<**PageTitle**>Главная</**PageTitle**>

<h3>Главная</h3>

Выберите действие

<div class="form-control">

<button type="button" class="btn btn-outline-secondary mb-2" @onclick="ToTemplates">Шаблоны...</button>

<button type="button" class="btn btn-outline-secondary mb-2" @onclick="ToDocuments">Документы...</button>

</div>

@code{

private void ToEditor() => NavigationManager.NavigateTo("/editor");

private void ToTemplates() => NavigationManager.NavigateTo("/templates");

private void ToDocuments() => NavigationManager.NavigateTo("/documents");

}

Файл «Documents/Create.razor»

@page "/documents/create"

<**PageTitle**>Создать документ</**PageTitle**>

<**PanelElements** />

<p class="alert">@Status</p>

@if (Request is null)

{

<p><em>Загрузка...</em></p>

}

else

{

<h3>Создать документ</h3>

<**EditForm** **Model**="Request" **OnValidSubmit**="Save" **FormName**="DocumentCreate">

<**DataAnnotationsValidator** />

<button class="btn btn-primary mb-2" type="submit">Создать</button>

<div class="mb-2">

<label class="form-label" for="name">Название</label>

<**InputText** id="name" class="form-control mb-2" @bind-Value="Request.Name" placeholder="Введите название документа" />

<**ValidationMessage** **For**="() => Request.Name" class="text-danger" />

</div>

<div class="mb-2">

<label class="form-label" for="user">Пользователь</label>

<**InputNumber** id="user" class="form-control" @bind-Value="Request.UserId" placeholder="Пользователь" />

<**ValidationMessage** **For**="() => Request.UserId" class="text-danger" />

</div>

<div class="form-control mb-2">

<div class="row d-flex" style="justify-self: center; width: 40%;">

@if (Inputs is null || !Inputs.Any())

{

<p>Нет тегов для заполнения</p>

}

else

{

foreach (var input in Inputs)

{

<label class="form-label mt-1 mb-0 fs-6 fst-italic" for="@input.Key">@input.Key</label>

<**InputTextArea** id="@input.Key" class="rounded-3 mb-2" @bind-Value="input.Value" placeholder="@input.Key" />

}

}

</div>

</div>

</**EditForm**>

}

@using System.Text

@using MVPv5.Application.Contracts.Document.v1

@using MVPv5.Application.Contracts.Template.v1

@inject HttpClient httpClient

@inject NavigationManager NavigationManager

@code {

[SupplyParameterFromQuery(Name = "id")]

public int IdTemplate { get; set; }

private string? Status { get; set; } = "...";

private string ButtonStyle => "btn btn-outline-primary btn-sm px-4";

private DocumentCreateRequest? Request { get; set; }

private TemplateReadResponse? Template;

private List<InputModel>? Inputs { get; set; }

private class InputModel

{

public required string Key { get; set; }

public string? Value { get; set; }

}

private async Task Save()

{

Status = string.Empty;

if (Template is null || Request is null) return;

try

{

Request.Data = Inputs?

.Where(i => !string.IsNullOrEmpty(i.Value))

.ToDictionary(keySelector: k => k.Key, elementSelector: v => v.Value!.Trim());

var response = await httpClient.PostAsJsonAsync("document/create", Request);

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

NavigationManager.NavigateTo("/documents");

}

catch (Exception ex)

{

Status = ex.Message;

}

StateHasChanged();

}

protected override async Task OnInitializedAsync()

{

Status = string.Empty;

try

{

var response = await httpClient.GetAsync($"Template/read/{IdTemplate}");

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

Template = await response.Content.ReadFromJsonAsync<TemplateReadResponse>();

if (Template is null)

{

Status = "Пустой ответ сервера";

return;

}

Request = new()

{

TemplateId = IdTemplate

};

if (Template.Tags is null) return;

Inputs = new(Template.Tags.Select(t => new InputModel() { Key = t }));

}

catch (Exception ex)

{

Status = ex.Message;

}

}

}

Файл «Documents/Edit.razor»

@page "/documents/edit"

<**PageTitle**>Редактировать документ</**PageTitle**>

<**PanelElements** />

@\* TODO - показ статуса на несколько секунд \*@

<p class="alert">@Status</p>

@if (Request is null)

{

<p><em>Загрузка...</em></p>

}

else

{

<h3>Редактировать шаблон - @Document?.Name</h3>

<**EditForm** **Model**="Request" **OnValidSubmit**="Save">

<**DataAnnotationsValidator** />

<button class="btn btn-primary mb-2" type="submit">Сохранить</button>

<div class="mb-2">

<label class="form-label" for="name">Название</label>

<**InputText** class="form-control mb-2" @bind-Value="Request.Name" placeholder="Введите название документа" />

<**ValidationMessage** **For**="() => Request.Name" class="text-danger" />

</div>

<div class="mb-2">

<button type="button" class="btn btn-secondary" @onclick="BuildAndDownloadFile">Скачать</button>

</div>

<div class="form-control mb-2">

<div class="row d-flex" style="justify-self: center; width: 40%;">

@if (Inputs is null || !Inputs.Any())

{

<p>Нет тегов для заполнения</p>

}

else

{

foreach (var input in Inputs)

{

<label class="form-label mt-1 mb-0 fs-6 fst-italic" for="@input.Key">@input.Key</label>

<**InputTextArea** id="@input.Key" class="rounded-3 mb-2" @bind-Value="input.Value" placeholder="@input.Key" />

}

}

</div>

</div>

</**EditForm**>

}

@using System.Text

@using MVPv5.Application.Contracts.Document.v1

@using MVPv5.Application.Contracts.Template.v1

@inject HttpClient httpClient

@code {

[SupplyParameterFromQuery(Name = "id")]

public int Id { get; set; }

private string? Status { get; set; }

private string ButtonStyle => "btn btn-outline-primary btn-sm px-4";

private DocumentReadResponse? Document { get; set; }

private TemplateReadResponse? template { get; set; }

private DocumentPatchRequest? Request { get; set; }

@inject IJSRuntime JS

private async Task BuildAndDownloadFile()

{

Status = string.Empty;

if (Request is null) return;

try

{

var stream = await httpClient.GetStreamAsync($"document/build-and-download/{Request.Id}");

using var memoryStream = new MemoryStream();

await stream.CopyToAsync(memoryStream);

var bytes = memoryStream.ToArray();

await JS.InvokeVoidAsync("downloadFromByteArray", new

{

byteBase64 = Convert.ToBase64String(bytes),

fileName = Request.Name,

mimeType = "application/octet-stream"

});

}

catch (Exception ex)

{

Status = ex.Message;

}

}

private List<InputModel>? Inputs { get; set; }

private class InputModel

{

public required string Key { get; set; }

public required string Value { get; set; }

}

private async Task Save()

{

Status = string.Empty;

if (Request is null) return;

try

{

Request.Data = Inputs?

.ToDictionary(keySelector: k => k.Key, elementSelector: v => v.Value.Trim());

var response = await httpClient.PatchAsJsonAsync("document/update", Request);

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

Status = "Сохранено";

}

catch (Exception ex)

{

Status = ex.Message;

}

StateHasChanged();

}

protected override async Task OnInitializedAsync()

{

Status = string.Empty;

try

{

var response = await httpClient.GetAsync($"document/read/{Id}");

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

Document = await response.Content.ReadFromJsonAsync<DocumentReadResponse>();

if (Document is null)

{

Status = "Пустой документ";

return;

}

var responseTmpl = await httpClient.GetAsync($"Template/read/{Document.TemplateId}");

if (!responseTmpl.IsSuccessStatusCode)

{

Status = await responseTmpl.Content.ReadAsStringAsync();

return;

}

template = await responseTmpl.Content.ReadFromJsonAsync<TemplateReadResponse>();

if (template is null)

{

Status = "Пустой шаблон";

return;

}

Request = new()

{

Id = this.Id,

Name = Document.Name,

Data = Document.Data

};

if (template.Tags is null || !template.Tags.Any()) return;

Inputs = new(template.Tags.Select(t => new InputModel()

{

Key = t,

Value = (Request.Data is not null) ? Request.Data.TryGetValue(t, out var value) ? value : string.Empty : string.Empty

}));

}

catch (Exception ex)

{

Status = ex.Message;

}

}

}

Файл «Documents/Index.razor»

@page "/documents"

<**PageTitle**>Документы</**PageTitle**>

<p class="alert">@Status</p>

@if (PreCreate)

{

if (Templates is null || !Templates.Any())

{

<p>Отсутствуют шаблоны для выбора</p>

}

else

{

<div class="mb-2">

<label class="form-label">Шаблон</label>

<select class="form-control" @onchange="TemplateChanged">

@foreach (var option in Templates.Select(i => i.Name))

{

<option value="@option">@option</option>

}

</select>

</div>

if (TemplateId != 0)

{

<button type="button" class="btn btn-primary" @onclick="CreateDoc">Создать документ</button>

}

}

}

else

{

<button type="button" class="btn btn-primary" @onclick="ToCreate">Создать</button>

}

@if (Searching is null)

{

<p><em>Загрузка...</em></p>

}

else if (!Searching.Any())

{

<p><em>Документов нет</em></p>

}

else

{

@using Microsoft.AspNetCore.Components.QuickGrid

<**QuickGrid** **Class**="table" **Items**="Searching" **Pagination**="pagination">

<**PropertyColumn** **Property**="m => m.Id" **Title**="Id" **Sortable**="true" />

<**PropertyColumn** **Property**="m => m.Name" **Title**="Название" **Sortable**="true" />

<**PropertyColumn** **Property**="m => m.UserId" **Title**="Пользователь" **Sortable**="true" />

<**PropertyColumn** **Property**="m => m.TemplateId" **Title**="Шаблон" **Sortable**="true" />

<**TemplateColumn** **Context**="p">

<div style="white-space: nowrap; overflow: hidden; text-overflow: ellipsis;">

<a href="@($"documents/edit?id={p.Id}")">Детали</a>

</div>

<div style="white-space: nowrap; overflow: hidden; text-overflow: ellipsis;">

<button type="button" class="btn btn-link" @onclick="() => BuildAndDownloadFile(p)">Собрать и скачать</button>

</div>

</**TemplateColumn**>

<**Paginator** **State**="pagination" />

</**QuickGrid**>

}

@using MVPv5.Application.Contracts.Document.v1

@using MVPv5.Application.Contracts.Template.v1

@inject HttpClient httpClient

@inject NavigationManager NavigationManager

@code {

private string? Status { get; set; }

private IQueryable<DocumentReadResponse>? Searching { get; set; }

static int size = 10;

PaginationState pagination = new PaginationState { ItemsPerPage = size };

protected override void OnParametersSet()

{

var count = 1;

if (Searching is not null && Searching.Any()) count = Searching.Count();

pagination = new PaginationState { ItemsPerPage = count < 10 ? count : size };

}

@inject IJSRuntime JS

private async Task BuildAndDownloadFile(DocumentReadResponse document)

{

try

{

var stream = await httpClient.GetStreamAsync($"document/build-and-download/{document.Id}");

using var memoryStream = new MemoryStream();

await stream.CopyToAsync(memoryStream);

var bytes = memoryStream.ToArray();

await JS.InvokeVoidAsync("downloadFromByteArray", new

{

byteBase64 = Convert.ToBase64String(bytes),

fileName = document.Name,

mimeType = "application/octet-stream"

});

}

catch (Exception ex)

{

Status = ex.Message;

}

}

protected override async Task OnInitializedAsync()

{

var response = await httpClient.SendAsync(

new HttpRequestMessage(HttpMethod.Get, "document/read-all")

.SetBrowserRequestCredentials(BrowserRequestCredentials.Include));

if (response.IsSuccessStatusCode)

{

Searching = (await response.Content.ReadFromJsonAsync<IEnumerable<DocumentReadResponse>>())!.AsQueryable();

}

}

private async void ToCreate()

{

Status = string.Empty;

var response = await httpClient.SendAsync(

new HttpRequestMessage(HttpMethod.Get, "Template/read-all")

.SetBrowserRequestCredentials(BrowserRequestCredentials.Include));

if (response.IsSuccessStatusCode)

{

Templates = await response.Content.ReadFromJsonAsync<IEnumerable<TemplateReadResponse>>();

if (Templates is not null) TemplateId = Templates.First().Id;

PreCreate = true;

}

else

{

Status = "Нет шаблонов в ответе";

}

StateHasChanged();

}

private void CreateDoc() => NavigationManager.NavigateTo($"/documents/create?id={TemplateId}");

private bool PreCreate { get; set; } = false;

private IEnumerable<TemplateReadResponse>? Templates { get; set; }

// private TemplateReadResponse? SelectedTemplate { get; set; }

private int TemplateId { get; set; }

private void TemplateChanged(ChangeEventArgs e)

{

Status = string.Empty;

if (Templates is null || e.Value is null)

{

Status = "Нет выбора";

}

else

{

TemplateId = Templates.FirstOrDefault(i => i.Name == e.Value.ToString())!.Id;

}

StateHasChanged();

}

}

Файл «Templates/Create.razor»

@page "/templates/create"

<**PageTitle**>Создать шаблон</**PageTitle**>

<**PanelElements** />

<p class="alert">@Status</p>

<h3>Создать шаблон</h3>

<**EditForm** **Model**="Request" **OnValidSubmit**="Save" **FormName**="TemplateCreate">

<**DataAnnotationsValidator** />

<button class="btn btn-primary mb-2" type="submit">Создать</button>

<div class="mb-2">

<**InputText** class="form-control mb-2" @bind-Value="Request.Name" placeholder="Введите название шаблона" />

<**ValidationMessage** **For**="() => Request.Name" class="text-danger" />

</div>

<div class="mb-2">

<**InputFile** class="form-control mb-2" **OnChange**="HandleFileSelected" title="Загрузить файл" />

<**ValidationMessage** **For**="() => Request.Content" class="text-danger" />

</div>

<div class="mb-2">

<**InputText** class="form-control" @bind-Value="Request.Type" placeholder="Тип работы" />

<**ValidationMessage** **For**="() => Request.Type" class="text-danger" />

</div>

<div class="mb-2">

<label class="form-label">Теги для замещения</label>

@if (Inputs is null || !Inputs.Any())

{

<button @onclick="() => Inputs = new(){new()}" class=@ButtonStyle>+</button>

}

else

{

<div class="row d-flex" style="justify-self: center; width: 40%;">

@foreach (var input in Inputs)

{

<**InputText** class="rounded-3 mt-2 mb-2" @bind-Value="input.Key" placeholder="@(!string.IsNullOrEmpty(input.Key) ? input.Key : "новый тег")" />

<div class="gap-3 d-flex justify-content-center">

<button type="button" @onclick="a => AddInput(Inputs.IndexOf(input))" class=@ButtonStyle>+</button>

<button type="button" @onclick="a => Remove(Inputs.IndexOf(input))" class=@ButtonStyle>-</button>

</div>

}

</div>

}

</div>

</**EditForm**>

@using System.Text

@using MVPv5.Application.Contracts.Template.v1

@inject HttpClient httpClient

@inject NavigationManager NavigationManager

@code {

private string? Status { get; set; }

private string ButtonStyle => "btn btn-outline-primary btn-sm px-4";

private TemplateCreateRequest Request { get; set; } = new();

private async void HandleFileSelected(InputFileChangeEventArgs args)

{

Status = string.Empty;

try

{

var file = args.File;

if (file is null) return;

MemoryStream? newFile;

await file.OpenReadStream(maxAllowedSize: 44 \* 1024 \* 1024).CopyToAsync(newFile = new());

Request.Content = newFile?.ToArray();

Request.ContentType = file.ContentType;

}

catch (Exception ex)

{

Status = ex.Message;

}

StateHasChanged();

}

private List<InputModel>? Inputs { get; set; }

private class InputModel

{

public string? Key { get; set; }

}

private void AddInput(int index)

{

Inputs?.Insert(index + 1, new());

}

private void Remove(int index)

{

if (Inputs?.Count() <= 1) return;

Inputs?.RemoveAt(index);

}

private async Task Save()

{

Status = string.Empty;

try

{

Request.Tags = Inputs?

.Where(i => !string.IsNullOrEmpty(i.Key))

.Select(i => i.Key!.Trim());

var response = await httpClient.PostAsJsonAsync("template/create", Request);

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

NavigationManager.NavigateTo("/templates");

}

catch (Exception ex)

{

Status = ex.Message;

}

this.StateHasChanged();

}

}

Файл «Templates/Edit.razor»

@page "/templates/edit"

<**PageTitle**>Редактировать шаблон</**PageTitle**>

<**PanelElements** />

@\* TODO - показ статуса на несколько секунд \*@

<p class="alert">@Status</p>

@if (Request is null)

{

<p><em>Загрузка...</em></p>

}

else

{

<h3>Редактировать шаблон - @template?.Name</h3>

<**EditForm** **Model**="Request" **OnValidSubmit**="Save" **FormName**="TemplateEdit">

<**DataAnnotationsValidator** />

<button class="btn btn-primary mb-2" type="submit">Сохранить</button>

<div class="mb-2">

<**InputText** class="form-control mb-2" @bind-Value="Request.Name" placeholder="Введите название шаблона" />

<**ValidationMessage** **For**="() => Request.Name" class="text-danger" />

</div>

<div class="mb-2">

<**InputFile** class="form-control mb-2" **OnChange**="HandleFileSelected" title="Загрузить файл" />

<**ValidationMessage** **For**="() => Request.Content" class="text-danger" />

</div>

<div class="mb-2">

<button class="btn btn-secondary" @onclick="DownloadFile">Скачать</button>

</div>

<div class="mb-2">

<**InputText** class="form-control" @bind-Value="Request.Type" placeholder="Тип работы" />

<**ValidationMessage** **For**="() => Request.Type" class="text-danger" />

</div>

<div class="mb-2">

<label class="form-label">Теги для замещения</label>

@if (Inputs is null || !Inputs.Any())

{

<button @onclick="() => Inputs = new(){new()}" class=@ButtonStyle>+</button>

}

else

{

<div class="row d-flex" style="justify-self: center; width: 40%;">

@foreach (var input in Inputs)

{

<**InputText** class="rounded-3 mt-2 mb-2" @bind-Value="input.Key" placeholder="@(!string.IsNullOrEmpty(input.Key) ? input.Key : "новый тег")" />

<div class="gap-3 d-flex justify-content-center">

<button type="button" @onclick="a => AddInput(Inputs.IndexOf(input))" class=@ButtonStyle>+</button>

<button type="button" @onclick="a => Remove(Inputs.IndexOf(input))" class=@ButtonStyle>-</button>

</div>

}

</div>

}

</div>

</**EditForm**>

}

@using System.Text

@using MVPv5.Application.Contracts.Template.v1

@inject HttpClient httpClient

@code {

[SupplyParameterFromQuery(Name = "id")]

public int Id { get; set; }

private string? Status { get; set; }

private string ButtonStyle => "btn btn-outline-primary btn-sm px-4";

private TemplateReadResponse? template { get; set; }

private TemplatePatchRequest? Request { get; set; }

@inject IJSRuntime JS

private async Task DownloadFile()

{

Status = string.Empty;

if (template is null) return;

try

{

await JS.InvokeVoidAsync("downloadFromByteArray", new

{

byteBase64 = Convert.ToBase64String(template.Content ?? new byte[0]),

fileName = template.Name,

mimeType = "application/octet-stream"

});

}

catch (Exception ex)

{

Status = ex.Message;

}

}

private async void HandleFileSelected(InputFileChangeEventArgs args)

{

Status = string.Empty;

if (Request is null) return;

try

{

var file = args.File;

if (file is null) return;

MemoryStream? newFile;

await file.OpenReadStream(maxAllowedSize: 44 \* 1024 \* 1024).CopyToAsync(newFile = new());

Request.Content = newFile?.ToArray();

Request.ContentType = file.ContentType;

}

catch (Exception ex)

{

Status = ex.Message;

}

StateHasChanged();

}

private List<InputModel>? Inputs { get; set; }

private class InputModel

{

public string? Key { get; set; }

}

private void AddInput(int index)

{

Inputs?.Insert(index + 1, new());

}

private void Remove(int index)

{

if (Inputs?.Count() <= 1) return;

Inputs?.RemoveAt(index);

}

private async void Save()

{

Status = string.Empty;

if (Request is null) return;

try

{

Request.Tags = Inputs?

.Where(i => !string.IsNullOrEmpty(i.Key))

.Select(i => i.Key!.Trim());

var response = await httpClient.PatchAsJsonAsync("template/update", Request);

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

Status = "Сохранено";

}

catch (Exception ex)

{

Status = ex.Message;

}

StateHasChanged();

}

protected override async Task OnInitializedAsync()

{

Status = string.Empty;

try

{

var response = await httpClient.GetAsync($"Template/read/{Id}");

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

template = await response.Content.ReadFromJsonAsync<TemplateReadResponse>();

if (template is null)

{

Status = "Пустой ответ сервера";

return;

}

Request = new()

{

Id = this.Id,

Name = template.Name,

Type = template.Type,

Content = template.Content,

ContentType = template.ContentType,

Tags = template.Tags

};

if (Request.Tags is null) return;

Inputs = new(Request.Tags.Select(t => new InputModel() { Key = t }));

}

catch (Exception ex)

{

Status = ex.Message;

}

}

}

Файл «Templates/Index.razor»

@page "/templates"

<**PageTitle**>Шаблоны</**PageTitle**>

@inject NavigationManager NavigationManager

<button class="btn btn-primary" @onclick="ToCreate">Создать</button>

@if (Searching is null)

{

<p><em>@Status</em></p>

}

else

{

@using Microsoft.AspNetCore.Components.QuickGrid

<**QuickGrid** **Class**="table" **Items**="Searching" **Pagination**="pagination">

<**PropertyColumn** **Property**="m => m.Id" **Title**="Id" **Sortable**="true" />

<**PropertyColumn** **Property**="m => m.Name" **Title**="Название" **Sortable**="true" />

<**PropertyColumn** **Property**="m => m.Type" **Title**="Тип работы" **Sortable**="true" />

<**TemplateColumn** **Context**="p">

<div style="white-space: nowrap; overflow: hidden; text-overflow: ellipsis;">

<a href="@($"templates/edit?id={p.Id}")">Детали</a>

</div>

</**TemplateColumn**>

<**Paginator** **State**="pagination" />

</**QuickGrid**>

}

@using MVPv5.Application.Contracts.Template.v1

@inject HttpClient httpClient

@code {

private string? Status { get; set; } = "Загрузка...";

private IQueryable<TemplateReadResponse>? Searching { get; set; }

static int size = 10;

PaginationState pagination = new PaginationState { ItemsPerPage = size };

protected override void OnParametersSet()

{

var count = 1;

if (Searching is not null && Searching.Any()) count = Searching.Count();

pagination = new PaginationState { ItemsPerPage = count < 10 ? count : size };

}

protected override async Task OnInitializedAsync()

{

var response = await httpClient.SendAsync(

new HttpRequestMessage(HttpMethod.Get, "Template/read-all")

.SetBrowserRequestCredentials(BrowserRequestCredentials.Include));

if (response.IsSuccessStatusCode)

{

Searching = (await response.Content.ReadFromJsonAsync<IEnumerable<TemplateReadResponse>>())!.AsQueryable();

}

else

{

Status = "Не получилось загрузить файлы\n"/\* + await response.Content.ReadAsStringAsync() \*/;

return;

}

}

private void ToCreate() => NavigationManager.NavigateTo("/templates/create");

}

Файл «Editor.razor»

@page "/editor"

<**PageTitle**>Editor</**PageTitle**>

<div class="upper-panel">

<**PanelElements** OnSave="Save" />

</div>

<p class="alert">@Status</p>

@if (Inputs is null)

{

<p><em>Loading...</em></p>

}

else

{

<div class="row d-flex" style="justify-self: center; width: 40%;">

@foreach (var input in Inputs)

{

<**InputTextArea** class="rounded-3 mt-2 mb-2" style="min-height:100px;" @bind-Value="input.Value" placeholder="@input.Key" />

<div class="gap-3 d-flex justify-content-center">

<button @onclick="a => AddInput(Inputs.IndexOf(input))" class=@buttonStyle>+</button>

<button @onclick="a => Remove(Inputs.IndexOf(input))" class=@buttonStyle>-</button>

</div>

}

</div>

}

@using MVPv5.Application.Contracts.Document.v1

@using System.Text

@using MVPv5.Application.Contracts.Template.v1

@code {

private string? Status { get; set; }

private string buttonStyle = "btn btn-outline-primary btn-sm px-4";

private TemplateReadResponse? document { get; set; }

private List<InputModel>? Inputs { get; set; }

private class InputModel

{

public required string Key { get; set; }

public required string Value { get; set; }

}

@inject HttpClient httpClient

protected override async Task OnInitializedAsync()

{

try

{

var response = await httpClient.GetAsync($"Template/read/5");

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

document = await response.Content.ReadFromJsonAsync<TemplateReadResponse>();

if (document?.Tags is null) return;

Inputs = new(document.Tags.Select(t => new InputModel() { Key = t, Value = t }));

}

catch (Exception ex)

{

Status = ex.Message;

}

}

private async void Save()

{

if (Inputs is null) return;

try

{

TemplatePatchRequest request = new()

{

Id = 5,

Tags = Inputs.Where(i => !string.IsNullOrEmpty(i.Value)).Select(i => i.Value)

};

var response = await httpClient.PatchAsJsonAsync("template/update-tags", request);

if (!response.IsSuccessStatusCode)

{

Status = await response.Content.ReadAsStringAsync();

return;

}

Status = "Сохранено";

}

catch (Exception ex)

{

Status = ex.Message;

}

StateHasChanged();

}

private void AddInput(int index)

{

Inputs?.Insert(index + 1, new() { Key = Guid.NewGuid().ToString(), Value = "" });

}

private void Remove(int index)

{

if (Inputs?.Count() <= 1) return;

Inputs?.RemoveAt(index);

}

}

Файл «Login.razor»

@page "/"

@page "/login"

<**PageTitle**>Login</**PageTitle**>

<div class="login-menu">

<div class="login-back">

<div class="login-main">

<button class="button-login active" @onclick="ToLog">Вход</button>

<button class="button-register" @onclick="ToReg">Регистрация</button>

<**EditForm** class="px-4" **Model**="loginedUser" **OnValidSubmit**="OnLogin">

<**DataAnnotationsValidator** />

<**ValidationSummary** class="text-danger" role="alert" />

<div class="mb-3 mt-2">

<**InputText** id="login" @bind-Value="loginedUser.Login" class="form-control input-format" aria-required="true" placeholder="Логин" />

<**ValidationMessage** **For**="() => loginedUser.Login" class="text-danger" />

</div>

<div class="mb-3">

<**InputText** id="password" @bind-Value="loginedUser.Password" class="form-control input-format" placeholder="Пароль" />

<**ValidationMessage** **For**="() => loginedUser.Password" class="text-danger" />

</div>

<button type="submit" class="btn btn-primary button-enter">Войти</button>

<button class="button-guest active" @onclick="EnterLikeGuest">Войти как гость</button>

<div class="justify-content-center mt-3">@Status</div>

</**EditForm**>

</div>

</div>

</div>

@using MVPv5.Application.Contracts.User.v1

@inject HttpClient httpClient

@inject NavigationManager NavigationManager

@code {

private string? Status { get; set; }

private UserLoginRequest loginedUser { get; set; } = new();

private async Task OnLogin()

{

var token = new CancellationTokenSource(3000);

var response = await httpClient.PostAsJsonAsync("User/check", loginedUser, token.Token);

if (response.IsSuccessStatusCode)

{

NavigationManager.NavigateTo("/main");

}

else

{

Status = await response.Content.ReadAsStringAsync();

}

}

private void EnterLikeGuest()

{

NavigationManager.NavigateTo("/main");

}

private void ToLog() => NavigationManager.NavigateTo("/login");

private void ToReg() => NavigationManager.NavigateTo("/registration");

}

Файл «Registration.razor»

@page "/registration"

<**PageTitle**>Register</**PageTitle**>

<div class="login-menu">

<div class="login-back">

<div class="login-main">

<button class="button-login" @onclick="ToLog">Вход</button>

<button class="button-register active" @onclick="ToReg">Регистрация</button>

<**EditForm** method="post" **Model**="createdUser" **OnValidSubmit**="OnRegister" **FormName**="Registration" **Enhance**>

<**DataAnnotationsValidator** />

<**ValidationSummary** class="text-danger" role="alert" />

<div class="mb-3 mt-2">

<**InputText** id="nickname" @bind-Value="createdUser.Nickname" class="form-control input-format" placeholder="Никнейм" />

<**ValidationMessage** **For**="() => createdUser.Nickname" class="text-danger" />

</div>

<div class="mb-3">

<**InputText** id="login" @bind-Value="createdUser.Login" class="form-control input-format" aria-required="true" placeholder="Логин" />

<**ValidationMessage** **For**="() => createdUser.Login" class="text-danger" />

</div>

<div class="mb-3">

<**InputText** id="password" @bind-Value="createdUser.Password" class="form-control input-format" placeholder="Пароль" />

<**ValidationMessage** **For**="() => createdUser.Password" class="text-danger" />

</div>

<div class="mb-3">

<**InputText** id="password" @bind-Value="createdUser.PasswordConfirm" class="form-control input-format" placeholder="Подтверждение пароля" />

<**ValidationMessage** **For**="() => createdUser.PasswordConfirm" class="text-danger" />

</div>

<button type="submit" class="btn btn-primary button-enter">Зарегистрироваться</button>

<div class="justify-content-center mt-3">@Status</div>

</**EditForm**>

</div>

</div>

</div>

@using MVPv5.Application.Contracts.User.v1

@inject HttpClient httpClient

@inject NavigationManager NavigationManager

@code {

private string Status { get; set; } = "";

private UserCreateRequest createdUser { get; set; } = new();

private async Task OnRegister()

{

var response = await httpClient.PostAsJsonAsync("User/Create", createdUser);

if (response.IsSuccessStatusCode)

{ NavigationManager.NavigateTo("/main");

}

else

{

Status = await response.Content.ReadAsStringAsync();

}

}

private void ToLog() => NavigationManager.NavigateTo("/login");

private void ToReg() => NavigationManager.NavigateTo("/registration");

}

Файл «App.razor»

<**Router** **AppAssembly**="@typeof(App).Assembly">

<**Found** **Context**="routeData">

<**RouteView** **RouteData**="@routeData" **DefaultLayout**="@typeof(UpperPanel)" />

<**FocusOnNavigate** **RouteData**="@routeData" **Selector**="h1" />

</**Found**>

<**NotFound**>

<**PageTitle**>Not found</**PageTitle**>

<**LayoutView** **Layout**="null">

<p role="alert">Страницы не существует.</p>

</**LayoutView**>

</**NotFound**>

</**Router**>

# Приложение 3

Руководство пользователя.

Общие сведения о программе

Программа создана для упрощения написания курсовых работ и отчётов по оформлению кафедры МПО ЭВМ, используя браузер в качестве посредника и отрисовки. Программа работает только с файлами формата «.docx», а ввод пользовательского текста в итоговый документ основан на замене так называемых «тегов».

Установка программы

Программа поставляется в архиве «CSharpRelease.zip». Необходимо распаковать в желаемую папку (рис. П3.1).

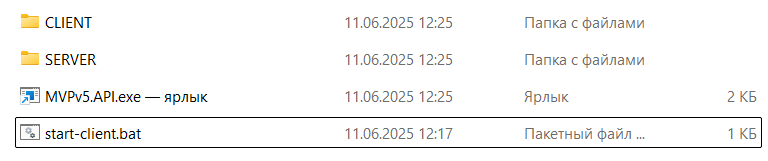


Рис. П3.1. Содержимое распакованного архива

В комплекте идут серверное и клиентское приложения, которые можно запускать на отличных ЭВМ (подробнее в разделе «Запуск программы»).

Для корректной работы серверного приложения требуется установленным пакет «PostgreSQL». Клиентскому приложению данный пакет не нужен.

Запуск программы

Запуск программы осуществляется последовательным нажатием на ярлыки «MVPv5.exe» и «start-client.bat» (см. рис. П3.1) – это серверное и клиентское приложения соответственно, – после чего откроются два консольных окна, которые нельзя закрывать до окончания работы с программой.

Приёмы работы в программе

Для начала работы необходимо зарегистрироваться или войти по логину и паролю. Поля имеют контроль входных значений, например, логином выступает почта домена «@chsu» (рис. П3.2-П3.3).



Рис. П3.2. Не пропускает почту не ЧГУ

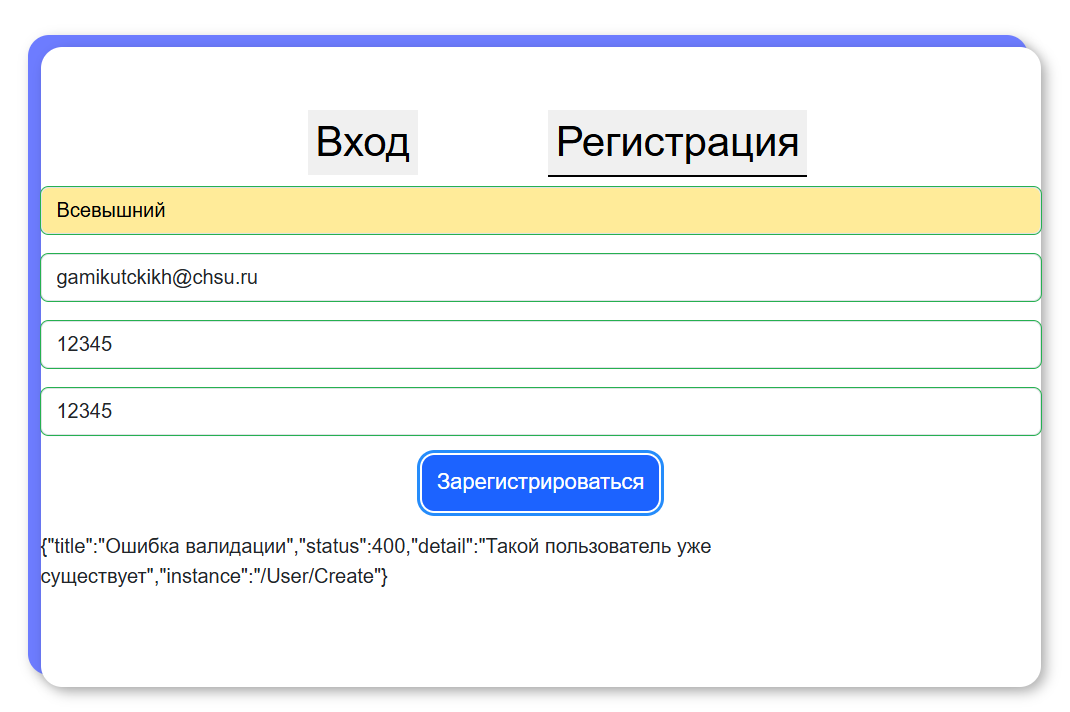


Рис. П3.3. Введён уже использующийся логин

Если пользователя не существует, программа выдаст соответствующее сообщение (рис. П3.4).

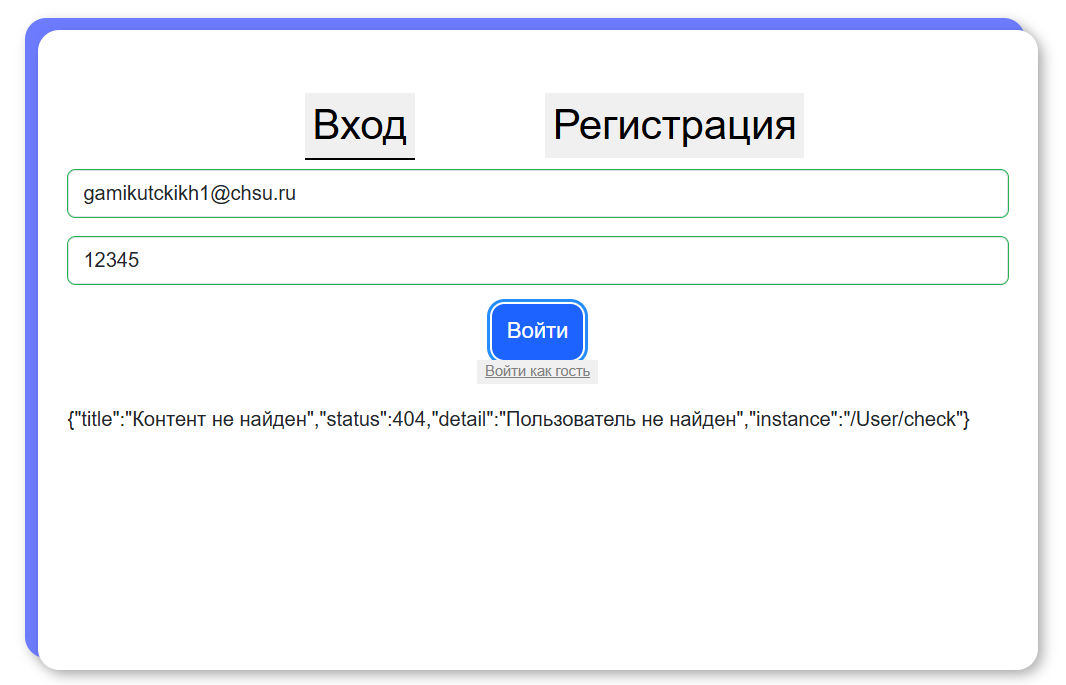


Рис. П3.4. Ошибка «Пользователь не найден»

После входа на выбор даётся два раздела: страница шаблонов и страница документов (рис. П3.5).

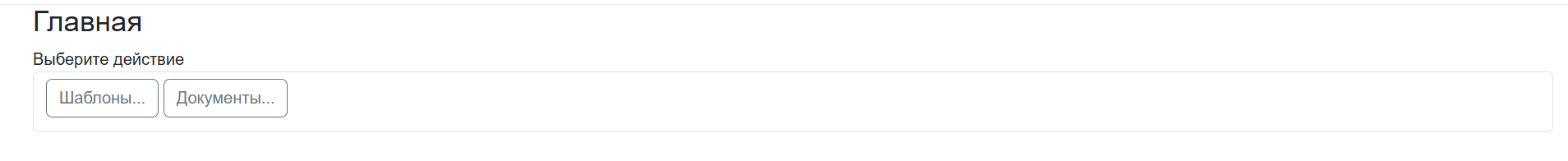


Рис. П3.5. Главная страница

Страница шаблонов выводит все существующие шаблоны, позволяет создать новый или изменить существующий шаблон (рис. П3.6).

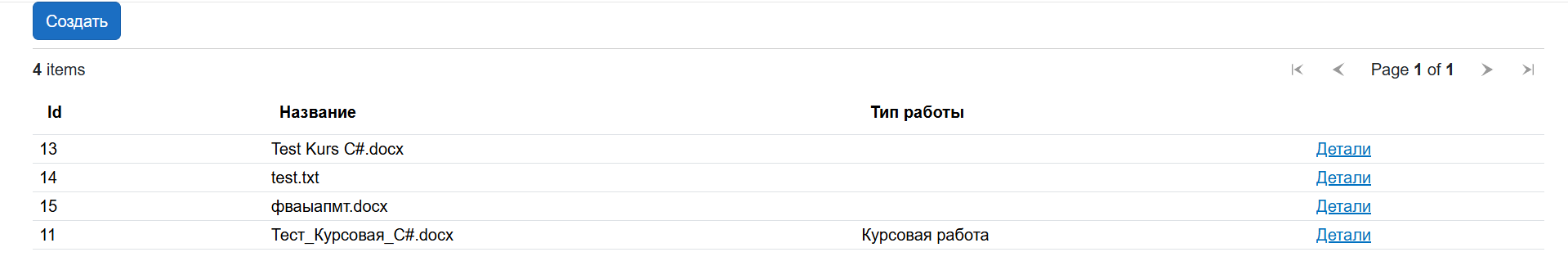


Рис. П3.6. Выведены все шаблоны из БД

Разделы «Создать» и «Детали» позволяют загружать свои «.docx»-документы, ввести имя шаблона и указать теги для замены. Пример заполненных данных шаблона представлен рис. П3.7.

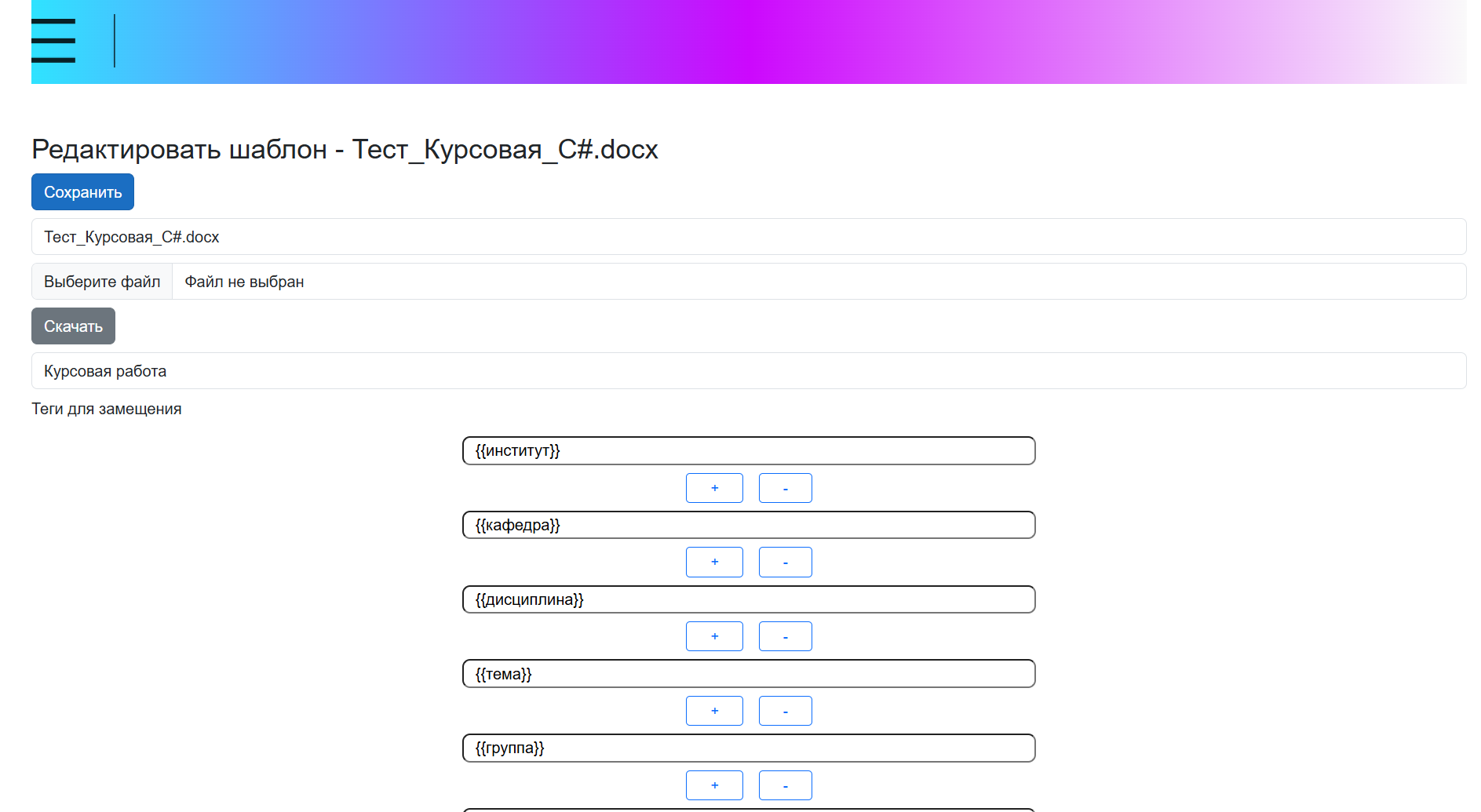


Рис. П3.7. Пример заполненных данных шаблона

Важно: стиль текста (в Word) должен быть один у всего тега. Для уверенности рекомендуется выполнить над тегом последовательно команды «Ctrl + X, Ctrl + V».

Для примера теги могут быть следующие:

* {{student}} – ФИО студента;
* {{teacher}} – ФИО руководителя;
* {{plot}} – тема работы;
* {{year}} – год сдачи работы;
* прочее (может быть обычным словом или текстом особого стиля/содержания).

Страница документов выводит все существующие документы, позволяет создать новый или изменить существующий (рис. П3.8).

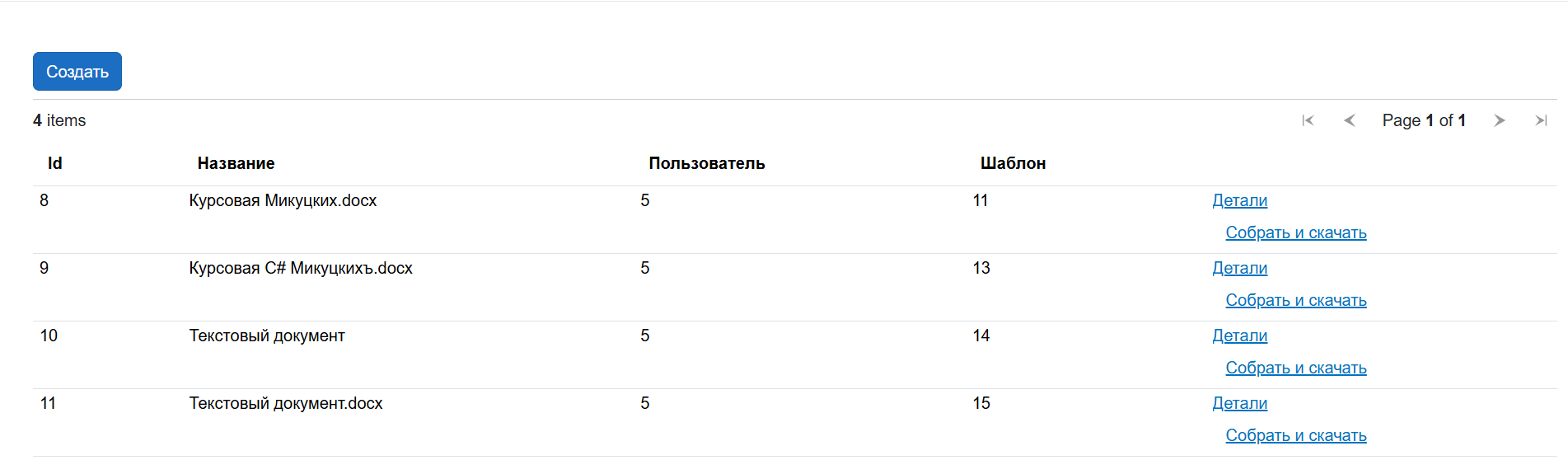


Рис. П3.8. Пример вывода всех документов

Кнопка «Создать» выводит список всех доступных шаблонов, который необходимо выбрать перед созданием документа (рис. П3.9).

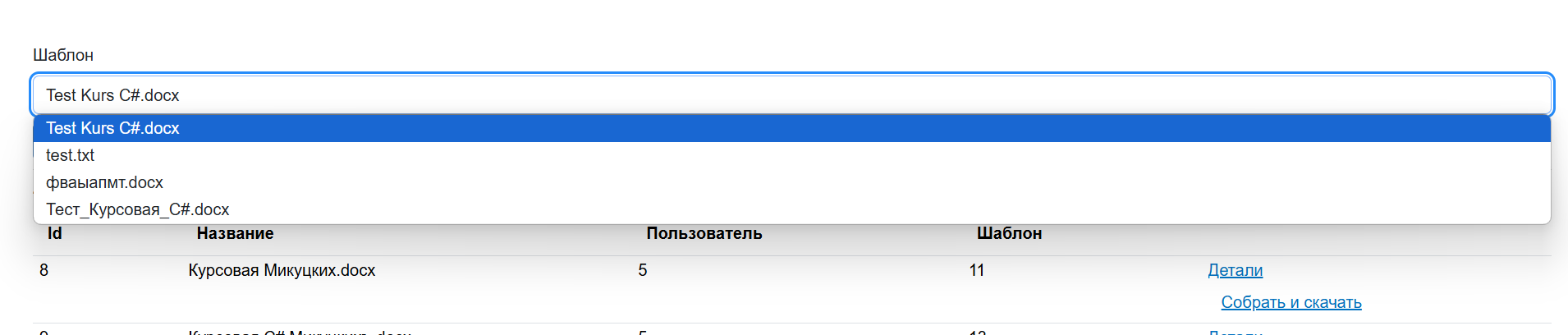


Рис. П3.9. Выпадающий список всех шаблонов

Страница для создания документа берёт теги шаблона и ставит им в соответствие пользовательское содержание. Для создания документа необходимо знать свой пользовательский ID, уточните его у администратора базы данных (рис. П3.10).

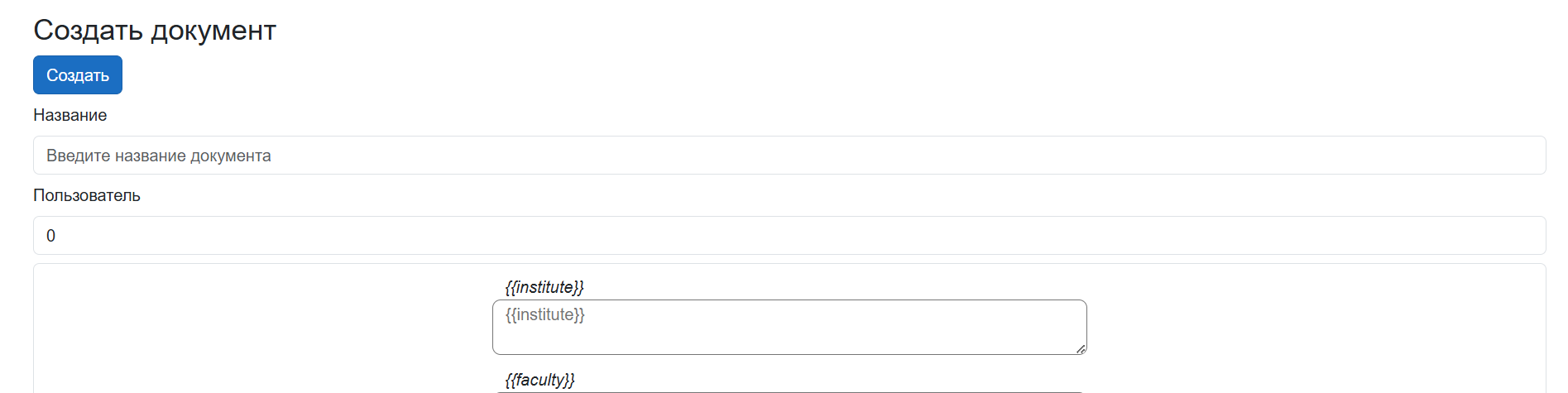


Рис. П3.10. Раздел «Создать Документ»

Пример заполненных данных документа представлен на рис. П3.11.

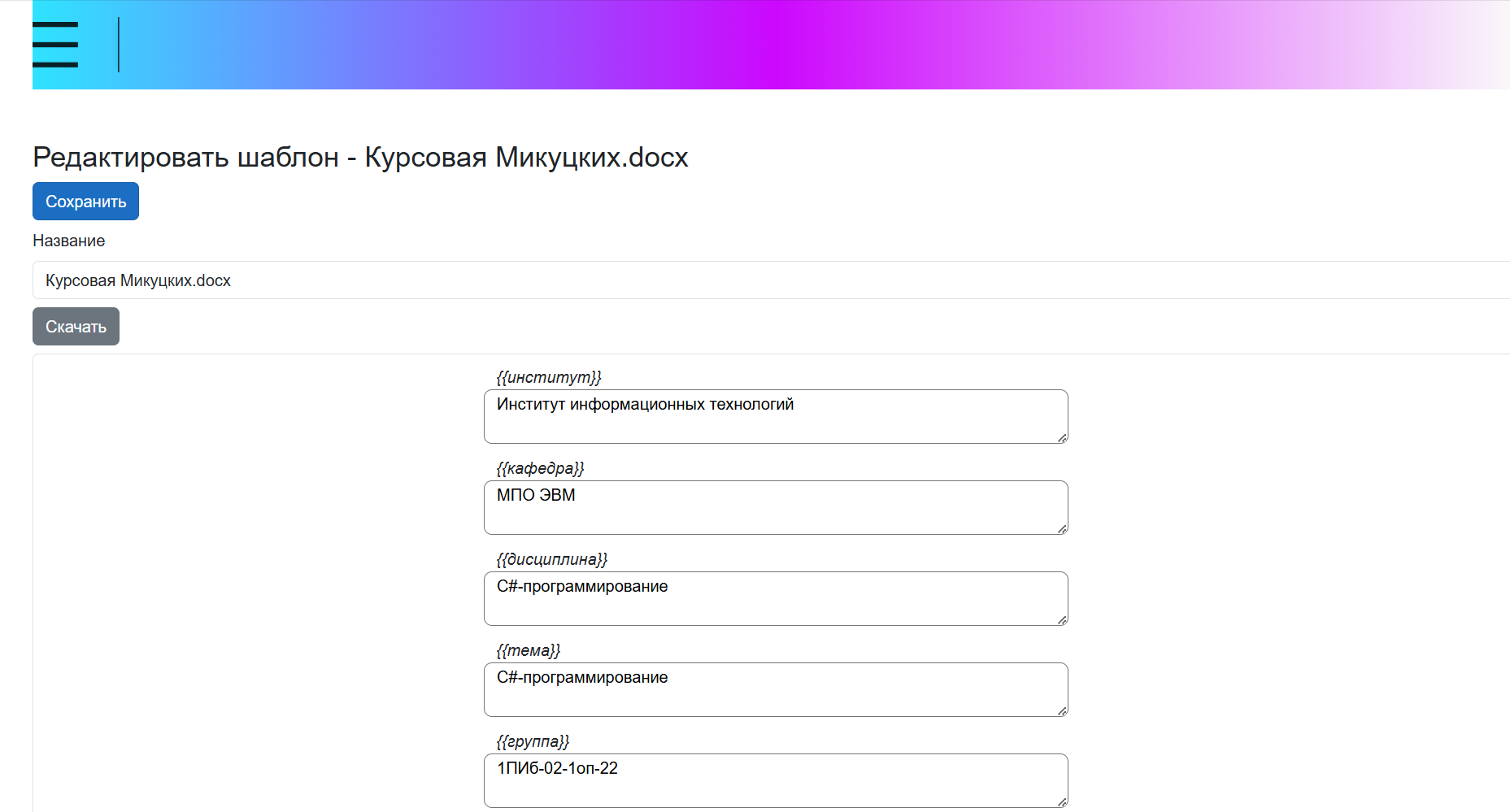


Рис. П3.11. Пример заполненных данных документа

Готовый документ можно собрать и скачать (см. рис. П3.8). После нажатия кнопки «Собрать и скачать» будет собран документ и через некоторое время инициирована загрузка итогового документа через браузер.

Результат работы программы представлен на рис. П3.12-П3.13.

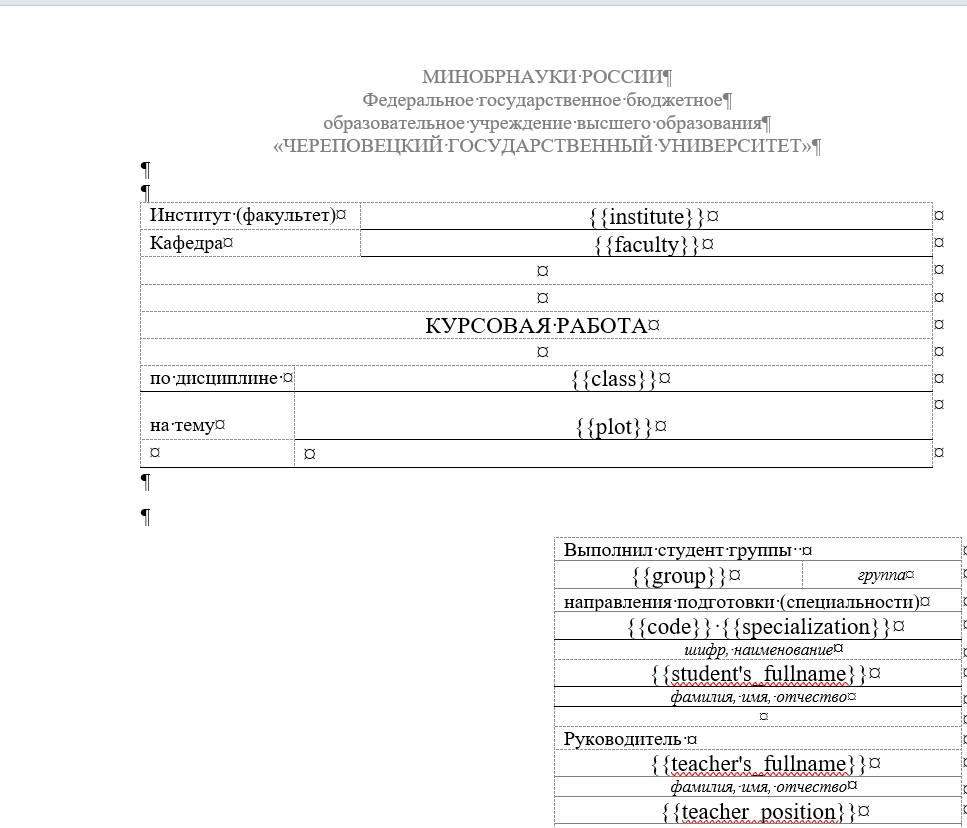


Рис. П3. 12. Пример шаблона

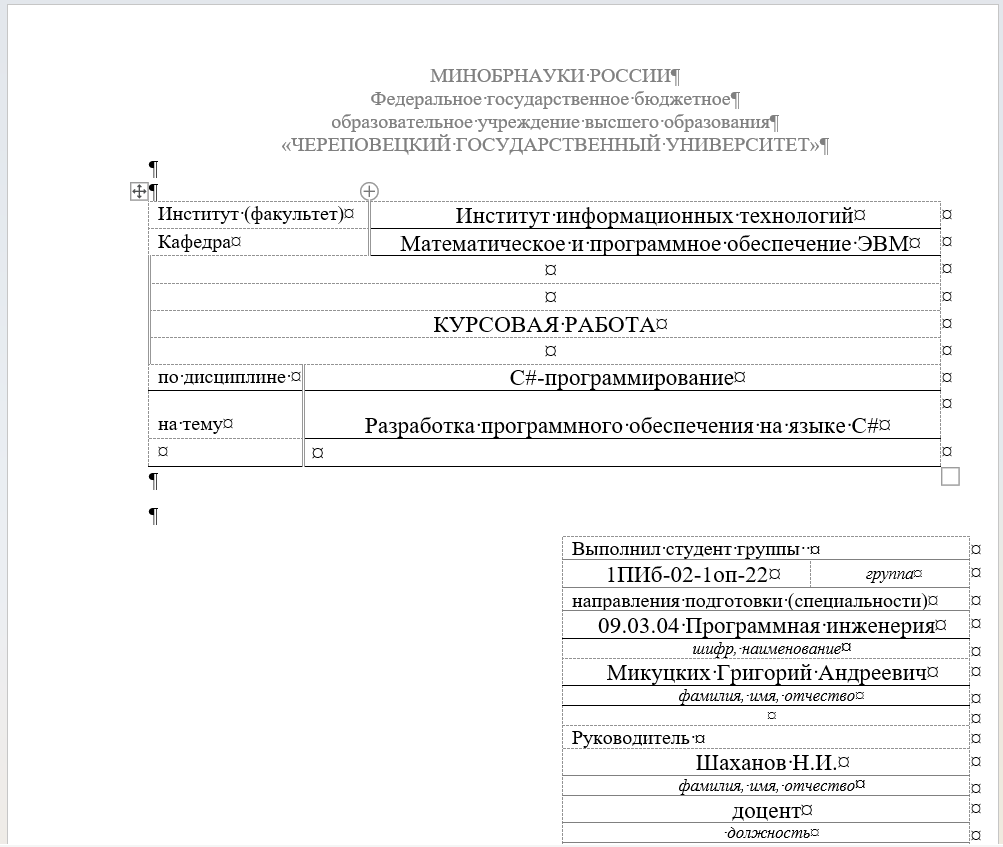
****

Рис. П3. 13. Пример вставленных данных

Завершение работы

Для завершения работы программы необходимо закрыть вкладки в браузере и два консольных окна, открывшихся при запуске программы. Закрытие программ осуществляется нажатием крестика «❌» в правом верхнем углу окна.