Санкт-Петербургский государственный университет

Коекин Ярослав Алексеевич

Выпускная квалификационная работа

Разработка веб-редактора для ведения электронных структурированных документов ECM-системы

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5006.2017 «Математическое обеспечение и администрирование информационных систем»*

Профиль *Системное программирование*

Научный руководитель:
к.ф.-м.н., доцент кафедры системного программирования Луцив Д. В.

Рецензент:
Старший разработчик серверных решений ООО «ДоксВижн» Кононов Н. С.

Санкт-Петербург
2021

Saint Petersburg State University

Iaroslav Koekin

Bachelor's Thesis

Web editor development for maintaining electronic structured documents of ECM system

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5006.2017 "Software and Administration of Information Systems"*

Profile: *System Programming*

Scientific supervisor:
C.Sc., docent Dmitry Luciv

Reviewer:
Senior developer of server solutions "DocsVision" Nikita Kononov

Saint Petersburg
2021

Содержание

Введение **4**

1 Постановка задачи **6**

2 Обзор **8**

2.1 Существующие текстовые редакторы 8

2.2 Технологии реализации совместного редактирования 11

2.2.1 OT- и CRDT-технологии 11

2.2.2 Решения для совместного редактирования документов . . 12

2.3 Вывод 13

3 Архитектура приложения **15**

3.1 Компоненты приложения 15

3.2 Текстовый редактор 16

4 Управление основными сущностями документа **19**

4.1 Управление документами и его блоками 19

4.2 Создание и редактирование атрибутов документа 20

5 Особенности реализации текстового редактора **21**

6 Тестирование **25**

7 Заключение **26**

Список используемой литературы **27**

Введение

ECM (англ. Enterprise Content Management, рус. Управление Корпоративным Контентом) — осуществление деятельности, направленное на хранение, обработку, распространение и управление цифровыми документами в рамках организации. ECM-система — программное обеспечение для управления корпоративным контентом. В список функциональных возможностей такой системы входят: управление документами (экспорт и импорт цифровых документов, контроль версий, архивирование контента), управление потоками работ (поддержка бизнес-процессов, передача контента по маршрутам), документоориентированное взаимодействие (поддержка возможности совместного использования цифровых документов пользователями) и др. Системы такого типа создаются в целях автоматизации и упрощения работы с большим количеством неструктурированной информации различного рода и формата.

Платформа Docsvision – полнофункциональная ECM-платформа, позволяющая упростить работу с корпоративным контентом.

На данный момент существует большое количество текстовых редакторов и форматов для создания и редактирования цифровых документов. Некоторые из них предоставляют инструменты для работы со структурными элементами текста, выходящие за рамки форматирования и выравнивания.

Основными объектами, управление которыми осуществляется ECM-системами, являются цифровые документы. Вместе с цифровыми документами в системе ECM хранятся дополнительные атрибуты: данные о документе, данные о пользователях, имеющих доступ к документу и др.

Упростить работу с цифровыми документами могут следующие решения.

- Автоматизация создания атрибутов и возможность интегрировать их в текст цифрового документа. Такое решение облегчает создание атрибутов документа и делает работу с этими данными более наглядной.

- Использование веб-приложения для просмотра и редактирования документов. Такой подход делает систему более доступной.

- Разделение документа на сгруппированные элементы (блоки) и поддержка независимости этих блоков. Это позволит работать с ними как с отдельными документами, что увеличит гибкость управления цифровым кон-

тентом. Например, в такой ситуации разные люди могут иметь доступ на редактирование различных параграфов одного документа при совместной работе над ним без доступа к целому документу.

Подобные рассуждения являются мотивацией для создания и реализации альтернативного подхода при работе с цифровыми документами в ECM-системах.

1 Постановка задачи

Классический сценарий работы с ECM-системой предлагает пользователю:

1. создавать цифровой документ, содержащий текст, графики, таблицы и другие объекты, формируя его в монолитный файл некоторого формата;
2. загружать такой документ в систему ECM;
3. создавать карточку, которая представляет собой структурированный набор атрибутов и файл документа (файлы документов).

У такого подхода есть ряд ограничений.

- Создание цифрового документа производится отдельно от ECM-системы.
- Задание атрибутов документа отделено от процесса редактирования документа.
- Невозможность регулирования прав доступа (просмотр, модификация) отдельных частей документа.

Альтернативной же работой с цифровыми документами является возможность создавать и редактировать их непосредственно в ECM-системе, интегрируя содержимое документа с атрибутами карточки. Кроме того, удобным кажется дать возможность работать с отдельными наборами элементов документа (блоками) как с самостоятельным документом.

Таким образом, целью данной работы является создание текстового веб-редактора для ведения электронных структурированных документов в системе ECM.

Для достижения данной цели были поставлены следующие задачи:

1. Изучить возможности и способы расширения существующих текстовых редакторов, проанализировав их на целесообразность использования в качестве основы для будущего редактора ECM-системы.
2. Разработать архитектуру веб-редактора.
3. Реализовать следующие функции веб-редактора:

- поддержка механизма интеграции атрибутов документа с его содержимым;
- возможность объединять элементы документа в блоки и управлять ими отдельно (задавать доступ к блокам документа для пользователей и версионировать блоки независимо);
- возможность создавать шаблоны на основе документа;
- поддержка совместного редактирования документа и блоков документа между пользователями;
- возможность экспорта документа в формат PDF;
- поддержание механизмов редактирования документа (форматирование текста, выравнивание, добавление таблиц, изображений и др.).

4. Провести тестирование полученного приложения.

2 Обзор

2.1 Существующие текстовые редакторы

Для того, чтобы достичь поставленную цель, необходимо провести анализ существующих текстовых редакторов, которые имеют все необходимые функциональные требования или которые могут быть расширены до удовлетворения всем требованиям.

Microsoft Word Online

Microsoft Word[1] является одним из самых популярных WYSIWYG[2] текстовых редакторов. Такой редактор позволяет создавать, редактировать и сохранять документы, включает в себя расширенные возможности по форматированию текста, проверяет орфографию и грамматику при вводе текста, позволяет отправлять документ по электронной почте, содержит встроенную систему подсказок и др. Благодаря популярности редактора, бинарный формат документа, которым он оперирует, является форматом де-факто для ECM-систем. Онлайн версия этого редактора Microsoft Word Online позволяет пользоваться многими функциями редактора без его непосредственной установки на ПК.

Microsoft Word Online наследует от Microsoft Word богатые возможности по форматированию текста, шаблонирование, совместное редактирование текста и др. Но, к сожалению, функциональных возможностей Microsoft Word Online недостаточно для удовлетворения всех поставленных требований.

Для расширения редактора Microsoft Word Online существует Word JavaScript API [1], который, основываясь на Office JavaScript API, предоставляет доступ к структурам документа и редактора. Office JavaScript API для редактора Word включает в себя две объектные модели[3]:

- Word JavaScript API – представлен вместе с Office 2016. Объединяет структуры, позволяющие взаимодействовать со специфичными для текстового редактора Microsoft Word объектами (доступ к документу и его метаданным, таблицам, форматированию и т. д.).

[1] https://www.microsoft.com/en/microsoft-365/word

[2] WYSIWYG (What You See Is What You Get) – что видишь, то и получаешь. Концепция различного рода интерфейсов, при котором промежуточный результат создания (например, редактирование) максимально похож на результат конечный.

[3] https://docs.microsoft.com/office/dev/add-ins/reference/overview/word-add-ins-reference-overview

- Common API – представлен вместе с Office 2013. Предоставляет доступ к структурам, которые могут взаимодействовать с общими для всей линейки редакторов Office объектами (пользовательский интерфейс, диалоги и т. д.).

Не смотря на это, возможности по расширению Microsoft Word Online весьма ограничены и не подходят для удовлетворения всех исходных требований. Так, к примеру, расширения Microsoft Word Online не позволяют иметь в контексте одного документа редактируемые и нередактируемые структурные единицы текста.

OnlyOffice

OnlyOffice[4] – офисный пакет с открытым исходным кодом. Его документный модуль, в состав которого входит и текстовый веб-редактор, так же, как и Microsoft Word, имеет расширенные возможности по редактированию текста: от разнообразных вариантов форматирования текста до сервиса совместного редактирования документов.

Как и Microsoft Word Online, текстовый веб-редактор OnlyOffice имеет плагинную систему OnlyOffice API [2], позволяющую дополнять функциональные возможности редактора. Однако, и её возможности достаточно ограничены и не позволяют в полной мере удовлетворить все функциональные требования к реализации текстового веб-редактора.

Имеется возможность изменения исходного кода OnlyOffice, но для такого комплексного программного продукта это является нетривиальной задачей и представляется нецелесообразным в рамках данного проекта. Более того, документный модуль OnlyOffice оперирует файловым форматом docx[5], имеющим широкие возможности по хранению и управлению контентом цифровых документов. Однако, в таком объёме возможностей в рамках данного проекта нет необходимости, и данный формат представляется перегруженным и сложным в поддержке.

TinyMCE

TinyMCE[6] – open-source веб WYSIWYG текстовый редактор, используемый в системах управления контентом, системах управления знаниями, систе-

[4] www.onlyoffice.com/

[5] https://docs.microsoft.com/openspecs/office_standards/ms-docx

[6] https://github.com/tinymce/tinymce

мах управления отношениями с клиентами и других. Являясь HTML редактором, он может быть интегрирован с существующими веб-приложениями. Редактор имеет пользовательский интерфейс приложения (API), который позволяет расширять уже существующие функциональные возможности [3]. А также у данного редактора есть богатая библиотека встроенных расширений (дополнительные возможности по форматированию текста, совместное редактирование документа, возможность оставлять комментарии к тексту и прочие). Манипулирование текстом в таком редакторе основывается на управлении объектной модели документа DOM[7]. Это приводит к ограничениям при расширении, которые связаны с тем, что DOM структура не знает про все свойства, присущие тексту. Более того, многие из этих расширений платные, а открытый код находится под лицензией LGPL-2.1 License, что не даёт свободно изменять и расширять существующий редактор.

Quill Rich Text Editor

Quill[8] – расширяемый текстовый веб редактор с открытым исходным кодом под лицензией BSD-3-Clause License. Архитектура такой системы основывается на способности редактора быть расширяемым [4]. Кроме этого, Quill имеет надстройку поверх DOM для работы с текстом, которая предоставляет дополнительные возможности при взаимодействии со структурными единицами текстового документа. Однако, Quill предоставляет широкие возможности только для работы с заранее заданным набором элементов, но предлагает нетривиальные способы при создании и управлении собственными.

slate.js

Slate[9] — фреймворк для создания богатых текстовых редакторов с открытым исходным кодом под лицензией MIT. Создание данной библиотеки было вдохновлено в том числе и библиотекой Quill. Slate имеет следующие преимущества.

- Ориентированность на расширяемость — Slate предоставляет ядро и механизмы для добавления новых функциональных возможностей к этому ядру.

[7] developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
[8] github.com/quilljs/quill
[9] https://github.com/ianstormtaylor/slate

- Модель документа внутри редактора представляет собой структурированное дерево в формате JSON и основывается на объектной модели документа DOM, что, в свою очередь, предоставляет возможность создания сложных вложенных структур.

- Использование атомарных команд при модифицировании текста. Каждое изменение структуры документа создаёт команду внутри Slate. Такой подход позволяет явно следить за изменениями документа и облегчает разработку инструментов, зависящих от них (например, реализация истории состояний документа или совместное редактирование текста).

- Представление пользовательского интерфейса происходит с использованием технологии React[10], что даёт доступ к обширной документации инструмента, а также опыт крупного сообщества технологии в случае возникновения проблем при разработке интерфейса [10].

Slate предоставляет текстовый редактор с минимальным количеством функциональных возможностей, но даёт инструменты для полной его кастомизации [5]. В сообществе существуют библиотеки различных расширений для Slate с открытым исходным кодом под лицензией MIT[11], в которую входят расширения для форматирования и выравнивания текста, расширения для работы с таблицами, изображениями и другими структурами, расширения для сериализации/десериализации структуры документа Slate в HTML, расширения для идентификации структурных единиц и другие.

2.2 Технологии реализации совместного редактирования

2.2.1 OT- и CRDT-технологии

Operational Transformation (OP) — технология совместного редактирования, при которой каждое действие над документом разбивается на операции, которые синхронизируются между пользователями. На данный момент имеет

[10] https://reactjs.org/

[11] например, https://github.com/udecode/slate-plugins

широкое распространение подход, при котором каждый экземпляр документа взаимодействует лишь с промежуточным сервером. Такой сервер принимает изменения каждого документа, разрешает конфликты, предоставляя доступ к изменениям остальным клиентам. Подобные подходы используются в Google Docs, Apache Wave, Tiny MCE и других редакторах. К сожалению, реализация таких технологий — задача комплексная и связана со сложностью реализации промежуточного сервера.

В целях уменьшения сложности при реализации были выдвинуты новые подходы к построению совместного редактирования. Подход, основанный на Conflict Free Replicated Data Types (CRDT) предполагает создание специальной копии документа (так называемой бесконфликтной структуры) в каждом клиентском приложении. Тогда изменения документа переводятся в эквивалентные изменения этих структур, которые гарантируют разрешение конфликтов самостоятельно. После этого изменения передаются оставшимся участникам совместного редактирования документа. Таким образом, достигается синхронизация между различными локальными бесконфликтными структурами. А благодаря синхронизации таких структур, синхронизируются и основные документы [13].

Простота поддержки являлась решающим фактором при выборе технологии. Именно поэтому было решено остановиться на методе, основанном на CRDT.

2.2.2 Решения для совместного редактирования документов

Automerge

Automerge[12] — JavaScript библиотека, реализующая подход к совместному редактированию, основанный на бесконфликтных структурах. Его особенностями являются.

- Независимость от способов передачи изменений. Библиотека не накладывает ограничения на виды взаимодействий между клиентами, и связь может быть реализована на основе клиент-серверного взаимодействия, peer-to-peer, Bluetooth и других.

[12]https://github.com/automerge/automerge

- Неизменяемые состояния. Объект Automerge имеет неизменяемое состояние, что приводит к совместимости с функциональным реактивным стилем, присуще инструментам React и Redux.

- Портативность. Automerge протестирована на различных платформах, таких как: Chrome, Safari, Electron, Node.js и др.

Yjs

Другой инструмент, реализующий подход на основе CRDT — Yjs[13]. Этот инструмент был в первую очередь разработан для передачи содержимого различных документов. Структура данной реализации основана на работе «Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types» [14] (Petru Nicolaescu, Kevin Jahns, Michael Derntl, Ralf Klamma). Данная реализация так же, как и предыдущая независима от способов передачи изменений. В отличие от объектов Automerge, объекты Yjs являются мутабельными. Yjs, будучи JavaScript библиотекой, также имеет порты на другие языки (например, C# и Rust), что может быть важным аспектом при реализации приложения, основанного на уровневой архитектуре и написанного на различных языках программирования.

2.3 Вывод

Таким образом, так как возможности Office JavaScript API для Microsoft Word Online и API плагинной системы документного модуля OnlyOffice ограничены и не подходят для достижения поставленной задачи, а формат файлов, с которым оперируют данные редакторы является избыточным для реализации структур данной работы, эти варианты были исключены.

Кроме этого, так как TinyMCE имеет неподходящие для данного проекта политики доступа к исходному коду, необходимо рассматривать 2 инструмента: Quill и Slate. Благодаря доступу к крупному репозиторию расширений, понятной внутренней архитектуре и поддержке популярной библиотеки для взаимодействия с редактором, целесообразно выбрать фреймворк Slate как основу для будущего редактора, удовлетворяющего всем функциональным требованиям.

В качестве инструмента для реализации совместного редактирования был выбран Yjs, так как он основывается на подходе CRDT, имеет порт на язык раз-

[13]https://github.com/yjs/yjs

работки серверной части приложения C# и является достаточно эффективным по-сравнению с другими похожими инструментами[14].

[14]https://github.com/dmonad/crdt-benchmarks

3 Архитектура приложения

3.1 Компоненты приложения

По своей структуре приложение имеет клиент-серверную архитектуру, где на стороне клиента в первую очередь находится текстовый редактор, а на стороне серверной части — бизнес-логика взаимодействия между основными сущностями: пользователями, документами, блоками и атрибутами документа. Архитектура приложения представлена на рис 1.

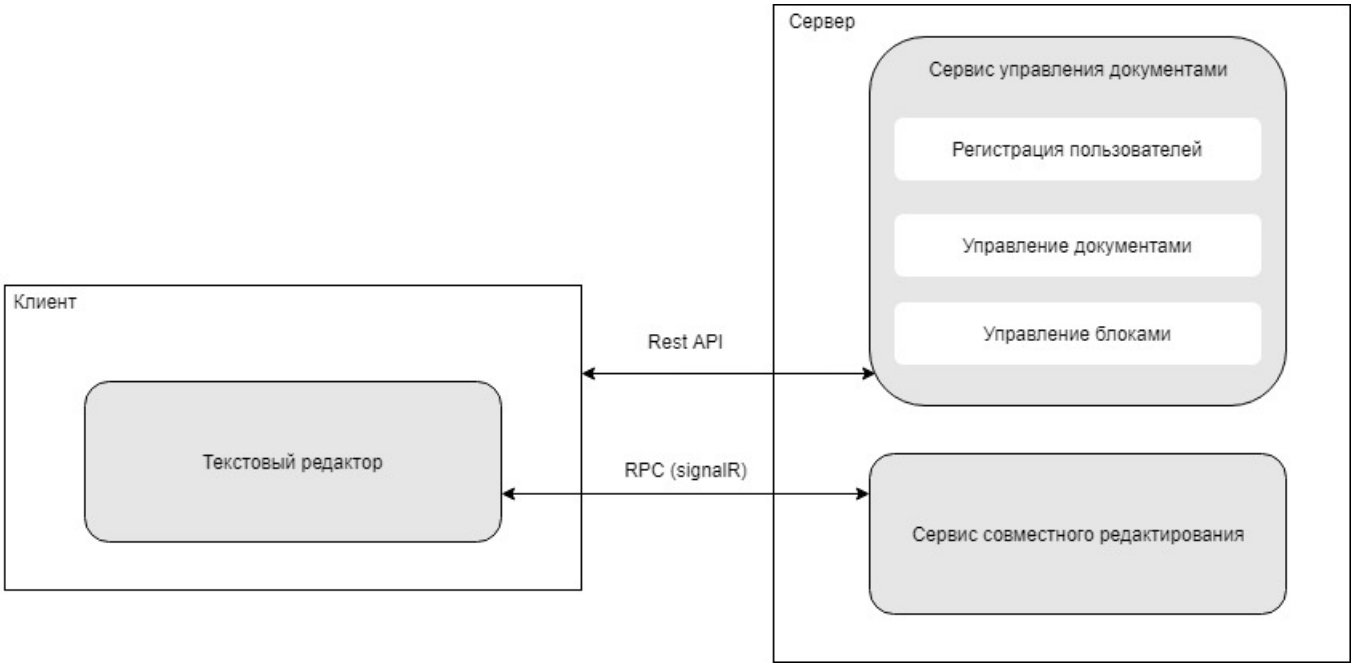

Рис. 1: Архитектура приложения

На изображении можно увидеть, что сервер состоит из двух частей.

1. Сервис управления документами, который отвечает за регистрацию пользователей, сохранение документов и управление доступами документов и блоков между пользователями. Связь осуществляется согласно архитектурному стилю REST API[15].

2. Сервис совместного редактирования, который управляет связями между клиентами и призван предоставить инструменты для параллельного редактирования документов и блоков между различными пользователями.

[15] https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Такое взаимодействие в режиме реального времени реализовано с помощью библиотеки SignalR [8].

Сам же текстовый веб-редактор является частью клиентского приложения. Помимо доступа к текстовому редактору, на стороне клиента находятся интерфейсы для регистрации пользователей и просмотра информации о документах и блоках.

3.2 Текстовый редактор

Текстовый редактор — основная часть приложения. Он позволяет: создавать и редактировать текст и элементы документа (параграфы, таблицы, графики, нумерации и т. д.), задавать атрибуты документа вместе с текстом, создавать шаблоны документа, экспортировать документ в формат PDF, а также имеет возможность объединять текст и элементы документа в блоки. Редактор позволяет версионировать документ и блоки, а также управлять доступами (редактирование или чтение) документа и блоков, из которых он состоит. Помимо этого, редактирование содержимого документа и блоков может происходить совместно с другими имеющими на это право пользователями.

Встраивание собственного редактора в фреймворк slate.js происходит с двух сторон:

- во-первых, необходимо описать свои собственные структуры содержимого документа, с которыми работает редактор;
- во-вторых, необходимо описать новые возможности, расширяющие базовую реализацию редактора slate.js.

Структура документа

С точки зрения структуры документа slate.js работает с двумя типами объектов: промежуточный и текстовый. Промежуточные объекты могут состоять из других промежуточных объектов или из множества текстовых объектов. Текстовые объекты, в свою очередь, помимо самого текста могут содержать различные добавочные метки (например, метка bold в текстовом объекте показывает, что его текст необходимо отображать жирным). Промежуточные объекты в такой древовидной структуре могут быть описаны разработчиком и используются, например, для описания элементов документа: параграфы, таблицы,

заголовки, графики и др. Многие такие элементы уже разработаны и доступны[16]. Структура документа, с которой работает разработанный веб-редактор, имеет древовидный вид и изображён на рис 2. Документ состоит из блоков, каждый из которых, в свою очередь, имеет объединение различных промежуточных объектов. Листьями такого дерева обязательно будут текстовые объекты. Встраивание собственной структуры блока в систему slate.js происходит путём наследования базового промежуточного элемента.

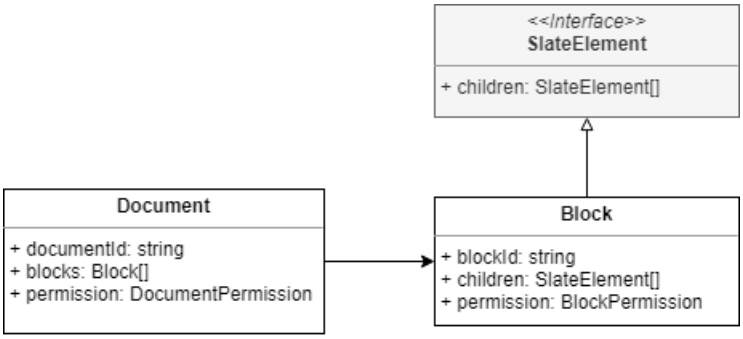

Рис. 2: Структура документа

Используя данную возможность, был реализован новый элемент — блок, позволяющий объединить базовые элементы в один набор и работать с такими наборами независимо от документа и остальных блоков.

Редактор

Архитектура текстового редактора представлена на рис. 3. На ней можно увидеть встраивание собственных инструментов в экземпляр базового редактора slate.js. Таким образом, редактор позволяет создавать и удалять блоки, сохранять документ, делиться им, сохранять и делиться блоками, а также создавать шаблоны и экспортировать содержимое документа.

[16]например, https://github.com/udecode/slate-plugins

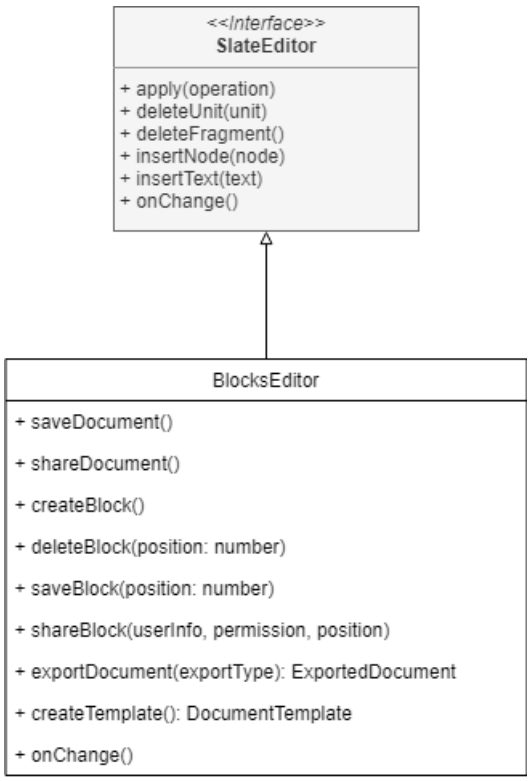

Рис. 3: Архитектура текстового редактора

Приложение позволяет просматривать и редактировать каждый блок отдельно. В этом случае, структура документа будет состоять только из одного блока, а редактор запрещать некоторые действия над блоком (сохранение и распространение документа, создание и удаление блока).

4 Управление основными сущностями документа

Серверная часть приложения оперирует основными сущностями приложения (пользователи, документы и блоки) согласно архитектуре, показанной на рис 4.

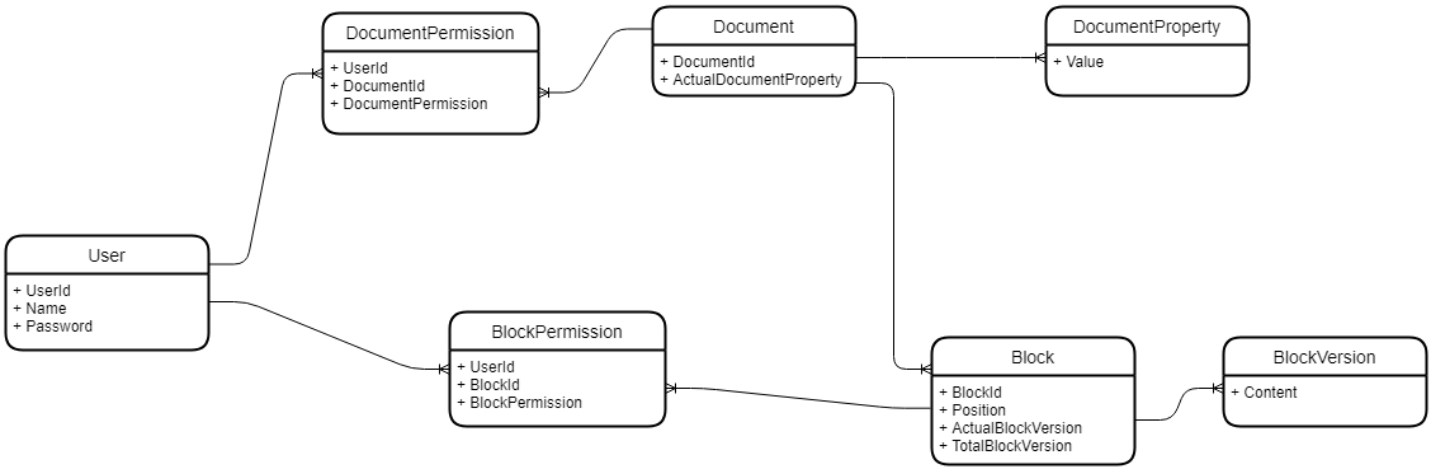

Рис. 4: Основные сущности

4.1 Управление документами и его блоками

Благодаря подходу, описанному на рис. 4, доступы к документу и к его блокам являются отдельными сущностями. Это позволяет комбинировать различные доступы. Например, иметь доступ к документу на редактирование, но доступ к отдельному его блоку только на чтение.

Помимо этого, каждый блок имеет множество версий, среди который одна актуальная. Таким образом, понятие «версия» применима не только ко всему документу, но и к отдельному блоку внутри него.

С точки зрения управления документами и его блоками, серверная часть приложения отвечает за:

- создание и удаление нового документа;
- создание документа на основе шаблона;

- предоставление информации о документе и его блоках;
- добавление и удаление блока внутри документа;
- сохранение документа и блоков;
- возможность делиться документом или его блоками между пользователями.

Кроме этого, серверная часть следит за корректностью выполнения этих действий в рамках разрешений между пользователями с одной стороны, и документами и его блоками с другой. Так, например, удаление документа пользователем, который создал этот документ, влечёт удаление документа у всех пользователей, с которыми он поделился этим документом. И наоборот, при попытке удалить документ пользователем, не являющимся владельцем этого документа, сотрётся лишь информация о связи между этим документом (и его блоками) и данным пользователем.

4.2 Создание и редактирование атрибутов документа

Документу в системах ECM присущи многие атрибуты. Примерами могу быть имя документа, его описание, названия организаций, номера телефонов и др. На уровне сервера каждый такой атрибут задаётся отдельно и имеет свою историю версий.

Древовидная структура документа текстового веб-редактора позволяет задавать дополнительные маркеры каждому текстовому объекту, для интеграции тех или иных атрибутов прямо в текст документа. Таким образом, задание атрибутов клиентом на стороне редактора осуществляется путём добавления информации об атрибуте в текстовые элементы документа. Извлечение же атрибутов документа на стороне серверной части приложения происходит путём нахождения текстовых объектов, помеченных маркером конкретного типа.

5 Особенности реализации текстового редактора

Совместное редактирование

Для того, чтобы совместное редактирование работало как и между экземплярами документа, так и между экземплярами блоков, решено было создавать отдельную бесконфликтную структуру для каждого блока внутри документа. Каждая такая бесконфликтная структура обновляется вместе со своим экземпляром блока и синхронизируется с такими же структурами всех активных на момент редактирования пользователей. Благодаря схожести структур данных, с которыми работает CRDT реализация Yjs, с объектами документа, перевод структуры блока в структуру Yjs является естественным и реализуется различными библиотеками[17].

На рис. 5 представлена диаграмма последовательностей инициализации совместного редактирования для документа.

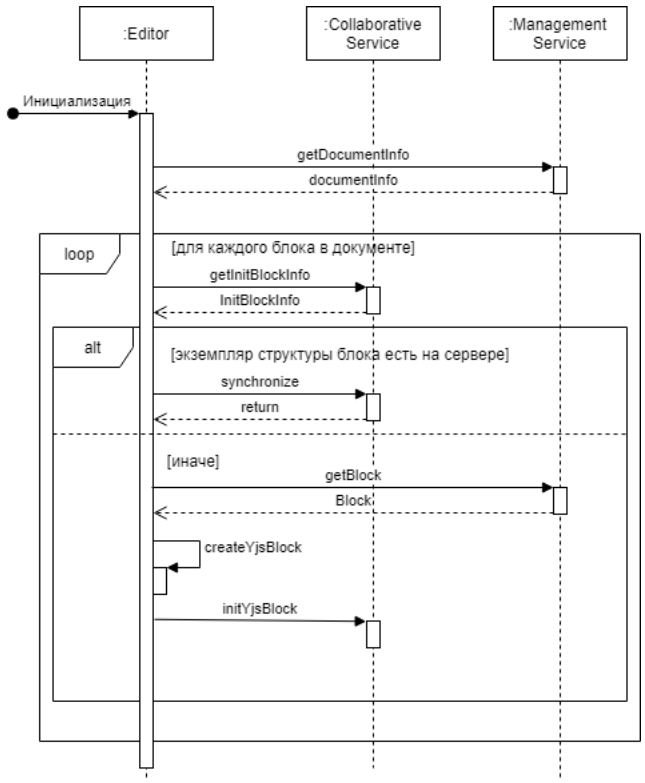

Рис. 5: Инициализация документа

Приложение клиента узнаёт у сервера основные данные о документе (иден-

[17]например, https://github.com/BitPhinix/slate-yjs

тификатор документа, идентификаторы блоков, из которых он состоит, информация о доступах к документу и каждому блоку). Далее, для каждого блока оно узнаёт, имеет ли сервис совместного редактирования доступ к его бесконфликтной структуре. Если сервис не имеет доступа, то инициализация блока происходит путём забора контента из хранилища, иначе путём восстановления блока из бесконфликтной структуры сервиса.

Редактирование и содержимого блока, и бесконфликтной структуры может происходить параллельно и с той, и с другой стороны: при изменении содержимого блока пользователем (например, при добавлении текста одному из параграфов блока), система параллельно изменяет бесконфликтную структуру данных. И при каждом таком изменении новые обновления посылаются сервису совместного редактирования, а тот, в свою очередь, доносит их до всех клиентов. И наоборот, когда приложение клиента принимает изменения бесконфликтной структуры от сервиса, оно переводит их в изменения содержимого блока, отображая их в структуре документа. Более подробно это показано на рис 6.

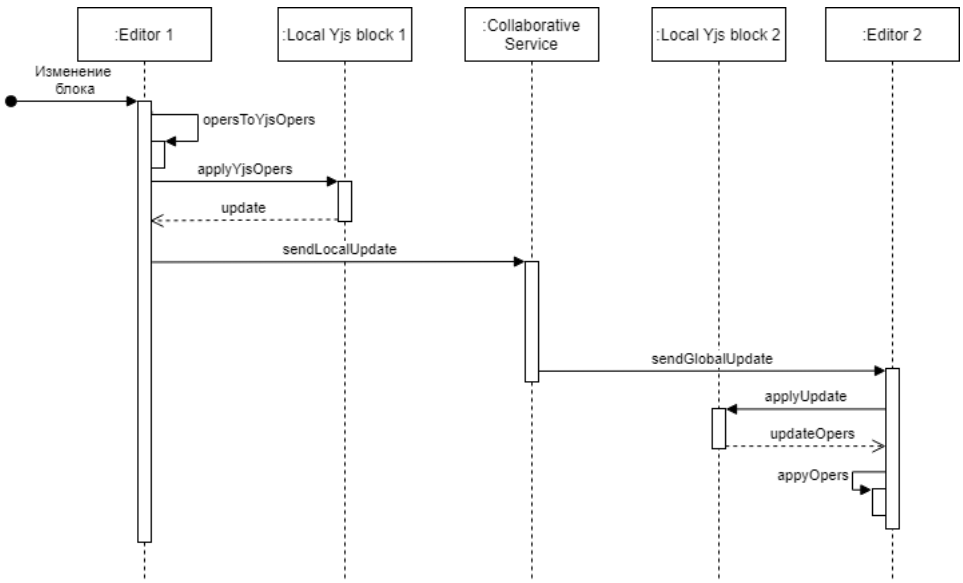

Рис. 6: Редактирование документа

Для предоставления вышеописанных функциональных возможностей был изменён редактор: переопределены методы по созданию и удалению блока, добавлены методы, переводящие изменения между блоками и бесконфликтными структурами Yjs. (рис. 7).

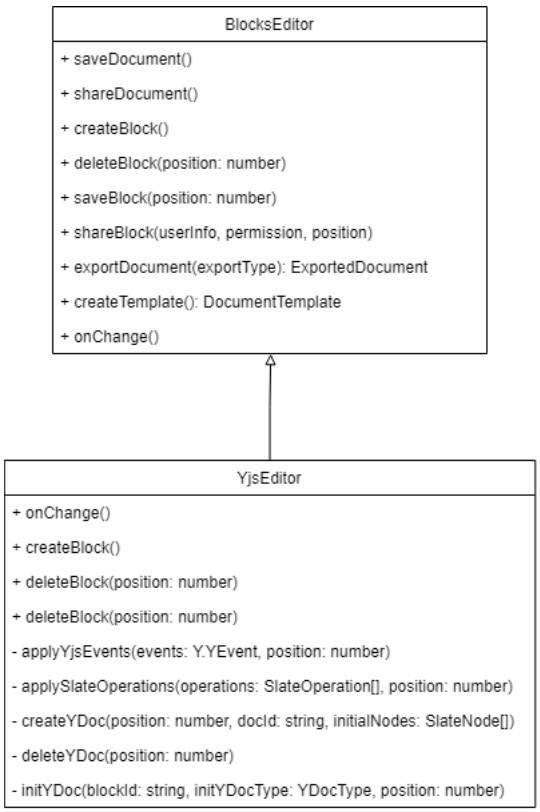

Рис. 7: Редактор с инструментами совместного редактирования

Шаблонирование

Шаблон создаётся на основе всего документа и представляет собой экземпляр документа, состоящий из содержимого актуальных на момент шаблонирования блоков. При этом атрибуты из оригинального документа извлекаются и заменяются на атрибуты-шаблоны, подсказывающие, что они требуют заполнения при создании собственного документа на основе шаблона. Структура сущностей, которыми оперирует серверная часть приложения, указана на рис 8.

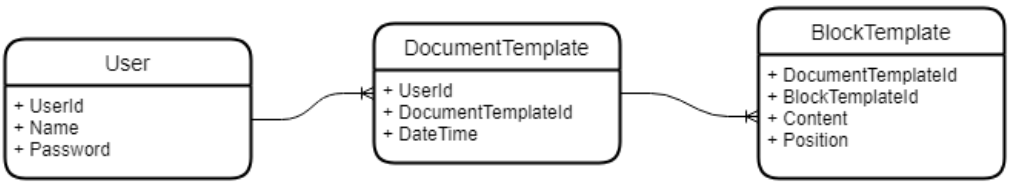

Рис. 8: Структура сущностей

При создании документа на основе шаблона создаются новый документ и

новые блоки со своими идентификаторами.

Экспорт документа

Благодаря тому, что структура документа имеет древовидный вид и хранится в формате JSON, перевод в различные форматы возможен с помощью тех или иных библиотек, конструирующих различные форматы. Так, для перевода собственной структуры документа в формат PDF используется инструмент pdfmake[18].

Экспорт искомого документа осуществляется поэлементно сверху-вниз, где каждый элемент документа транслируется в аналогичный элемент, интерпретируемый библиотекой pdfmake.

[18] http://pdfmake.org/#/

6 Тестирование

Реализованное приложение покрыто модульными тестами с использованием Mock-объектов.

Системное же тестирование было проведено с учётом тестовых сценариев от разработчиков ООО «ДоксВижн» и в целом показала соответствие требованиям реализованного приложения. Тестирование проводилось вручную.

Помимо этого, реализация прототипа была продемонстрирована на встрече со специалистами ООО «ДоксВижн». В рамках этого показа были отмечены широкие возможности по управлению атрибутами и блоками документа, а также по форматированию содержимого документа. Реализованное приложение получило положительные оценки и на данном этапе рассматриваются идеи его развития и интеграции в Docsvision ECM.

Результаты тестирования

В результате ручного тестирования был выявлен ряд дефектов совместного редактирования документов. В силу большого количества ошибок в коде порта библиотеки Yjs на язык C# было решено отказаться от этого порта и заменить подход к реализации совместного редактирования: от поддержания аналогичной клиентам бесконфликтной структуры на серверной части приложения было решено отказаться в пользу peer-to-peer соединения между клиентами. Данная реализация совместного редактирования представлена в этой работе (раздел совместное редактирование). Это позволяет не зависеть от инструмента с большим количеством дефектов и использовать основную библиотеку Yjs, основным языком реализации которой является язык JavaScript, в которой данные ошибки отсутствуют.

В таком случае, сервис совместного редактирования является своего рода арбитром, задача которого связывать клиентов с документами и блоками, с которыми они взаимодействуют и пересылать данные об изменениях бесконфликтных структур от одного клиента другому.

7 Заключение

В ходе данной работы были достигнуты следующие результаты.

- Проанализированы существующие текстовые редакторы и способы их расширения: Microsoft Office Word Online, OnlyOffice, TinyMCE, Quill, Slate.js. В качестве основы для реализации собственного веб-редактора выбран Slate.js.

- Спроектирована архитектура веб-редактора, описаны свойства и сценарии взаимодействия между пользователями приложения, документами и блоками документа, построена структура текстового редактора.

- Выполнена реализация текстового редактора: форматирование, шаблонирование, совместное редактирование, экспорт в формат PDF, объединение элементов документа в блоки (с использованием технологий: TypeScript, React.js, Slate.js, signalR, Yjs).

- Реализованы основные функции приложения: интеграция атрибутов документа с содержимым, раздельное управление блоками документа (используя ASP.NET Core, Entity Framework, TypeScript).

- Проведено тестирование приложения.

Список используемой литературы

[1] Word JavaScript API overview. — URL: https://docs.microsoft.com/office/dev/add-ins/reference/overview/word-add-ins-reference-overview (дата обращения: 25.04.2021)

[2] OnlyOffice API — URL: https://api.onlyoffice.com/ (дата обращения: 23.05.2021)

[3] TinyMCE 5 documentation. — URL: https://www.tiny.cloud/docs/ (дата обращения: 25.04.2021)

[4] Quill DevGuide. — URL https://quilljs.com/docs/quickstart/ (дата обращения: 25.04.2021)

[5] Slate documentation. — URL: https://docs.slatejs.org/ (дата обращения: 25.04.2021)

[6] ASP.NET Core documentation. — URL: https://docs.microsoft.com/aspnet/core/ (дата обращения: 25.04.2021)

[7] Dino Esposito. *Programming ASP.NET Core (1st Edition)* — Microsoft Press, 2018

[8] ASP.NET Core SignalR documentation. — URL: https://docs.microsoft.com/aspnet/core/signalr (дата обращения: 25.04.2021)

[9] Entity Framework Core documentation. — URL: https://docs.microsoft.com/ef/core (дата обращения: 25.04.2021)

[10] React documentation. — URL: https://reactjs.org/docs (дата обращения: 25.04.2021)

[11] Jon P Smith. *Entity Framework Core in Action* — Manning Publications, 2018

[12] TypeScript Documentation. — URL: https://www.typescriptlang.org/docs/ (дата обращения: 25.04.2021)

[13] Chengzheng Sun, David Sun, Agustina Ng, Weiwei Cai, and Bryden Cho. 2020. Real Differences between OT and CRDT under a General Transformation

Framework for Consistency Maintenance in Co-Editors. Proc. ACM Hum.-Comput. Interact. 4, GROUP, Article 06 (January 2020), 26 pages. DOI: https://doi.org/10.1145/3375186

[14] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. 2016. Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. In Proceedings of the 19th International Conference on Supporting Group Work (GROUP '16). Association for Computing Machinery, New York, NY, USA, 39–49. DOI: https://doi.org/10.1145/2957276.2957310

[15] Yjs documentation. — URL: https://docs.yjs.dev/ (дата обращения: 25.04.2021)