

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА
на тему
СИСТЕМА АВТОМАТИЗАЦИИ
ДОКУМЕНТООБОРОТА
КОЛЛЕГИИ АДВОКАТОВ

Студент _____ Глазунов Александр Сергеевич _____
(подпись, дата) (ФИО полностью)

Руководитель _____ Шибанов С.В. _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В. _____
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю. _____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А. _____
(подпись) (фамилия, инициалы)

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»

Заведующий кафедрой

_____ А.Ю. Козлов

«__» _____ 2023 г

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент Глазунов Александр Сергеевич гр. 19ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы «Система автоматизации документооборота коллегии адвокатов»

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Шибанов С. В.

3. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: приложение для закрытого использования внутри коллегии адвокатов
для автоматизации хранения и работы с данными, автоматизации документооборота и
упрощение работы с юридическими нюансами в предметной области.

В качестве автоматизации документооборота подразумевается создание и
управление шаблонами, возможность использования переменных в шаблонах, возможность
генерации документов на основе документов.

Средства выполнения: Операционная система MacOS (версия 13.1)

Средства разработки: VSCode, Helix Editor

Языки программирования: JavaScript, TypeScript, Go, PL/pgSQL

Технологии разработки: Svelte, Docker

4. Перечень подлежащих разработке вопросов

Анализ предметной области и постановка задачи на разработку

Разработка требований к программным средствам

Разработка программных средств

Разработка архитектуры развёртывания окружения

Контроль качества программных средств

5. Календарный график выполнения работы

№	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи на разработку	10%	12.04.2023 – 16.04.2023	
2	Разработка требований к программным средствам	20%	17.04.2023 – 26.04.2023	
3	Разработка программных средств	20%	27.04.2023 – 11.05.2023	
4	Разработка архитектуры развёртывания окружения	10%	12.05.2023 – 16.05.2023	
5	Контроль качества программных средств	20%	17.05.2023 – 21.05.2023	
6	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022 г.

Руководитель бакалаврской работы

(подпись, дата)

Задание к исполнению принял студент

(подпись, дата)

Работу к защите допустить

Декан ФВТ

(подпись, дата) Л.Р. Фионова

Реферат

Пояснительная записка содержит 111 листов, 13 таблиц, 30 рисунков, 23 использованных источника и 5 приложений.

КОЛЛЕГИЯ АДВОКАТОВ, LAWFIRM, АВТОМАТИЗАЦИЯ, ДОКУМЕНТООБОРОТ, ГЕНЕРАЦИЯ ДОКУМЕНТОВ, DOCX, MICROSOFT WORD, VSCODE, SVELTE, JAVASCRIPT, GOLANG, REDIS, POSTGRES, DOCKER, CI\CD, GITLAB.

Объект разработки: необходимо разработать системы для автоматизации документооборота внутри коллегии адвокатов. Основными критериями для системы является защищённость данных, скорость генерации документов и обработки запросов пользователей и наличие мощного инструмента управления шаблонами.

Цель работы: разработка и развёртывание системы автоматизации документооборота для внутреннего использования в коллегии адвокатов.

Результаты работы: с учётом состояние рынка по данной предметной области в российском и американском регионах была разработана система, предоставляющая возможность полностью управлять входными и выходными шаблонами, имеющая быструю скорость работы и предоставляет необходимые защитные механизмы для данных клиентов, также учитывает все достоинства и недостатки своих аналогов.

Средства разработки: VSCode, Helix Editor, Docker, Svelte, Golang, Redis, Postgres.

					ПГУ 09.03.02 – 8БР191.06ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Глазунов А. С.			Система автоматизации документооборота коллегии адвокатов	Лит.	Лист
Провер.		Шибанов С. В.					
						4	111
Н. Контр.		Такташкин Д. В.				группа 19ВИ1	
Утв.		Козлов А. Ю.					

СОДЕРЖАНИЕ

Введение	7
1 Анализ предметной области и постановка задачи на разработку.....	9
1.1 Анализ предметной области адвокатской конторы	9
1.1.1 Анализ структуры предметной области	9
1.1.2 Анализ документооборота и требований заказчика.....	13
1.2 Сравнительный анализ аналогичных разработок.....	14
1.2.1 Обоснование выбора систем-аналогов	14
1.2.2 Обзор основных возможностей систем-аналогов	16
1.2.3 Результаты сравнительного анализа систем-аналогов.....	17
1.3 Постановка задачи на разработку программных средств.....	19
1.3.1 Назначение программных средств и классы пользователей	19
1.3.2 Перечень автоматизируемых функций.....	20
1.4 Функционально-стоимостной анализ разработки	21
1.4.1 Планирование процесса разработки	21
1.4.2 Стоимостная оценка проекта.....	24
1.4.3 Анализ временных и финансовых затрат	25
2 Разработка требований к программным средствам	27
2.1 Функциональные требования	27
2.2 Требования к концептуальной модели.....	30
2.3 Требования к обобщённой архитектуре приложения	31
2.4 Обоснование выбора технологий и средств разработки.....	33
2.5 Требования к аппаратному и техническому обеспечению.....	35
3 Разработка программных средств доступа к данным	36
3.1 Разработка принципа доступа к данным	36
3.2 Разработка физической модели базы данных	38
3.3 Согласование протоколов взаимодействия.....	41
3.4 Разработка программных средств серверной части	43
3.5 Разработка программных средств пользовательской части	44

3.6	Разработка программных средств генерации документов	46
3.6.1	Синтаксис внутренних переменных документов	46
3.6.2	Процесс генерации документов	49
4	Развёртывание окружения	51
4.1	Разработка архитектуры.....	51
4.2	Развёртывание локального окружения для разработки	52
4.3	Развёртывание реального окружения	54
5	Контроль качества программных средств.....	57
5.1	Планирование процесса контроля качества программных средств	57
5.1.1	Мероприятия по контролю качества программных средств	57
5.1.2	Обоснование выбора модели тестирования	58
5.2	Контроль качества программного кода	62
5.2.1	Использования метода покрытие условий и условий	62
5.2.2	Юнит-тестирование программного кода.....	64
5.2.3	Оценка качества кода с использованием CI\CD технологий	66
	Заключение	70
	Список использованных источников	72
	Приложение А. Глоссарий	75
	Приложение Б. Скрипт создания базы данных.....	79
	Приложение В. Исходный код генерации документов	90
	Приложение Г. Исходный код скачивания документов	99
	Приложение Д. Рекомендательное письмо, оригинал	106
	Приложение Е. Рекомендательное письмо, перевод.....	108

Введение

Адвокат – это человек, имеющий статус адвоката, которое даёт ему право участвовать в судебном процессе в качестве защитника либо советника. Он считается независимым экспертом, представляющим интересы одной из сторон.

У адвоката может иметься множество дел и судебных процессов, часто несколько из которых идут параллельно. Для управления таким объёмом информации приходится прибегать к использованию сторонних сервисов. Каждый адвокат может делать это по-разному, кто-то хранить файлы на компьютере в различных папках, кто-то может использовать сторонние сервисы общего назначения, такие как Google Docs или Google Sheets. Также существуют отдельные специализированные системы, большинство из которых работает по модели подписки.

Основные минусы подобного хранения – это невозможность повлиять на данные, если со сторонним сервисом что-нибудь случилось, будь то окончание подписки или технический сбой. Конфиденциальность информации тоже является не маловажным фактором, поэтому её тоже можно рассматривать как один из недостатков.

Основной задачей данного проекта является полноценная миграция хранения файлов из Google Sheets, минусы которого описаны выше, в самостоятельную, локально-защищённую систему, которая изготавливается «под ключ». Данная система должна полностью автоматизировать ту часть процесса, для которой наличие человека не является обязательным.

Система должна удовлетворять следующим критериям:

- быть защищённой и изолированной от посторонних пользователей;
- информация должна передаваться по безопасным каналам связи;
- иметь возможность генерации и кастомизации шаблонов документов;
- быть отказоустойчивой;
- быть доступной с мобильных устройств хотя бы для просмотра информации.

Также существует ряд необязательных критериев, которые улучшают опыт пользователей по работе с системой:

- интеграция с Google Calendar;
- интеграция с Synology Drive;
- возможность импорта и экспорта информации, желательно .csv формат;
- полноценная пользовательская система;
- иметь возможность сильно модифицировать систему для потенциальных покупателей (адвокатов);
- внедрение других типов API (кроме REST) для интеграции с другими приложениями.

В рамках дипломного проекта представляется функционал системы, реализованный на момент «апрель 2023 г.». Заказчиком данного проекта является компания «Patricoski Law Offices» [1]. Система продолжает разрабатываться и совершенствоваться, а часть вышеперечисленных необязательных критериев уже была реализована. На текущем этапе разработки производится внедрение альфа-версии системы для полноценного тестирования и сборы отзывов от конечного пользователя.

В качестве отзыва о проделанной работе и в знак благодарности, в приложениях Д и Ж представлено рекомендательное письмо заказчика.

1 Анализ предметной области и постановка задачи на разработку

1.1 Анализ предметной области адвокатской конторы

1.1.1 Анализ структуры предметной области

Данный дипломный проект является реальной коммерческой задачей и выполняется по техническому заданию заказчика на предметной области адвокатской конторы в США. Соответственно, все далее упомянутые законы (при надобности) и общая терминология могут не совпадать с русскими вариантами, даже если они имеют одинаковый перевод. Основная задача – создать приложение, позволяющее автоматизировать и систематизировать часть бизнес-процессов и реализовать полноценное и защищенное хранилище данных. Ниже приведён список терминологии и приблизительный перевод, необходимый для понимания предметной области. Данный список можно так же присутствует в глоссарии, приложении А.

Терминология:

- Adjuster – (юридический) консультант (далее – консультант);
- Attorney – прокурор;
- Carrier – (правовой) носитель (далее – носитель);
- Case – (судебное) дело;
- Client – клиент;
- County – округ;
- Defendant – ответчик, противоположная сторона (далее – ответчик).

Конкретные обязанности каждой сущности со стороны законодательства нам, как разработчикам, не важны, но важны их зависимости.

Дело – это основная сущность, имеющая ряд информации, которая относится непосредственно к делу (например – дата происшествия) и имеет как минимум одного клиента. К делу так же может относиться несколько клиентов и ответчиков. При наличии нескольких клиентов – один из клиентов имеет пометку «Основной». Так же, дело содержит информацию о округе, так как часть

данных и необходимый набор документов может варьироваться в зависимости от округа.

Клиент это одна из основных сущностей, которая содержит необходимую информацию об одном клиенте, такую как: контакты, личные данные, данные, относящиеся к делу. В большинстве случаев основной клиент будет являться истцом в судебном деле.

Ответчик это одна из основных сущностей. Как и клиент – содержит информацию об ответчике, а также данные о юридических консультантов и прокуроров, связанных с ним. Связанных консультантов и прокуроров может не быть совсем, а может быть несколько. В случае, когда их больше, чем 1 – используется тот, что был связан самым последним. У каждого ответчика есть правовой носитель, он может быть как одинаковым, так и разным для ответчиков в разрезе одного дела [2].

Правовой носитель является страховой компанией ответчика, которая предоставляет ему услуги консультанта и прокурора [3].

Со слов заказчика – менять консультантов и прокуроров – вполне обычная практика, но иногда она может использоваться чтобы замедлить общий ход дела, т. к. им необходимо время на изучение дела и это позволяет отсрочить судебный процесс. Для таких случаев необходимо сохранять время назначения каждого из них, чтобы увидеть и зафиксировать подобные операции.

Консультанты и Прокуроры – очень похожие сущности в плане информации, т. к. за исключением своего имени – они имеют одинаковую структуру данных, необходимых для дела. Каждый из них имеет носителя, но в отдельную связь мы это не выделяем, потому что их носитель всегда одинаков с носителем ответчика, с которым они связаны [4].

Подводя итог всех сущностей, описанных выше, можно их проиллюстрировать для более наглядного представления зависимостей (рисунок 1.1).

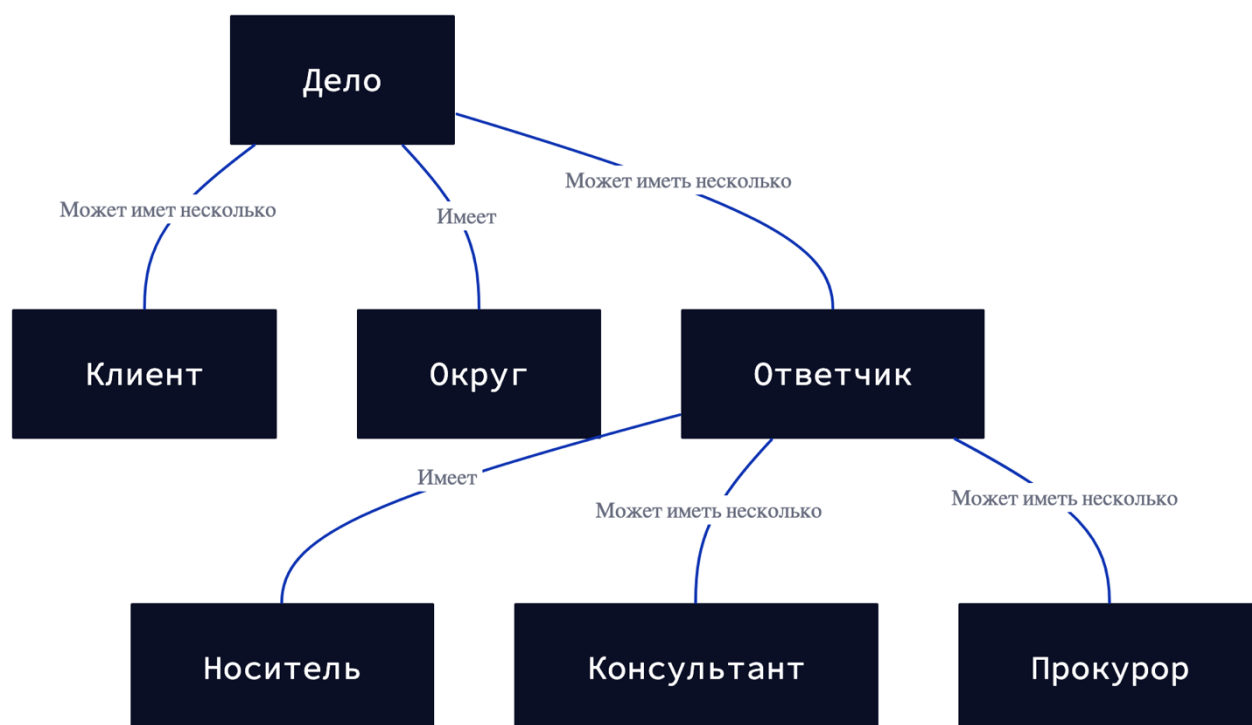


Рисунок 1.1 – Зависимости сущностей

Каждый из пунктов на рисунке будет являться своей программной сущностью, кроме времени присоединения, которое должно храниться отдельно, и не быть частью какой-либо другой сущности.

У дела присутствует особая связь с клиентом, которая называется «Основной клиент».

«Клиент» и «Основной клиент» являются одной и той же сущностью, однако имеют разное назначение в рамках одного дела. Само по себе дело не имеет названия или имени и обычно используется фамилия и имя основного клиента.

Основной бизнес-процесс представлен на рисунке 1.2 и показывает общий перечень действий, которые выполняет пользователь при работе с системой.

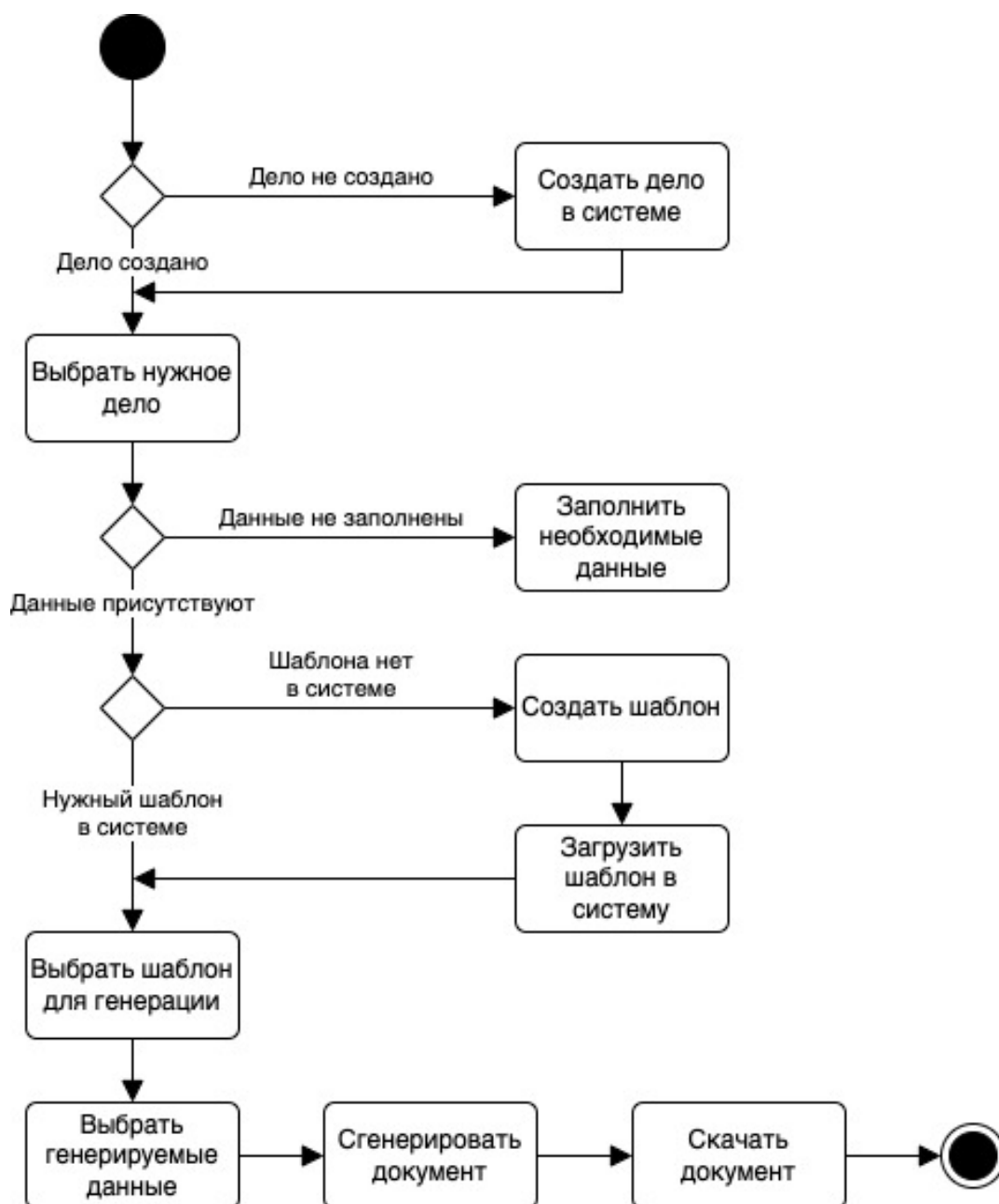


Рисунок 1.2 – Порядок действий при работе с системой

Данная диаграмма показывает набор действий при создании дела и шаблона с самого начала, однако все из этих действий могут быть выполнены отдельно и не имеют прямого влияния друг на друга.

Предполагается, что концом работы с системой является генерация документа, т. к. это и есть основное назначение данной системы.

Работа с шаблонизатором опущена на данной диаграмме, т. к. не является основным бизнес-процессом.

1.1.2 Анализ документооборота и требований заказчика

На текущий момент старая версия данного проекта насчитывает порядка 130 уникальных документов, начиная от простых уведомительных писем и заканчивая полноценных (но однотипных) отчётов по делам, в которые присутствует изменяющаяся информация о деле, клиентах и других необходимых нюансов.

Так как объём документов довольно большой и может увеличиваться в дальнейшем, какого-то жёсткого разделения на категории не будет, так одно из требований к системе – это полная автономность без участия разработчика.

Решение данной проблемы – это автоматизация обоих процессов: создания шаблонов и непосредственно самой генерации документов. У пользователя системы должна быть возможность загружать свои шаблоны в систему и использовать их с любыми делами. Таким способом будет убрана лишняя необходимость обслуживания системы третьими лицами и предоставим пользователям гибкость в создании документов любого рода и назначения.

После обсуждения изначальных требований с заказчиком был сформирован следующий список необходимых критериев, которым должна удовлетворять система:

- автономность;
- доступность;
- лёгкость в использовании;
- поддержка тёмной темы;
- защита данных;
- резервные копии.

Автономность – после завершения работы над системой она должна функционировать автономно без вмешательства разработчиков. Исключением является исправление багов, если таковые были найдены уже после введения в эксплуатацию.

Доступность – система должна быть доступна для просмотра информации в любое время и с любого места, где есть интернет.

Лёгкость в использовании – в будущем данной системой будет пользоваться не только заказчик, но и другие сотрудники фирмы, поэтому интерфейс пользователя необходимо спроектировать, используя лучшие практики расположения элементов.

Поддержка тёмной темы – система должна иметь возможность переключения между светлой и тёмной темами для работы в любое время суток и меньшей нагрузки на глаза.

Защита данных – система должна хранить и обрабатывать данные, максимально защищая их, насколько это возможно. Так как работа ведётся с личными данными реальных людей, утечка которых может представлять реальную угрозу для их жизни – информация должна быть максимально защищена.

Резервные копии – в дополнении к предыдущему пункту – через некоторый промежуток времени на постоянной основе должна производиться резервная копия всей системы – данных по каждой из сущностей, их связи, файлы макеты шаблонов и так далее.

Основным и главным требованием из этого прежде всего будет являться безопасность хранимой информации, поэтому при дальнейшем проектировании системы это стоит учитывать в первую очередь.

1.2 Сравнительный анализ аналогичных разработок

1.2.1 Обоснование выбора систем-аналогов

Так как данный проект прежде всего ориентирован на американский рынок, то поиск основных конкурентов будет проводиться именно там. Некоторые из этих аналогов уже были опробованы заказчиком, другие – были найдены во время проведения анализа. Общий список представлен ниже:

- Google Sheets;

- Clio (Clio Manage);
- Smokeball;
- 1С: Предприятие.

Данные о системах представлены в таблице 1.1.

Таблица 1.1 – Системы аналоги

№	Система	Производитель	Стоимость USD	Стоимость RUB
1	2	3	4	5
1	Google Sheets	Google LLC	Бесплатно	Бесплатно
2	Clio Manage	Clio	\$129	~10,500
3	Smokeball	Themis Solutions Inc.	\$149	~12,700
4	1С: Предприятие	1С	~\$86	7,000

В столбцах Стоимость (4 и 5) цены указаны на одного сотрудника. Префикс «~» означает примерную стоимость. Конвертация валюты произведена на момент апреля 2023 г.

Google Sheets [5] – это онлайн-система управления таблицами, которая позволяет создавать и форматировать информацию в виде Excel-таблиц. Изначально эта система откуда осуществлялась миграция на текущий проект, так как управлять целой фирмой посредством множества табличных документов очень сложно, трудоёмко и попросту неудобно. Автоматизация данного процесса так же очень сложная, потому что все действия ограничены API от Google, и во многих случаях приходится искать обходные пути для решения, казалось бы, тривиальных проблем.

Clio [6] – компания, которая разрабатывает Clio Manage – ПО для управления адвокатской конторой, организации дел и работать с клиентами. Всё это в одном ПО. Clio Manage позволяет управлять всем из облака, которое они предоставляют.

Smokeball [7] – это стороннее ПО, которое предоставляет свои услуги в сфере управления адвокатскими конторами в виде SaaS (англ. Software as a Service – Приложение как сервис) либо как облачные решения. Данное решение помогает фирмам вести учёт по делам, имеет автоматизацию при работе с

документами, имеет возможности работы и отправки с Email адресами и календарными событиями.

На Российском рынке тоже есть системы, которые подходят по техническим требованиям, однако не удовлетворяют другим факторам. Из основных русских аналогов можно выделить только одного – 1С: Предприятие.

1С: Предприятие – инструмент для комплексной автоматизации бухгалтерского и налогового учёта. Однако у 1С есть минус, который полностью перекрывает всего его плюсы – это Российское ПО, и по законам страны заказчика, официально он не имеет права хранить там данные.

1.2.2 Обзор основных возможностей систем-аналогов

Google Sheets – прежде всего система управления табличными документами. Этот инструмент предназначен совершенно для других целей, поэтому рассматривать его подробно не имеет смысла, так как данная система была использована лишь из-за ограничений других и отсутствия аналогов.

Clio Manage – система которая позволяет автоматизировать задачи, минимизировать объём бумажной работы. Одна из отличительных особенностей – возможность работы с электронными подписями. Так же данная система предлагает свою, внутреннюю, систему оплаты для компаний, что, возможно, поможет в ведении учёта и статистики доходов и расходов.

Smokeball предоставляет автоматический контроль и учёт времени в разрезе адвокатов и дел, автоматизацию генерации документов, отдельный клиентский портал для взаимодействия, электронные подписи, мобильное приложение, создание динамических «workflow» (англ. потоки задач), работу с календарём, возможность автоматического создания и управления банковскими счетами.

Последний рассматриваемый аналог – русская система «1С: Предприятие». Если затрагивать только техническую сторону вопроса – то даже при условии, что система должна быть выполнена «под ключ», 1С сможет покрыть практически все технические требования заказчика. Так как это

инструмент предоставляющий контроль над тем, как данные будут храниться, то воссоздать требуемую структуру данных не составит никаких проблем. Основной объём работы будет связан с просмотром и изменением данных, с чем 1С тоже справится. Единственное техническое требование, которому 1С не удовлетворяет – это автономность работы системы. При появлении новых шаблонов (т. е. новых форм документов) клиент не сможет сам добавить их при необходимости, и ему придётся прибегать к помощи специалистов либо самому разбираться в конфигураторе.

1.2.3 Результаты сравнительного анализа систем-аналогов

Общий сравнительный анализ представлен в таблице 1.2.

Для простоты восприятия и ввиду большого их количества – заголовки в таблице представлены в виде цифр, обозначение каждого из столбцов приведено после таблицы.

Иконка «✓» внутри таблицы означает что система удовлетворяет необходимым критериям заказчика, а «×» – наоборот, что система не удовлетворяет требованиям. Часть требований уточнить не удалось ввиду закрытости данных систем и невозможности самим полностью их протестировать. В таких случаях в ячейке будет находиться «знак вопроса».

Таблица 1.2 – Сравнительный анализ

Системы \ Критерии	Генерация документов	Автономность	Доступность	Лёгкость в использовании	Тёмная тема	Контроль над данными	Резервные копии
1	2	3	4	5	6	7	8
Google Sheets	✓*	×	✓	✓	×	×	✓**
Clio Manage	✓	?	×	?	?	×	?
Smokeball	✓	?	✓	?	?	×	?
1С: Предприятие	✓	×	×	×	✓***	×	?

* – С использованием Google Sheet API и Google Docs API, со встроенными ограничениями от Google.

** – Только при ручном создании копий.

*** – 1С предоставляет возможность настраивать темы, однако в изначальной комплектации тёмная тема не предусмотрена.

Выбирая сторонние сервисы для предметных областей, где речь идёт о личных данных пользователей – одним из основных критериев должен являться критерий сохранности и безопасности данных. «Clio Manage» и «Smokeball» позиционируют себя как SaaS (т. е. предоставляют сервисы в виде подписочных моделей) либо как Cloud-сервисы. Вполне возможно, что при окончании подписки данные могут быть уничтожены без возможности восстановления. «1С: Предприятие» может предоставляться как локальное решение, однако оно позиционирует себя как готовый продукт и все данные скрыты во внутренней системе. Подводя итог – не в одной из трёх систем нет прямого доступа к данным, а, соответственно, разработчик не в состоянии их полностью контролировать.

Каждая из перечисленных систем (за исключением Google Sheets) имеет модель подписки либо покупки. «1С: Предприятие» распространяется по модели единоразовой покупки на количество рабочих мест, а Smokeball и Clio Manage имеют модель оплаты за 1 пользователя в месяц.

Так как два последних продукта нацелены на получение прибыли от подписок – то большинство услуг, описанных ранее, такие как генерация документов, скрыты за более дорогими «тирами» (англ. tier – уровень) подписок. В следствии этого сервис теряет свою необходимость, если выбирать самый дешёвый вариант.

В обоих случаях на самых низких тирах система позволяет лишь работать с делами в общем виде и отслеживать незначительные вещи, не предоставляя доступ к работе с календарём, генерацией документов, работе с Email адресами и другим сервисам, обозначенными как «одни из основных особенностей» каждого из сервисов.

1.3 Постановка задачи на разработку программных средств

1.3.1 Назначение программных средств и классы пользователей

Программные средства – комплекс средств, созданных для их эксплуатации в компьютерных системах. Целью разработки и назначением программных средств данного дипломного проекта является разработка автоматизированной системы для адвокатской конторы, соответственно она должна обладать следующими возможностями:

- возможность размещать и управлять информацией в одном, централизованном месте;
- возможность управлять документами и их шаблонами, особенно как обычный пользователь, а не разработчик системы;
- возможность ведения промежуточной документации и отчётности (экспорт в .csv для дополнительного анализа);
- должна быть удобной в использовании и лёгкой для изменений при необходимости добавления нового функционала.

Можно перечислить ещё много необходимых особенностей для удобного пользования системой, однако самым главным всё равно остаётся автоматизация хранения и экспорта информации, в любых видах.

Так как данный проект создаётся как частный заказ под небольшую компанию, то у него будет только один основной класс пользователей – юристы и (или) адвокаты. Предполагается, что данный класс может не владеть ПК в совершенстве, поэтому нужно учитывать это при проектировании функций и пользовательского интерфейса.

Несмотря на то, что количество пользователей может со временем увеличиваться, с заказчиком было согласовано, что только он может давать доступ к системе. При этом права любого из пользователей будут одинаковы, т. е. возможность создавать, редактировать и читать объекты, генерировать и изменять шаблоны сможет любой пользователь.

1.3.2 Перечень автоматизируемых функций

В системе планируется следующий перечень функций для автоматизации:

- быстрое создание и управление сущностями, такими как: дело, клиент, ответчик и т. п.;
- полное управление шаблонами со стороны клиента;
- генерация документов на основе заранее определённых шаблонов;
- возможность создания собственных документов.

В быстром создании и управлении предполагается, что иногда необходимо быстро вносить информацию во время телефонного звонка, поэтому нужно учитывать, что у пользователей системы не всегда есть время внести полную либо корректную информацию. В следствии этого поля должны быть доступны для ввода любого значения, которое может быть преобразовано в значение нужного формата, например поле для ввода телефона.

Под полным управлением шаблонами подразумевается возможность создавать новые категории шаблонов, сами шаблоны (не документы), перемещать их между категориями при необходимости.

Генерация документов – это основная функция системы, которая будет доступна всем пользователям. Генератор берёт готовые документы, заменяет в них необходимые значения (например – имя клиента, текущую дату и др.) и предоставляет ссылку для скачивания пользователю для дальнейшей работы с документом.

Для создания шаблонов для документов была создана отдельная программа внутри приложения – «Placeholder Wizard» (от англ. – помощник по работе с переменными шаблонов). С помощью неё пользователи могут выбирать необходимые им переменные шаблонов из уже готового списка, что существенно облегчает порог входа для потенциально-новых работников компании и позволяет компании работать автономно, не завися от разработчиков.

Так же планируются некоторые возможности, приоритет выполнения которых на данный момент очень низкий:

- трекинг действий – кто и что сделал, в какое время было произведено изменение;
- полноценная пользовательская система;
- интеграция с Google Calendar, Google Sheets, Google Docs.

Для некоторых действий необходимо знать кто совершил некоторое действие и когда это действие было совершено. С связи с этим необходимо вести учёт некоторых действий. Самый простой пример такого – когда и кем было отправлено Email-письмо с некой важной информацией.

Пользовательская система нужна в основном для учёта действий. Политика авторизации как таковая не планируется ввиду совершенно другой системы аутентификации, поэтому необходимость в пользователях как инструмента входа в систему при помощи пароля – отсутствует на данном этапе разработки, однако возможно будет добавлена в будущих обновлениях.

Интеграция с сервисами Google планируется ввиду их распространённости и удобства и на данный момент имеет низший приоритет, ввиду её минимальной необходимости, за исключением Google Calendar. Внутри компании он используются всеми сотрудниками, поэтому приоритет конкретно этого сервиса – немного выше, чем у остальных.

1.4 Функционально-стоимостной анализ разработки

1.4.1 Планирование процесса разработки

Планирование процесса разработки начинается сразу после полной постановки задачи, согласования технических требований и урегулировании сроков выполнения проекта. В процессе планирования назначается порядок действий, необходимый для достижения поставленных целей проекта.

Диаграмма Ганта – это специальная диаграмма, которая используется для планирования и визуализации плана задач по какому-либо проекту [8].

Данная диаграмма состоит из полос, где самая полоса обозначает какую-либо одну задачу, в то время как длина этой полосы – время, необходимое для выполнения данной задачи. Задачи можно объединять в совокупные задачи. Данная диаграмма была выбрана, потому что:

- имеется наглядное отображение временных рамок проекта;
- понятное отображение зависимостей;
- простое отслеживание прогресса;
- возможность планирования будущих задач.

В данном дипломном проекте будет использоваться диаграмма Ганта для наглядности планирования процесса выполнения задач и анализа временных затрат.

В качестве программы для создания данной диаграммы будет использоваться независимый проект Mermaid JS [9] для создания диаграмм через написание кода, который единственный из всех аналогов поддерживает создание диаграмм Ганта. Это являлось основной причиной, однако у Mermaid JS есть и другие плюсы:

- описание диаграмм в текстовом формате;
- интеграция с «Markdown» разметкой;
- поддержка экспорта в разные форматы, такие как PNG, SVG и PDF;
- простота и удобство в использовании;
- возможность использования других видов диаграмм, что может пригодиться в будущем;
- наличие онлайн-редактора, что позволяет создавать, редактировать и просматривать диаграммы на любом устройстве с доступом в интернет.

Дополнительно, присутствует встроенный функционал для подсчёта времени для задач и возможности задания цепочек задач для диаграмм Ганта, что позволит автоматически изменять будущие задачи при изменении предыдущих.

Далее, на рисунке 1.3, представлена диаграмма Ганта.

Диаграмма ганта разработки проекта для коллегии адвокатов

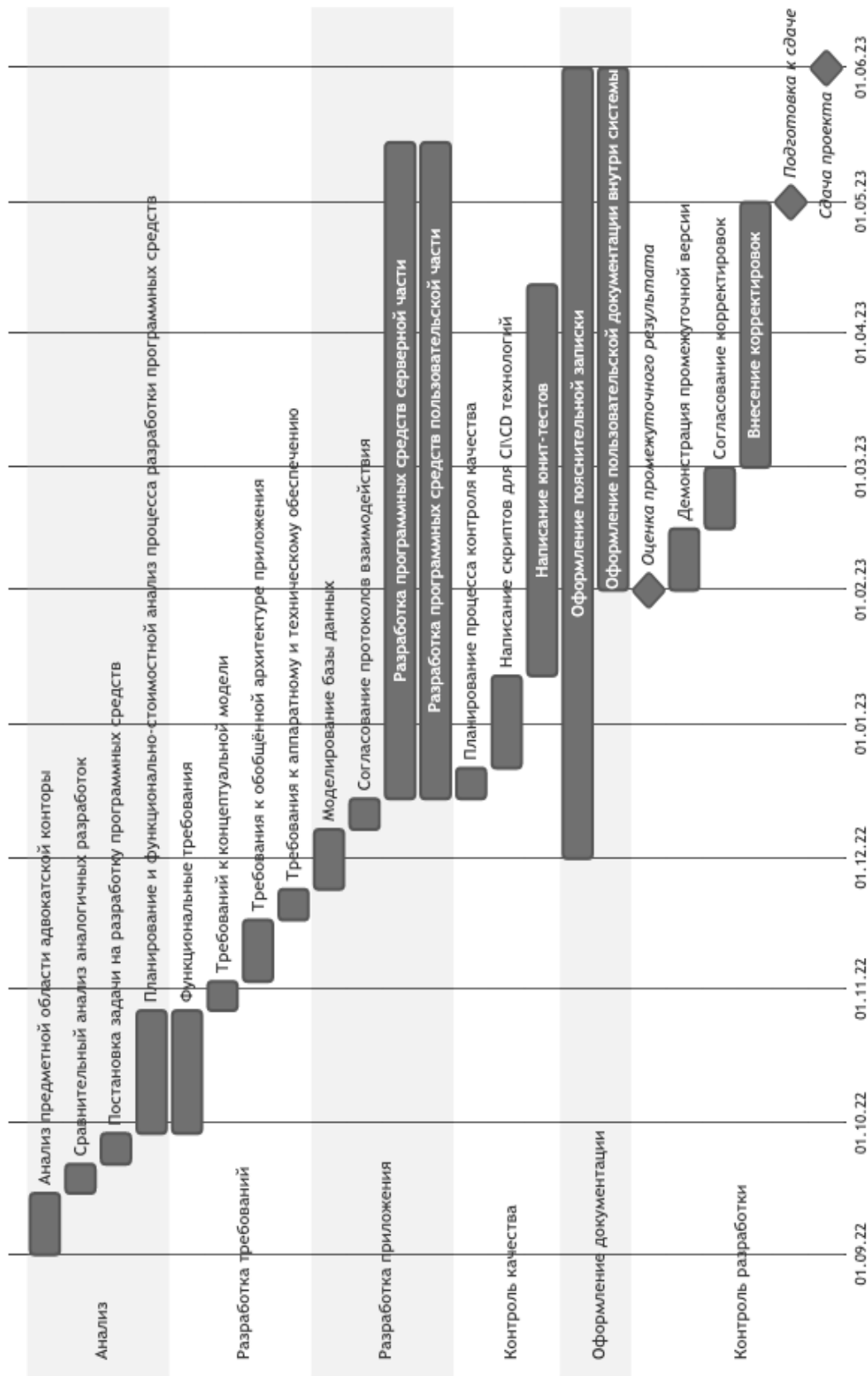


Рисунок 1.3 – Планирование процесса разработки (диаграмма Ганта)

На диаграмме присутствуют следующие секции:

- анализ;
- разработка требований;
- разработка приложения;
- контроль качества;
- оформление документации;
- контроль разработки.

Каждая из секций отвечает за определённый набор связанных действий, а в конце временного промежутка последней секции считается, что она была успешно выполнена.

Начальной точкой отчёта является начало учебного года, т. е. 1 сентября 2022 г., основной продукт разрабатывался до этого, однако это не является частью данного дипломного проекта.

Некоторые задачи и секции пересекаются по времени, что означает что данные задачи выполняются параллельно. Вполне возможно, что одна задача блокирует другую, из-за чего вторая может занимать больше времени, чем планировалось. Примером такой блокировки служит задача «Написание юнит-тестов» в секции «Контроль качества». Эту задачу нельзя выполнить после написания кода, т. к. в процессе разработки часть функционала может ломаться, и автоматические позволяют сразу находить подобные проблемы, а завершиться данная задача не может т. к. разработка продукта ещё не была закончена.

Финальной точкой является 1 июня 2023 г. В качестве дипломного проекта будет представлена часть функционала для демонстрации работоспособности, однако работа над основным проектом всё ещё продолжается и в него добавляется новый функционал.

1.4.2 Стоимостная оценка проекта

Одна из основных задач при планировании и разработки систем – стоимостная оценка процесса разработки. Очень часто такие оценки меняются в

процессе разработки ввиду таких изменений как: дополнительные требования или правки заказчика, неверное изначальное планирование задач, в следствии чего разработка занимает больше времени, чем планировалось.

Далее представлен список сотрудников, необходимых для реализации проекта. Данные ставки были взяты как среднестатистические за 2022 г. (таблица 1.3).

Таблица 1.3 – Сотрудники

Сотрудник	Ставка в день
1	2
Разработчик	2500 Р
Тестировщик	1000 Р

Стоит учитывать, что нанимать сотрудника на весь период разработки не нужно. Например, после проведения анализа задач для Аналитика больше не останется. Это позволяет прибегать к услугам сторонних компаний, если необходимо.

Далее представлен список необходимых ресурсов для реализации проекта. Цены взяты от 01.04.2023 г. по среднему значению (таблица 1.4).

Таблица 1.4 – Необходимые ресурсы

Ресурс	Количество	Цена	Стоимость
1	2	3	4
Интернет	2	800	1600
Компьютер	2	15000	30000
Удалённый сервер	1	10000	10000

Ресурсы – это траты, которые необходимо выполнить единожды, поэтому каких-либо сложных расчётов для них проводить не нужно [8].

Таким образом, имея данные о затратах, можно провести анализ финансовых затрат.

1.4.3 Анализ временных и финансовых затрат

Далее для каждой из секций на диаграмме будет приведён расчёт временных и финансовых затрат. Сам анализ представлен на таблице 1.5.

Таблица 1.5 – Анализ затрат

Секция	Затрачено дней	Цена	Стоимость
1	2	3	4
Анализ	$14+7+7+28 = 56$	2,500	140,000 Р
Разработка требований	$28+7+14+7 = 56$	2,500	140,000 Р
Разработка приложения	$14+7+150 = 171$	2,500	427,500 Р
Контроль качества	$7+21+90 = 118$	1,000	118,000 Р
Оформление документации	30	1,000 + 2,500	105,000 Р
Всего			930,000 Р

Под временными затратами подразумевается количество рабочих дней, потраченных на определённую задачу. Выходные и праздничные дни включены в общее количество, однако никаких действий в эти дни не производится. Под финансовыми затратами подразумевается зарплата сотрудникам и покупка необходимых ресурсов. Если сотрудник выполнил свои обязанности в рамках проекта – то зарплата ему больше не выплачивается.

Так как трата на ресурсы единоразовая, и будет равняться $1,600 + 30,000 + 10,000 = 41,600$ Р, то можно посчитать, что общая сумма затрат на проект равняется $41,600 + 930,000 = 1,166,500$ Р. Суммарное количество дней, которые потребовались для разработки, включая выходные и праздничные дни – 273 дня, т. к. разработка началась 01.09.2022 г., а закончилась – 01.06.2023 г. Таким образом можно узнать стоимость одного рабочего дня – $930,500 / 273 = 3,172$ Р.

Расчёты были произведены с полным учётом покупки рабочего оборудования и необходимого количества персонала.

2 Разработка требований к программным средствам

2.1 Функциональные требования

Исходя из требований заказчика были составлены следующие функциональные требования:

- система должна предоставлять возможность проводить операции создания, редактирования, чтения и удаления с основными сущностями;
- для части сущностей должна присутствовать возможность связи их с другими сущностями;
- у пользователей системы должна быть возможность создавать и редактировать свои шаблоны в формате .docx;
- генерация шаблонов не должна влиять на их конечный внешний вид;
- должен присутствовать механизм резервной копии данных, не обязательно как внутренняя часть системы.

В конечном итоге система должна быть установлена и запущена на частных серверах компании. Желательным, но не обязательным, критерием так же является автономная работа без сети «Интернет», в случае чего уровень безопасности данные возрастёт в разы.

На одном из более поздних этапов разработки, было обговорено что в систему будут иметь доступ не более 5 пользователей, но, не смотря на данное условие, полноценная система создания и редактирования пользователей должна присутствовать.

Так как данная система будет совершенствоваться и развиваться в будущем, при добавлении нового функционала необходимо учитывать возможность потенциального расширения системы и простоту поддержки кода. При написании функционала, который сложно поддерживать, технический долг проекта будет роста в геометрической прогрессии и поддерживать систему станет невозможно.

На основе данных требований была сформирована диаграмма вариантов использования, которая представлена на рисунке 2.1.

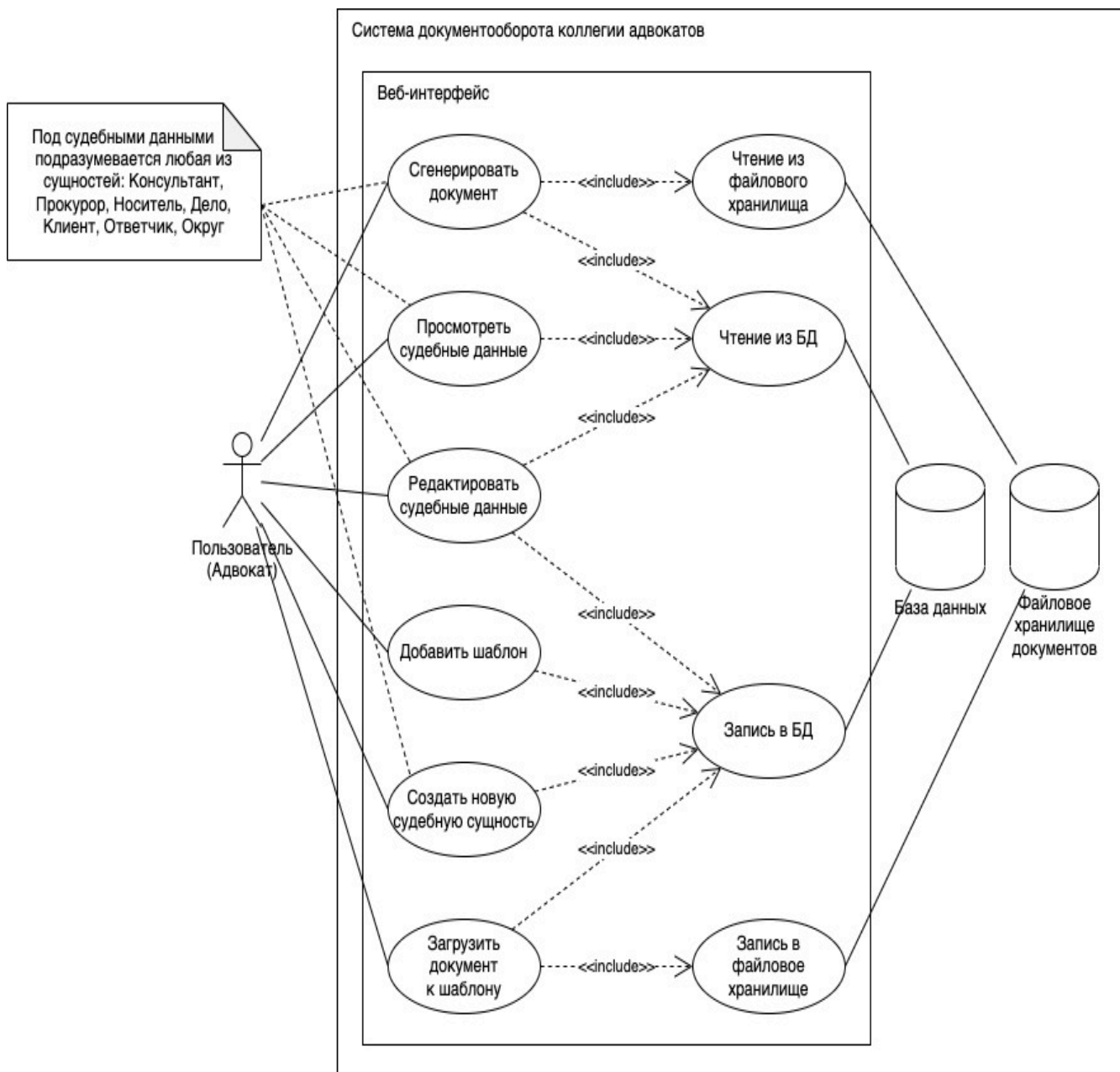


Рисунок 2.1 – Варианты использования Web-Приложения

В системе присутствует один актёр – Пользователь, он же – Адвокат и 2 основных хранилища: хранилище основной информации (База данных) и файловое хранилище, для хранения документов шаблонов. Так или иначе, каждое действие пользователя будет сопровождаться работе с этими хранилищами.

Общие операции над судебными сущностями представлены в виде одного варианта использования, так как кроме разных полей данных бизнес-логика приложения не меняется.

Далее, в таблице 2.1, приведены типовые сценарии работы пользователя.

Таблица 2.1 – Сценарии работы системы

№	Описание	Набор действий	Результат
1	2	3	4
1	Просмотр данных	1 Зайти в список желаемой сущности 2 Выбрать необходимую сущность 3 Нажать «Просмотреть»	Данные о выбранной сущности
2	Создание нового дела	1 Зайти в меню создания определённой сущности 2 Ввести изначальную информацию 3 Нажать «Создать»	Созданная сущность или ошибка с пояснением
3	Редактирование данных	1 Зайти на страницу сущности, которую необходимо отредактировать 2 Отредактировать необходимую информацию 3 Нажать «Сохранить»	Сущность изменилась
4	Загрузка шаблона для документа	1 Зайти на страницу списка шаблонов 2 Выбрать шаблон, нажать «Загрузить .docx» 3 Дождаться окончания загрузки шаблона	Шаблон загружен на сервер или сообщение об ошибке
5	Генерация документа	1 Зайти на страницу дела, для которого необходимо сгенерировать документ 2 Выбрать необходимые сущности (их может быть несколько) 3 Нажать «Сгенерировать»	Скаченный файл с изменёнными переменными под текущее дело

Самыми частыми сценариями будут: генерация документов и просмотр данных.

Загрузка шаблонов осуществляется один раз, и как только все необходимые шаблоны будут загружены, данная функция будет использоваться очень редко.

Создание и редактирование данных, теоретически, могут применяться часто, однако ввиду особенности предметной области, (судебные) дела

решаются не очень быстро и большинство реальной работы состоит из поиска уже имеющихся данных.

2.2 Требования к концептуальной модели

На основе требований заказчика была разработана концептуальная модель, которая представлена на рисунке 2.2.

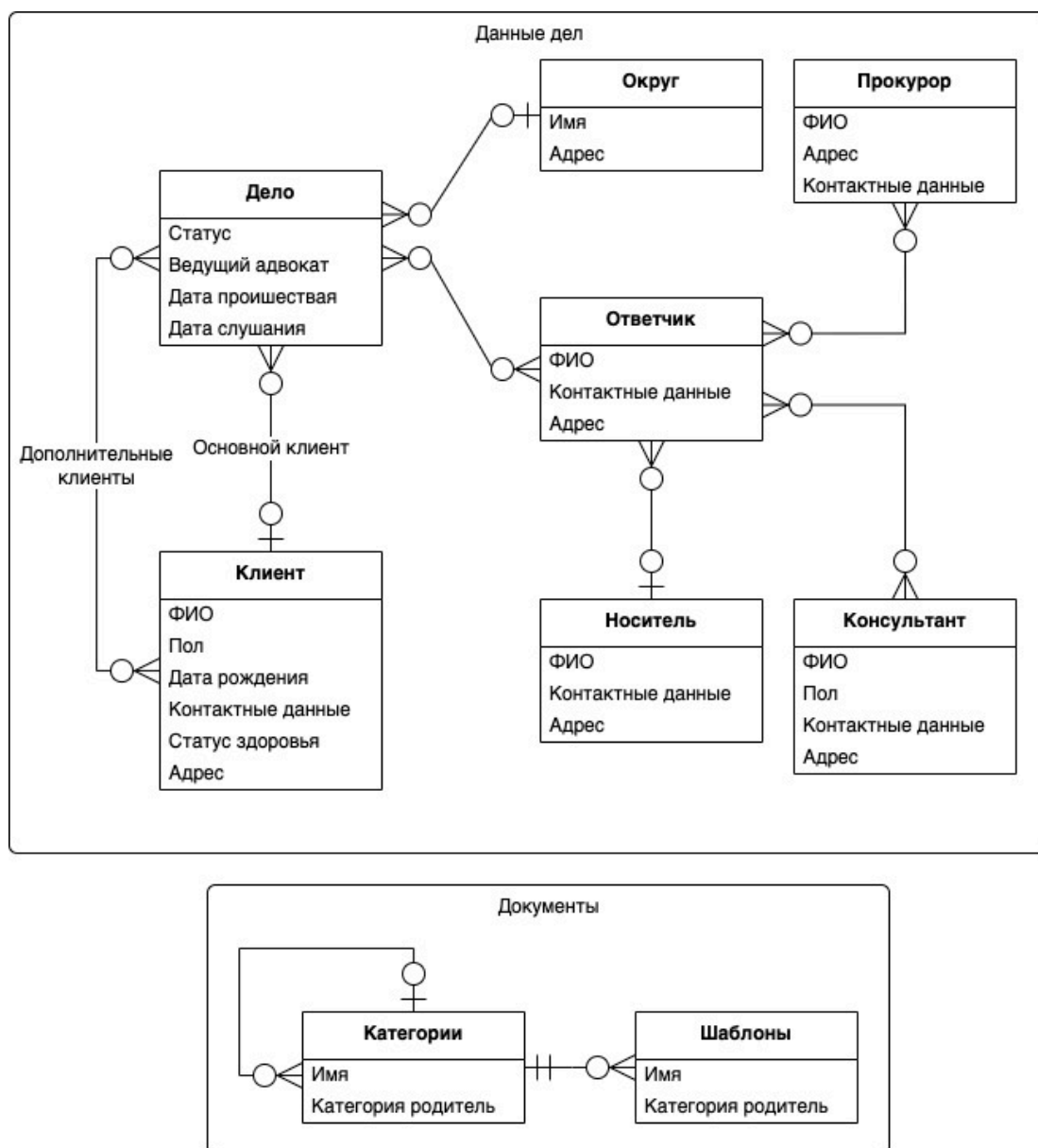


Рисунок 2.2 – Концептуальная модель предметной области

Ввиду большого количества полей (до 40) у некоторых сущностей – они были скрыты, и отображены лишь некоторые. Основной задачей являлась правильно распределить предметную область на сущности и сопоставить связи

между ними. На данной диаграмме используется нотация Мартина, более известная как «Crow's Foot Notation» [10].

На концептуальной модели представлено 2 основные предметные области (в контексте баз данных – схемы): «данные дел» и «документы».

Документы – это небольшая область, которая состоит из двух таблиц, в одной из которых хранятся документы, а в другой – категории, в которых эти документы находятся. У категории могут быть категории родители и это оформлено через внешний ключ таблицы на саму себя. Файлы (документы) и папки (категории) – будет прекрасной и общедоступной аналогией.

Данные дел – это основная область, в которой находится большинство сущностей. Основная сущность, от которой происходит начало любого бизнес-процесса в системе – Дело (case). К ней, через связи многие ко многим привязаны сущности Клиент (client) и Ответчик (defendant), а через связи ноль или один ко многим относятся Клиент и Носитель (carrier).

У дела практически всегда есть основной клиент, на основе его имени и формируется название дела, и, иногда бывают несколько связанных, но не основных клиентов. Они могут быть важны для дела, однако основным клиентом может быть только один. Пример – семейная пара, где оба будут являться пострадавшими от чего либо, однако юридически всё заполняется от лица мужа.

Ответчиков, как и клиентов, может быть несколько. К каждому ответчику либо от лица его Носителя (carrier), либо он может найти их сам, приставляются Юридический консультант (adjuster) и Прокурор (attorney). Последних может быть несколько, поэтому они тоже оформлены связью многие ко многим. Исходя из технического задания – очень важно иметь информацию о том, когда консультанты и прокуроры были приставлены к ответчику. Для этого в их связях присутствует дополнительное поле, которое хранит в себе время создания связи.

2.3 Требования к обобщённой архитектуре приложения

При проектировании архитектуры проекта в расчёт принимались следующие параметры:

- данные – самая главная часть всей системы, которая должна быть максимально защищена;
- для потенциального расширения на другие платформы необходимо использовать API подход без логики на клиенте;
- архитектура должна быть спроектирована таким образом, чтобы в дальнейшем не переписывать проект с нуля при переходе на микросервисную архитектуру, если она потребуется.

На основе данных требований была сформирована обобщённая диаграмма архитектуры системы, представленная на рисунке 2.3.

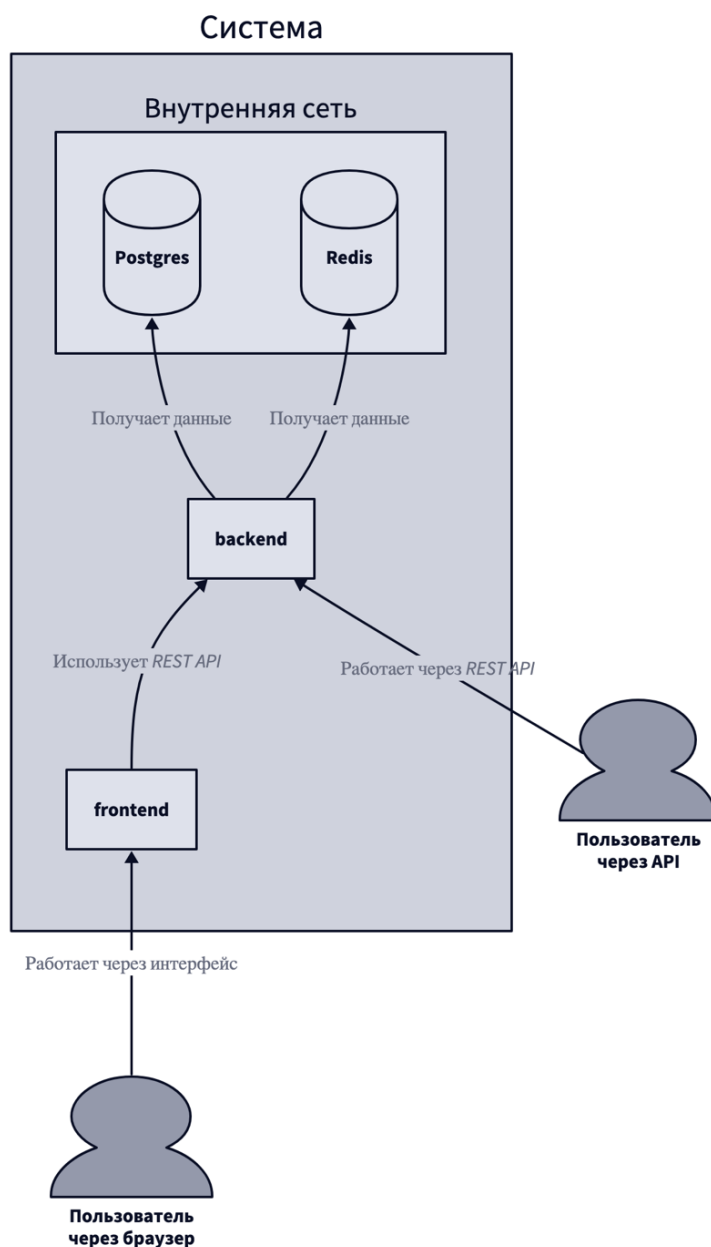


Рисунок 2.3 – Обобщённая архитектура системы

Взаимодействие с системой возможно двумя способами – через основное веб-приложение и через API. Важно учитывать, чтобы при изменении способа взаимодействия функционал системы не терялся.

Frontend должен представлять собой некое web-приложение с полноценным графическим интерфейсом, удобным для пользователя. Желательно использовать «реактивный» подход, т. е. чтобы работа с данными происходила без лишних перезагрузок страниц, ненужных анимаций и других, не имеющих дела, деталей.

Основная задача Backend сервера – принимать и обрабатывать запросы. Для уменьшения среднего времени обработки запроса необходимо использовать технологию кэширования, именно для этого и нужна вторая СУБД – Redis.

Redis это СУБД, которая хранит данные в памяти сервера, позволяя гораздо быстрее выполнять к ним обращение. Она не должна полностью копировать объём информации других баз, лишь временно запоминать данные, к которым недавно обращались [11].

Postgres является основной базой для хранения данных ввиду своей стабильности, проверенной многолетним опытом других компаний [12].

2.4 Обоснование выбора технологий и средств разработки

На основе функциональных и технических требований были выбраны следующие технологии для разработки проекта.

Docker является практически обязательным средством при разработке ввиду его простоте при работе с окружениями. Конфигурация, написанная однажды будет работать на любом другом компьютере, вне зависимости от внешней среды [13].

В качестве Frontend фреймворка на момент начала разработки (конец 2021 г., начало 2022 г.) основной выбор между React, и каким-либо новым, многообещающим, реактивным аналогом. Был найден Svelte, и несколько других, однако последние имели меньших размер сообщества и плохие отзывы пользователей, которые их использовали. React – это гигант,

который на рынке более 10 лет, но и он в сравнении оказался хуже. Основными минусами стали: большой порог входа, большое количество boilerplate кода, оптимизация. В реалиях 2023 г. к аналогам добавились бы Qwik, SolidJS. Статистика использования различных JS фреймворков приведена источником State Of JavaScript [14] и представлена на рисунке 2.4.

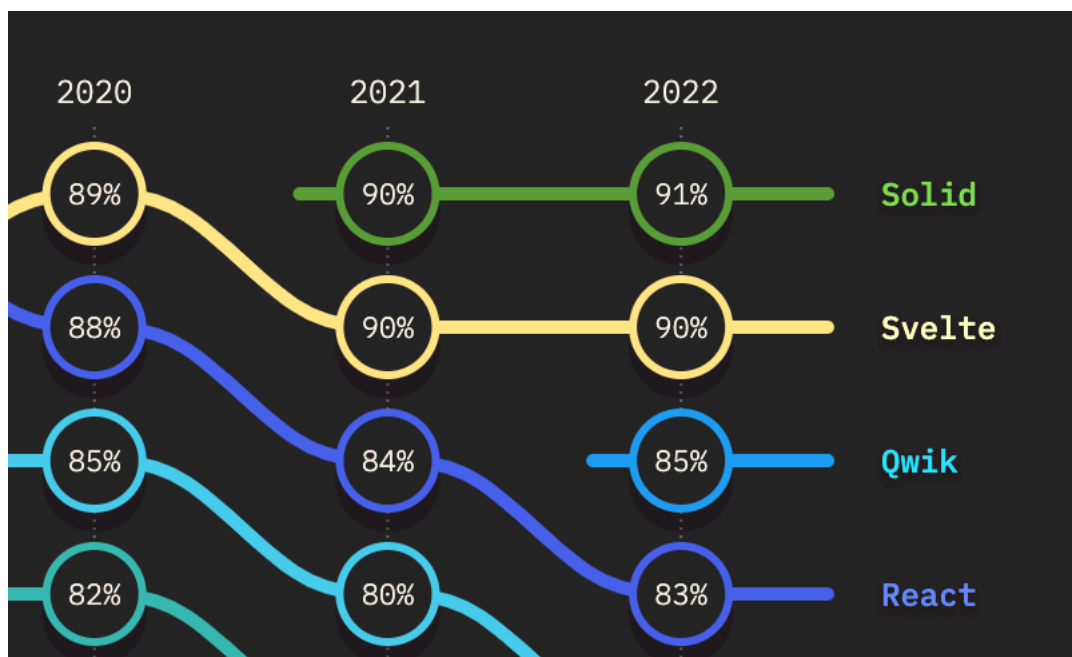


Рисунок 2.4 – Статистика JS фреймворков

Для Backend сервера был выбран Go, хотя Python, Java и Spring Boot или любой NodeJS фреймворк могли быть использованы с таким же успехом. Для Python основным минусом стала его скорость, ибо наша основная задача – чтение и генерация документов, с чем у Python есть проблемы из-за его внутренней архитектуры. Spring Boot предоставляет очень гибкий интерфейс для создания CRUD приложений, однако добавляет магию, что могло вызвать проблемы при разработке и использовании фреймворка. Под магией подразумевается большое количество сокрытых от пользователя действий, которые выполняют автоматически. Общим минусом для NodeJS и Python так же является отсутствие строгой типизации. Для NodeJS так же является однопоточным приложением и не очень хорошая реализация управления асинхронным кодом. В большинстве случаев против других альтернатив Go выигрывал за счёт своей простоты и скорости.

Redis и Postgres – СУБД, выбранные для использования в проекте. Основной особенностью Redis является хранение данных в памяти, что делает её идеальной в качестве СУБД для кэша. Postgres – практически золотой стандарт реляционных баз данных на текущий момент. Рассматривалась альтернатива в виде MongoDB, однако систематичность хранения данных была очень важна, а Mongo – это не реляционная СУБД.

2.5 Требования к аппаратному и техническому обеспечению.

Аппаратное обеспечение – физические компоненты, которые подключены к ПК, такие как: процессор, оперативная память и др. Основным источником требований для системы будет являться Docker. Он может быть установлен на все основные ОС, однако требования могут варьироваться. Стоит учитывать, что минимальных требований может быть недостаточно для постоянной стабильной работы системы, а лишь для её запуска и тестирования. Далее приведены общие требования для установки и запуска Docker:

- 64-разрядный процессор с поддержкой виртуализации;
- 2 ГБ оперативной памяти;
- 20 ГБ свободного места на жёстком диске.

Техническое обеспечение – различного рода вспомогательные программы и утилиты, которые составляют логическую часть компьютерной системы. Благодаря особенностям контейнеризации – не нужно устанавливать все зависимости на ПК, Docker сделает это сам, дополнительно изолируя их от основного окружения пользователя.

Таким образом, единственное техническое требование для запуска – Docker с поддержкой Docker-Compose.

3 Разработка программных средств доступа к данным

3.1 Разработка принципа доступа к данным

Одна из задач, которая стояла при разработке – это предоставление быстрого доступа к данным. При увеличении объёма хранимой информации скорость выборки напрямую из СУБД будет только падать, поэтому необходимо было найти решение, которое позволяло бы сохранять константную скорость доступа при любом объёме данных.

В качестве решения было решено использовать промежуточную СУБД Redis, так как основным её преимуществом является скорость, за счёт того, что она хранит данные в памяти, а не на жёстком диске.

Redis – база данных типа «ключ-значение», где в качестве ключа выступает уникальная строка, а значением – набор хранимых данных.

Для формирования ключа используются следующий формат, представленный на рисунке 3.1.

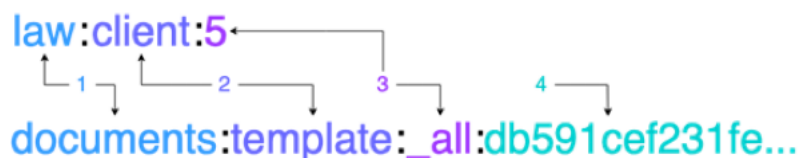


Рисунок 3.1 – Формат ключа в Redis

Обозначение, представленные на рисунке, следующие:

1 – схема внутри основной базы данных PostgreSQL позволяет в рамках одной базы данных отделить фрагменты предметной области друг от друга;

2 – название таблицы в основной базе данных PostgreSQL соответствует сущности предметной области;

3 – уникальный идентификатор записи (группы записей) в таблице, для единичной записи указывается значение идентификатора, для набора всех записей используется константа _all;

4 – хэш-значение, соответствующее набору фильтров выборки, если выборка не статична, а имеет множество вариаций, в зависимости от фильтров и параметров, которые пришли от пользователя.

Основной объём данных всё равно хранится в главной СУБД (Postgres), однако используется промежуточное хранилище, которое хранит уже готовую выборку в формате JSON, т. к. именно он используется в качестве ответов клиентской части.

Общий алгоритм получения данные представлен на рисунке 3.2.

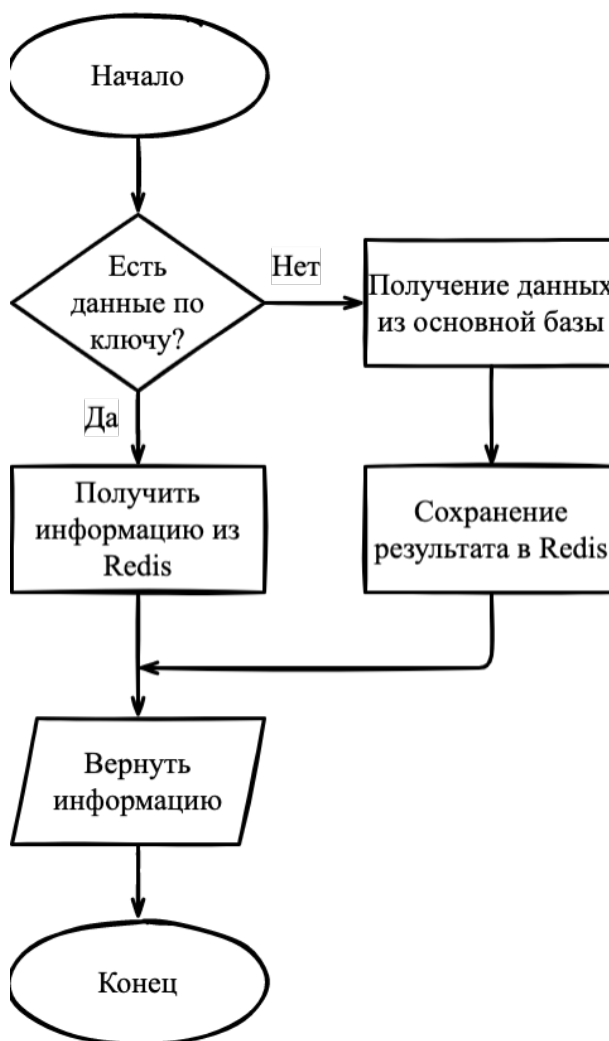


Рисунок 3.2 – Алгоритм выборки данных

На основе созданного решения была опубликована статья «Применение промежуточного хранилища на платформе Redis для повышения скорости выборки данных», подробно описывающая принципы хранения и обработки информации [15].

3.2 Разработка физической модели базы данных

Для моделирования баз данных был использован инструмент с открытыми исходным кодом – pgModeler [16]. Он позволяет наглядно увидеть структуры таблиц, схем и связей, имеет поддержку цветов для лучшего визуального восприятия, упрощает работу с созданием таблиц, внешних и первичных ключей, перечислений и изначальных данных. pgModeler имеет несколько режимов импорта и экспорта, самый важный из которых – Diff. Данный режим позволяет автоматически проверить конечную базу данных либо модель на совместимость и применить к ней изменения динамически, не затрагивая уже имеющиеся данные.

Единственным минусом pgModeler его «условная» бесплатность. Само ПО является бесплатным, однако предоставляется лишь в виде исходного кода, который пользователю самому необходимо скомпилировать, что для большинства обычных пользователей может вызвать затруднения.

В качестве аналогов были так же рассмотрено часть проектов, описанных ниже, однако в каждом из был один или несколько недостатков, не позволяющий вести комфортную разработку.

pgAdmin – утилита для управления базами данными Postgres. Изначально это было лишь веб-приложением, и только в недавнем времени был выпущен порт как десктоп-приложение, однако оптимизация и UX не позволяют полноценно использовать его в разработке.

DbVisualizer – инструмент, позволяющий проектировать СУБД, доступен в платной и бесплатной версиях. Бесплатная версия покрывает практически все нужны, за исключением отсутствия аналога Diff операции.

Далее представлена физическая модель базы данных в pgModeler (рисунок 3.3).

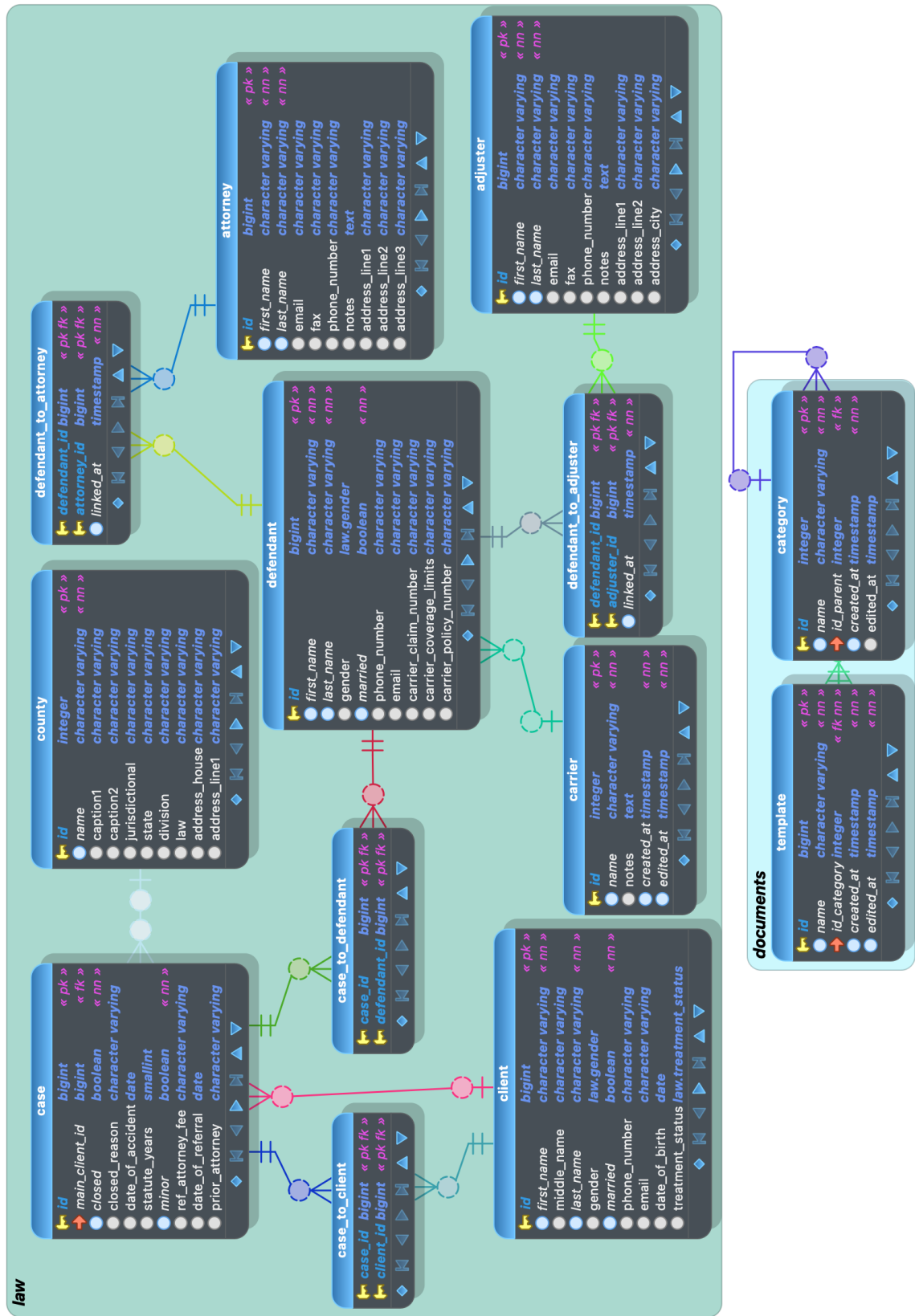


Рисунок 3.3 – Физическая модель базы данных

Далее, в таблице 3.1 приведёт список таблиц и их назначение в системе.

Таблица 3.1 – Список таблиц физической модели

№	Наименование	Назначение	Кол-во полей	Связи
1	2	3	4	5
1	case	Содержит данные о делах	35	client defendant
2	county	Содержит данные о округах	16	case
3	client	Содержит данные о округах	33	case
4	defendant	Содержит данные об ответчиках	39	attorney adjuster carrier case
5	carrier	Содержит данные о носителях	5	defendant
6	attorney	Содержит данные о прокурорах	16	defendant
7	adjuster	Содержит данные о консультантах	15	defendant
8	template	Содержит данные о шаблонах	5	category
9	category	Содержит данные о категориях	5	category template

Для большинства основных сущностей характерна связь «Многие ко многим», для чего используется следующий перечень вспомогательных таблиц:

- «case_to_client»;
- «case_to_defendant»;
- «defendant_to_attorney»;
- «defendant_to_adjuster».

Данные таблицы состоят из двух полей: внешних ключей на «id» сущностей, которые они соединяют.

В случае «defendant_to_attorney» и «defendant_to_adjuster» в таблицах связи имеются дополнительные параметры.

Таблица «category» имеет особую рекурсивную связь, на саму себя. В ней хранятся категории, к которым относятся шаблоны, а данная связь используется для создания иерархической структуры.

Для более простого понимания можно думать о категориях как о папках, а шаблонах – файлах, которые находятся в них.

Если категория имеет ссылку на категорию родителя – знает она находится внутри какой-то другой, более большой, категории.

3.3 Согласование протоколов взаимодействия

В качестве протокола взаимодействия был выбран протокол HTTP с использованием REST API методологии. В качестве альтернативы рассматривались «GraphQL» и «gRPC», но обе технологии оказались слишком большими для размеров текущего проекта. REST был выбран по следующим причинам [17]:

- простота и лёгкость в использовании и интеграции;
- масштабируемость;
- гибкость;
- независимость от ЯП клиента и сервера.

В системе были реализованы следующие конечные точки, которые представлены в таблице 3.2.

Таблица 3.2 – Список конечных точек.

№	Кон. Точка	Описание	Поддерживаемые методы
1	2	3	4
1	/adjuster	Адрес для получения консультантов	GET, POST, PUT, DELETE
2	/attorneys	Адрес для получения прокуроров	GET, POST, PUT, DELETE
3	/carriers	Адрес для получения носителей	GET, POST, PUT, DELETE
4	/cases	Адрес для получения дел	GET, POST, PUT, DELETE
5	/categories	Адрес для получения категорий шаблонов	GET, POST, PUT, DELETE
6	/clients	Адрес для получения клиентов	GET, POST, PUT, DELETE
7	/counties	Адрес для получения округов	GET, POST, PUT, DELETE
8	/defendants	Адрес для получения ответчиков	GET, POST, PUT, DELETE
9	/download	Адрес для скачивания генерируемых шаблонов	GET
10	/generate	Адрес для генерации документов, который возвращает ссылки на «/download»	POST
11	/hierarchy	Адрес для получения иерархии категорий и шаблонов	GET
12	/templates	Адрес для получения метаданных о шаблонах	GET, POST, PUT, DELETE

REST API используют терминологию `endpoint` как адреса, по которому может быть получена информация по определённому ресурсу. Соответственно, `endpoint` `«/cases»` означает что данный адрес доступ по следующей ссылке: `«https://server_address.com/cases»`, а в случае локального сервера это будет выглядеть как `«localhost/cases»`.

В системе существует несколько «особых» конечных точек:

- `/download`;
- `/generate`;
- `/hierarchy`.

`«/generate»` – принимает запрос для генерации нового документа. Пример принимаемых данных приведёт на листинге 3.1.

Листинг 3.1 – Запрос на генерацию документа

```
{
  "template_id": 1,
  "case_id": 1,
  "client_id": 1,
  "defendant_id": 1,
  "adjuster_id": 1,
  "attorney_id": 1
}
```

В качестве ответа возвращается уникальный идентификатор для скачивания готового документа. Пример ответа приведёт на листинге 3.2.

Листинг 3.2 – Ответ на запрос о генерации

```
{
  "data": {
    "uuid": "ca81dd3c-21d9-4ec3-854f-e4f0db2178f9"
  }
}
```

`«/download»` – используется для скачивания уже сгенерированных шаблонов. Например, для того чтобы скачать шаблон, сгенерированный в примере выше, необходимо перейти по ссылке `«http://localhost/download/ca81dd3c-21d9-4ec3-854f-e4f0db2178f9»`. Данная ссылка активна в течении пяти минут с момента генерации.

«/hierarchy» – используется для создания обобщённой структуры для категорий и шаблонов. Гораздо удобнее иметь готовый общий механизм, что воссоздавать иерархию на клиенте.

3.4 Разработка программных средств серверной части

Язык Go не привязан к файловой структуре проекта, поэтому имена файлов и папок могут быть любыми. Каждая папка является модулем с таким же идентичным названием, а сама структура серверной части представлена на диаграмме компонентов (рисунок 3.4).

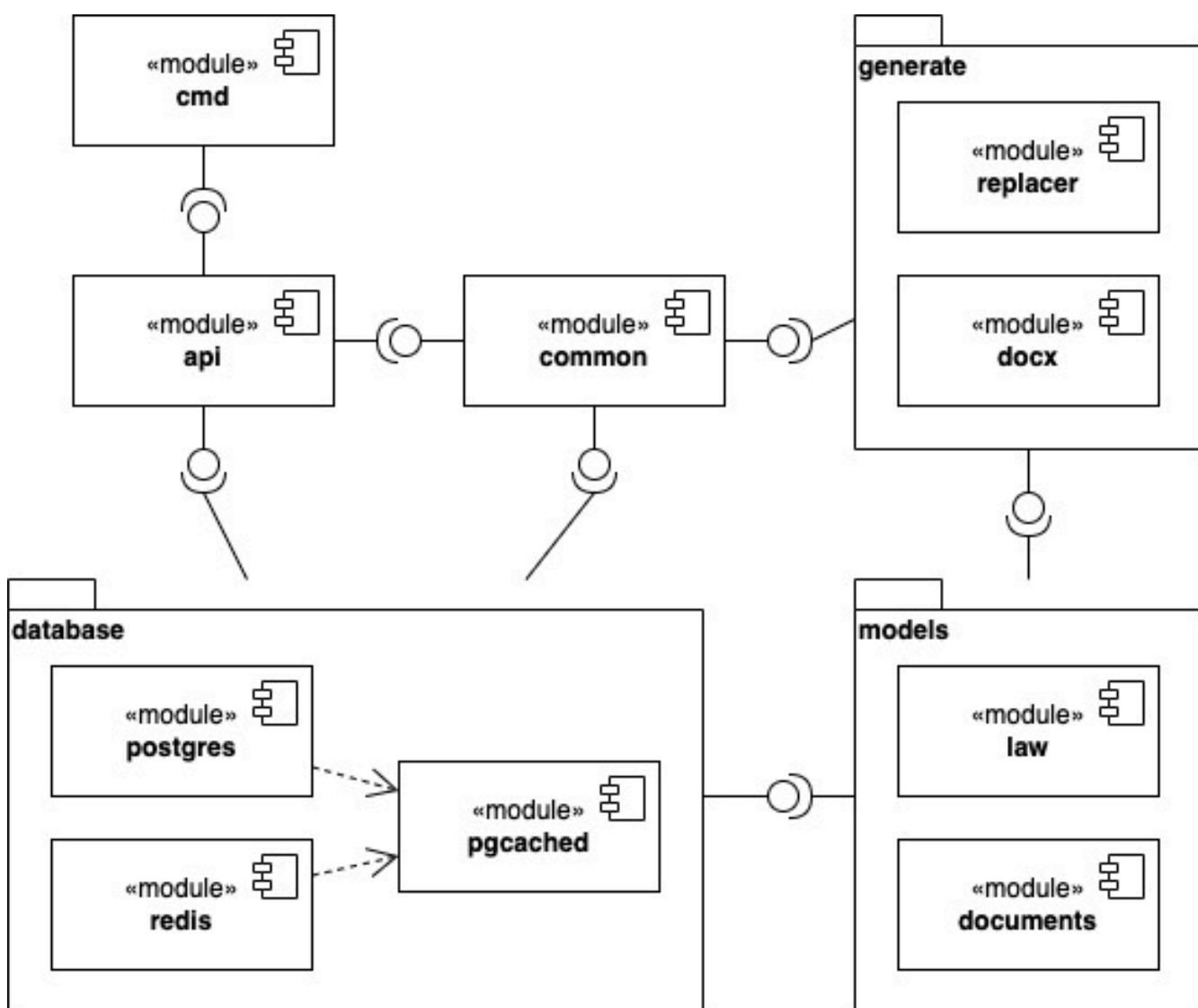


Рисунок 3.4 – Компоненты серверной части

Модуль «api» отвечает за взаимодействие с сервером через REST API. В качестве роутера используется «httprouter». «api/endpoints» – модуль, где

хранятся данные об адресах и их обработчиков. «api/middleware» – содержит в себе «перехватчиков» запросов к серверу. На данный момент он лишь логирует входящие запросы, однако в последующих версиях здесь будет находиться обработка сессии и верификации пользователей.

Модуль «cmd» отвечает за различного рода файлы запуска, в нашем случае он всего лишь один – «cmd/server».

Модуль «common» содержит общие данные, для разных слоёв проекта, для того чтобы избежать рекурсивных зависимостей. Тут находятся общие ошибки, сигнатуры функций и параметры фильтрации.

В модуле «database», хранится всё связанное с базами данных. «database/postgres» является драйвером для работы с Postgres, «database/redis» – драйвером для Redis, «database/pgcached» – надстройкой над предыдущими двумя, который реализует функционал хэш-хранилища.

Модуль «models» содержит различного рода модели, как для работы с базой данных, так и просто перечисления для внутренних ограничений.

Папка «uploads» содержит загруженные шаблоны, сжатые в «.gzip» формат.

Во время разработки была опубликована отдельная библиотека, упрощающая работу с экспортированием тегов из структур – Textra [18]. Внутри проекта она используется для автоматической генерации переменных в шаблонах. С недавнего времени библиотека была добавлена в репозиторий go-awesome [19], где хранится список полезных Open Source проектов и находится в разделе Go Tools.

3.5 Разработка программных средств пользовательской части

Svelte – это фреймворк для Javascript, который упрощает работу с реактивностью и позволяет быстро разрабатывать клиентские приложения. На текущее время он является вторым фреймворком по соотношению хороших отзывов к популярности, уступая лишь SolidJS. Для работы большинства фреймворков на текущий момент требуется внутренняя платформа для

разработки, в качестве которой была взята Vite, как стандартный вариант, с которым рекомендуется использоваться Svelte. Далее, на рисунке 3.5, приведена файловая структура сервера.

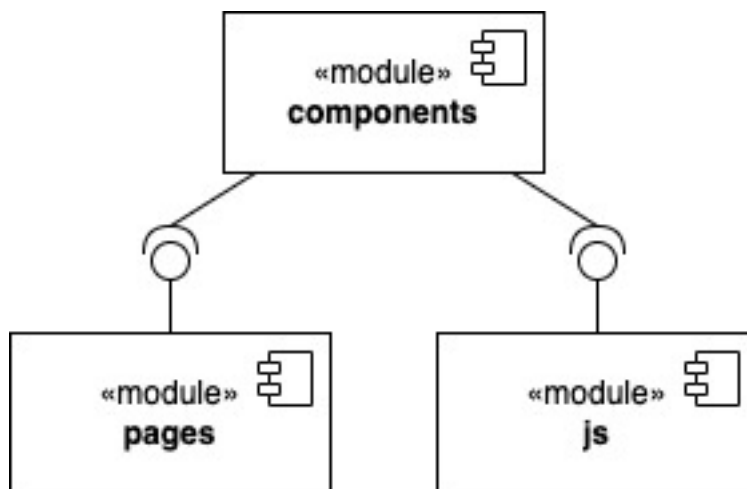


Рисунок 3.5 – Компоненты клиентской части

В данном проекте находятся 3 основные директории:

- «components» – файлы компонентов;
- «js» – вспомогательные JS файлы;
- «pages» – список и компоненты страниц (ссылок) в приложении.

В папке «components» лежат различные вспомогательные компоненты, которые могут быть переиспользованы, в чём и состоит основная причина, зачем логически разделять их. Например, один раз создав компонент «Button.svelte» можно импортировать его из любого файла проекта, и данная кнопка везде будет одинаковой. При изменении исходного кода кнопка она так же поменяется во всех местах, что очень удобно.

В папке «js» лежат несколько вспомогательных JavaScript модулей и утилит для общих строковых операций, работе с локальными хранилищами, различного рода парсеры и функции для API запросов.

Папка «pages» особенная, так как каждый файл или подпапка внутри неё образует полноценную ссылку. Например, при переходе по пути «localhost/adjusters» пользователю будет выведен файл

«/pages/adjusters/index.svelte», если папка «adjusters» существует, либо «/pages/adjusters.svelte», если файл «pages/adjusters.svelte» существует.

Соответственно, чтобы создать страницу на сайте – необходимо просто создать папку либо файл с таким названием и сделать на него ссылку.

Внутри папки cases есть особая папка – «[case]». Данная папка называется «slug», что означает она используется как переменная в пути, т. е. при переходе на «localhost/cases/14» – особая переменная внутри этого шаблона примет значение 14, что позволяет динамически генерировать страницы.

3.6 Разработка программных средств генерации документов

3.6.1 Синтаксис внутренних переменных документов

Основной функцией данной информационной системы является автоматизация документооборота в автономном режиме, что означает что все необходимые изменения, связанные с документами, может и должен выполнять конечный пользователь, а не разработчик. Шаблоны для документов со временем будут меняться и просить разработчиков их изменять просто нецелесообразно.

На основе этих требований было решено разработать самостоятельный шаблонизатор документов и предоставить документацию к нему. В этом случае, пользователь (заказчик) сам сможет создавать нужные ему шаблоны в уже известном ему .docx формате, а разработчику не придётся вникать в тонкости каждого документа.

В качестве решения заказчику будет предоставлен набор переменных, которые он может вставлять в изначальный шаблон, после чего они будут заменены на реальные данные из дела, для которого будет генерироваться шаблон.

В качестве переменной выступает любое значение, заключённое в фигурные скобки {variable}, это и является самым минимально-возможным

суффиксом, однако сам синтаксис переменной немного сложнее и представлен на рисунке 3.6.

{ prefix# body \$suffix | suffix2 }

Рисунок 3.6 – Синтаксис переменных

Переменная разделена на 3 части:

- префикс (prefix);
- тело (body);
- суффикс (suffix).

Префикс находится до символа «#» указывает на то, из какой сущности необходимо использовать набор переменных тела, так как их имена могут быть идентичными (например – поле first_name присутствует как минимум в 4 сущностях). В особых случаях префикс может быть опущен, когда переменная обозначает общие данные, например – текущее время. Перечень доступных префиксов представлен в таблице 3.3.

Таблица 3.3 – Перечень префиксов

№	Префикс	Сущность (англ.)	Сущность (рус.)
1	2		3
1	adjuster#	Adjuster	Консультант
2	attorney#	Attorney	Прокурор
3	carrier#	Carrier	Носитель
4	case#	Case	Дело
5	client#	Client	Клиент
6	county#	County	Округ
7	defendant#	Defendant	Ответчик
8	Пустой	-	Общие данные

Тело находится между символами «#» и «\$» и относится к текущему префиксу, и на основе него решается какая переменная будет подставлена. Список тел практически идентичен списку полей сущности в базе данных. Например, если нужно подставить имя клиента в документ – необходимо использовать переменную «{client#first_name}».

Суффикс – это дополнительные визуальные преобразования над уже имеющимися данными, самый простой пример таких – перевод текста в нижний или верхний регистр или работа с датами. Например, если необходимо перевести имя клиента в верхний регистр – нужно использовать переменную «{client#first_name\$uppercase}». Некоторые суффиксы могут быть использованы только на определённый тип данных, например суффиксы, который преобразуют даты в определённый формат. Полный перечень суффиксов представлен на таблице 3.4.

Таблица 3.4 – Перечень суффиксов

№	Суффикс	Описание	Может быть использован на	Преобразование	
				Исходный текст	Результат
1	2	3	4	5	6
1	\$capitalize	Делает первую букву заглавной	Все строки	пример	Пример
2	\$uppercase	Верхний регистр	Все строки	пример	ПРИМЕР
3	\$lowercase	Нижний регистр	Все строки	ПРИМЕР	пример
4	\$US	Стандартный формат дат в США	Даты	21.09.2001	September 21, 2001
5	\$notary	Особый формат дат для адвокатов	Даты	21.09.2001	This 21st day of September, 2001
6	\$pronoun	Местоимение пола	Пол	Male	he
7	\$possessive	Притяж. местоимение	Пол	Male	his
8	\$salutation	Приветственное местоимение	Пол	Male	mr.

В процессе разработки оказалось, что даже данного функционала недостаточно, что полностью обеспечить желаемую независимость, поэтому позже был добавлен оператор «|», который работает идентично своему аналогу в UNIX и позволяет использовать несколько суффиксов (вплоть до 5), один после другого.

Такой приём был назван «Цепные суффиксы», а пример его использования приведён на рисунке 3.7.

Порядок:	1	->	2
{defendant#gender}		->	Female
{defendant#gender\$salutation}		->	Ms.
{defendant#gender\$salutation uppercase}		->	MS.
{defendant#gender\$salutation lowercase}		->	ms.

Рисунок 3.7 – Цепные суффиксы

Используя данную технику, можно преобразовывать переменные в совершенно различные форматы, при этом подставляя их в предложения с корректной пунктуацией.

3.6.2 Процесс генерации документов

Пример запроса на генерацию и загрузку документов представлен в разделе 3.2, на листингах 3.1, 3.2.

При получении запроса система обрабатывает его и пытается получить данные о сущностях с теми идентификаторами, что ей были отправлены.

Если сущность не существует – система пропустит её, и переменные с префиксом данной сущности заменены не будут. Исключением является идентификатор шаблона, который не может быть опущен.

После успешной замены все переменных шаблон сохраняется в памяти на 5 минут, в течении которых он может быть скачан по ссылке. Самый простой пример шаблона представлен на рисунке 3.8.

Test suffixes:

```
current_date$notary – {current_date$notary}
current_date$US – {current_date$US}
client#first_name$capitalize – {client#first_name$capitalize}
client#first_name$uppercase – {client#first_name$uppercase}
client#first_name$lowercase – {client#first_name$lowercase}
```

Рисунок 3.8 – Пример шаблона

А результат генерации документа на основе этого шаблона представлен на рисунке 3.9.

Test suffixes:

current_date\$notary – this 30th day of May, 2023

current_date\$US – May 30, 2023

client#first_name\$capitalize – Имя

client#first_name\$uppercase – ИМЯ

client#first_name\$lowercase – имя

Рисунок 3.9 – Результат генерации

После скачивания документ имеет имя «Фамилия, Имя – Название шаблона», а в случае необходимости, переименовать его можно уже после скачивания.

Исходные коды генерации документов на серверной части представлены в приложении В, а клиентской части – в приложении Г.

4 Развёртывание окружения

4.1 Разработка архитектуры

В качестве основной платформы для разработки архитектуры будет использоваться Docker, который позволяет контейнеризировать компоненты системы и запускать их в одинаковых условиях на разных машинах. Соответственно, для разработки и развёртывания необходим предустановленный Docker на любой из машин, где система будет запускаться. Подробная архитектура представлена на рисунке 4.1.

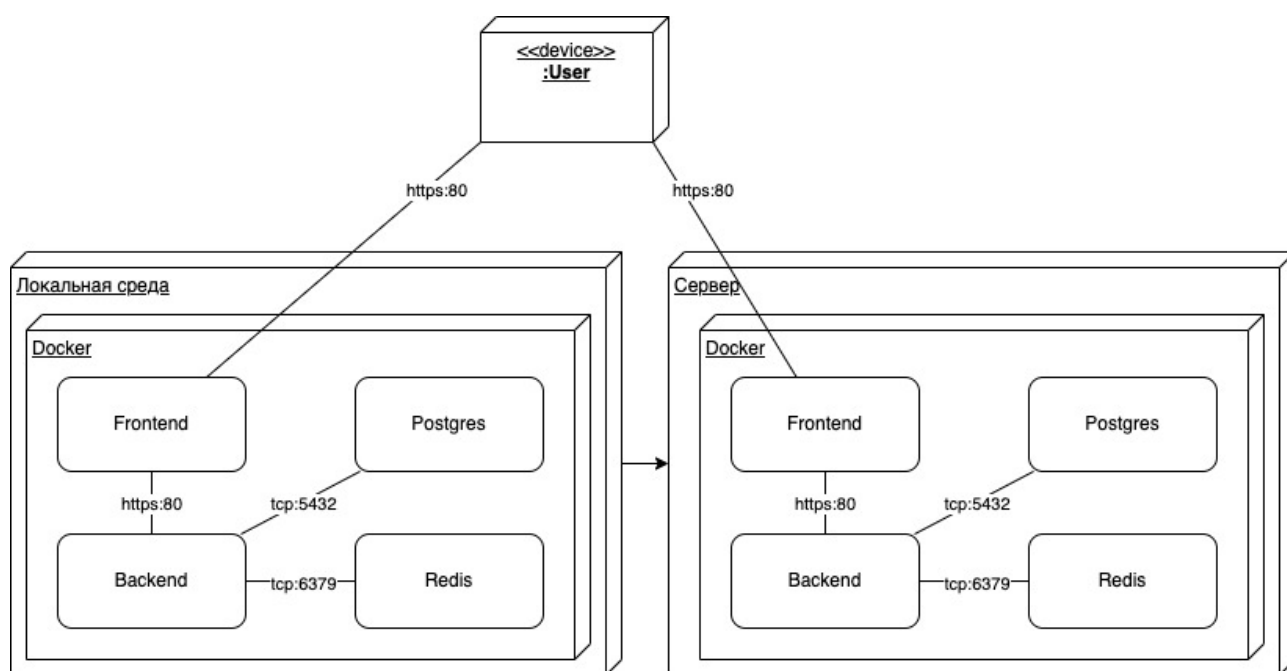


Рисунок 4.1 – Архитектура развёртывания

Более подробное описание контейнеров представлено ниже, на таблице 4.1.

Таблица 4.1 – Спецификации контейнеров

№	Контейнер	Образ основы	Доступен для пользователя?	Порт
1	2	3	4	5
1	frontend	node-19:alpine	Да	80
2	backend	cosmtrek/air:latest	Да	5000
3	redis	redis:7-alpine	Нет	6379
4	postgres	postgres:15-alpine	Нет	5432

Образ основы – это изначальный образ настроенной системы, в которую будет загружаться и запускаться код или другие компоненты.

Контейнеры `redis` и `postgres` доступны для подключения к ним в режиме разработки, но будут закрыты от внешних источников при развёртывании на реальном сервере.

В большинстве случаев используются образы дистрибутива `alpine`, так как они самые легковесные по сравнению со всеми остальными, соответственно скачивание и запуск не будет занимать долго времени.

4.2 Развёртывание локального окружения для разработки

Во время разработки очень важным является возможность полностью воссоздать рабочее окружение на другом ПК, для того чтобы сторонние лица, такие как: разработчик, тестировщик, и, иногда заказчик смогли работать с полностью идентичной копией ПО. В случае данного проекта для запуска полноценной локальной копии программы необходимо запустить службы:

- «frontend», которым является Javascript Vite сервер;
- «backend», которым является приложение сервера, написанное на языке Go;
- «redis» – сервер с базой данных Redis;
- «postgres» – сервер с базой данных Postgres.

Установка и индивидуальная настройка каждой из вышеперечисленных служб – очень трудоёмкий и затратный по времени процесс, умножая это на количество потенциальных сторонних лиц, которым нужно выполнить данные действия можно понять, что ручной вариант рассматривать не имеет смысла. Для решения данной проблемы была использована программная платформа `Docker`.

`Docker` – платформа, которая позволяет разработчикам написать конфигурацию и использовать её на любой машине, убирая все возможные ошибки, связанные с локальными настройками на любом ПК. Этот процесс происходит при помощи использования виртуальных операционных систем, на которые устанавливаются необходимые компоненты и файлы конфигурации. Это позволяет сократить время, потраченную на настройку и исправление локальных ошибок и уделить время разработке реального продукта.

Существует несколько вариантов интеграции Docker в процесс разработки. В данном проекте было выбрано использование дополнительного конфигурационного файла «docker-compose.dev.yml», пример которого представлен на листинге 4.1.

Листинг 4.1 – Файл docker-compose.dev.yml

```
version: '3.9'
services:
  proxy:
    container_name: lawfirm-nginx-proxy
    image: nginx:1.23-alpine
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    ports:
      - 80:80
    environment:
      - NGINX_PORT=80
    depends_on:
      - backend
      - frontend
  frontend:
    ...
  backend:
    ...
  redis:
    ...
  postgres:
    ...
```

На данном конфигурационном файле присутствует ряд сервисов, которые описаны в словаре «services»: «proxy», «frontend», «backend», «redis», «postgres». Для каждого из сервисов присутствует ряд конфигурируемых параметров, часть из них для примера представлена у сервиса «проху». Полное объяснение каждого из значений можно найти в официальной документации [20].

При первом запуске Docker скачает необходимые зависимости для запуска сервисов, выполнит все конфигурационные команды и подготовит окружение для работы. При последующих запусках повторно скачивать и настраивать окружение не нужно, будет использована уже готовая конфигурация.

Таким образом можно передавать код всего проекта людям, которые могут не знать его внутренние особенности, и они без проблем могут запустить его на своём локальном ПК.

Так как данное окружение предоставляется, прежде всего, для разработки, то для него характерны некоторые особенности по сравнению с основной конфигурацией, а именно:

- открытый доступ (порты) и возможность подключения к базам данных «redis» и «postgres» для разработчиков;
- дополнительные надстройки для «frontend» и «backend» сервисов, позволяющие вести удалённую отладку кода;
- игнорирование части параметров безопасности ввиду их ненужности.

При необходимости, можно вносить дополнительные надстройки или даже целые сервисы для разработки, они никак не будут влиять на основное окружение, даже при использовании одинаковой кодовой базы.

4.3 Развёртывание реального окружения

Также, как и в случае с локальным окружением – производить развёртывание реального окружения в ручном режиме – это большая потеря времени для процесса разработки, поэтому это тоже было необходимо автоматизировать.

Конечный результат данного процесса предполагает, что на какой-либо реальный сервер в сети, доступный реальным пользователям будет загружена или обновлена программа и с ней можно будет работать в этот же момент. Docker предоставляет лишь оболочку для запуска и конфигурацию, однако загрузить и запустить сервисы на удалённой машине не представляется возможным. Для этого были использованы CI\CD возможности GitLab.

CI\CD – это, в переводе с английского, «непрерывная интеграция» и «непрерывная поставка», набор практик и технологий, которые позволяют разработчикам автоматизировать различные процессы жизненного цикла ПО. Чаще всего под автоматизацию попадают: тестирование, сборка, линтинг, развёртывание окружений [21].

Процесс автоматического развёртывания подразумевает, что при выполнении некоторых условий запускается такой же Docker сервис, в задачи

которого входит подключиться к удалённому серверу, загрузить на него обновлённую версию кодовой базы и запустить её. Сервис считает свою работу успешной, если при выполнении заданного перечня действий не произошло ошибок. Пример конфигурационного файла для GitLab CI/CD на листинге 4.2.

Листинг 4.2 – Файл .gitlab-ci.yml

```
...
stages:
  ...
  - deploy
...

deploy_staging:
  stage: deploy
  environment:
    name: staging
    url: ... # Адрес сервера
  before_script:
    - apt-get update -qq
    - apt-get install -qq git
    # Setup SSH deploy keys
    - 'which ssh-agent || ( apt-get install -qq openssh-client ) '
    - eval $(ssh-agent -s)
    - ssh-add <(echo "$SSH_PRIVATE_KEY")
    - mkdir -p ~/.ssh
    - '[[ -f ~/.dockerenv ]] && echo -e "Host *\n\tStrictHostKeyChecking
no\n\n" > ~/.ssh/config'
  script:
    - ssh root@{адрес сервера} "cd /app && git checkout main && git pull
&& docker compose build && docker compose stop && docker compose start && exit"
  only:
    - main
```

В данном конфигурационном файле в качестве одной из стадий указана «deploy_staging», где в списке «before_script» представлен набор действий для UNIX-подобной операционной системы и выполняется данная стадия будет только при условии «only: main», что означает – только при загрузке коммита на ветку «main».

Триггером (инициатором) выполнения всех стадий является коммит загруженный в любую ветку репозитория проекта.

Примерный набор действий в данном файле, следующий:

- установить и обновить зависимости;
- добавить ssh-ключ для доступа к файлам репозитория;

- изменить конфигурацию ssh;
- подключиться по ssh к серверу;
- обновить кодовую базу;
- оставить текущие docker-контейнеры и запустить новые;
- выйти.

После выполнения данных действий на сервере будет обновлена кодовая база, перезапущены docker-контейнеры и сервер будет готов к работе.

В случае необходимости добавления новых серверов, либо сервисов на уже существующие сервера присутствует возможность «наследования» конфигураций, т. е. можно использовать часть действий до подключения к серверу как базовую конфигурацию, а после добавлять необходимые действия. Данная возможность не используется в рамках проекта ввиду его относительно малого размера, однако о ней стоит помнить.

5 Контроль качества программных средств

5.1 Планирование процесса контроля качества программных средств

5.1.1 Мероприятия по контролю качества программных средств

Контроль качества программных средств – проверка соответствия функционала ПО к изначальным требованиям заказчика. В этот процесс так же входит поиск и устранение багов, неполадок, граничных случаев системы. В рамках данного приложения было осуществлено тестирование клиентской и серверной части методом белого ящика.

Для полноценного тестирования, проверки всех возможных вариантов отказов и ошибок системы и последующей интеграции ПО в реальную среду тестирование будет проводиться в несколько этапов. Ввиду ограниченности человеческих ресурсов в проекте стоит учитывать, что тестирование лучше всего

Первый этап – ручное тестирование. Данный вид тестирования является самым распространённым и применяется практически каждым разработчиком прямо во время разработки. Каждый раз после внесения изменений программист запускает программу чтобы убедиться, что она работает корректно, в противном случае процесс повторяется.

Второй этап – автоматизированное тестирование. На данном этапе происходит тестирование программы другой программой. Самый распространённый вариант такого тестирования – Юнит тестирование. Такой тип тестирования позволяет писать тест-программы для основных программ, которые эмулируют реальное поведение, позволяя сразу обнаруживать ошибки, которые могли быть незамечены при ручном тестировании, ввиду ограниченности ресурсов.

Третий этап – тестирование с помощью CI\CD технологий. Прежде чем перейти к тому, что такое CI\CD, необходимо понять на основе чего он работает.

Git – это система контроля версий, применяемая при разработке ПО, позволяющая вести полный учёт изменений файлов и легко организовывать работу от маленьких до очень больших команд разработчиков. При работе с ним программисты постоянно загружают свой код на сервер, либо скачивают его оттуда, чтобы увидеть добавленные изменения других разработчиков. Процесс скачивания кода называют – «Pull», а загрузки своего – «Push».

CI/CD (англ. continuous integration, continuous delivery) – это процесс непрерывной интеграции и непрерывного развёртывания при разработке. Он позволяет выполнять некоторый, заранее известный порядок действий, который обычно состоит из автоматического запуска юнит-тестов и загрузки кода на сервер. В большинстве сервисов Git, таких как GitHub и Gitlab, данные процессы можно запустить вручную, однако чаще всего они привязаны к действию загрузки кода (Push).

Существует отдельная группа программ, задача которой выявлять ошибки, баги, стилистические неточности и странные (возможно – опасные) конструкции на основе статического анализа кода. Такие утилиты называются Линтеры. Для языка программирования Go, на котором написана серверная часть данного проекта существует великолепный линтер – GolangCi-Lint [22]. Это сборник из 120 отдельных линтеров, предоставляющий глубокий анализ проекта на всевозможные ошибки.

5.1.2 Обоснование выбора модели тестирования

В качестве ручных методов тестирования были рассмотрены следующие методы: покрытие решений и покрытие условий.

Покрытие решений – это способ тестирования, при котором проверяется корректность результат при наличии ветвлений. При использовании данного метода необходимо составить такое число тестовых данных, с использованием которых каждое из условий будет принимать истинное и ложное значение хотя бы 1 раз.

Покрытие условий – это более улучшенный способ, основанный на покрытии решений. Имеет смысл его использовать только в тех частях, где присутствуют сложные условия, включающие в себя такие логические выражения как «и» или «или». При использовании данного способа необходимо что каждое составное условие принимало значения истина и ложь хотя бы 1 раз.

Как основной способ тестирования было выбрано юнит тестирование методом белого ящика, ввиду его лёгкой интеграцией с CI\CD процессом.

Несмотря на то, что метод белого ящика менее эффективен, ввиду того что разработчик не всегда может адекватно протестировать свой собственный код на предмет граничных случаев, использовать тестирование методом чёрного ящика не представляется возможным.

Язык программирования Go обладает встроенным модулем тестирования и встроенным модулем проверки покрытия, который позволяет текстовым редакторам получить информацию о том, какие строки были покрыты тестами, а какие нет. Пример представлен на рисунке 5.1.

```
// returns a placeholder splitted into 3 parts.
// case#name$uppercase -> case, name, uppercase.
func ParsePlaceholder(ph string) (prefix, body, suffix string) {
    body = ph

    prefixIndex := strings.Index(body, "#")
    if prefixIndex != -1 {
        prefix = body[:prefixIndex]
        body = body[prefixIndex+1:]
    }

    suffixIndex := strings.Index(body, "$")
    if suffixIndex != -1 {
        suffix = body[suffixIndex+1:]
        body = body[:suffixIndex]
    }

    return prefix, body, suffix
}
```

Рисунок 5.1 – Полное покрытие тестами

Как видно, в данном случае все строки выделены светлым фоном, что означает что тесты, написанные для этой функции, протестировали её функционал на 100%.

Для примера, закомментируем несколько тестов и запустим проверку покрытия ещё раз. Результат проверки представлен на рисунке 5.2.

```
// returns a placeholder splitted into 3 parts.
// case#name$uppercase -> case, name, uppercase.
func ParsePlaceholder(ph string) (prefix, body, suffix string) {
    body = ph

    prefixIndex := strings.Index(body, "#")
    if prefixIndex != -1 {
        prefix = body[:prefixIndex]
        body = body[prefixIndex+1:]
    }

    suffixIndex := strings.Index(body, "$")
    if suffixIndex != -1 {
        suffix = body[suffixIndex+1:]
        body = body[:suffixIndex]
    }

    return prefix, body, suffix
}
```

Рисунок 5.2 – Неполное покрытие тестами

Исходя из рисунка, сразу можно увидеть, что часть строк уже непокрыта тестами, а значит – корректность результата выполнения такой функции точно неизвестна.

Так как «Golangci-Lint» предоставляет широкий список [23] линтеров для анализа, среди есть и такие, которые позволяют выдавать сообщения либо ошибки при достижении этими метриками критических значений. Для проверки метрик были использованы следующие линтеры:

- «cyclop», «gosyclo» – проверка цикломатической сложности функций;
- «funlen» – проверка на длину функций;
- «gocognit» – проверка на когнитивную сложность;
- «maintidx» – метрика оценивающая лёгкость поддержки кода;
- «nestif» – проверка на слишком большую вложенность операторов if.

Рассмотрим на примере работу некоторых из них.

«cyclor» – линтер, позволяющий проверить код на цикломатическую сложность. Он умеет считать цикломатическую сложность как отдельных функций, так и целых пакетов. Пример его работы (со специальность заниженной границей срабатывания до 3) представлен на рисунке 5.3.

```
~/Coding/Greg/lawfirm/backend/database/postgres feature/search* ↑ } cyclor -maxComplexity 3 .
/Users/alex/Coding/Greg/lawfirm/backend/database/postgres/filters.go:47:1: calculated cyclomatic
complexity for function Order is 4, max is 3
/Users/alex/Coding/Greg/lawfirm/backend/database/postgres/filters.go:78:1: calculated cyclomatic
complexity for function Search is 4, max is 3
~/Coding/Greg/lawfirm/backend/database/postgres feature/search* ↑ } 03:05:12 PM
```

Рисунок 5.3 – Линтер «cyclor»

Следующий линтер – это «gocognit». Он позволяет оценить когнитивную сложность отдельной функции. Разработчики предоставили примеры подсчётов в своём github-репозитории, которые можно увидеть на рисунках 5.4, 5.5.

```
1 func GetWords(number int) string {
2     if number == 1 {           // +1
3         return "one"
4     } else if number == 2 {     // +1
5         return "a couple"
6     } else if number == 3 {     // +1
7         return "a few"
8     } else {                   // +1
9         return "lots"
10    }
11 } // Cognitive complexity = 4
```

Рисунок 5.4 – Когнитивная сложность

```
1 func SumOfPrimes(max int) int {
2     var total int
3
4     for i := 1; i < max; i++ {   // +1
5         for j := 2; j < i; j++ { // +2 (nesting = 1)
6             if i%j == 0 {       // +3 (nesting = 2)
7                 continue OUT    // +1
8             }
9         }
10        total += i
11    }
12
13    return total
14 } // Cognitive complexity = 7
```

Рисунок 5.5 – Когнитивная вложенная сложность

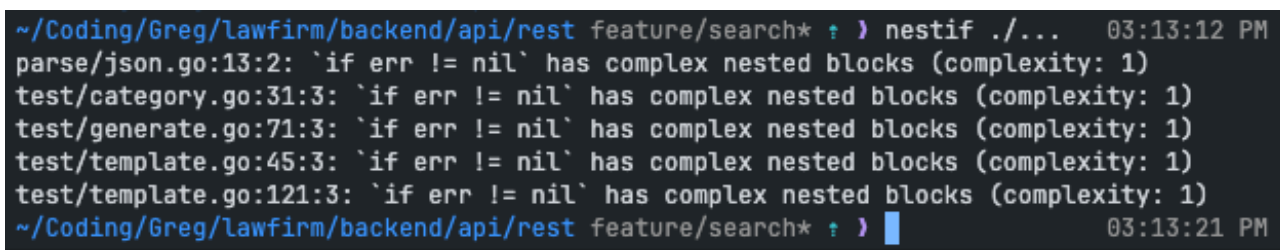
«gocyclo» – это ещё один линтер, который считает цикломатическую сложность. Формула расчёта, следующая: 1 как базовая сложность и 1 за каждый «if», «for», «case», «&&» и «||». Соответственно, результаты могут отличаться от «cyclop», даже при анализе одного и того же кода. Пример работы линтера представлен на рисунке 5.6.



```
~/Coding/Greg/lawfirm/backend/database/postgres feature/search* ↑ } gocyclo -over 4 .
7 postgres (*Client).SaveCase test/case_repo.go:57:1
5 postgres (*Client).SaveDefendant test/defendant_repo.go:54:1
~/Coding/Greg/lawfirm/backend/database/postgres feature/search* ↑ } 03:09:57 PM
```

Рисунок 5.6 – Линтер «gocyclo»

Последний рассматриваемый линтер – «nestif». Это довольно простой линтер, который считает метрику вложенности операторов «if». Пример работы можно увидеть на рисунке 5.7.



```
~/Coding/Greg/lawfirm/backend/api/rest feature/search* ↑ } nestif ./... 03:13:12 PM
parse/json.go:13:2: `if err != nil` has complex nested blocks (complexity: 1)
test/category.go:31:3: `if err != nil` has complex nested blocks (complexity: 1)
test/generate.go:71:3: `if err != nil` has complex nested blocks (complexity: 1)
test/template.go:45:3: `if err != nil` has complex nested blocks (complexity: 1)
test/template.go:121:3: `if err != nil` has complex nested blocks (complexity: 1)
~/Coding/Greg/lawfirm/backend/api/rest feature/search* ↑ } 03:13:21 PM
```

Рисунок 5.7 – Линтер «nestif»

В конечном итоге, результаты всех линтеров импортируются в «Golangci-Lint», и разработчик видит полный отчёт после анализа системы, на основе чего может принимать необходимые решения по исправлению или изменению модулей, если необходимо.

5.2 Контроль качества программного кода

5.2.1 Использование метода покрытие условий и условий

При использовании данного метода необходимо составить такое число тестовых данных, чтобы любые вариации каждого из условий, включая составные, соединённые операторами «и» или «или», выполнялись хотя бы 1 раз.

Прежде чем переходить к составлению данных – необходимо изобразить на блок-схеме все возможные варианты переходов. В качестве тестируемой

функции используем код функции «ParsePlaceholder», уже ранее представленной на рисунках 5.1 и 5.2. Блок-схема переходов представлена на рисунке 5.8.

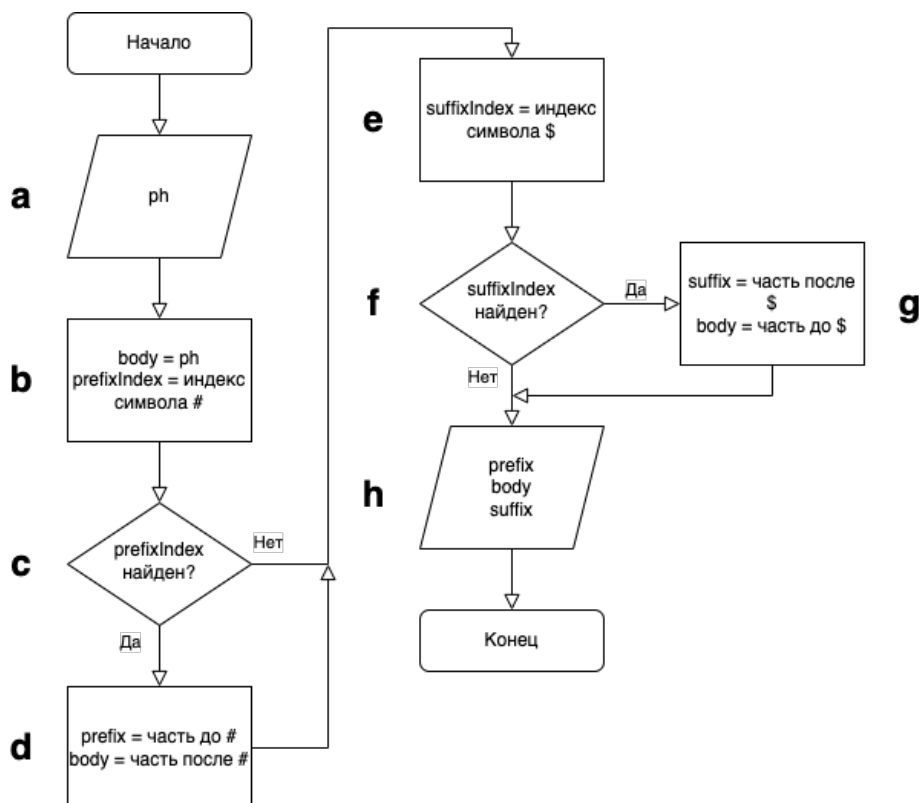


Рисунок 5.8 – Блок-схема функции «ParsePlaceholder»

Для того чтобы покрыть все возможные вариации использования данной функции, необходимо чтобы каждый из блоков «с» и «f» принимал значение истина или ложь хотя бы 1 раз. В соответствии с данными условие был составлен набор тестовых данных, представленный в таблице 5.1.

Таблица 5.1 – Покрывание условий.

№	Решения	Входные данные	Выходные данные
1	2	3	4
1	a, b, c(да), d, e, f(да), g, h	{str1#main\$suff}	prefix=str1 body=main suffix=suff
2	a, b, c(да), d, e, f(нет), h	{str1#main}	prefix=str1 body=main
3	a, b, c(нет), e, f(да), g, h	{main\$suff}	prefix=пустая строка body=main suffix=suff
4	a, b, c(нет), e, f(нет), h	{main}	prefix=пустая строка body=main suffix=пустая строка

Как видно из таблицы тестовых данных, к представленной функции был составлен такой набор тестовых данных, который позволяет проверить все уникальные комбинации сочетаний блоков «с» и «f».

5.2.2 Юнит-тестирование программного кода

Как уже было оговорено, тестирование будет осуществляться с использованием юнит-тестов.

В качестве примера рассмотрим код, тестирующий функцию «ParsePlaceholder», представленной в разделе 5.1.2, которая должна обрабатывать входные данные о переменных подстановки и разбивать их на 3 части – префикс, тело и суффикс. Префикс или суффикс могут отсутствовать, в таком случае на месте их результата должна быть пустая строка. Код представлен на листинге 5.1.

Листинг 5.1 – Пример юнит-теста для функции ParsePlaceholder

```
func TestParsePlaceholder(t *testing.T) {
    t.Parallel()

    type testCase struct {
        input      string
        wantPrefix string
        wantBody   string
        wantSuffix string
    }

    testCases := [...]testCase{
        {"prefix#", "prefix", "", ""},
        {"prefix#body", "prefix", "body", ""},
        {"prefix#body$suffix", "prefix", "body", "suffix"},
        {"body$suffix", "", "body", "suffix"},
        {"body", "", "body", ""},
    }

    for _, testCase := range testCases {
        testCase := testCase
        t.Run(testCase.input, func(t *testing.T) {
            gotPrefix, gotBody, gotSuffix :=
replacer.Parse(testCase.input)

            if gotPrefix != testCase.wantPrefix {
                t.Errorf(...)
            }

            // Похожие операции для тела и суффикса
            // ...
        })
    }
}
```

Прежде всего начнём с сигнатуры и названия тестовой функции. Обязательным критерием является префикс «Test*» и аргумент типа «*testing.T». Так система понимает, что данная функция является тестовой и автоматически будет выполнять её при запуске тестов.

Далее, по усмотрению разработчика, можно указать чтобы функция тестировалась параллельно с другими тестовыми функциями, для этого необходимо вызвать метод «t.Parallel()».

В большинстве случаев разработчики предпочитают использовать именно такой метод, ибо он позволяет ускорить быстродействие выполнения тестов и, иногда, найти ошибки, связанные с гонкой данных (англ. data race) в случае многопоточных программ.

Для выполнения тестов необходимо задать тестовые данные. Обычно для этого создают дополнительную структуру, а исходные данные записывают как массив данных структур.

Далее, в цикле «for» выполняется перебор входных данных, вызов исходной функции, сверка желаемого и реального полученного результата. На основе данной тестовой функции и формируется покрытие кода, описанное в разделе 5.1.2.

Подобный шаблон тестовой функции применяется в проекте повсеместно, поэтому все остальные функции имеют схожую структуру, за исключением проверок и тестовых данных. Данный тип тестирования называют табличными тестами, за счёт того, что тестовые данные размещают в структуре, напоминающую таблицу.

Его удобство в том, что утилита Go для тестов умеет распознавать табличные тесты как отдельные внутри одной тестовой функции и даёт возможность запускать их независимо друг от друга, что позволяет экономить время при отладке и тестировании приложения.

Весь список существующих тестов представлен в таблице 5.2.

Таблица 5.2 – Перечень юнит-тестов

№	Тест	Проверяет функционал
1	2	3
1	TestConvertReplaceMap	Правильность конвертации объекта «ключ-значение», где ключ – переменная, а значение – подставляемое значение
2	TestMakeName	Правильность генерации имени шаблона
3	TestExtractPlaceholders	Корректность извлечения переменных из изначального шаблона
4	TestParsePlaceholder	Корректность обработки синтаксиса переменной
5	TestNotaryDate	Корректность замены суффикса нотариальной даты

На текущем этапе разработки нецелесообразно покрывать тестами весь функционал приложения, так как он с большой вероятностью изменится, соответственно, тесты тоже будет необходимо изменять.

5.2.3 Оценка качества кода с использованием CI\CD технологий

При загрузке данных на сервер контроля версий, для него автоматически запускаются процедуры тестирования, линтинга, и, в случае успеха предыдущих и выполнении дополнительных условий, данная версия кода загружается на рабочий сервер. Данный процесс так же представлен на диаграмме деятельности (рисунок 5.9).

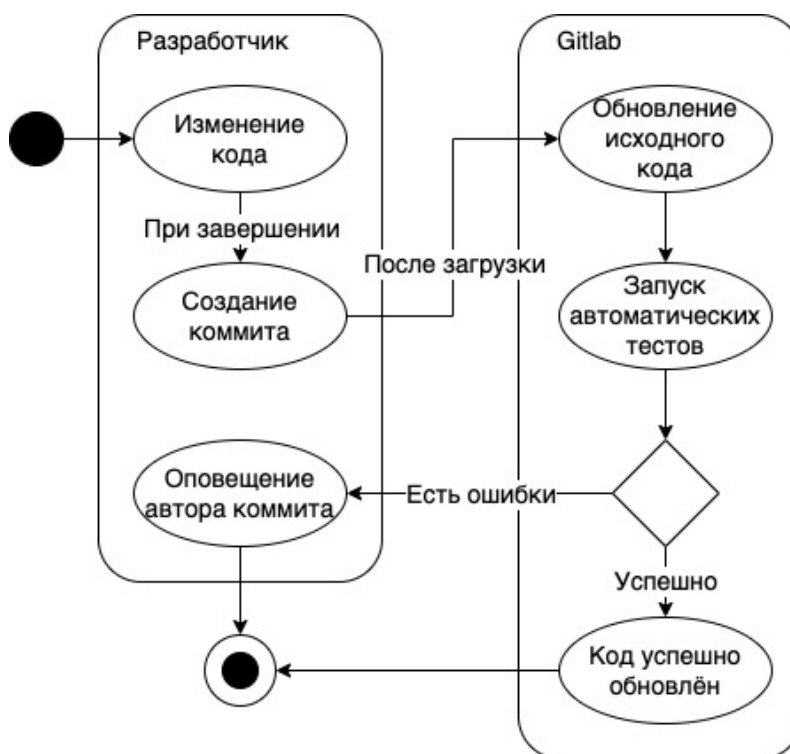


Рисунок 5.9 – CI\CD в системе GitLab

Под загрузкой подразумевается Merge Request (Pull Request) – это процесс, при котором разработчик загружает свои изменения для их оценки другими, но его изменения ещё не загружены в основную кодовую базу. Это позволяет находить и исправлять ошибки ещё до их попадания в реальный код.

Как раз на этом этапе GitLab и запускает все необходимые тесты, если они были созданы и настроены.

В качестве оповещений GitLab использует встроенную систему нотификаций либо обычное письмо на Email адрес пользователя, если необходимо.

Внутри системы разработчик сам может посмотреть статус проверки прямо на странице Merge Request. Пример удачного и неудачного процессов представлен на рисунке 5.10.

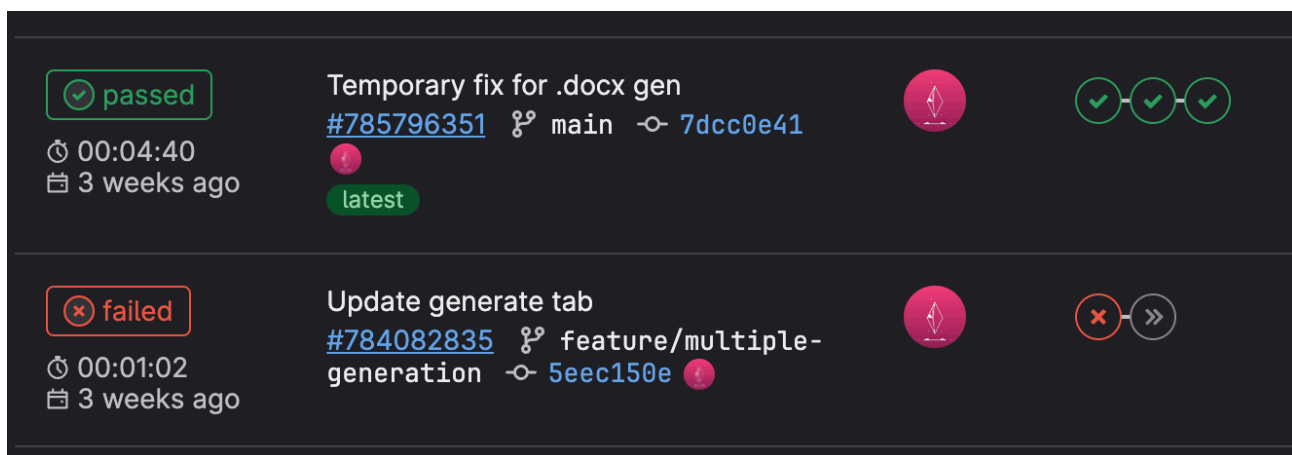


Рисунок 5.10 – CI\CD в системе Gitlab

У каждого процесса есть результат «passed» или «failed», «triggerer» (тот, кто запустил его), и «stages» (этапы, кружки в правой части рисунка). Он подразделяется на 3 стадии: линтинг, тестирование, загрузка на сервер. Проверка происходит именно в таком порядке и может быть изменена разработчиком, а загрузка на сервер осуществляется только при условии, что код был загружен в ветку «main».

Для примера рассмотрим нижний процесс под названием «Update generate tab». Сразу отметим, что названия эти происходят от имени коммита, с которого процессы были вызваны. У данного процесса есть 2 этапа, один из которых

провалился, а второй был пропущен, т. к. необходимости выполнять его уже не было. Рассмотрим причину «провала» процесса на рисунке 5.11.

```

107 generate/replacer/suffix/gender.go:22:7: string `Corporation` has 3 occurrences, make it a constant (gocons
t)
108     case "Corporation":
109         ^
110 models/law/enums/gender/gender.go:9:2: SA9004: only the first constant in this group has an explicit type
(staticcheck)
111     Corporation Gender = "Corporation"
112     ^
✓ 114 Cleaning up project directory and file based variables 00:01
116 ERROR: Job failed: exit code 2

```

Рисунок 5.11 – Провальный этап линтинга

На рисунке видно, что этап был провален, потому что линтер нашёл слишком много ошибок. Для удобства разработчика – каждая из этих ошибок представлена в итоговом выводе, с именем файла, строки и названием ошибки.

Предположим, что стадия линтинга прошла успешно, значит следующая стадия – стадия тестирования. На данном этапе просто вызывается команда протестировать всё, и в случае провала хотя бы одного теста стадия должна провалиться. Пример успешной стадии представлен на рисунке 5.12.

```

84 ?      gitlab.com/patricoski-law/lawfirm/backend/models/law/enums/case/status [no test files]
85 ?      gitlab.com/patricoski-law/lawfirm/backend/models/law/enums/gender   [no test files]
86 ?      gitlab.com/patricoski-law/lawfirm/backend/models/law/enums/treatment [no test files]
✓ 88 Cleaning up project directory and file based variables 00:00
90 Job succeeded

```

Рисунок 5.12 – Успешное прохождение стадии тестирования

На рисунке можно увидеть, что тесты были пройдены успешно, поэтому стадия была успешно пройдена.

Всю информацию об одном конкретном процессе можно посмотреть, если нажать на кнопку «passed» \ «failed». Далее, на рисунке 5.13 показано визуальное представление и порядок стадий.

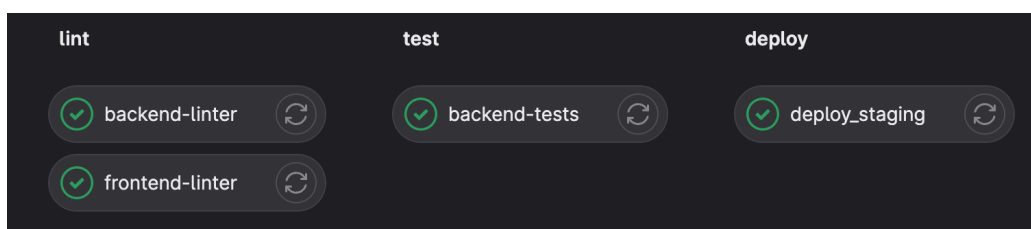


Рисунок 5.13 – Визуальное представление процесса

Вся конфигурация процесса находится в файле «.gitlab-ci.yml» и в упрощённом варианте представлена на листинге 5.2 для ознакомления.

Листинг 5.2 – Файл .gitlab-ci.yml

```
variables:
  GO_IMAGE: golang:1.19-alpine
  GOLINT_IMAGE: golangci/golangci-lint:latest-alpine

stages:
  - lint
  - test
  - deploy

backend-linter:
  image: $GOLINT_IMAGE
  stage: lint
  before_script:
    - cd backend
  script:
    - golangci-lint run ./... --out-format colored-line-number -c
      .golangci.yml
// ...

backend-tests:
  image: $GO_IMAGE
  stage: test
  before_script:
    - apk add gcc musl-dev
    - cd backend
  script:
    - go test -race $(go list ./... | grep -v /vendor/)
//...
```

Данный файл позволяет использовать переменные, определить имена стадий, имена и действия каждого из действий внутри стадии, установить порядок выполнения и зависимости, если необходимо. В данном примере было рассмотрено только по 1 задаче в каждой из стадий, однако в больших проектах количества таких задач, как и самих стадий, может исчисляться десятками, и, в редкий случаях, сотнями.

Заключение

Основной задачей данного дипломного проекта являлось создание, тестирование и ввод в эксплуатацию системы, позволяющую заменить «псевдо-систему», реализованную в виде Google-таблиц для коллегии адвокатов.

На текущей стадии разработки система удовлетворяет следующим требованиям:

- система защищена и изолирована от посторонних пользователей;
- информация передаётся по безопасным каналам связи;
- имеется возможность генерации и кастомизации шаблонов документов;
- система отказоустойчива;
- системой можно пользоваться, практически в полноценном объёме, с мобильных устройств;
- имеется возможность контролировать множественные тонкости делопроизводства, которые прямым образом влияют на качество генерируемых документов.

В рамках дипломного проекта представлен функционал системы, реализованный на момент «апрель 2023 г.». Система продолжает разрабатываться и совершенствоваться. На данный момент присутствуют следующие изменения:

- множественные изменения в способе хранения данных;
- интеграция с Google Calendar;
- система управления (внутренними) пользователями и авторизация;
- система логирования.

Сейчас обсуждается ввод в эксплуатацию системы, как альфа-версии продукта, для более масштабного тестирования.

Таким образом, задачи, поставленные в дипломном проекте – выполнены, а цели – достигнуты.

Во время разработки данного проекта была написана и опубликована статья: «Применение промежуточного хранилища на платформе Redis для повышения скорости выборки данных» в рамках ВНК-2023 «Информационные технологии в науке и образовании. проблемы и перспективы» [15].

В качестве финальной точки данного дипломного проекта была разработана альфа-версия программного продукта, которая тестируется у заказчика

Также, в качестве благодарности, в приложениях Д и Ж представлено рекомендательное письмо на языке оригинала и с заверенным переводом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Patricoski Law Offices // Patricoski Law Offices [Электронный ресурс] URL: <https://www.plolegal.com/> (дата обращения: 01.05.2023)
2. Civil vs. Criminal Case Guide // The Kryder Law Group, LLC [Электронный ресурс] URL: <https://www.kryderlaw.com/blog/civil-vs-criminal-case-guide-the-plaintiff-defendant-and-burden-of-proof/> (дата обращения: 10.01.2023);
3. Law Careers in the US // Shorelight [Электронный ресурс] URL: <https://shorelight.com/student-stories/law-careers-in-the-usa/> (дата обращения: 10.01.2023);
4. What is the difference between public adjusters and attorneys? // Busciglio Sheridan & Schoeb [Электронный ресурс] URL: <https://mytampafirm.com/blog/what-is-the-difference-between-public-adjusters-and-attorneys/> (дата обращения: 10.01.2023);
5. Google Sheets // Google [Электронный ресурс] URL: <https://docs.google.com/spreadsheets/> (дата обращения: 10.01.2023);
6. Clio // Clio, The Industry #1 Legal Software [Электронный ресурс] URL: <https://www.clio.com/> (дата обращения: 14.01.2023);
7. Law Practice Management Software for Attorneys // Smokeball [Электронный ресурс] URL: <https://www.smokeball.com/> (дата обращения: 17.01.2023);
8. Кухноцев П., Разбираем диаграмму Ганта // Skillbox Media [Электронный ресурс] URL: <https://skillbox.ru/media/management/razbiraem-diagrammu-ganta-instrument-kotoryy-dolzhen-znat-kazhdyy-menedzher/> (дата обращения: 01.02.2023);
9. Mermaid, Diagramming and charting tool // Mermaid [Электронный ресурс] URL: <https://mermaid.js.org/> (дата обращения: 08.02.2023);
10. Нотация Мартина (Crow's Foot) // Учебные материалы для студентов [Электронный ресурс] URL:

https://studme.org/77223/informatika/notatsiya_martina_crows_foot (дата обращения: 10.02.2023);

11. Redis // Redis [Электронный ресурс] URL: <https://redis.io/> (дата обращения: 22.02.2023);

12. PostgreSQL // PostgreSQL – The most advanced open-source database [Электронный ресурс] URL: <https://www.postgresql.org/> (дата обращения: 22.02.2023);

13. Docker // Docker [Электронный ресурс] URL: <https://www.docker.com/> (дата обращения: 04.03.2023);

14. Front-end frameworks // State of Javascript 2022 [Электронный ресурс] URL: <https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/> (дата обращения: 08.03.2023);

15. Глазунов, А. С., Применение промежуточного хранилища на платформе Redis для повышения скорости выборки данных / А. С. Глазунов, С. В. Шибанов // Информационные технологии в науке и образовании. Проблемы и перспективы: материалы X Всерос. науч.-техн. конф. – Пенза: Изд-во ПГУ, 2023. – С. 413 – 420;

16. pgModeler // pgModeler [Электронный ресурс] URL: <https://pgmodeler.io/> (дата обращения: 15.03.2023);

17. Арно Л., Проектирование API / Арно Л. – ДМК Пресс, 2020. – 440 с.;

18. Textra // Github [Электронный ресурс] URL: <https://github.com/ravsii/textra> (дата обращения: 15.04.2023);

19. A curated list of awesome Go frameworks // Github [Электронный ресурс] URL: <https://github.com/avelino/awesome-go#go-tools> (дата обращения: 15.04.2023);

20. Compose file version 3 reference // Docker Documentation [Электронный ресурс] URL: <https://docs.docker.com/compose/compose-file/compose-file-v3/> (дата обращения: 10.04.2023);

21. Что такое CI/CD? Разбираемся с непрерывной интеграцией и непрерывной поставкой // Блог компании OPUS, habrahabr [Электронный ресурс]

URL: <https://habr.com/ru/companies/otus/articles/515078/> (дата обращения: 24.03.2023);

22. Introduction // golangci-lint [Электронный ресурс] URL: <https://golangci-lint.run/> (дата обращения: 18.04.2023);

23. Linters // golangci-lint [Электронный ресурс] URL: <https://golangci-lint.run/usage/linters/> (дата обращения: 24.04.2023).

Приложение А
Глоссарий
(справочное)

Adjuster – (юридический) консультант – прикреплен к ответчику, отвечает за сбор информации о материальном и нематериальном ущербе клиента (ответчика).

Attorney – прокурор – прикреплен к ответчику, отвечает за представление ответчика перед судом и управляет поступающей и исходящей информацией от и к ответчику, в том числе и приватной.

API – Application Programming Interface – Набор интерфейсов, который предоставляют разработчики для взаимодействия с их системой или сервисами.

Backend – Серверная часть – Часть приложения, отвечающая за бизнес-логику, обработку и хранение данных, обеспечение взаимодействия с базой данных.

Carrier – (правовой) носитель – некий аналог страховой компании для ответчика, который предоставляет консультанты и прокурора в случае, если он не способен (материально) или не желает находить их сам.

Case – (судебное) дело – иногда называют судебным иском.

CI/CD – Continuous Integration \ Continuous Deployment – Непрерывная интеграция и непрерывное развертывание. Это набор операций и процессов, которые используются для автоматизации тестирования, сборки и развертывания приложений.

Client – клиент – Клиент, которого компания защищает в рамках одного дела. Может быть несколько в рамках одного дела. Пример – семейная пара.

County – округ – географический округ, к которому прикреплен клиент и в котором проводится судебный процесс. В отличие от нашего судебного процесса, там законы в разных округах могут варьироваться.

Cloud – Облако – набор технологий и инфраструктуры, которые предоставляют вычислительные ресурсы и хранилище данных как услуги.

Db Visualizer – Инструмент для работы с базами данных.

Defendant – ответчик – Основной клиент противоположной стороны, к которому предъявляется иск. Им может быть как физическое, так и юридическое лицо. В некоторых случаях возможно наличие более чем одного ответчика.

Docker – инструмент для контейнеризации приложений.

Frontend – Клиентская часть – часть приложения, отвечающая за интерфейс и взаимодействие с пользователем.

JSON – формат обмена данными, широко используемый в разработке.

MongoDB – Документно-ориентированная NoSQL СУБД, которая хранит данные в JSON формате.

Postgres – реляционная СУБД с открытым исходным кодом.

REST API – Representational State Transfer API – Набор спецификаций и стандартов для обмена данными на основе протокола HTTP.

Redis – СУБД с открытым исходным кодом, которая работает в оперативной памяти, за счёт чего гораздо быстрее обычных СУБД.

SaaS – Software as a Service – Программное обеспечение как услуга, модель предоставления при которой приложение предоставляет доступ как услуга через интернет.

Wizard / Placeholder Wizard – Помощник – Внутренняя система, позволяющая заказчику легко находить и использовать переменные шаблонов

Workflow – Рабочий процесс – Последовательность шагов и действий, которые определяют процесс выполнения работы.

Boilerplate Code – фрагменты кода, которые очень часто повторяются и выполняют практически идентичные задачи.

Endpoint – Конечная точка – Адрес веб-сервиса, на который необходимо зайти, чтобы получить ответ или отправить запрос.

gRPC – Google's Remote Procedure Call – Высокопроизводительный фреймворк для удалённого вызова процедур.

GraphQL – язык запросов для получения данных с сервера. Отличительной особенностью являются возможно указания необходимых полей.

Middleware – компонент программы, который находится между клиентским и серверным приложением.

pgAdmin – популярный веб-интерфейс для работы с СУБД Postgres.

pgModeler – инструмент для визуального проектирования СУБД Postgres.

Proxy – промежуточный сервер, который выполняет запросы от имени другого сервера, позволяя сохранить анонимность.

СУБД – система управления базами данных.

ПО – Программное обеспечение.

Фреймворк – Набор инструментов, библиотек и стандартов который облегчает разработку приложения, обеспечивая решения типовых проблем.

ЯП – Язык программирования.

Приложение Б
Скрипт создания базы данных
(рекомендуемое)

Листинг Б.1 – Скрипт создания базы данных

```
-- Database generated with pgModeler (PostgreSQL Database Modeler).
-- pgModeler version: 1.1.0-alpha
-- PostgreSQL version: 15.0
-- Project Site: pgmodeler.io
-- Model Author: ---

-- Database creation must be performed outside a multi lined SQL file.
-- These commands were put in this file only as a convenience.
--
-- object: law_firm | type: DATABASE --
-- DROP DATABASE IF EXISTS law_firm;
CREATE DATABASE law_firm
    ENCODING = 'UTF8'
    LC_COLLATE = 'en_US.utf8'
    LC_CTYPE = 'en_US.utf8'
    TABLESPACE = pg_default;
-- ddl-end --

-- object: law | type: SCHEMA --
-- DROP SCHEMA IF EXISTS law CASCADE;
CREATE SCHEMA law;
-- ddl-end --

-- object: documents | type: SCHEMA --
-- DROP SCHEMA IF EXISTS documents CASCADE;
CREATE SCHEMA documents;
-- ddl-end --

SET search_path TO pg_catalog,public,law,documents;
-- ddl-end --

-- object: law.gender | type: TYPE --
-- DROP TYPE IF EXISTS law.gender CASCADE;
CREATE TYPE law.gender AS
ENUM ('Male','Female','Corporation');
-- ddl-end --

-- object: law.treatment_status | type: TYPE --
-- DROP TYPE IF EXISTS law.treatment_status CASCADE;
CREATE TYPE law.treatment_status AS
ENUM ('Treating','Done Treating','No Treatment','Unknown');
-- ddl-end --

-- object: law.county_enum | type: TYPE --
-- DROP TYPE IF EXISTS law.county_enum CASCADE;
CREATE TYPE law.county_enum AS
ENUM ('Cook County','DeKalb','DuPage','Kane','Kendall','Lake','Will');
-- ddl-end --

-- object: law.adjuster | type: TABLE --
-- DROP TABLE IF EXISTS law.adjuster CASCADE;
CREATE TABLE law.adjuster (
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT BY 1 MINVALUE
1 MAXVALUE 9223372036854775807 START WITH 1 CACHE 1 ),
    first_name character varying NOT NULL,
    last_name character varying NOT NULL,
    email character varying,
    fax character varying,
    phone_number character varying,
```

```

        notes text,
        address_line1 character varying,
        address_line2 character varying,
        address_city character varying,
        address_state character varying,
        address_zipcode character varying,
        address_county law.county_enum,
        created_at timestamp NOT NULL DEFAULT now(),
        edited_at timestamp NOT NULL DEFAULT now(),
        CONSTRAINT adjuster_pk PRIMARY KEY (id)
    );
-- ddl-end --

-- -- object: law.adjuster_id_seq | type: SEQUENCE --
-- -- DROP SEQUENCE IF EXISTS law.adjuster_id_seq CASCADE;
-- CREATE SEQUENCE law.adjuster_id_seq
--     INCREMENT BY 1
--     MINVALUE 1
--     MAXVALUE 9223372036854775807
--     START WITH 1
--     CACHE 1
--     NO CYCLE
--     OWNED BY NONE;
--
-- -- ddl-end --
--
-- object: law.defendant | type: TABLE --
-- DROP TABLE IF EXISTS law.defendant CASCADE;
CREATE TABLE law.defendant (
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT BY 1 MINVALUE
1 MAXVALUE 9223372036854775807 START WITH 1 CACHE 1 ),
    first_name character varying NOT NULL,
    last_name character varying NOT NULL,
    gender law.gender,
    married boolean NOT NULL DEFAULT false,
    phone_number character varying,
    email character varying,
    carrier_claim_number character varying,
    carrier_coverage_limits character varying,
    carrier_policy_number character varying,
    carrier_lien_sent boolean NOT NULL DEFAULT false,
    carrier_lien_sent_date timestamp,
    carrier_tracking_number character varying,
    notes text,
    address_line1 character varying,
    address_line2 character varying,
    address_city character varying,
    address_state character varying,
    address_zipcode character varying,
    address_county law.county_enum,
    pd_first_name character varying,
    pd_last_name character varying,
    pd_phone_number character varying,
    pd_email character varying,
    pd_fax character varying,
    pd_claim_number character varying,
    pd_notes text,
    medpay_first_name character varying,
    medpay_last_name character varying,
    medpay_phone_number character varying,
    medpay_email character varying,
    medpay_fax character varying,
    medpay_claim_number character varying,

```

```

        medpay_lien_sent boolean NOT NULL DEFAULT false,
        medpay_lien_sent_date timestamp,
        medpay_notes text,
        created_at timestamp NOT NULL DEFAULT now(),
        edited_at timestamp NOT NULL DEFAULT now(),
        carrier_id integer,
        CONSTRAINT defendant_pk PRIMARY KEY (id)
    );
-- ddl-end --

-- -- object: law.defendant_id_seq | type: SEQUENCE --
-- -- DROP SEQUENCE IF EXISTS law.defendant_id_seq CASCADE;
-- CREATE SEQUENCE law.defendant_id_seq
--     INCREMENT BY 1
--     MINVALUE 1
--     MAXVALUE 9223372036854775807
--     START WITH 1
--     CACHE 1
--     NO CYCLE
--     OWNED BY NONE;
--
-- -- ddl-end --
--
-- object: law.attorney | type: TABLE --
-- DROP TABLE IF EXISTS law.attorney CASCADE;
CREATE TABLE law.attorney (
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT BY 1 MINVALUE
1 MAXVALUE 9223372036854775807 START WITH 1 CACHE 1 ),
    first_name character varying NOT NULL,
    last_name character varying NOT NULL,
    email character varying,
    fax character varying,
    phone_number character varying,
    notes text,
    address_line1 character varying,
    address_line2 character varying,
    address_line3 character varying,
    address_city character varying,
    address_state character varying,
    address_zipcode character varying,
    address_county law.county_enum,
    created_at timestamp NOT NULL DEFAULT now(),
    edited_at timestamp NOT NULL DEFAULT now(),
    CONSTRAINT attorney_pk PRIMARY KEY (id)
);
-- ddl-end --

-- -- object: law.attorney_id_seq | type: SEQUENCE --
-- -- DROP SEQUENCE IF EXISTS law.attorney_id_seq CASCADE;
-- CREATE SEQUENCE law.attorney_id_seq
--     INCREMENT BY 1
--     MINVALUE 1
--     MAXVALUE 9223372036854775807
--     START WITH 1
--     CACHE 1
--     NO CYCLE
--     OWNED BY NONE;
--
-- -- ddl-end --
--
-- object: law.sign_up_status | type: TYPE --
-- DROP TYPE IF EXISTS law.sign_up_status CASCADE;
CREATE TYPE law.sign_up_status AS

```

```

ENUM ('Under Evaluation','Rejected','Sign Up Pending','Signed Up');
-- ddl-end --

-- object: law.police_report_status | type: TYPE --
-- DROP TYPE IF EXISTS law.police_report_status CASCADE;
CREATE TYPE law.police_report_status AS
ENUM ('In File','Too Soon','Ordered','Client To Send In','No');
-- ddl-end --

-- object: law.case_status | type: TYPE --
-- DROP TYPE IF EXISTS law.case_status CASCADE;
CREATE TYPE law.case_status AS
ENUM ('Not In Suit','In Suit','Settled - Disbursement Pending','Settled -
Partially Disbursed','Settled - Fully Disbursed');
-- ddl-end --

-- object: law.property_damage | type: TYPE --
-- DROP TYPE IF EXISTS law.property_damage CASCADE;
CREATE TYPE law.property_damage AS
ENUM ('Total','Serious','Limited','None');
-- ddl-end --
COMMENT ON TYPE law.property_damage IS 'property';
-- ddl-end --

-- object: law.case_type | type: TYPE --
-- DROP TYPE IF EXISTS law.case_type CASCADE;
CREATE TYPE law.case_type AS
ENUM ('WC','UM','UIM','Third Party','Homeowners','Renters');
-- ddl-end --

-- object: law.case_priority | type: TYPE --
-- DROP TYPE IF EXISTS law.case_priority CASCADE;
CREATE TYPE law.case_priority AS
ENUM ('N/A','Pro
Bono','Hourly','$5,000+','$25,000','$50,000','$100,000','$250,000','$1,000,000+'
);
-- ddl-end --

-- object: law.client | type: TABLE --
-- DROP TABLE IF EXISTS law.client CASCADE;
CREATE TABLE law.client (
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT BY 1 MINVALUE
1 MAXVALUE 9223372036854775807 START WITH 1 CACHE 1 ),
    first_name character varying NOT NULL,
    middle_name character varying,
    last_name character varying NOT NULL,
    gender law.gender,
    married boolean NOT NULL DEFAULT false,
    phone_number character varying,
    email character varying,
    date_of_birth date,
    treatment_status law.treatment_status,
    ssn character varying,
    martial_info character varying,
    family_contact_info character varying,
    related_files character varying,
    insurance character varying,
    health_card boolean DEFAULT false,
    medicare boolean DEFAULT false,
    medicare_card boolean DEFAULT false,
    medicare_notes text,
    physical_address_line1 character varying,
    physical_address_line2 character varying,

```

```

    physical_address_city character varying,
    physical_address_state character varying,
    physical_address_zipcode character varying,
    physical_address_county law.county_enum,
    mailing_address_line1 character varying,
    mailing_address_line2 character varying,
    mailing_address_city character varying,
    mailing_address_state character varying,
    mailing_address_zipcode character varying,
    mailing_address_county law.county_enum,
    created_at timestamp NOT NULL DEFAULT now(),
    edited_at timestamp NOT NULL DEFAULT now(),
    CONSTRAINT clients_pk PRIMARY KEY (id)
);
-- ddl-end --

-- -- object: law.client_id_seq | type: SEQUENCE --
-- -- DROP SEQUENCE IF EXISTS law.client_id_seq CASCADE;
-- CREATE SEQUENCE law.client_id_seq
--     INCREMENT BY 1
--     MINVALUE 1
--     MAXVALUE 9223372036854775807
--     START WITH 1
--     CACHE 1
--     NO CYCLE
--     OWNED BY NONE;
--
-- -- ddl-end --
--
-- object: documents.template | type: TABLE --
-- DROP TABLE IF EXISTS documents.template CASCADE;
CREATE TABLE documents.template (
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT BY 1 MINVALUE
1 MAXVALUE 9223372036854775807 START WITH 1 CACHE 1 ),
    name character varying NOT NULL,
    id_category integer NOT NULL,
    created_at timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    edited_at timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT template_pk PRIMARY KEY (id)
);
-- ddl-end --

-- -- object: documents.template_id_seq | type: SEQUENCE --
-- -- DROP SEQUENCE IF EXISTS documents.template_id_seq CASCADE;
-- CREATE SEQUENCE documents.template_id_seq
--     INCREMENT BY 1
--     MINVALUE 1
--     MAXVALUE 9223372036854775807
--     START WITH 1
--     CACHE 1
--     NO CYCLE
--     OWNED BY NONE;
--
-- -- ddl-end --
--
-- object: documents.category | type: TABLE --
-- DROP TABLE IF EXISTS documents.category CASCADE;
CREATE TABLE documents.category (
    id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT BY 1 MINVALUE
1 MAXVALUE 2147483647 START WITH 1 CACHE 1 ),
    name character varying NOT NULL,
    id_parent integer,
    created_at timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,

```

```

        edited_at timestamp DEFAULT CURRENT_TIMESTAMP,
        CONSTRAINT category_pk PRIMARY KEY (id)
    );
-- ddl-end --

-- -- object: documents.category_id_seq | type: SEQUENCE --
-- -- DROP SEQUENCE IF EXISTS documents.category_id_seq CASCADE;
-- CREATE SEQUENCE documents.category_id_seq
--     INCREMENT BY 1
--     MINVALUE 1
--     MAXVALUE 2147483647
--     START WITH 1
--     CACHE 1
--     NO CYCLE
--     OWNED BY NONE;
--
-- -- ddl-end --
--
-- object: law.carrier_id_seq | type: SEQUENCE --
-- DROP SEQUENCE IF EXISTS law.carrier_id_seq CASCADE;
CREATE SEQUENCE law.carrier_id_seq
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 2147483647
    START WITH 1
    CACHE 1
    NO CYCLE
    OWNED BY NONE;

-- ddl-end --

-- object: law.carrier | type: TABLE --
-- DROP TABLE IF EXISTS law.carrier CASCADE;
CREATE TABLE law.carrier (
    id integer NOT NULL DEFAULT nextval('law.carrier_id_seq'::regclass),
    name character varying NOT NULL,
    notes text,
    created_at timestamp NOT NULL DEFAULT now(),
    edited_at timestamp NOT NULL DEFAULT now(),
    CONSTRAINT carrier_pk PRIMARY KEY (id)
);
-- ddl-end --

-- object: law.county_id_seq | type: SEQUENCE --
-- DROP SEQUENCE IF EXISTS law.county_id_seq CASCADE;
CREATE SEQUENCE law.county_id_seq
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 2147483647
    START WITH 1
    CACHE 1
    NO CYCLE
    OWNED BY NONE;

-- ddl-end --

-- object: law.county | type: TABLE --
-- DROP TABLE IF EXISTS law.county CASCADE;
CREATE TABLE law.county (
    id integer NOT NULL DEFAULT nextval('law.county_id_seq'::regclass),
    name character varying NOT NULL,
    caption1 character varying,
    caption2 character varying,

```

```

jurisdictional character varying,
state character varying,
division character varying,
law character varying,
address_house character varying,
address_line1 character varying,
address_line2 character varying,
address_city character varying,
address_state character varying,
address_zipcode character varying,
created_at timestamp NOT NULL DEFAULT now(),
edited_at timestamp NOT NULL DEFAULT now(),
CONSTRAINT county_pk PRIMARY KEY (id)
);
-- ddl-end --

-- object: law.defendant_to_adjuster | type: TABLE --
-- DROP TABLE IF EXISTS law.defendant_to_adjuster CASCADE;
CREATE TABLE law.defendant_to_adjuster (
    defendant_id bigint NOT NULL,
    adjuster_id bigint NOT NULL,
    linked_at timestamp NOT NULL DEFAULT now(),
    CONSTRAINT defendant_to_adjuster_pk PRIMARY KEY (defendant_id,adjuster_id)
);
-- ddl-end --

-- object: law.defendant_to_attorney | type: TABLE --
-- DROP TABLE IF EXISTS law.defendant_to_attorney CASCADE;
CREATE TABLE law.defendant_to_attorney (
    defendant_id bigint NOT NULL,
    attorney_id bigint NOT NULL,
    linked_at timestamp NOT NULL DEFAULT now(),
    CONSTRAINT defendant_to_attorney_pk PRIMARY KEY (defendant_id,attorney_id)
);
-- ddl-end --

-- object: law."case" | type: TABLE --
-- DROP TABLE IF EXISTS law."case" CASCADE;
CREATE TABLE law."case" (
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT BY 1 MINVALUE
1 MAXVALUE 9223372036854775807 START WITH 1 CACHE 1 ),
    main_client_id bigint,
    closed boolean NOT NULL DEFAULT false,
    closed_reason character varying,
    date_of_accident date,
    statute_years smallint DEFAULT 0,
    minor boolean NOT NULL DEFAULT false,
    ref_attorney_fee character varying,
    date_of_referral date,
    prior_attorney character varying,
    ref_non_attorney character varying,
    priority law.case_priority,
    status law.case_status,
    sign_up_status law.sign_up_status,
    police_report_status law.police_report_status,
    officer_info character varying,
    liability smallint DEFAULT 0,
    property_damage law.property_damage,
    type law.case_type,
    coverage_value character varying,
    notes text,
    caller_first_name character varying,
    caller_last_name character varying,

```

```

        caller_phone_number character varying,
        caller_email character varying,
        caller_relationship character varying,
        court_case_number character varying,
        court_county_id integer,
        court_date_filed date,
        court_judge character varying,
        court_room character varying,
        court_notes text,
        created_at timestamp NOT NULL DEFAULT now(),
        edited_at timestamp NOT NULL DEFAULT now(),
        closed_at timestamp,
        CONSTRAINT case_pk PRIMARY KEY (id)
    );
-- ddl-end --

-- -- object: law.case_id_seq | type: SEQUENCE --
-- -- DROP SEQUENCE IF EXISTS law.case_id_seq CASCADE;
-- CREATE SEQUENCE law.case_id_seq
--     INCREMENT BY 1
--     MINVALUE 1
--     MAXVALUE 9223372036854775807
--     START WITH 1
--     CACHE 1
--     NO CYCLE
--     OWNED BY NONE;
--
-- -- ddl-end --
--
-- object: law.case_to_client | type: TABLE --
-- DROP TABLE IF EXISTS law.case_to_client CASCADE;
CREATE TABLE law.case_to_client (
    case_id bigint NOT NULL,
    client_id bigint NOT NULL,
    CONSTRAINT case_to_client_pk PRIMARY KEY (case_id,client_id)
);
-- ddl-end --

-- object: law.case_to_defendant | type: TABLE --
-- DROP TABLE IF EXISTS law.case_to_defendant CASCADE;
CREATE TABLE law.case_to_defendant (
    case_id bigint NOT NULL,
    defendant_id bigint NOT NULL,
    CONSTRAINT case_to_defendant_pk PRIMARY KEY (case_id,defendant_id)
);
-- ddl-end --

-- object: carrier_fk | type: CONSTRAINT --
-- ALTER TABLE law.defendant DROP CONSTRAINT IF EXISTS carrier_fk CASCADE;
ALTER TABLE law.defendant ADD CONSTRAINT carrier_fk FOREIGN KEY (carrier_id)
REFERENCES law.carrier (id) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;
-- ddl-end --

-- object: category_fk | type: CONSTRAINT --
-- ALTER TABLE documents.template DROP CONSTRAINT IF EXISTS category_fk CASCADE;
ALTER TABLE documents.template ADD CONSTRAINT category_fk FOREIGN KEY
(id_category)
REFERENCES documents.category (id) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;
-- ddl-end --

-- object: id_parent_fk | type: CONSTRAINT --

```

```

-- ALTER TABLE documents.category DROP CONSTRAINT IF EXISTS id_parent_fk
CASCADE;
ALTER TABLE documents.category ADD CONSTRAINT id_parent_fk FOREIGN KEY
(id_parent)
REFERENCES documents.category (id) MATCH SIMPLE
ON DELETE SET NULL ON UPDATE CASCADE;
-- ddl-end --

-- object: defendant_fk | type: CONSTRAINT --
-- ALTER TABLE law.defendant_to_adjuster DROP CONSTRAINT IF EXISTS defendant_fk
CASCADE;
ALTER TABLE law.defendant_to_adjuster ADD CONSTRAINT defendant_fk FOREIGN KEY
(defendant_id)
REFERENCES law.defendant (id) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;
-- ddl-end --

-- object: adjuster_fk | type: CONSTRAINT --
-- ALTER TABLE law.defendant_to_adjuster DROP CONSTRAINT IF EXISTS adjuster_fk
CASCADE;
ALTER TABLE law.defendant_to_adjuster ADD CONSTRAINT adjuster_fk FOREIGN KEY
(adjuster_id)
REFERENCES law.adjuster (id) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;
-- ddl-end --

-- object: defendant_fk | type: CONSTRAINT --
-- ALTER TABLE law.defendant_to_attorney DROP CONSTRAINT IF EXISTS defendant_fk
CASCADE;
ALTER TABLE law.defendant_to_attorney ADD CONSTRAINT defendant_fk FOREIGN KEY
(defendant_id)
REFERENCES law.defendant (id) MATCH FULL
ON DELETE CASCADE ON UPDATE CASCADE;
-- ddl-end --

-- object: attorney_fk | type: CONSTRAINT --
-- ALTER TABLE law.defendant_to_attorney DROP CONSTRAINT IF EXISTS attorney_fk
CASCADE;
ALTER TABLE law.defendant_to_attorney ADD CONSTRAINT attorney_fk FOREIGN KEY
(attorney_id)
REFERENCES law.attorney (id) MATCH FULL
ON DELETE CASCADE ON UPDATE CASCADE;
-- ddl-end --

-- object: county_fk | type: CONSTRAINT --
-- ALTER TABLE law."case" DROP CONSTRAINT IF EXISTS county_fk CASCADE;
ALTER TABLE law."case" ADD CONSTRAINT county_fk FOREIGN KEY (court_county_id)
REFERENCES law.county (id) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;
-- ddl-end --

-- object: client_fk | type: CONSTRAINT --
-- ALTER TABLE law."case" DROP CONSTRAINT IF EXISTS client_fk CASCADE;
ALTER TABLE law."case" ADD CONSTRAINT client_fk FOREIGN KEY (main_client_id)
REFERENCES law.client (id) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;
-- ddl-end --

-- object: case_fk | type: CONSTRAINT --
-- ALTER TABLE law.case_to_client DROP CONSTRAINT IF EXISTS case_fk CASCADE;
ALTER TABLE law.case_to_client ADD CONSTRAINT case_fk FOREIGN KEY (case_id)
REFERENCES law."case" (id) MATCH FULL
ON DELETE CASCADE ON UPDATE CASCADE;

```

```

-- ddl-end --

-- object: client_fk | type: CONSTRAINT --
-- ALTER TABLE law.case_to_client DROP CONSTRAINT IF EXISTS client_fk CASCADE;
ALTER TABLE law.case_to_client ADD CONSTRAINT client_fk FOREIGN KEY (client_id)
REFERENCES law.client (id) MATCH FULL
ON DELETE CASCADE ON UPDATE CASCADE;
-- ddl-end --

-- object: case_fk | type: CONSTRAINT --
-- ALTER TABLE law.case_to_defendant DROP CONSTRAINT IF EXISTS case_fk CASCADE;
ALTER TABLE law.case_to_defendant ADD CONSTRAINT case_fk FOREIGN KEY (case_id)
REFERENCES law."case" (id) MATCH FULL
ON DELETE CASCADE ON UPDATE CASCADE;
-- ddl-end --

-- object: defendant_fk | type: CONSTRAINT --
-- ALTER TABLE law.case_to_defendant DROP CONSTRAINT IF EXISTS defendant_fk
CASCADE;
ALTER TABLE law.case_to_defendant ADD CONSTRAINT defendant_fk FOREIGN KEY
(defendant_id)
REFERENCES law.defendant (id) MATCH FULL
ON DELETE CASCADE ON UPDATE CASCADE;
-- ddl-end --

-- object: "grant_U_cd8e46e7b6" | type: PERMISSION --
GRANT USAGE
    ON SCHEMA public
    TO PUBLIC;
-- ddl-end -

```

Приложение В
Исходный код генерации документов
(рекомендуемое)

Листинг В.1 – generate/docx/generate.go

```
package docx

import (
    "archive/zip"
    "bytes"
    "errors"
    "fmt"
    "io"

    libDocx "github.com/lukasjarosch/go-docx"
    "gitlab.com/patricoski-law/lawfirm/backend/generate/replacer"
)

const docXMLPath = "word/document.xml"

var ErrNilTemplate = errors.New("template can't be nil")

type Document struct {
    Name string
    File []byte
    Size int
}

// Generate accepts a defined replaced with sources and generates a document.
func Generate(replacer *replacer.Replacer) (*Document, error) {
    if replacer.Template == nil {
        return nil, ErrNilTemplate
    }

    templateBytes, err := replacer.Template.File()
    if err != nil {
        return nil, fmt.Errorf("can't get template file: %w", err)
    }

    tplLibDocx, err := libDocx.OpenBytes(templateBytes)
    if err != nil {
        return nil, fmt.Errorf("open bytes: %w", err)
    }

    placeholders, err := ExtractPlaceholders(templateBytes, tplLibDocx)
    if err != nil {
        return nil, fmt.Errorf("extract placeholders: %w", err)
    }

    mm, err := replacer.MapFor(placeholders)
    if err != nil {
        return nil, fmt.Errorf("make replace map: %w", err)
    }

    err = tplLibDocx.ReplaceAll(ConvertReplaceMap(mm))
    if err != nil {
        return nil, fmt.Errorf("replace: %w", err)
    }

    var generatedFile bytes.Buffer

    err = tplLibDocx.Write(&generatedFile)
    if err != nil {
        return nil, fmt.Errorf("write: %w", err)
    }
}
```

```

        return &Document{
            Name: MakeName(replacer),
            File: generatedFile.Bytes(),
            Size: generatedFile.Len(),
        }, nil
    }

    // ConvertReplaceMap converts replacer's ReplaceMap to libDocx.Placeholder map.
    func ConvertReplaceMap(m replacer.ReplaceMap) libDocx.PlaceholderMap {
        phmap := make(libDocx.PlaceholderMap, len(m))
        for k, v := range m {
            phmap[k] = v
        }

        return phmap
    }

    // MakeName returns a name for a template for a given request.
    func MakeName(replacer *replacer.Replacer) string {
        if replacer.Client != nil {
            return fmt.Sprintf("%s - %s.docx", replacer.Client.BackwardsName(),
                replacer.Template.Name)
        }

        return fmt.Sprintf("[%d] %s.docx", replacer.Case.ID,
            replacer.Template.Name)
    }

    // ExtractPlaceholders parses a file of a given template and returns found
    placeholders
    // as a string slice.
    func ExtractPlaceholders(tplBytes []byte, doc *libDocx.Document) ([]string,
        error) {
        zipReader, err := zip.NewReader(bytes.NewReader(tplBytes),
            int64(len(tplBytes)))
        if err != nil {
            return nil, fmt.Errorf("zip reader: %w", err)
        }

        var xmlBuf bytes.Buffer

        for _, file := range zipReader.File {
            if file.Name == docXMLPath {
                rc, err := file.Open()
                if err != nil {
                    return nil, fmt.Errorf("xml can't open: %w", err)
                }

                _, err = io.CopyN(&xmlBuf, rc,
                    file.FileHeader.FileInfo().Size())
                if err != nil {
                    return nil, fmt.Errorf("copy: %w", err)
                }

                rc.Close()
            }
        }

        needToReplace := make([]string, 0, len(doc.Placeholders()))

        for _, ph := range doc.Placeholders() {
            phWithBrackets := ph.Text(xmlBuf.Bytes())

```

```

        phNoBrackets := phWithBrackets[1 : len(phWithBrackets)-1]
        needToReplace = append(needToReplace, phNoBrackets)
    }

    return needToReplace, nil
}

```

Листинг В.2 – generate/replacer/suffix/suffix.go

```

package suffix

import (
    "errors"
    "fmt"
    "strings"
    "time"
)

// MaxChainLength is the maximum amount of chained suffixes parsed.
//
// 2 would mean that in {...$uppercase|uppercase|uppercase}
// only the first 2 would be parsed and the rest will be ignored.
//
// The final string would appear as if we'd only use 2 "uppercase" suffixes.
const MaxChainLength = 5

var ErrUnknownSuffix = errors.New("unknown suffix")

type Params struct {
    Married bool
}

// Parse mutate and returns a string depending on the suffix.
func Parse(str, suffix string, p *Params) (string, error) {
    // If no suffix is present, we'd still try to parse it as a time into the
    // US format.
    if suffix == "" {
        // If no error occurred then its time.Time
        if t, err := time.Parse(TimeFormat, str); err == nil {
            return t.Format(DateLayoutUS), nil
        }

        return str, nil
    }

    suffixes := strings.Split(suffix, "|")

    if len(suffixes) > MaxChainLength {
        suffixes = suffixes[:MaxChainLength]
    }

    var err error

    for _, s := range suffixes {
        str, err = parsePlaceholder(str, s, p)
        if err != nil {
            return "", fmt.Errorf("parse placeholder: %w", err)
        }
    }

    return str, nil
}

```

```

func parsePlaceholder(str, suffix string, p *Params) (string, error) {
    switch suffix {
    case "US", "notary":
        return parseTime(str, suffix)
    case "uppercase", "lowercase", "capitalize":
        return parseString(str, suffix), nil
    case "pronoun", "possessive", "salutation":
        return parseGender(str, suffix, p.Married), nil
    default:
        return str, ErrUnknownSuffix
    }
}

func parseString(str, suffix string) string {
    switch suffix {
    case "uppercase":
        return strings.ToUpper(str)
    case "lowercase":
        return strings.ToLower(str)
    case "capitalize":
        return strings.Title(str) //nolint:staticcheck
    default:
        return str
    }
}

const (
    DateLayoutUS = "January 2, 2006"
    TimeFormat    = "2006-01-02 15:04:05 -0700 MST"
)

func parseTime(str, suffix string) (string, error) {
    t, err := time.Parse(TimeFormat, str)
    if err != nil {
        return "", fmt.Errorf("time parse (%s): %w", str, err)
    }

    switch suffix {
    case "US":
        return t.Format(DateLayoutUS), nil
    case "notary":
        return NotaryDate(t), nil
    default:
        return str, nil
    }
}

// NotaryDateSpecified returns a given time in a "this 22nd day of March, 2022"
// format.
func NotaryDate(t time.Time) string {
    var ending string

    // 11, 12, 13 have to be ...th
    if d := t.Day(); d < 11 || d > 13 {
        switch d % 10 {
        case 1:
            ending = "st"
        case 2:
            ending = "nd"
        case 3:
            ending = "rd"
        default:
            ending = "th"
        }
    }
    return t.Format("this " + ending + " day of March, 2022")
}

```

```

        ending = "th"
    }
} else {
    ending = "th"
}

return fmt.Sprintf("this %d%s day of %s, %d", t.Day(), ending,
t.Month().String(), t.Year())
}

```

Листинг В.3 – generate/replacer/extract.go

```

package replacer

import (
    "fmt"
    "reflect"

    "github.com/ravsii/textra"
)

// GetValueByJSONTag tries to find a field with a tag `json: jsonTag`.
func GetValueByJSONTag(obj any, jsonTag string) (value string, found bool, err
error) {
    fields := textra.Extract(obj).Only("json")

    for _, field := range fields {
        if field.Tag.Value == jsonTag {
            return extractFieldValue(obj, field.Name)
        }
    }

    return "", false, nil
}

// extractFieldValue uses some *_#rEfleCtioN mAgIC#_* to get field value from a
struct
// - if a field is a non-pointer field, simply returns its string value (false
for bool, etc)
// - if a field is a pointer, tries to dereference it and returns its value as
well.
// - if this pointer is nil, returns ValueNotSet, true, nil.
//
// Any other value is returned using fmt's default %v format.
func extractFieldValue(obj any, fieldName string) (string, bool, error) {
    field := reflect.ValueOf(obj).Elem().FieldByName(fieldName)

    // Although v.Interface() shouldn't be valid for any of our fields, let's
double check it anyways
    if !field.CanInterface() {
        return "", false, fmt.Errorf("can't convert %s: %w", fieldName,
ErrConvert)
    }

    value := field.Interface()
    vof := reflect.ValueOf(value)

    // If a value is a pointer like *string or *Struct
    if vof.Kind() == reflect.Pointer {
        value = vof.Elem() // dereference it
    }
}

```

```

        if !vof.IsValid() || vof.IsZero() {
            return "", true, ErrValueNotSet
        }

        return fmt.Sprintf("%v", value), true, nil
    }
}

```

Листинг В.4 – generate/replacer/replace.go

```

package replacer

import (
    "context"
    "errors"
    "fmt"
    "strings"
    "sync"
    "time"

    "gitlab.com/patricoski-law/lawfirm/backend/generate/replacer/suffix"
    "gitlab.com/patricoski-law/lawfirm/backend/models/documents"
    "gitlab.com/patricoski-law/lawfirm/backend/models/law"
    "golang.org/x/sync/errgroup"
)

var (
    ErrNoInfo          = errors.New("no info provided")
    ErrPlaceholderNotExists = errors.New("placeholder doesn't exist")
    ErrValueNotSet      = errors.New("value not set")
    ErrConvert          = errors.New("can't convert")
)

type ReplaceMap map[string]string

type Replacer struct {
    Adjuster *law.Adjuster
    Attorney *law.Attorney
    Case     *law.Case
    Client   *law.Client
    Defendant *law.Defendant
    Template *documents.Template

    rMap ReplaceMap
    m     sync.Mutex
}

// NewReplacer accepts a number of additional sources (like case, client,
// etc...)
// which would be used for replacements later.
// For sources use "WithCase(...)", "WithClient(...)", ...
func NewReplacer(source ...SourceFunc) (*Replacer, error) {
    r := &Replacer{}

    for _, fn := range source {
        err := fn(r)
        if err != nil {
            return nil, fmt.Errorf("source: %w", err)
        }
    }

    return r, nil
}

```

```

func (r *Replacer) MapFor(placeholders []string) (ReplaceMap, error) {
    r.rMap = make(ReplaceMap, len(placeholders))

    eg, _ := errgroup.WithContext(context.Background())
    for _, ph := range placeholders {
        eg.Go(r.replaceWorker(ph))
    }

    if err := eg.Wait(); err != nil {
        return nil, fmt.Errorf("worker: %w", err)
    }

    return r.rMap, nil
}

func (r *Replacer) replaceWorker(ph string) func() error {
    return func() error {
        value, err := r.replace(ph)
        if err != nil {
            return fmt.Errorf("can't insert placeholder %s: %w", ph, err)
        }

        r.m.Lock()
        if _, exists := r.rMap[ph]; !exists {
            r.rMap[ph] = value
        }
        r.m.Unlock()

        return nil
    }
}

// replace takes a placeholder, parses it and returns a replaced value.
func (r *Replacer) replace(placeholder string) (string, error) { //nolint:cyclop
    prefixStr, bodyStr, suffixStr := ParsePlaceholder(placeholder)

    var (
        result string
        err     error
    )

    suffixParams := &suffix.Params{}

    switch prefixStr {
    case "carrier":
        result, err = replaceEntity(bodyStr, r.Defendant.Carrier)
    case "case":
        result, err = replaceEntity(bodyStr, r.Case)
    case "client":
        result, err = replaceEntity(bodyStr, r.Client)
        suffixParams.Married = *r.Client.Married
    case "county":
        result, err = replaceEntity(bodyStr, r.Case.CourtCounty)
    case "defendant":
        result, err = replaceEntity(bodyStr, r.Defendant)
    case "adjuster":
        result, err = replaceEntity(bodyStr, r.Adjuster)
    case "attorney":
        result, err = replaceEntity(bodyStr, r.Attorney)
    default:
        result = replaceCommon(bodyStr)
    }
}

```

```

        if err != nil {
            if errors.Is(err, ErrPlaceholderNotExists) || errors.Is(err,
ErrValueNotSet) {
                return "{" + placeholder + "}", nil
            }

            return "", fmt.Errorf("error while replacing: %w", err)
        }

        result, err = suffix.Parse(result, suffixStr, suffixParams)
        if err != nil {
            return "", fmt.Errorf("suffix: %w", err)
        }

        return result, nil
    }

func replaceCommon(body string) string {
    switch body {
    case "current_date":
        return time.Now().Format(suffix.TimeFormat)
    default:
        return body
    }
}

func replaceEntity[E any](body string, e *E) (string, error) {
    if e == nil {
        return "", ErrNoInfo
    }

    replaceValue, found, err := GetValueByJSONTag(e, body)
    if err != nil {
        return "", err
    }

    if !found {
        return "", ErrPlaceholderNotExists
    }

    return replaceValue, nil
}

// returns a placeholder splitted into 3 parts.
// case#name$uppercase -> case, name, uppercase.
func ParsePlaceholder(ph string) (prefix, body, suffix string) {
    body = ph

    prefixIndex := strings.Index(body, "#")
    if prefixIndex != -1 {
        prefix = body[:prefixIndex]
        body = body[prefixIndex+1:]
    }

    suffixIndex := strings.Index(body, "$")
    if suffixIndex != -1 {
        suffix = body[suffixIndex+1:]
        body = body[:suffixIndex]
    }

    return prefix, body, suffix
}

```

Приложение Г
Исходный код скачивания документов
(рекомендуемое)

Листинг Г.1 – src/components/cases/generate_old/GenerateDocumentModel.svelte

```
<script>
import {onMount} from 'svelte'
import Fa from 'svelte-fa'
import {BaseURL, endpoints, paramsJson} from '../../../js/api'
import icons from '../../../js/icons'
import Modal from '../../../Modal.svelte'
import {notify} from '../../../Notifications.svelte'
import AdvancedSelectedItem from './AdvancedSelectedItem.svelte'
import TemplateLabel from './TemplateLabel.svelte'

let show, hide, _case
export {_case as case}

// Data vars
let document_link,
    request_data = {},
    defendant,
    loading = false

const clickGenerate = async () => {
    loading = true
    document_link = null

    if (!request_data.template_id) {
        loading = false
        notify('Please select a template before generating', 'error')
        return
    }

    const url = new URL(BaseURL + endpoints.generate)
    const params = paramsJson(request_data)
    params.method = 'POST'

    const result = await fetch(url.href, params)
    const response = await result.json()
    loading = false
    if (result.status !== 200 || !result.ok) {
        if (response?.error) {
            notify(`${response.error.message}
($ {response.error.details})`, 'error')
            return
        }

        notify('Unable to reach server', 'error')
        return
    }

    const uuid = response?.data?.uuid
    if (uuid) {
        document_link = new URL(BaseURL + endpoints.download + uuid)
    } else {
        notify('Request was fulfilled, but no download link returned',
'error')
    }
}

const reset = () => {
    request_data = {
        template_id: null,
        case_id: _case.id,
        client_id: _case?.main_client?.id ?? null,
    }
}
```

```

        defendant_id: null,
        adjuster_id: null,
        attorney_id: null,
    }
    loading = false
}

async function handleChangeDefendant() {
    const url = new URL(
        BaseURL + endpoints.defendant + request_data?.defendant_id,
    )
    const params = {method: 'GET'}
    const result = await fetch(url.href, params)
    const response = await result.json()
    loading = false
    if (result.status !== 200 || !result.ok) {
        if (response?.error) {
            notify(`${response.error.message}
($ {response.error.details})`, 'error')
            return
        }

        notify('Unable to reach server', 'error')
        return
    }

    defendant = response
    request_data.adjuster_id = null
    request_data.attorney_id = null
    handleLockAdjuster()
    handleLockAttorney()
}

function handleLockClient() {
    request_data.client_id = _case?.main_client?.id ?? null
}

function handleLockAdjuster() {
    if (defendant?.adjusters?.length > 0) {
        request_data.adjuster_id = defendant.adjusters[0].id
    } else {
        request_data.adjuster_id = null
    }
}

function handleLockAttorney() {
    if (defendant?.attorneys?.length > 0) {
        request_data.attorney_id = defendant.attorneys[0].id
    } else {
        request_data.attorney_id = null
    }
}

onMount(reset)
</script>

<button class="gap-2 btn btn-outline btn-secondary" on:click="{show}">
    <Fa icon="{icons.docx}" />
    <span class="hidden md:inline-block">Generate (old)</span>
</button>

<Modal
    title="Generate Document"

```

```

bind:show="{show}"
bind:hide="{hide}"
on:close="{reset}">
<div class="flex flex-col w-full p-1 overflow-x-hidden overflow-y-auto
gap-4">
  <TemplateLabel bind:value="{request_data.template_id}" />
  <div class="grid grid-cols-[fit-content(100%)_auto] gap-4">
    <AdvancedSelectItem
      bind:value="{request_data.client_id}"
      mainClient="{_case?.main_client}"
      on:lock="{handleLockClient}"
      options="{_case?.clients}"
      locked="{true}"
      tooltipText="Use Main Client"
      title="Client" />
    <AdvancedSelectItem
      bind:value="{request_data.defendant_id}"
      options="{_case?.defendants}"
      on:change="{handleChangeDefendant}"
      showPlaceholder="{true}"
      lockable="{false}"
      title="Defendant" />
    {#if request_data.defendant_id != null}
      <AdvancedSelectItem
        bind:value="{request_data.adjuster_id}"
        on:lock="{handleLockAdjuster}"
        options="{defendant?.adjusters}"
        locked="{true}"
        tooltipText="Use Recent Adjuster"
        title="Adjuster" />
      <AdvancedSelectItem
        bind:value="{request_data.attorney_id}"
        on:lock="{handleLockAttorney}"
        options="{defendant?.attorneys}"
        locked="{true}"
        tooltipText="Use Recent Attorney"
        title="Attorney" />
    {/if}
  </div>
</div>

<svelte:fragment slot="actions">
  <button
    class:loading="{loading}"
    class="gap-2 btn btn-secondary"
    disabled="{loading}"
    on:click="{clickGenerate}">
    {#if loading}
      Generating...
    {/if}
  </button>
  {#if document_link}
    <a href="{document_link}" class="gap-2 btn btn-success"
download>
      <Fa icon="{icons.docx}" size="lg" />
      Download
    </a>
  {/if}
</svelte:fragment>
</Modal>

```

Листинг Г.2 – src/components/cases/generate_old/TemplateLabel.svelte

```

<script>
import {onMount} from 'svelte'
import Fa from 'svelte-fa'
import {BaseURL, endpoints} from '../../../js/api'
import icons from '../../../js/icons'
import {notify} from '../../../Notifications.svelte'
import GenerateRecursiveSelectItem from './GenerateRecursiveSelectItem.svelte'

export let value = null

let hierarchy

async function handleUpdate() {
  const result = await fetch(new URL(BaseURL + endpoints.hierarchy), {
    method: 'GET',
  })
  const response = await result.json()
  if (result.status !== 200 || !result.ok) {
    if (response?.error) {
      notify(`${response.error.message}
($ {response.error.details})`, 'error')
      return
    }

    notify('Unable to reach server', 'error')
    return
  }

  hierarchy = response?.data
}

onMount(handleUpdate)
</script>

<div>
  <div class="label">
    <span class="text-xl label-text">Template:</span>
  </div>
  <div class="flex gap-2">
    <select class="grow select select-secondary" bind:value="{value}">
      <option value="{null}" selected disabled>Choose a
template</option>
      {#each hierarchy ?? [] as category}
        <GenerateRecursiveSelectItem category="{category}" />
      {:else}
        <option disabled>Loading</option>
      {/each}
    </select>
    <button
      class="gap-2 btn btn-secondary btn-outline btn-square"
      on:click="{handleUpdate}">
      <Fa icon="{icons.refresh}" />
    </button>
  </div>
</div>

```

Листинг Г.3 – src/components/cases/generate_old/AdvancedSelectItem.svelte

```

<script>
import {createEventDispatcher} from 'svelte'
import Fa from 'svelte-fa'

```

```

import Time from 'svelte-time'
import icons from '../.../js/icons'
import {firstLastName} from '../.../js/utils'

export let title,
  value,
  options,
  mainClient,
  tooltipText = '',
  lockable = true,
  showPlaceholder = false,
  nameFunc = firstLastName,
  locked

const dispatch = createEventDispatcher(),
  format = 'MMM/DD'

function lock() {
  locked = !locked

  if (locked) {
    dispatch('lock')
  }
}

$: isValidSelect = options?.length >= 1
</script>

<div class="label">
  <span class="capitalize label-text {title}">{title}</span>
</div>
<div class="flex gap-2 place-items-center">
  {#if lockable}
    <div
      class:tooltip="{tooltipText}"
      class="cursor-pointer place-items-center"
      data-tip="{tooltipText}"
      on:click|preventDefault="{lock}">
      <button
        class:btn-secondary="{!locked}"
        class:btn-outline="{!locked}"
        disabled="{!isValidSelect}"
        class="btn btn-circle btn-xs">
        {#if locked || !isValidSelect}
          <Fa icon="{icons.case_close}" />
        {:else}
          <Fa icon="{icons.case_open}" />
        {/if}
      </button>
    </div>
    <span>&rsquo;</span>
  {/if}
  <select
    class="grow select select-secondary select-sm"
    bind:value="{value}"
    on:change
    disabled="{locked || !options}">
    {#if showPlaceholder || !value}
      <option disabled value="{null}">Select an option</option>
    {/if}
    {#if mainClient}
      <option disabled>Main Client</option>
      <option value="{mainClient.id}">

```

```

        {firstName(mainClient)}
      </option>
      <option disabled>Related Clients</option>
    {/if}
    {#each options ?? [] as option}
      <option value="{option.id}">
        {nameFunc(option)}
        {#if option?.linked_at}
          (<Time format="{format}"
timestamp="{option?.linked_at}" />)
        {/if}
      </option>
    {/each}
    <option selected disabled value="{null}">No options</option>
  {/each}
</select>
</div>

```

Листинг Г.4 –

src/components/cases/generate_old/GenerateRecursiveSelectItem.svelte

```

<script>
export let category,
  level = 0

level = parseInt(level)
</script>

{#each category?.templates ?? [] as template}
  <option value="{template.id}" disabled="{!template.uploaded}">
    {#if !template.uploaded}
      (no template)
    {/if}
    {template.name}
  </option>
{/each}
{#each category?.categories ?? [] as category}
  <svelte:self category="{category}" level="{level + 1}" />
{/each}

```

Приложение Д
Рекомендательное письмо, оригинал
(рекомендуемое)

PATRICOSKI

LAW OFFICES

To Whom It May Concern:

My name is Gregory A. Patricoski, I am an attorney in Illinois, United States, practicing complex civil litigation, primarily in Chicagoland. I also have a Masters of Information Systems from Northwestern University, Chicago, Illinois. I would like to take a moment to express in the clearest and most passionate terms possible my highest recommendation for Alexandr Glazunov (Alex) in any technical employment he may be seeking. I have had the occasion to know Alex in a professional capacity for over four (4) years, two (2) of which he and I have worked closely together. I first met Alex when he came on as an intern for an attorney software company I was a co-founder of. I later determined to commission and highly sophisticated and custom software package for my law firm as I plan to scale the firm larger and manage file information and rapidly create dynamic documents (including, for example, attorney correspondence and even pleadings). I reached out to Alex without hesitation and he singlehandedly designed, developed, launched, and refined a custom case file management system that is more advanced, robust, simple, and convenient than anything I viewed on the market. With basic and limited direction from myself, he created a program meeting all my needs that fully managed my law firm and practice. Some of the many highlights include:

- Clear and easy display of a very complex and diverse amount of data for each case;
- Client, attorney, and insurance company management; and my personal favorite
- Powerful dynamic document automation with the ability to add templates for customized document automation from data within the system.

This project required not only great technical skills but also the ability to understand complex and nuanced laws and legal practice in the State of Illinois. As is true in most American law, every single rule had at least one exception – nothing about this application's needs was straightforward. Alex, who speaks perfect academic English, was not only able to understand the goals and needs of the software but also the rules and limitations of the law, which are of mission-critical importance to myself¹. Furthermore, he developed this software under his own project management and goal setting/advancement, writing detailed and proper documentation and addressing bugs with minimal or no supervision from me. What's more, he was able to do this as a full-time student and difficult course load.

With this said, I must extend not only my highest praise to Alex but my most sincere recommendation that anyone with the opportunity to hire him do so without haste. He has a skill and ability to distill complex information and deal with difficult and demanding principals (such as myself) that is rare and cannot be taught. Thank you, and please do not hesitate to contact me to further discuss Alex and his qualifications.

/s/ Gregory A. Patricoski
gregory@markpatlaw.com
 (312) 500-2020
May 16, 2023

¹ I have often joked with him that he was able to understand the complexity of Illinois Civil Procedure better than many attorneys I know.

Приложение Е
Рекомендательное письмо, перевод
(рекомендуемое)

Юридические бюро Патрикоски

18 мая 2023 г.

Всем заинтересованным лицам

Меня зовут Грегори А. Патрикоски, я адвокат из Иллинойса, Соединенные Штаты Америки, занимающийся комплексным гражданским судопроизводством, главным образом в городской конгломерации Чикаго. У меня также есть степень магистра информационных систем Северо-Западного университета Чикаго, штат Иллинойс. Я хотел бы воспользоваться возможностью со всей ясностью и горячей поддержкой дать свою наивысшую рекомендацию Александру (Алексу) Глазунову для любой работы в технической области, соискателем которой он является. У меня была возможность наблюдать Александра в профессиональном качестве на протяжении более четырех (4) лет, два (2) года из которых мы с ним тесно сотрудничали. Впервые я познакомился с Алексом, когда он пришел стажироваться в компанию по разработке программного обеспечения для юристов, одним из основателем которой я являлся. Позже я решил заказать очень сложный индивидуализированный программный пакет для своей юридической фирмы, поскольку я планирую расширить деятельность фирмы, управлять файловой информацией и быстро создавать динамические документы (включая, например, переписку адвокатов и даже состязательные бумаги). Я без колебаний обратился к Алексу, и он в одиночку спроектировал, разработал, запустил и отладил пользовательскую систему управления материалами дел, которая оказалась более продвинутой, надежной, простой и удобной, чем все то, что я видел на рынке. Руководствуясь лишь моими общими и ограниченными инструкциями, он создал программу, отвечающую всем моим потребностям, с помощью которой я управлял всей своей юридической фирмой и практикой. Некоторые из ее многочисленных преимуществ включают в себя:

- четкое и ясное отображение очень сложного и разнообразного объема данных для каждого конкретного дела;
- управление работой с клиентами, адвокатами и страховыми компаниями; и моя любимая функция —
- мощная динамическая автоматизация документооборота с возможностью добавления шаблонов для индивидуализированной автоматизации документооборота на основе данных из системы.

Этот проект требовал не только отличных технических навыков, но и умения разбираться в сложных и содержащих множество нюансов законах и юридической практике штата Иллинойс. Как и в большинстве американских законов, у каждого правила было по крайней мере одно исключение — ни одно из требований к этому приложению не было простым и однозначным. Алекс, в совершенстве владеющий академическим английским, смог понять не только цели и требования программного обеспечения, но и нормы и ограничения законодательства, которые имеют для меня критически важное значение¹. Кроме того, он разработал это программное обеспечение, сам осуществляя руководство проектом, ставя цели и выполняя их, сопровождая это написанием подробной и надлежащей документации и устраняя программные ошибки при минимальном

¹ Общаясь с ним, я часто шутил, что он смог разобраться в сложностях гражданского судопроизводства штата Иллинойс лучше, чем многие адвокаты, которых я знаю.

контроле с моей стороны и даже вовсе без него. Более того, он смог сделать это, будучи студентом очной формы обучения и имея сложную учебную нагрузку.

С учетом всего вышесказанного, я не только обязан дать Алексу наивысшую оценку, но и искренне посоветовать всем, кто рассматривает его кандидатуру, делать это без спешки. Он обладает навыками и способностью извлекать сложную информацию и иметь дело с непростыми и требовательными заказчиками (такими, как я), что является редким качеством, которому нельзя научиться. Благодарю вас и прошу без колебаний обращаться ко мне для дальнейшего обсуждения Александра и его квалификации.

/Подпись/ Грегори А. Патрикоски _____

Юридические бюро Патрикоски

Грегори А. Патрикоски

1755 С. Напервилль Род, № 100 Уитон, Иллинойс 60189

Офис (630) 933-8000

gregory@markpatlaw.com

www.markpatlaw.com

Рисунок Е. 2 – Рекомендательное письмо, перевод, вторая часть

Certification of Translation Accuracy

Translation of **Letter Of Recommendation** from **English** to **Russian**

As an authorized representative of RushTranslate, a professional translation services agency, I hereby certify that the above-mentioned document has been translated by an experienced, qualified and competent professional translator, fluent in the above-mentioned language pair and that, in my best judgment, the translated text truly reflects the content, meaning, and style of the original text and constitutes in every respect a complete and accurate translation of the original document. This document has not been translated for a family member, friend, or business associate.

This is to certify the correctness of the translation only. I do not make any claims or guarantees about the authenticity or content of the original document. Further, RushTranslate assumes no liability for the way in which the translation is used by the customer or any third party, including end-users of the translation.

A copy of the translation is attached to this certification.



Mike Bortscheller
 Authorized Representative
 Order Date: May 18, 2023

RushTranslate
 640 South Fourth St
 Suite 300
 Louisville, KY 40202
 United States

