Минобрнауки россии

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения и администрирования

информационных систем

Разработка и реализация веб-приложения для подписи документов «Распишитесь здесь»

Дипломная работа

Специальность 010503 Математическое обеспечение

и администрирование информационных систем

Специализация Информационные системы

Допущено к защите в ГЭК \_\_.\_\_.2015

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ д. ф.-м.н., проф. М.А. Артемов

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П.А. Томаровский

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ к. ф.-м.н., доц. Е.Е. Михайлова

Воронеж 2015

**Аннотация**

Работа посвящена разработке и созданию веб-приложения, предоставляющего возможность подписи документов. Пользуясь им, пользователи смогут самостоятельно, обладая лишь навыками пользователя ПК, осуществлять подпись документов в электронном виде.

Исходя из поставленной задачи, в дипломной работе рассмотрены современные технологические и технические средства для реализации подобных приложений.

**Содержание**

[Введение 5](#_Toc421694008)

[Основные термины 7](#_Toc421694009)

[1. Постановка задачи 8](#_Toc421694010)

[2. Обзор существующих веб-приложений 9](#_Toc421694011)

[3. Анализ выбранных технологий 10](#_Toc421694012)

[3.1. Шаблон проектирования Model-view-controller (MVC) 10](#_Toc421694013)

[3.2. ORM технология ADO.Net Entity Framework 11](#_Toc421694014)

[3.2.1. Шаблоны разработки Entity Framework 13](#_Toc421694015)

[3.2.2. Стратегия Code First 14](#_Toc421694016)

[3.3. JavaScript-фреймворк AngularJS 14](#_Toc421694017)

[4. Средства реализации 16](#_Toc421694018)

[5. Требования к аппаратному и программному обеспечению 17](#_Toc421694019)

[5.1 Требования к серверной части 17](#_Toc421694020)

[5.2 Требования для веб-приложения 17](#_Toc421694021)

[6. Интерфейс пользователя 18](#_Toc421694022)

[6.1. Регистрация и вход в систему 18](#_Toc421694023)

[6.2. Главная страница. Работа с документами 19](#_Toc421694024)

[6.3. Пользовательские настройки 24](#_Toc421694025)

[6.4. Запросы 28](#_Toc421694026)

[6.5. Процесс подписания документа 28](#_Toc421694027)

[7. Реализация 33](#_Toc421694028)

[7.1. Построение концептуальной модели 33](#_Toc421694029)

[7.2. Выбор конкретной СУБД 34](#_Toc421694030)

[7.3. Отображение концептуальной схемы на физическую модель 35](#_Toc421694031)

[7.4. Описание созданных таблиц базы 37](#_Toc421694032)

[7.5. Общая архитектура и схема работы приложения 44](#_Toc421694033)

[7.6. Модель 44](#_Toc421694034)

[7.7. Паттерн Репозиторий 46](#_Toc421694035)

[7.8. Сервисы 47](#_Toc421694036)

[8. План тестирования 49](#_Toc421694037)

[Заключение 51](#_Toc421694038)

[Список литературы 52](#_Toc421694039)

[Приложение 1. Листинг интерфейсов и классов для работы с таблицами User и Request 53](#_Toc421694040)

[Приложение 2. Таблица системных состояний 55](#_Toc421694041)

[Приложение 3. Пример представления страниц логина и регистрации 57](#_Toc421694042)

# Введение

Актуальность темы дипломной работы обуславливается тем, что в нашей стране, в организациях, всё чаще применяются электронные документы. Многие работодатели считают особо важным увеличить эффективность работы на своем предприятии, так как за последнее время резко увеличился поток информации. Также возросла необходимость в более качественной обработке и удобном хранении информации.

По данным Siemens Business Services [1], при использовании в организации электронных документов:

* производительность труда персонала увеличивается на 20-25%;
* стоимость архивного хранения электронных документов на 80% ниже в сравнении со стоимостью хранения бумажных архивов.

Принято также считать, что при использовании документов в электронном виде приобретаются тактические и стратегические выгоды. Тактические выгоды определяются сокращением расходов, связанным с: освобождением физического места для хранения документов; уменьшением затрат на копирование и доставку документов в бумажном виде; снижением расходов на персонал и оборудование и др. К стратегическим относятся преимущества, связанные с повышением эффективности работы предприятия или организации.

Но преградой на пути к безбумажному документообороту является его якобы ненадежность, по сравнению с проверенным временем бумажным. Хотя электронные документы имеют точно такую же юридическую значимость, как и бумажные. Их не нужно дополнительно распечатывать, а также можно предоставить по требованию налоговой инспекции или в суде. Также немаловажную роль играет человеческий фактор, так как многим сотрудникам удобнее читать по бумаге и делать на ней записи, чем работать на компьютере или в программе.

Однако в целесообразности перехода на электронные документы сегодня не приходится сомневаться: достаточно обратиться к опыту зарубежных стран, где так называемый e-invoicing (обмен электронными счетами-фактурами) работает уже более 6 лет. Так, например, с момента перехода на электронный документооборот Германия к 2012 году сэкономила 54 млрд. евро, а в Дании бизнес на электронном декларировании налогов ежегодно экономит 50 млн. евро [2].

В дипломной работе будет представлено веб-приложение «Распишитесь здесь». С его помощью можно будет подписывать документы, хранящиеся в электронном виде, минуя процессы печати, подписи от руки, а затем сканирования и загрузки обратно в электронный вид. Приложение позволяет расписаться с помощью мышки (планшета) непосредственно на изображении документа, либо добавить в документ заранее подготовленное изображение подписи.

# Основные термины

* Подпись – графический файл, содержащий изображение подписи.
* Рисунок – графический файл, содержащий произвольное изображение, например, логотип компании.
* Надпись – поле, содержащее произвольный текст, который может быть отредактирован в процессе подготовки документа.

# 1. Постановка задачи

Разработать и реализовать веб-приложение, предоставляющее возможность подписывать документы. Система должна обладать следующей функциональностью:

1. Авторизация пользователей в системе;
2. Загрузка документов;
3. Размещение на страницах документа компонента(-ов) для подписи;
4. Отправка документа на подпись одному человеку;
5. Отправка документа на подпись группе людей;
6. Подписание документа человеком, который:
   1. является пользователем системы;
   2. не является пользователем системы.

# 2. Обзор существующих веб-приложений

В наше время существует ряд сайтов и программ, помогающих пользователям упростить процесс подписи и отправки документов.

Было рассмотрено около десяти различных видов приложений для подписи документов, начиная с программ для компьютеров и заканчивая приложениями для телефонов. Все они, кроме возможности электронной подписи, обладают целым рядом дополнительного функционала и, соответственно, высокой ценой.

Наиболее подходящей по критериям была программа «На Подпись!» [3]. Однако, стоимость данного продукта слишком высока: стоимость минимального комплекта версии Pro для сервера терминалов (на 5 пользователей) составляет 12600 руб. А стоимость одной лицензии программы для рабочей станции составляет 2800 руб.

3. Анализ выбранных технологий

## 3.1. Шаблон проектирования Model-view-controller (MVC)

Шаблон проектирования Model-View-Controller (MVC) предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо [4]. Стандартная схема архитектуры «Модель-Вид-Контроллер» изображена на следующем рис. 3.1:

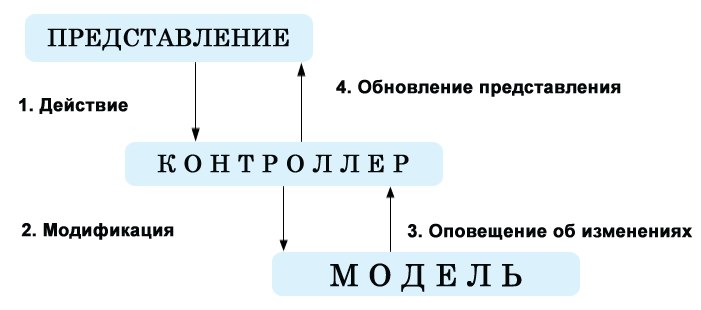


Рис. 3.1. Схема архитектуры «Модель-Вид-Контроллер»

* Представление. Модуль вывода информации. Это может быть шаблонизатор или что-либо подобное, цель которого является только в выводе информации в виде HTML на основе каких-либо готовых данных.
* Контроллер. Модуль управления вводом и выводом данных. Данный модуль должен следить за переданными в систему данными (через форму, строку запроса, cookie или любым другим способом) и на основе введённых данных решить:
* Передавать ли их в модель.
* Вывести сообщение об ошибке и запросить повторный ввод (заставить модуль представление обновить страницу с учётом изменившихся условий).

Кроме того, контроллер обязан определять тип данных, полученных от модели (есть ли это готовый результат, отсутствие оного, либо сообщение об ошибке) и передавать информацию в модуль представления.

* Модель. Модуль, отвечающий за непосредственный расчёт чего-либо на основе полученных от пользователя данных. Результат, полученный этим модулем, должен быть передан в контроллер, и не должен содержать ничего, относящегося к непосредственному выводу (то есть должен быть представлен во внутреннем формате приложения) [5].

## 3.2. ORM технология ADO.Net Entity Framework

Использование реляционных систем управления базами данных имеет много преимуществ, среди которых: простота использования, математическая точность реляционной алгебры и широкое распространение продуктов от различных производителей. Но использование реляционной базы данных для хранения объектно-ориентированных данных приводит к возникновению ряда задач, которые должны быть решены программистами. Одна из таких задач: преобразование объектов приложения в удобную для сохранения в реляционных базах данных форму. Также существует обратная задача: преобразование реляционных данных в объекты с сохранением свойств этих объектов и отношений между ними. Решая такие задачи, разработчикам приходится использовать языки запросов и писать большое количество кода, который часто однообразен и сильно подвержен ошибкам [5].

Сейчас существуют механизмы, которые могут автоматически решать такие задачи – инициализация объектов программы данными из таблиц базы данных с возможностью передачи всех изменений обратно, в записи базы данных. Они основаны на технологии object-relational mapping (ORM) – объектно-реляционное отображение. Благодаря такой технологии программисты могут манипулировать привычными элементами языка программирования: классами, объектами, атрибутами, методами. А специальная ORM система, имея список таблиц в базе данных и список таблиц в программе, сможет автоматически преобразовывать запросы объектного вида в запросы реляционного вида (рис. 3.2).

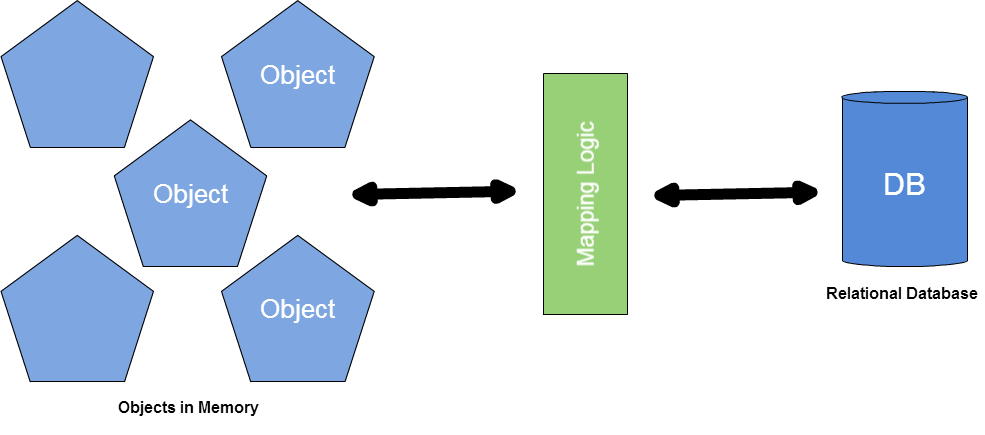


Рис. 3.2. Функционирование ORM системы

Преимущества ORM систем:

1. Производительность. Использование ORM систем повышает производительность разработки, избавляя разработчика от необходимости работы с языками запросов и написания большого количества однообразного кода.
2. Дизайн приложения. ORM система, разрабатываемая опытными программными архитекторами, реализует эффективные паттерны проектирования, которые заставляют использовать положительные практики при создании слоя доступа к данным. Соответственно, ORM система будет благоприятно влиять на дизайн приложения.
3. Повторное использование. При использовании ORM систем для доступа к данным, приложение не будет содержать код, привязанный к конкретной системе управления базами данных или же к конкретной схеме базы данных, то есть приложению ничего не нужно знать о структуре базы данных, а базе данных ничего о том, как данные организованы в приложении. Благодаря этому, схема хранения может изменяться, не затрагивая код, а изменение кода не повлечет за собой изменения схемы хранения.
4. Сопровождаемость. Код, генерируемый ORM системами, предположительно, хорошо проверен и оптимизирован. Поэтому, как правило, не стоит заботиться о его детальном и глубоком тестировании.

Технология Entity Framework – это ORM решение компании The Microsoft для программной платформы .NET Framework. Entity Framework реализует все функции объектно-реляционного отображения и предоставляет возможность взаимодействия с объектами посредством технологии Linq To Entities или технологией Entity SQL.

Технология Entity Framework предоставляет специальную структуру данных Entity Data Model (EDM), описывающую данные на основе понятия «сущность» и «связь», которая не зависит ни от схем хранения, ни от особенностей синтаксиса языка прикладного уровня. Такая структура данных называется концептуальной моделью сущности и предметной области (концептуальной моделью) [6].

### 3.2.1. Шаблоны разработки Entity Framework

Entity Framework предлагает три различных шаблона разработки (рис. 3.3):

* Database First – разработка начинается со схемы существующей базы данных, на основе которой создается концептуальная модель и классы сущностей предметной области.
* Model First – разработка начинается с создания концептуальной модели сущностей на основе которой создаются программируемые классы и схемы базы данных.
* Code First – разработка начинается с написания кода. Entity Framework использует классы для динамического построения концептуальной модели и схемы базы данных.

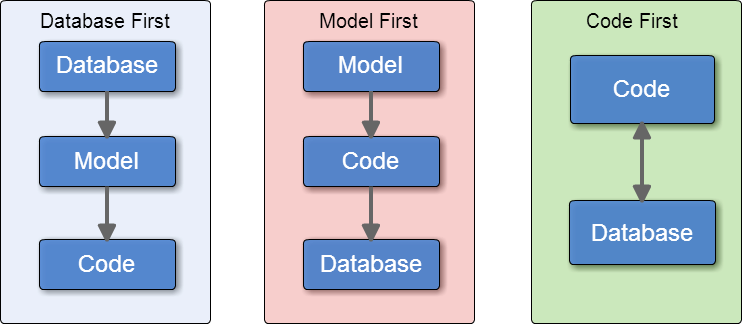


Рис. 3.3. Шаблоны разработки, предлагаемые Entity Framework

### 3.2.2. Стратегия Code First

Разработка начинается сразу же с написания кода, без использования графических дизайнеров. При использовании такого подхода все необходимые объекты, которые нужны для описания модели, будут генерироваться автоматически во время выполнения. Стратегия Code First позволяет описывать модель сразу в виде классов предметной области. При выполнении, концептуальная модель будет описана на основе этих классов. Конечно же, классы и их члены не могут в полной мере описать концептуальную модель, поэтому дополнительная информация задается или же с помощью декларативного подхода (с использованием атрибутов), или же с помощью "текучего" API (Fluent API). При этом подходе предполагается, что у вас еще нет базы данных, с которой будет работать ваше приложение. Поэтому Code First так же предоставляет возможность создать необходимую базу данных на основе динамически построенной модели [7].

## 3.3. JavaScript-фреймворк AngularJS

AngularJS — JavaScript-фреймворк с открытым исходным кодом. Предназначен для разработки одностраничных приложений. Его цель — расширение браузерных приложений на основе MVC шаблона, а также упрощение тестирования и разработки.

Фреймворк работает с HTML, содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных [8].

Схема работы AngularJS представлена на рис. 3.4:

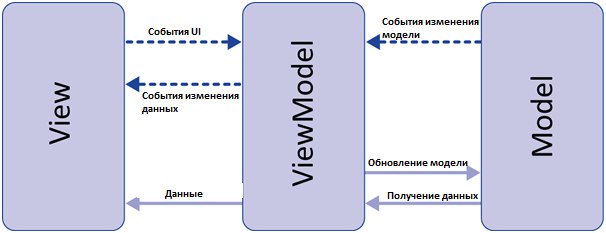


Рис. 3.4. Структурная схема работы AngularJS

AngularJS на стадии загрузки страницы превращает код в angular‑приложение. HTML-разметка страницы с angular-директивами и фильтрами в ней компилируются в дерево шаблонов, соответствующие области видимости (scope) и контроллеры присоединяются к ним в нужных местах, внутренний цикл работы приложения обеспечивает правильную привязку данных между представлением и моделью. Данная схема реализована в полном согласии с принципами MVC и обеспечивает максимальное разделение между представлением, контроллером и моделью.

# 4. Средства реализации

1. Интерактивная среда разработки – Microsoft Visual Studio 2012.
2. Языки разработки – C#, JavaScript.
3. Система контроля версий – Git [9].
4. Реляционная база данных MS SQL Server.

# 5. Требования к аппаратному и программному обеспечению

5.1 Требования к серверной части

Требования к аппаратному обеспечению:

* ПК типа IBM PC;
* процессор с тактовой частотой не менее 2.4 ГГц;
* оперативная память не менее 2048 Мб;
* постоянная память не менее 1 Гб;
* подключение к сети Интернет со скоростью не менее 100 Мбит/с.

Требования к программному обеспечению:

* IIS версии 6.0;
* Microsoft SQL Server.

5.2 Требования для веб-приложения

Требования к аппаратному обеспечению:

* ПК типа IBM PC;
* процессор с тактовой частотой не менее 1 ГГц;
* оперативная память не менее 2 Гб;
* постоянная память не менее 32 Мб;
* подключение к сети Интернет со скоростью не менее 100 Кбит/с.

Требования к программному обеспечению: приложение является кроссбраузерным. Рекомендуемые версии браузеров: Safari – версия 5.1.7, Opera – версия 11.64, Google Chrome – версия 43.0.2357.124, Mozilla Firefox – версия 38.0 , Internet Explorer – версии 11.

# 6. Интерфейс пользователя

## 6.1. Регистрация и вход в систему

Если пользователь впервые заходит на сайт, то ему нужно перейти в раздел регистрации нового пользователя и заполнить форму (рис. 6.1.):

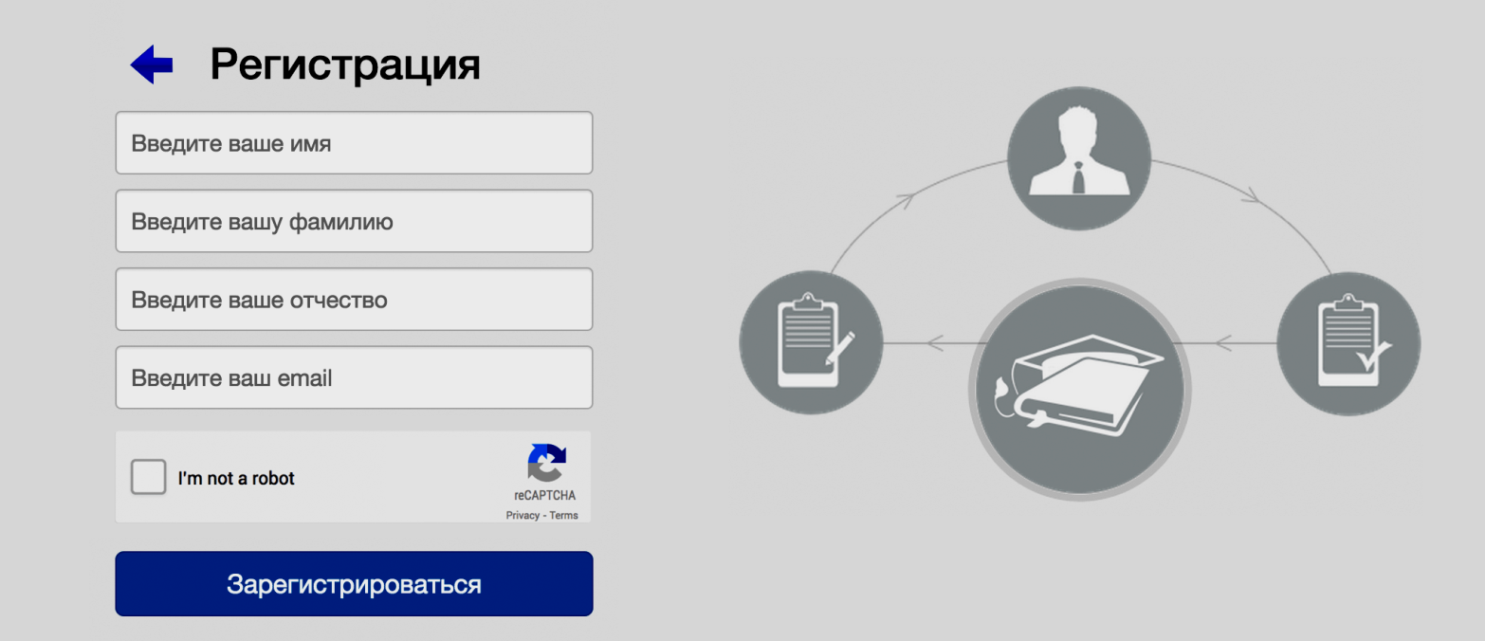


Рис. 6.1. Регистрация

Для того чтобы авторизоваться в системе «Распишитесь здесь», необходимо зайти на страницу входа и ввести свой E-mail адрес и пароль (рис. 6.2).

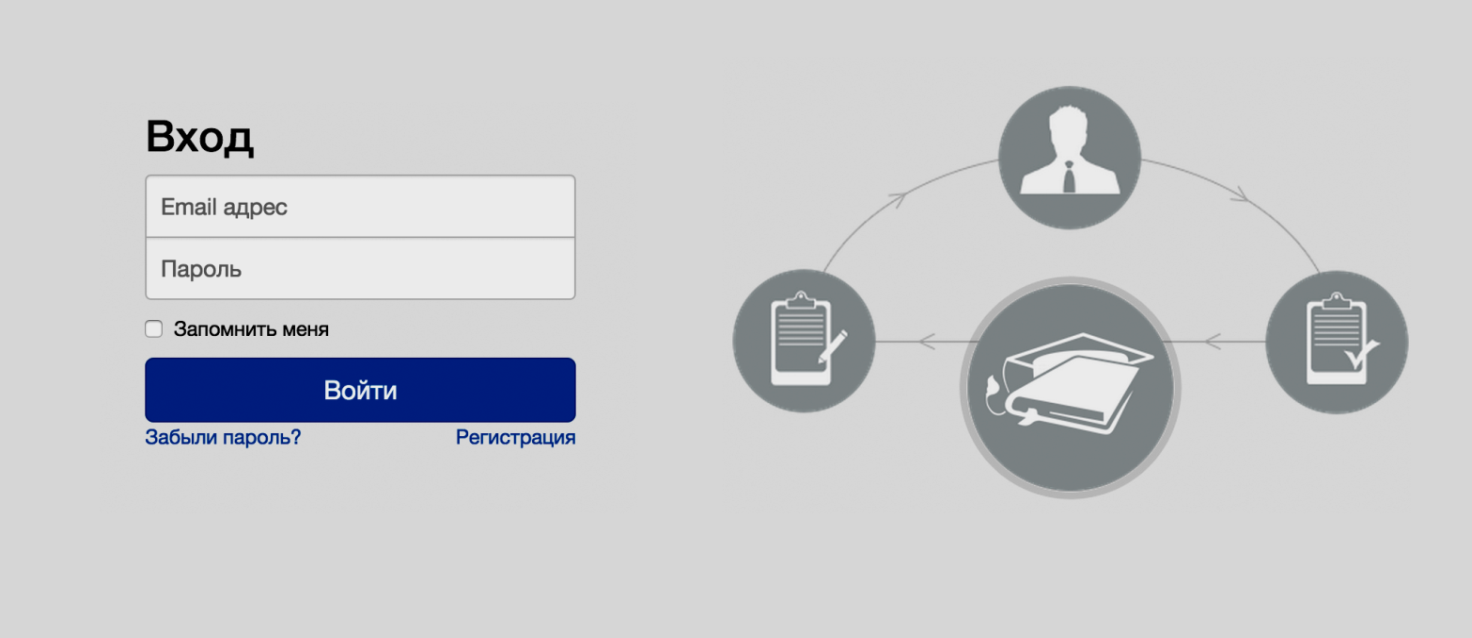


Рис. 6.2. Авторизация пользователя

## 6.2. Главная страница. Работа с документами

Авторизовавшись в системе, пользователь видит главную страницу. На ней расположены графики с текущей статистикой по запросам и документам (рис. 6.3).

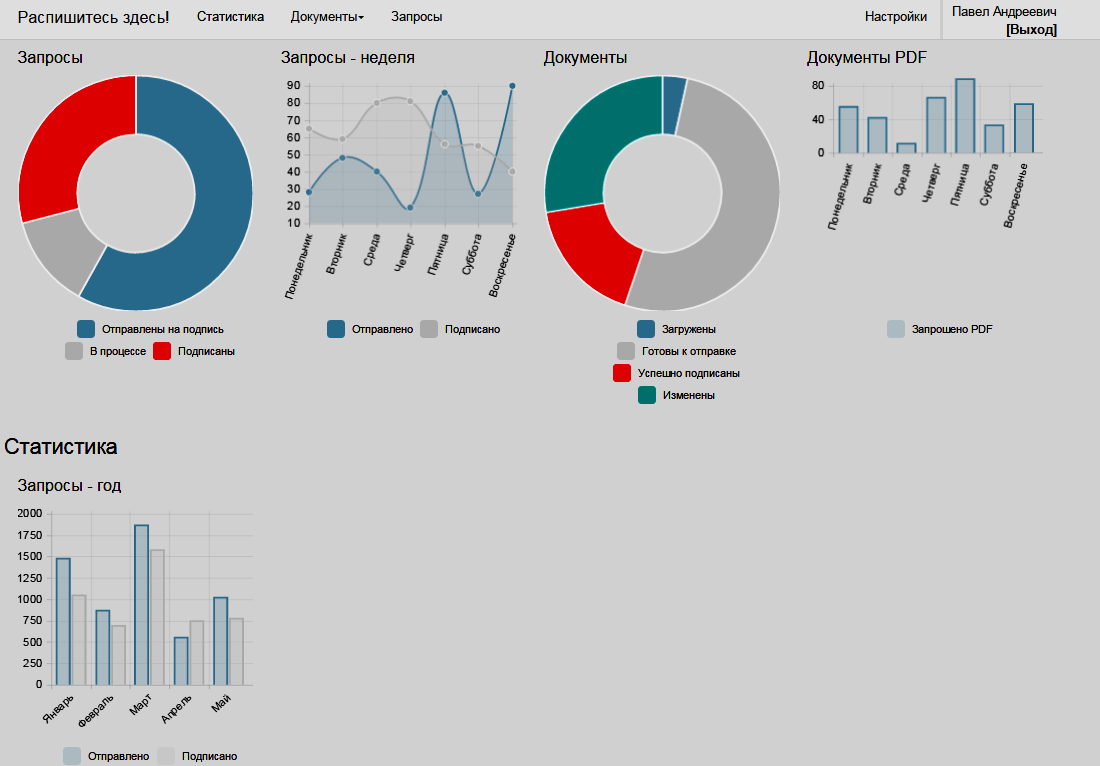


Рис. 6.3. Главная страница

В случае если работа с системой уже производилась ранее, статистика будет непустой. В виде графиков отображается следующая информация:

* Общее количество запросов пользователя, которые «Отправлены на подпись», «В процессе» и «Подписаны».
* График отправленных и подписанных запросов за последние семь дней.
* Общее количество документов пользователя, которые «Загружены», «Готовы к отправке», «Успешно подписаны», «Изменены».
* График количества загруженных пользователем документов за последние семь дней.
* График годовой статистики количества документов, которые были «Отправлены» и «Подписаны».

Вверху страницы расположено основное меню для навигации по системе (рис. 6.4).

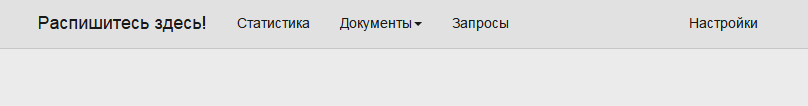


Рис. 6.4. Меню

Для того чтобы загрузить документ, необходимо перейти через основное меню на страницу «Загрузить документ». Далее, пользователь выбирает необходимое ему количество документов для загрузки и нажимает: кнопку «Начать загрузку» – для одновременной загрузки всех документов или кнопку «Загрузить» – для загрузки документов по одному (рис. 6.5).

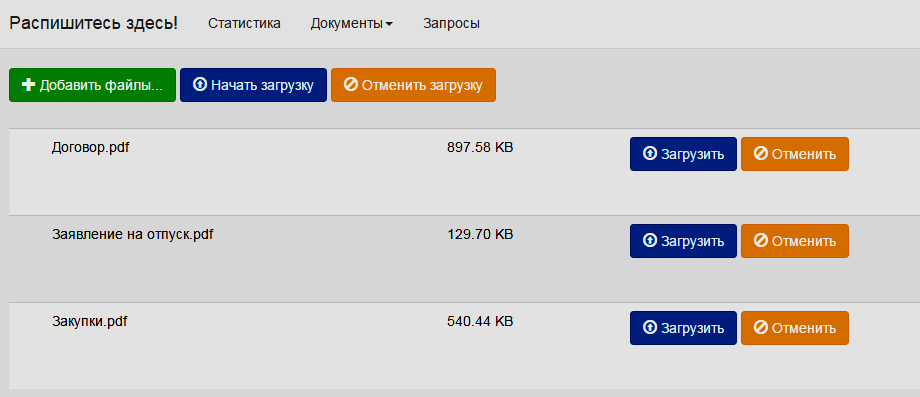


Рис. 6.5. Загрузка документов

Просмотреть уже загруженные в систему документы можно перейдя в основном меню на страницу «Список документов». На данной странице в виде списка представлены все документы пользователя, также здесь можно увидеть информацию по каждому из них, а именно: присвоенный документу статус, количество страниц (рис. 6.6).



Рис. 6.6. Список документов

Кликнув по названию необходимого документа, пользователь перейдет на страницу работы с документом (рис. 6.7).

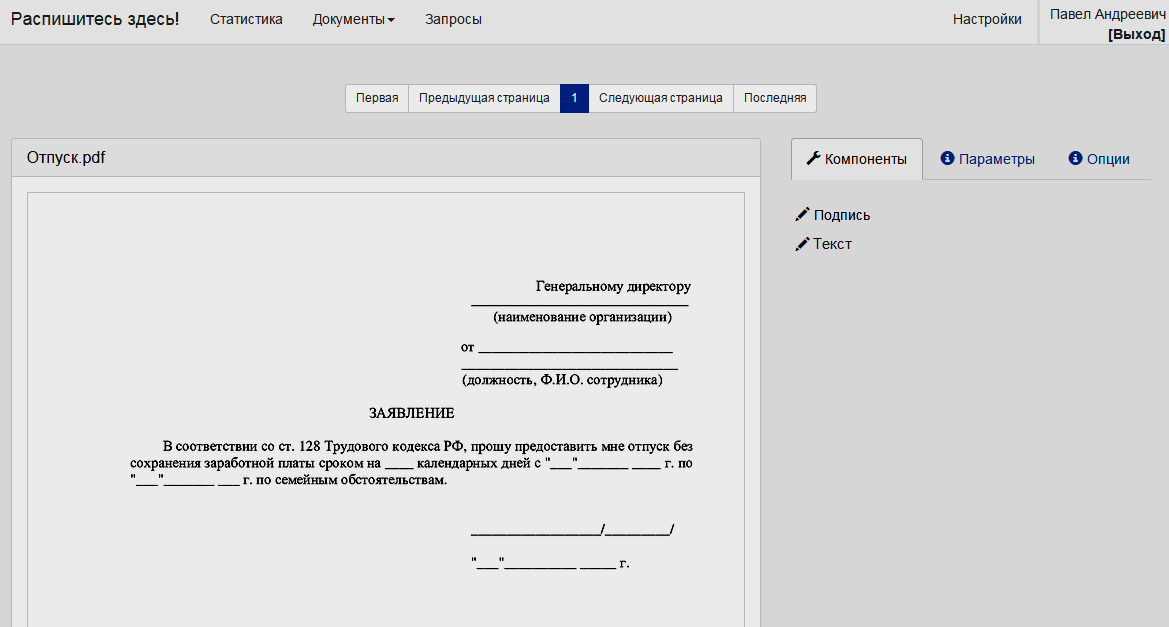


Рис. 6.7. Просмотр документа

Также, здесь пользователь может добавлять необходимое количество компонентов «Подпись» и «Текст» (рис. 6.8).

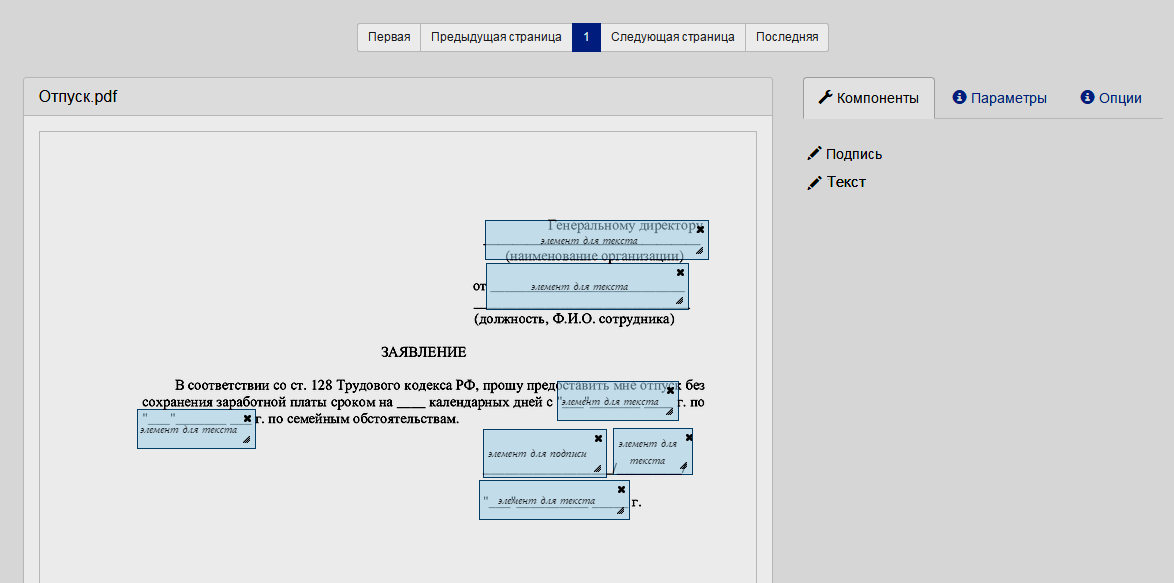


Рис. 6.8. Добавление компонентов

В разделе опции можно изменить название документа и присвоить ему статус «Готов к отправке» (рис. 6.9).

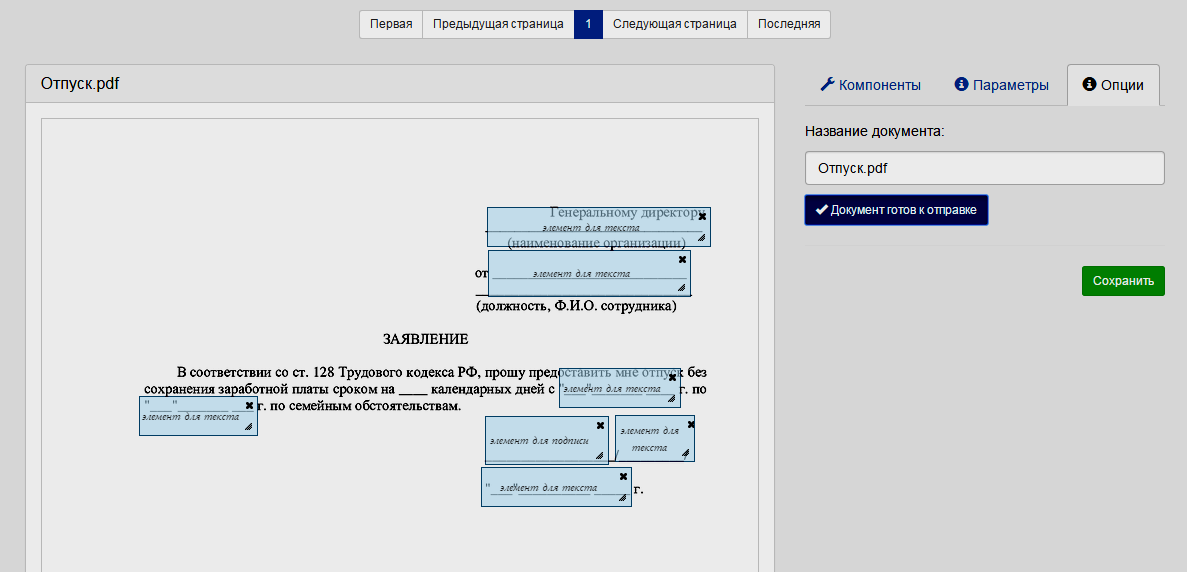


Рис. 6.9. Пометка документа «Готов к отправке»

После того, как пользователь добавит в документ компонент «Подпись» и поставит ему статус «Готов к отправке», то в общем списке, около данного документа, появится кнопка «Отправить на подпись» (рис. 6.10). При нажатии на данную кнопку, пользователю будет предложено ввести e-mail адреса и имена получателей, которые должны поставить на документе свою подпись (рис. 6.11).

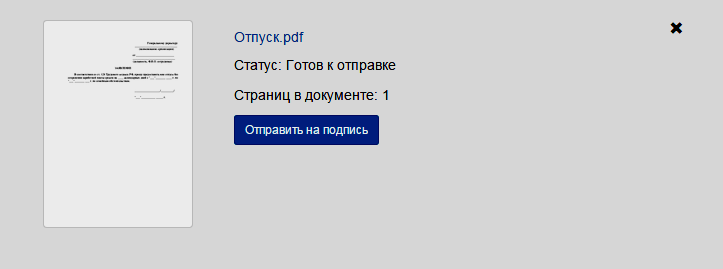


Рис. 6.10. Документ готов к отправке

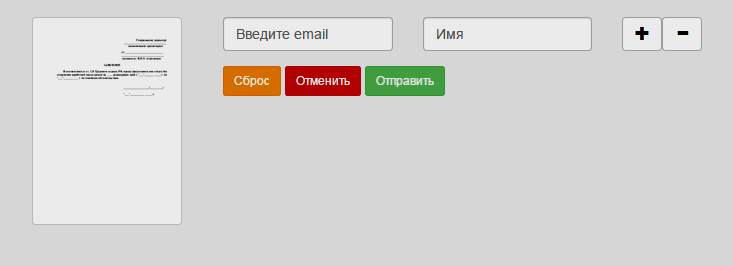


Рис. 6.11. Отправка документа на подпись

## 6.3. Пользовательские настройки

Раздел «Настройки» включает в себя подразделы:

* Аккаунт.
* Параметры:
  + основные;
  + запросы;
  + процесс подписи;
  + документы.
* Настройки почты.

В настройках аккаунта пользователь может изменить адрес электронной почты, на которую приходят уведомления от системы и с помощью которой происходит авторизация, также можно изменить свои данные (Ф.И.О.) и пароль (рис. 6.12).

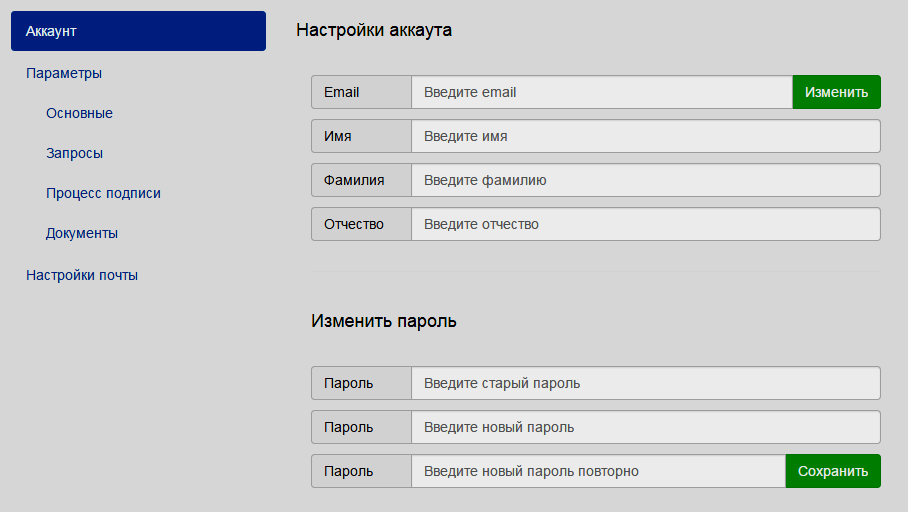


Рис. 6.12. Настройка аккаунта

Подраздел «Основные настройки» позволяет установить следующее (рис. 6.13):

1. Необходимо ли пользователю получать уведомления о завершении процессов подписания документов.
2. Тип уведомлений:
   1. Email;
   2. SMS;
   3. Личное сообщение.
3. Прикреплять ли к письмам подписанные документы.
4. Посылать ли участникам подписания документов письма об том, что документ всеми подписан.
5. Напоминать ли человеку о том, что ему необходимо подписать документ, и количество дней до напоминания.
6. Переключаться ли, при возможности, на мобильную версию сайта.

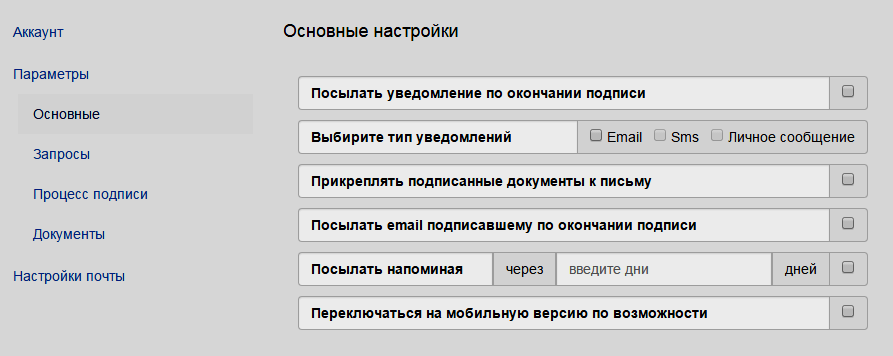


Рис. 6.13. Основные настройки

В настройках «Запросов» (рис. 6.14) можно установить количество дней, в течение которых запрос на подпись документа будет действителен. Галочка в строке «отправлять себе копию документа (по умолчанию)» подразумевает, что пользователь будет получать в письме с уведомлением о завершении процесса подписи копию подписанного документа. «Уведомление о прочтении письма с запросом (по умолчанию)» означает, что пользователю будут приходить оповещения о прочтении писем с запросом на подпись документа, когда письма будут читаться каждым из участников процесса подписания. Помимо уведомления, запросу будет присваиваться статус «Документ просмотрен». Если пользователь хочет получать нотификации и оповещения на другой e‑mail адрес, то ему достаточно указать этот адрес в строке «использовать другой email для нотификаций».

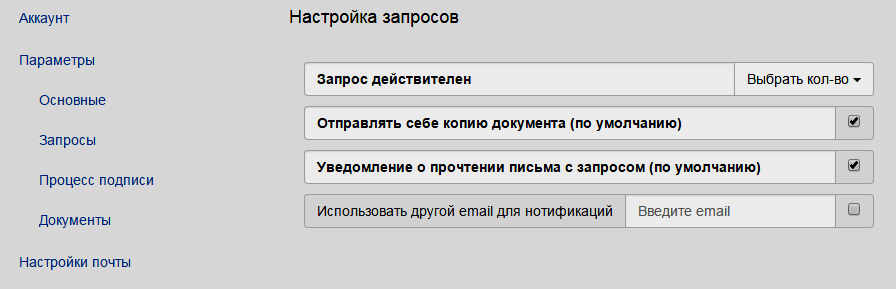


Рис. 6.14. Настройки запросов

В настройках «Процесса подписи» (рис. 6.15) пользователь может сделать следующее:

1. Указать, рисовать подпись или разрешить использовать шаблон с подписью.
2. Установить разрешение на подпись в «один клик», когда вместо подписи ставится галочка, что пользователь ознакомлен и согласен с содержанием документа.
3. Установить ограничение на подпись только в «один клик».
4. Разрешить скачивать документы, которые еще не подписаны всеми участниками процесса подписания.

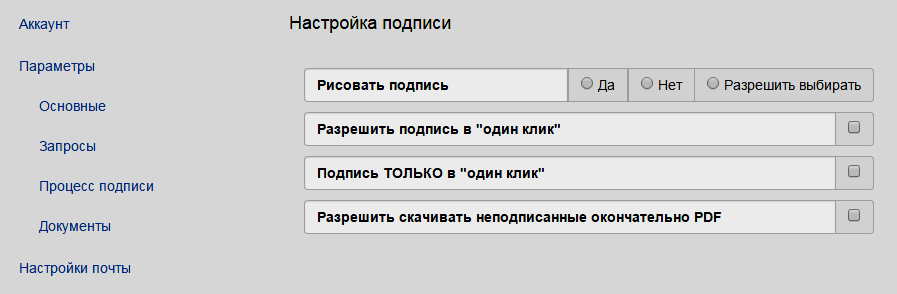


Рис. 6.15. Настройки подписи

В подразделе настройки «Документов» (рис. 6.16) есть возможность установить количество дней, в течение которых будет действителен запрос на подпись. Также можно указать, необходимо ли добавлять страницу информации к письму с запросом.

Пользователь может указать, в каком качестве необходимо сохранять документы при загрузке в систему – от 100 до 600 dpi.

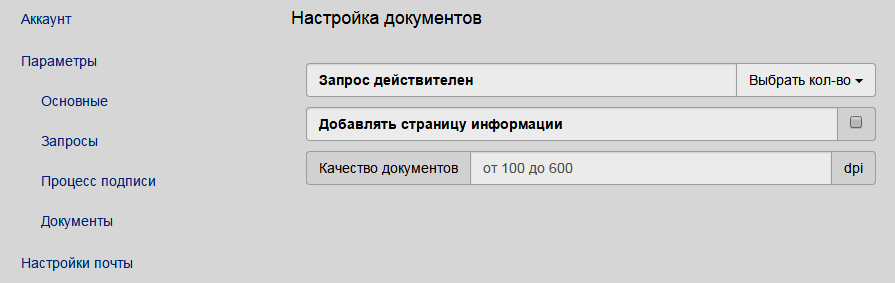


Рис. 6.16. Настройки документов

## 6.4. Запросы

Перейдя на страницу «Запросы», пользователь может просмотреть список его запросов. Они представлены в табличном виде (рис. 6.17):

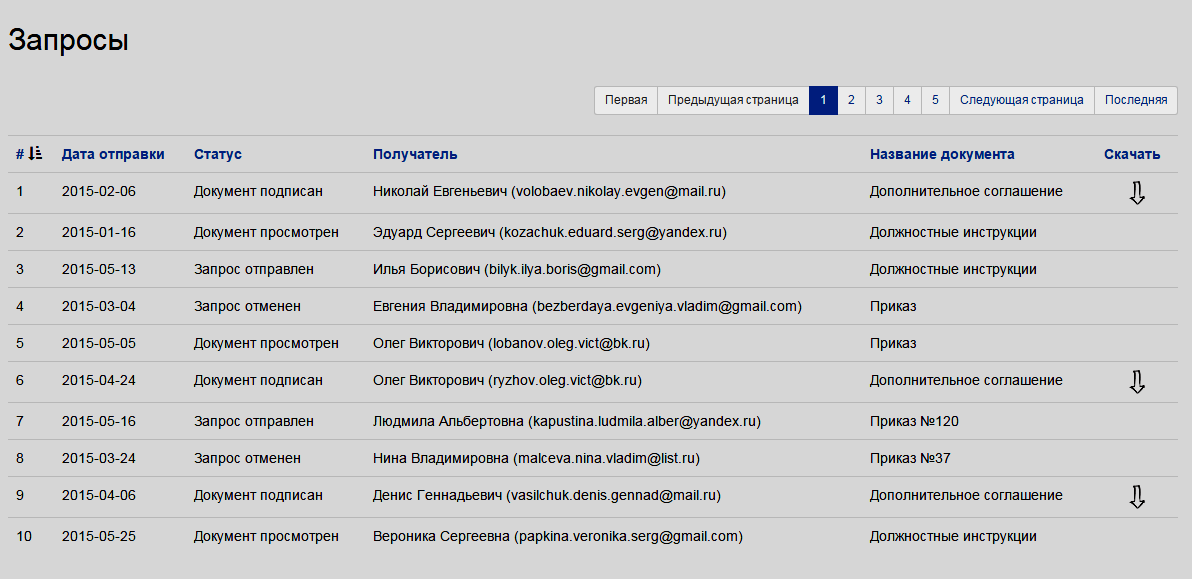


Рис. 6.17. Список запросов

К скачиванию доступны только окончательно подписанные документы.

Для удобства поиска документов, пользователь может отсортировать таблицу по любому столбцу.

## 6.5. Процесс подписания документа

После указания адреса человека, чью подпись необходимо получить для утверждения документа, ему будет выслано письмо-уведомление и специальная секретная ссылка для работы с документом в интерфейсе приложения (рис. 6.18):

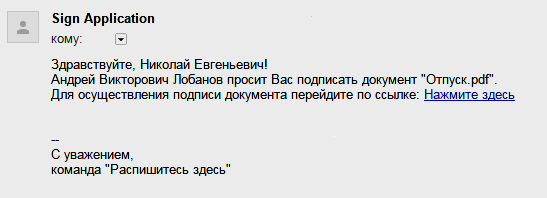


Рис. 6.18. Письмо с запросом на подпись

Перейдя по ссылке, пользователь попадает на страницу приложения, где постранично отображается документ с необходимыми элементами для ввода текста и подписи (рис. 6.19).

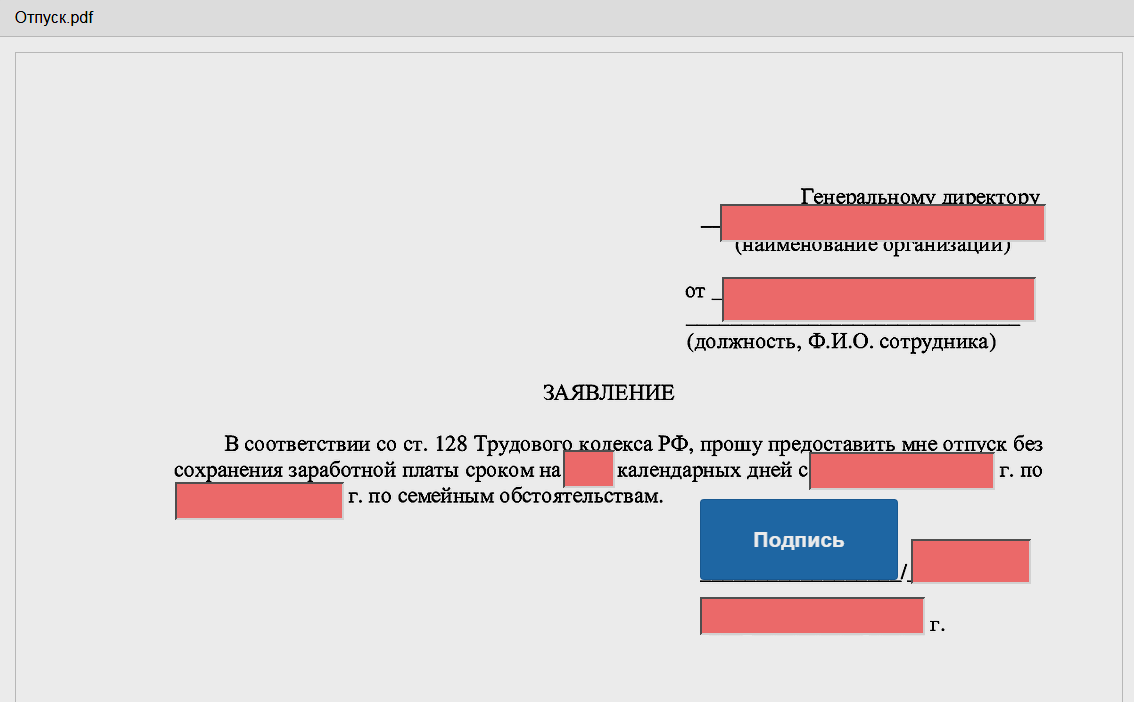


Рис. 6.19. Документ с размещенными на нем элементами

На каждом поле для ввода текста стоит проверка на заполнение. Пока пользователь не заполнит все текстовые поля, элемент «Подпись» доступен не будет.

На рисунке 6.20 изображена ситуация, когда пользователь заполнил все текстовые поля:

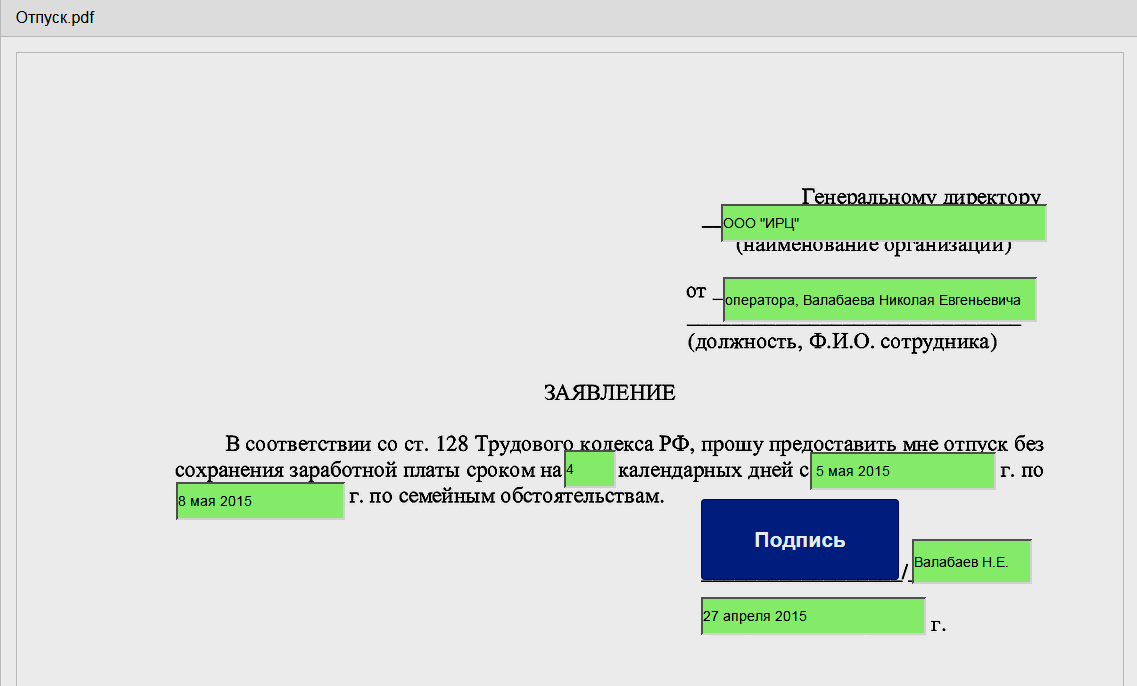


Рис. 6.20. Документ с заполненными текстовыми элементами

Подписывать документ нужно собственноручной подписью, которую вы рисуете в специальном окошке, появившемся при нажатии на элемент «Подпись» (рис. 6.21).

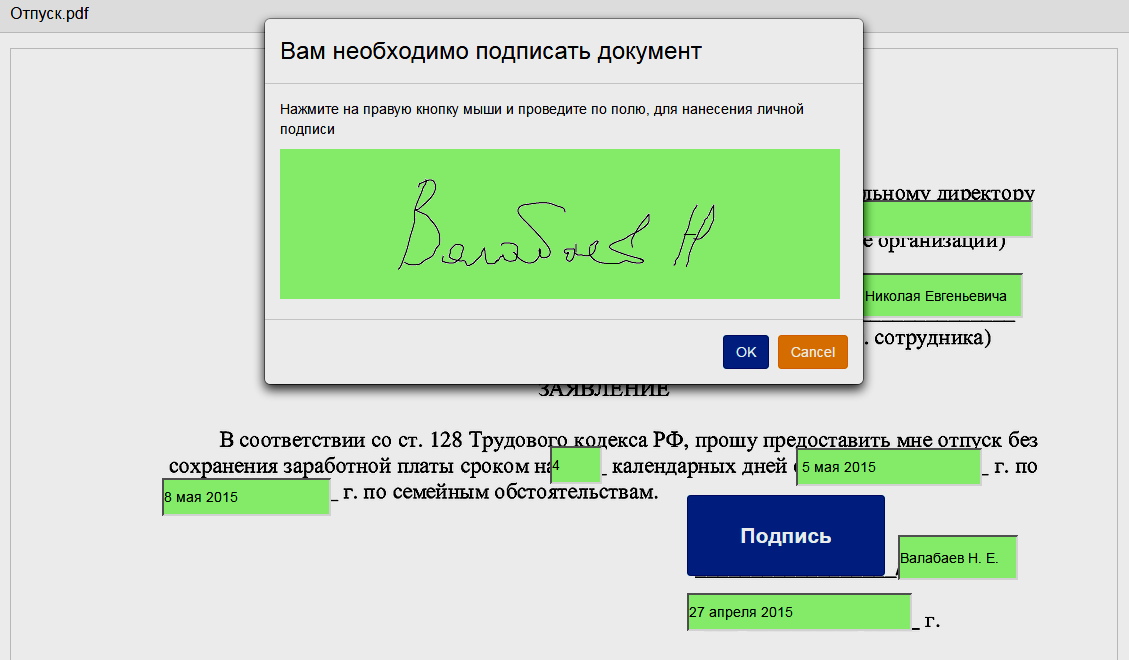


Рис. 6.21. Окно для собственноручной подписи

После подписания документа, отправителю запроса приходит письмо с уведомлением (рис. 6.22):

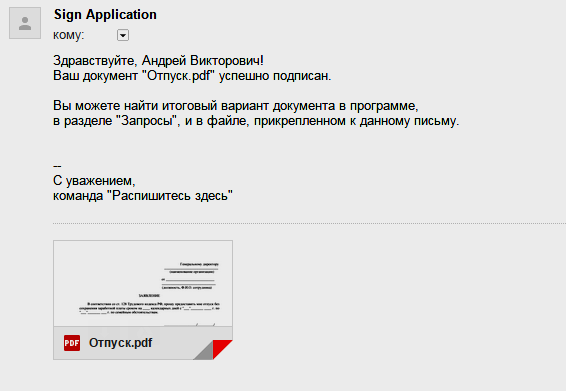


Рис. 6.22. Письмо-уведомление отправителю запроса

А каждому участнику процесса подписания документа, приходит письмо со следующим содержанием (рис. 6.23):

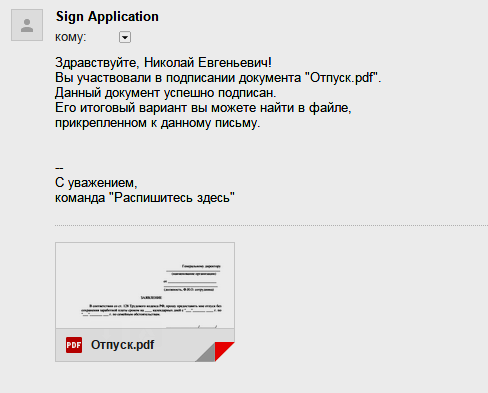


Рис. 6.23. Письмо-уведомление участникам подписания документа

Окончательный вариант подписанного и доступного к скачиванию документа выглядит следующим образом (рис. 6.24):

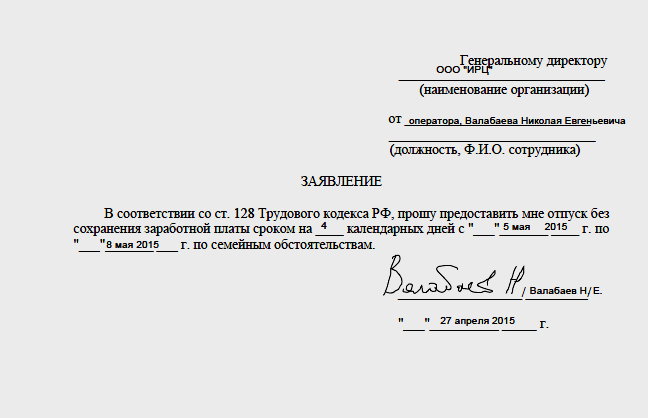


Рис. 6.24. Заполненный документ с подписью

# 7. Реализация

## 7.1. Построение концептуальной модели

В простых случаях для построения концептуальной схемы используют традиционные методы агрегации и обобщения. При агрегации объединяются информационные объекты (элементы данных) в один в соответствии с семантическими связями между объектами. Методом агрегации создается информационный объект (сущность), с присущими ему атрибутами. При обобщении информационные объекты (элементы данных) объединяются в родовой объект (рис. 7.1):

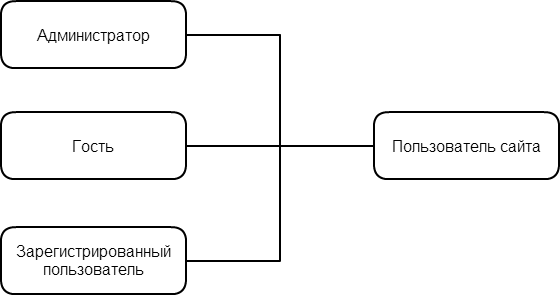


Рис. 7.1. Обобщение информационных объектов в родовой объект

Выбор модели диктуется, прежде всего, характером предметной области и требованиями к БД. Другим немаловажным обстоятельством является независимость концептуальной модели от СУБД, которая должна быть выбрана после построения концептуальной схемы [10].

Модель “Сущность-связь”, дающая возможность представлять структуру и ограничения реального мира, а затем трансформировать их в соответствии с возможностями промышленных СУБД, является весьма распространенной. На рисунке 7.2 изображена концептуальная модель “Сущность-связь”, которая представлена в виде графической схемы.

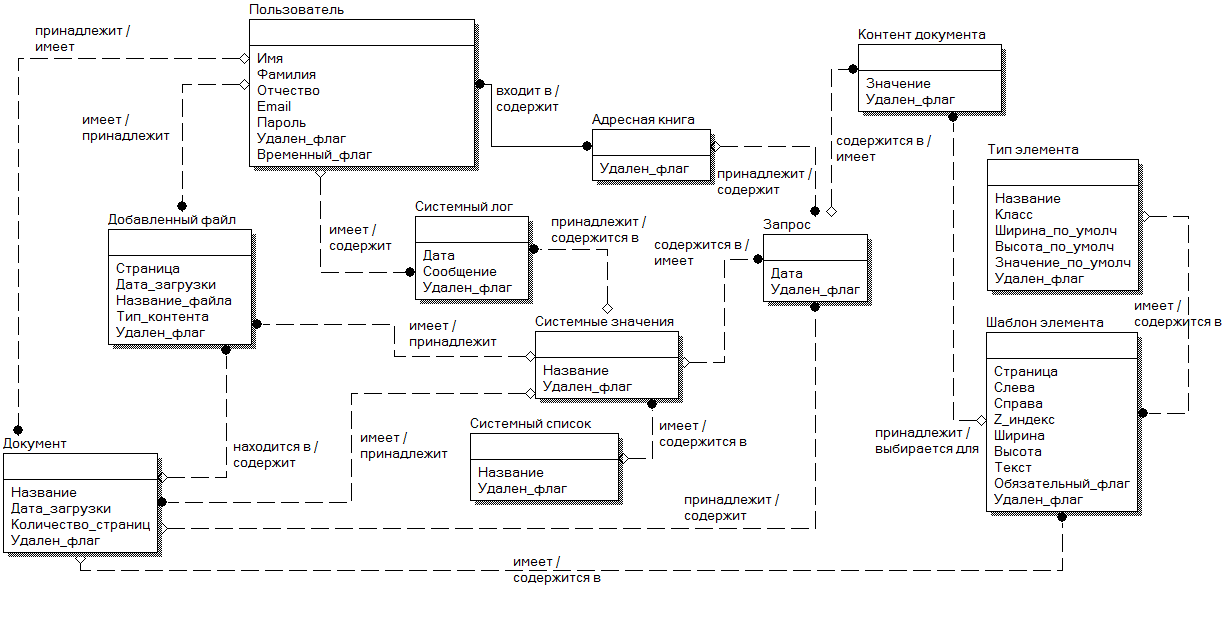


Рис. 7.2. Концептуальная модель веб-приложения

## 7.2. Выбор конкретной СУБД

Одним из основных критериев выбора СУБД, в данном случае это MS SQL, является оценка того, насколько эффективно внутренняя модель данных, поддерживаемая системой, способна описать концептуальную схему. Системы управления базами данных, ориентированные на персональные компьютеры, как правило, поддерживают реляционную или сетевую модель данных. Подавляющее большинство современных СУБД – реляционные.

Конструирование баз данных на основе реляционной модели имеет ряд важных преимуществ перед другими моделями:

* Независимость логической структуры от физического и пользовательского представления.
* Гибкость структуры базы данных – конструктивные решения не ограничивают возможности разработчика БД выполнять в будущем самые разнообразные запросы.

Так как реляционная модель не требует описания всех возможных связей между данными, впоследствии разработчик может задавать запросы о любых логических взаимосвязях, содержащихся в базе, а не только о тех, которые планировались первоначально.

## 7.3. Отображение концептуальной схемы на физическую модель

1. Каждая простая сущность превращается в таблицу (отношение). Имя сущности становится именем таблицы.
2. Каждый атрибут становится возможным столбцом с тем же именем. Столбцы соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы соответствующие обязательным атрибутам, - не могут. Если атрибут является множественным, то для него строится отдельное отношение.
3. Компоненты уникального идентификатора сущности превращаются в первичный ключ. Если имеется несколько возможных уникальных идентификаторов, из них выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, то к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящаяся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.
4. Связи многие к одному и один к одному становятся внешними ключами. То есть создается копия уникального идентификатора с конца связи один, и соответствующие столбцы составляют внешний ключ.
5. Индексы создаются для первичного ключа (уникальный индекс), а также внешних ключей и тех атрибутов, которые будут часто использоваться в запросах.
6. Если в концептуальной схеме используются подтипы, то возможны два варианта.

Все подтипы хранятся в одной таблице, которая создается для самого

внешнего супертипа, а для подтипов создаются представления. В таблицы добавляется, по крайней мере, один столбец, содержащий код ТИПА, и он становится частью первичного ключа.

Во втором случае для каждого подтипа создается отдельная таблица (для более нижних - представление) и для каждого подтипа первого уровня супертип создается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы – столбцы супертипа).

1. Если остающиеся внешние ключи все принадлежат одному домену, то есть имеют общий формат, то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различных связей. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи [11].

Разработанная выше диаграмма является концептуальной (рис. 7.2). Это означает, что диаграмма не учитывает особенности конкретной СУБД. По данной концептуальной диаграмме можно построить физическую диаграмму, в которой уже будут учитываться такие особенности СУБД, как допустимые типы и наименования полей и таблиц, ограничения целостности и т.п. Физический вариант диаграммы, приведен на рис. 7.3.

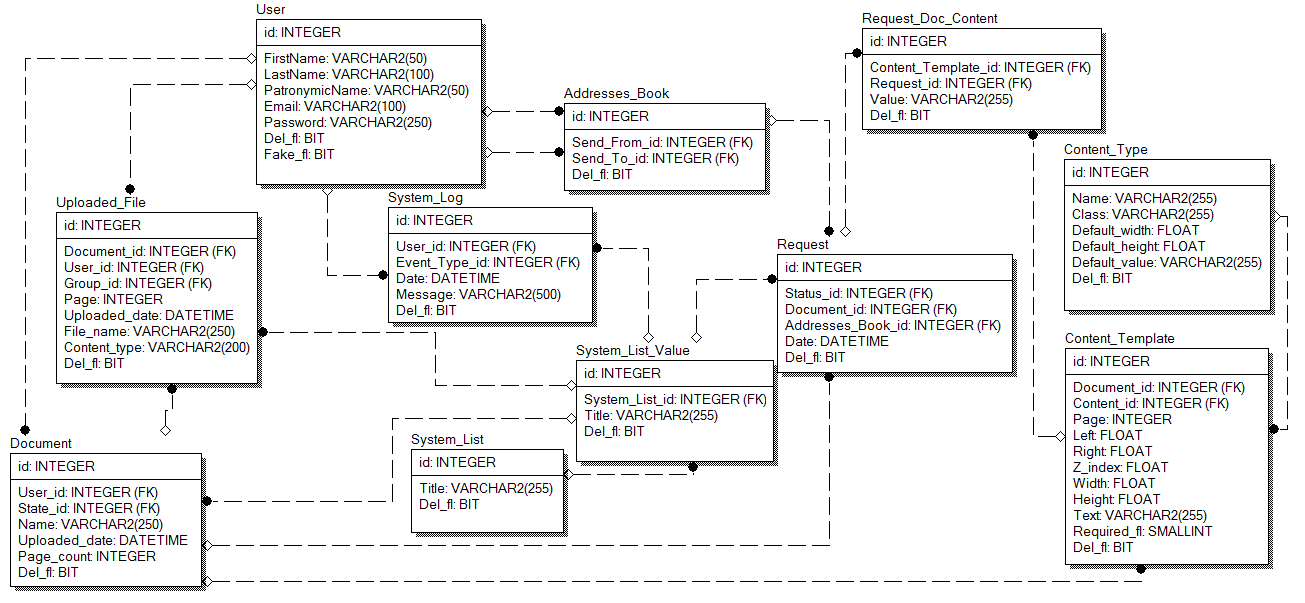


Рис. 7.3. Физическая модель базы данных веб-приложения

На данной диаграмме каждая сущность представляет собой таблицу базы данных, каждый атрибут становится колонкой соответствующей таблицы. Во всех таблицах, например, появились атрибуты, которых не было в концептуальной модели. Это ключевые атрибуты родительских таблиц, мигрировавших в дочерние таблицы для того, чтобы обеспечить связь между таблицами посредством внешних ключей, а так же обычные атрибуты.

## 7.4. Описание созданных таблиц базы

Исходя из представленной на рисунке 7.3 физической модели, в базе данных имеется одиннадцать таблиц.

Ниже рассмотрена каждая из них, дано подробное описание ее предназначения и всех полей.

Таблица User содержит информацию обо всех пользователях (табл. 7.1). В поле Fake\_fl заносится показатель: зарегистрирован пользователь в системе или нет.

Таблица 7.1. Структура таблицы User

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор пользователя |
| FirstName | varchar | Имя пользователя |
| LastName | varchar | Фамилия пользователя |
| PatronymicName | varchar | Отчество пользователя |
| Email | varchar | Электронная почта пользователя |
| Password | varchar | Пароль пользователя |
| Del\_fl | bit | Флаг удаления |
| Fake\_fl | bit | Флаг регистрации |

В таблице SystemLog хранятся системные логи программы (табл. 7.2). В поле ID заносится идентификатор записи.

Таблица 7.2. Структура таблицы SystemLog

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор записи |
| EventType | int | Тип события |
| UserID | int | Идентификатор пользователя |
| Date | datetime | Дата |
| Message | varchar | Текст записи |
| Del\_fl | bit | Флаг удаления |

Таблица Document содержит информацию о загруженных файлах (табл. 7.3). В поле UserID заносится идентификатор пользователя, загрузившего файл. Поле StateID содержит системное значение о текущем состоянии документа.

Таблица 7.3. Структура таблицы Document

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор документа |
| Name | varchar | Название документа |
| UserID | int | Идентификатор пользователя |
| UploadedDate | datetime | Дата загрузки |
| StateID | int | Идентификатор состояния |
| PageCount | int | Количество страниц |
| Del\_fl | bit | Флаг удаления |

Таблица AddressesBook используется в качестве адресной книги и содержит информацию о том, кому пользователь отправлял запросы на подпись документов (табл. 7.4). В поле SendFromID заносится идентификатор пользователя, отправлявшего запрос. Поле SendToID содержит идентификатор получателя запроса.

Таблица 7.4. Структура таблицы AddressesBook

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор записи |
| SendFromID | int | Идентификатор пользователя |
| SendToID | int | Идентификатор пользователя |

В таблице SystemList хранятся системные таблицы, используемые в программе (табл. 7.5). В поле Title заносится название таблицы.

Таблица 7.5. Структура таблицы SystemList

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор записи |
| Title | varchar | Название |
| Del\_fl | bit | Флаг удаления |

Таблица UploadedFile содержит информацию о загруженных файлах (табл. 7.6). В поле GroupID заносится идентификатор типа файла (исходный документ, элемент подписи, страница документа и пр.). Поле DocumentID показывает идентификатор документа, к которому относится данный файл.

Таблица 7.6. Структура таблицы UploadedFile

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор файла |
| GroupID | int | Идентификатор типа документа |
| DocumentID | int | Идентификатор документа |
| Page | int | Номер страницы |

Продолжение табл. 7.6.

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| UploadedDate | datetime | Дата |
| FileName | varchar | Название файла |
| Del\_fl | bit | Флаг удаления |
| ContentType | varchar | Тип контента |
| UserID | int | Идентификатор пользователя |

Таблица SystemListValue содержит информацию о загруженных файлах (табл. 7.7). В поле SystemListID заносится идентификатор одной из системных таблиц (например, состояние документа, состояние заявки и пр.).

Таблица 7.7. Структура таблицы SystemListValue

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор файла |
| SystemListID | int | Идентификатор системной таблицы |
| Title | varchar | Название |
| Del\_fl | bit | Флаг удаления |

В таблице Request хранятся отправленные запросы на подпись документов (табл. 7.8). В поле StatusID записывается идентификатор системного значения, показывающего статус запроса (например, запрос отправлен, запрос отменен).

Таблица 7.8. Структура таблицы Request

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор запроса |
| AddressesBookID | int | Идентификатор записи в адресной книге |
| Date | datetime | Дата отправки запроса |
| DocumentID | int | Идентификатор документа |
| StatusID | int | Идентификатор статуса заявки |
| Del\_fl | bit | Флаг удаления |

В таблице RequestDocContent хранятся элементы, прикрепленные к документу, отправленному на подпись (табл. 7.9). Например, элемент подписи или элемент для написания текста.

Таблица 7.9. Структура таблицы RequestDocContent

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор контента |
| RequestID | int | Идентификатор запроса |
| ContentTemplateID | int | Идентификатор элемента |
| Value | varchar | Значение |
| Del\_fl | bit | Флаг удаления |

В таблицу ContentTemplate записана информация о шаблонах элементов, которые прикреплены к документам (табл. 7.10).

Таблица 7.10. Структура таблицы ContentTemplate

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор файла |
| DocumentID | int | Идентификатор документа |
| ContentID | int | Идентификатор типа элемента |
| Page | int | Номер страницы документа |
| Left | float | Расположение элемента относительно левого края |
| Zindex | float | Расположение элемента относительно левого края |
| Width | float | Ширина элемента |
| Height | float | Высота элемента |
| Text | varchar | Текст в элементе |
| Required\_fl | bit | Флаг необходимости |
| Del\_fl | bit | Флаг удаления |

Таблица ContentType содержит типы элементов, которые могут быть прикреплены к документам во время их редактирования. Также в этой таблице задаются значения по умолчанию для каждого элемента (табл. 7.11).

Таблица 7.11. Структура таблицы ContentType

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор документа |
| Title | varchar | Название элемента |
| Class | varchar | Класс элемента |
| DefaultWidth | float | Длина элемента по умолчанию |
| DefaultHeight | float | Ширина элемента по умолчанию |
| DefaultValue | varchar | Значение элемента по умолчанию |
| Del\_fl | bit | Флаг удаления |

## 7.5. Общая архитектура и схема работы приложения

Взаимодействие внутренних компонентов системы схематично представлено на рис. 7.4.

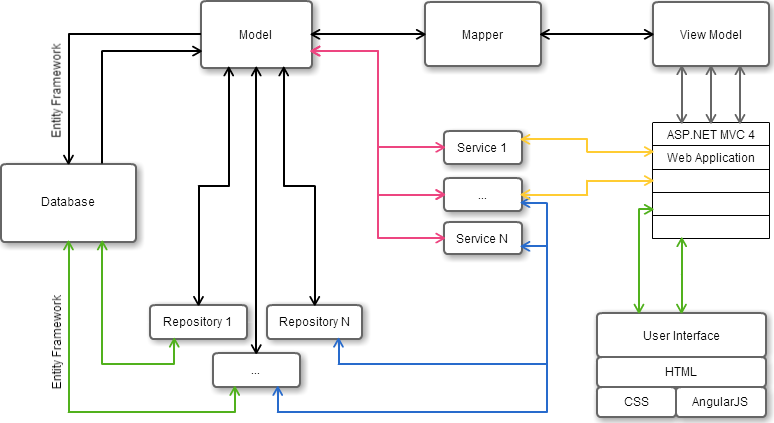


Рис. 7.4. Структура взаимодействия компонентов внутри системы

База данных связывается с моделью при помощи технологии Entity Framework. Далее, создаются репозитории (содержат методы для доступа к базе данных) для каждой таблицы из базы данных. И, с помощью Dependency Injection (внедрение зависимостей), внедряем каждый репозиторий в сервис.

После этого, также внедряем зависимости между сервисами и контроллерами (модуль управления вводом и выводом данных).

## 7.6. Модель

Entity Framework позволяет использовать собственные доменные классы для представления модели, на которую опирается EF при выполнении запросов, отслеживании изменений и обновлении функций. Code First использует шаблон программирования под названием «соглашение выше конфигурации». Это означает, что Code First предполагает, что классы следуют соглашениям, применяемым платформой Entity Framework. В этом случае платформа EF сможет выработать подробные сведения, необходимые для выполнения задания. Если классы не следуют этим соглашениям, имеется возможность добавления настроек к классам для обеспечения EF необходимыми сведениями. Code First предоставляет способ DataAnnotations – добавление настроек к классам с помощью атрибутов. В качестве примера написания доменного класса и использования в нем атрибутов приведен класс User:

[Table("User")]

public class User

{

public User()

{ }

[Key]

[Column("ID", TypeName = "int")]

public int ID { set; get; }

[Column("FirstName", TypeName = "varchar")]

[MaxLength(250)]

public string FirstName { set; get; }

[Column("LastName", TypeName = "varchar")]

[MaxLength(250)]

public string LastName { set; get; }

[Column("PatronymicName", TypeName = "varchar")]

[MaxLength(250)]

public string PatronymicName { set; get; }

[Column("Email", TypeName = "varchar")]

[MaxLength(250)]

public string EMail { set; get; }

[Column("Password", TypeName = "varchar")]

public string Password { set; get; }

[Column("Del\_fl", TypeName = "bit")]

public bool IsDelete { set; get; }

[Column("Fake\_fl", TypeName = "bit")]

public bool IsFake { set; get; }

public bool InRoles(string roles)

{

return true;

}

}

Использованные в программе атрибуты:

1. [Table(string name)]. Атрибут [Table] используется для классов и определяет произвольное имя (name) для создаваемой таблицы.
2. [Column(string name, TypeName = string)]. Этот атрибут применяется для свойства классов Модели и определяет имя колонки (name), связанной с ним. TypeName – позволяет уточнить тип, который будет использоваться в таблице для данного свойства.
3. [Key]. Позволяет отметить одно или несколько свойств, которые будут рассматриваться как первичный ключ для создаваемой таблицы. Атрибут применим только к простым типам.
4. [MaxLength(int length)]. Данный атрибут устанавливает максимальную длину массива (length) и может быть использован только со свойствами типа byte[] или string. Указать максимально возможный размер (max) можно используя константу "–1" в качестве параметра length. При этом другие отрицательные величины не допустимы.
5. [ForeignKey(string name)]. Указывает навигационному свойству связанное с ним свойство под именем name, которое является внешним ключом.
6. [Required]. Указывает на то, что данное свойство и соответствующее поле в базе данных обязательно должны содержать значение, то есть не могут быть равны null [12].

## 7.7. Паттерн Репозиторий

Репозиторий – это фасад для доступа к базе данных. Весь код приложения за пределами репозитория работает с базой данных только через него. Таким образом, репозиторий инкапсулирует в себе логику работы с базой данных, это слой объектно-реляционного отображения в приложении. Более точно, репозиторий, или хранилище, это интерфейс для доступа к данным одного типа – один класс модели, одна таблица базы данных в простейшем случае. Доступ к данным организуется через совокупность всех репозиториев.

В качестве примера в приложении 1 приведена реализация интерфейсов и классов для работы с таблицами User и Request.

## 7.8. Сервисы

Сервис-слой приложения содержит интерфейсы, в которых описано, что делать с данными или, другими словами, бизнес-логика приложения. В отличие от репозиториев, сервисы практически не содержат логики взаимодействия с БД, а только предоставляют более удобный механизм для этого взаимодействия. Также сервисы содержат практические реализации этих интерфейсов.

Например, в приложении реализован сервис по работе с документами. Интерфейс IDocumentService содержит методы для получения конкретного документа, получения списка документов, определенных страниц документа и пр.

public interface IDocumentService

{

IQueryable<DocumentView> GetDocuments(User aCurrentUser, string aDocFilePath);

IQueryable<DocumentView> GetDocument(User aCurrentUser, int aDocumentID, string aDocFilePath, int aPage);

IQueryable<ContentTemplateView> GetDocumentElements(int aDocumentID, int aPage);

IQueryable<ContentTypeView> GetTemplateElements();

ContentTemplateView UpdateDocumentElement(ContentTemplateView aElement);

void UpdateDocument(DocumentView aDocument);

}

Класс DocumentService наследуется от интерфейса IDocumentService. В нем реализованы все методы, описанные выше.

public class DocumentService : IDocumentService

{

private readonly int DefaultZindex = 100;

[Inject]

public IDocumentRepository DocumentRepository { get; set; }

[Inject]

public ISystemListValueRepository SystemListValueRepository { get; set; }

[Inject]

public IUploadedFileRepository UploadedFileRepository { get; set; }

[Inject]

public IContentTemplateRepository ContentTemplateRepository { get; set; }

[Inject]

public IContentTypeRepository ContentTypeRepository { get; set; }

[Inject]

public IFileService FileService { get; set; }

public IQueryable<DocumentView> GetDocuments(User aCurrentUser, string aDocFilePath)

{

...

}

public IQueryable<DocumentView> GetDocument(User aCurrentUser, int aDocumentID, string aDocFilePath, int aPage)

{

...

}

public IQueryable<ContentTemplateView> GetDocumentElements(int aDocumentID, int aPage)

{

...

}

public IQueryable<ContentTypeView> GetTemplateElements()

{

...

}

public ContentTemplateView UpdateDocumentElement(ContentTemplateView aElement)

{

...

}

public void UpdateDocument(DocumentView aDocument)

{

...

}

}

# 8. План тестирования

Основная функциональность реализованного веб-приложения была покрыта автоматическими модульными тестами. Дополнительно был проведён ряд ручных функциональных тестов, представленных ниже.

**Тест 1**. Авторизация (верные данные).

Цель: проверка системы авторизации.

Порядок проведения: открыть страницу авторизации и ввести верный логин и пароль.

Результат: веб-приложение перенаправило пользователя на основную страницу, что соответствует требованиям.

**Тест 2**. Загрузка документа.

Цель: проверка функционала загрузки документов.

Порядок проведения: открыть страницу загрузки документов. Нажать на кнопку «Добавить файлы», в открывшемся окне выбрать документ. Нажать на кнопку «Начать загрузку».

Результат: документ появился в списке документов пользователя.

**Тест 3**. Размещение и сохранение на странице документа компонента «Подпись».

Цель: проверка возможности размещения компонентов на странице документа.

Порядок проведения: перейти на страницу со списком документов. Выбрать документ, кликнуть левой кнопкой мышки на его название. На загрузившейся странице слева будут отображаться страницы документа, справа инструменты для работы. Поместить компонент «Подпись» на странице документа, перетащив его и бросив в необходимом месте.

Результат: отображение компонента на необходимом месте. При повторном переходе из списка документов на страницу редактирования документа, элемент отображается в том же месте.

**Тест 4**. Отправка запроса на подпись документа.

Цель: проверить работу функционала отправки запросов.

Порядок проведения: перейти на страницу со списком документов. Выбрать документ с доступной кнопкой «Отправить на подпись». Ввести данные и электронную почту пользователя, которому будет отправлен запрос. Нажать кнопку «Отправить».

Результат: на почту пришел запрос с просьбой подписать документ. На странице «Запросы» появилась запись о текущем отправленном запросе.

**Тест 5**. Подписание документа.

Цель: проверка работы функционала по подписанию документа.

Порядок проведения: отправить на почту запрос на подпись документа. В полученном письме перейти по ссылке, откроется страница для подписи документа. Подписать документ и нажать на кнопку «Принять».

Результат: каждому участнику на почту пришло уведомление о том, что документ подписан. К уведомлениям был прикреплен экземпляр подписанного документа. На странице «Запросы» у запроса сменился статус на «Документ подписан» и документ доступен для скачивания.

# Заключение

В результате проделанной работы была реализована система, обладающая следующей функциональностью:

1. Авторизация пользователей в системе;
2. Загрузка документов;
3. Размещение на страницах документа компонента(-ов) для подписи;
4. Отправка документа на подпись одному человеку;
5. Отправка документа на подпись группе людей;
6. Подписание документа человеком, который:
   1. является пользователем системы;
   2. не является пользователем системы.

# Список литературы

1. Информационное агентство «Референт». – (http://www.baikaldoc.ru/).
2. Контур. – (http://kontur.ru/).
3. «наПодпись!». – (http://www.ventasign.ru/).
4. Чедвик Д. ASP.NET MVC 4: разработка реальных веб-приложений с помощью ASP.NET MVC / Д. Чедвик, Т. Снайдер, Х. Панда. – Москва : Издательский дом «Вильямс», 2013. – 432 с.
5. Freeman A. Pro ASP.NET MVC 4 / A. Freeman. – New-York : “Apress”, 2012. – 756 c.
6. Lerman J. Programming Entity Framework, 2nd Edition Building Data Centric Apps with the ADO.NET Entity Framework 4 / J. Lerman. – Sebastopol : O’Reilly Media, 2009. – 920 c.
7. Lerman J. Programming Entity Framework: Code First / J. Lerman, R. Miller. – Sebastopol: O’Reilly Media, 2009. – 192 c.
8. Google, AngularJS. HTML enhanced for web apps! – (http://angularjs.org).
9. Chacon S. Pro Git / S. Chacon, B. Straub. – New-York: “Apress”, 2012. – 282 c.
10. Дейт К.Дж. Введение в системы баз данных / К.Дж. Дейт – Москва : Издательский дом «Вильямс», 1999. – 1316 с.
11. Чень П. Модель «Сущность-связь» – шаг к единому представлению данных / П. Чень. // СУБД. – 1995. – № 3. – С. 137–158.
12. Lerman J. Programming Entity Framework: DbContext / J. Lerman, R. Miller. – Sebastopol : O’Reilly Media, 2009. – 258 c.

# Приложение 1. Листинг интерфейсов и классов

# для работы с таблицами User и Request

public interface IUserRepository

{

IQueryable<User> Users { get; }

void CreateUser(User aUser);

User GetUser(int aID);

User GetUser(string aEmail);

bool UpdateUser(User aUser);

bool DeleteUser(User aUser);

User Login(string aUserName, string aPassword);

}

public class UserRepository : IUserRepository

{

public IQueryable<User> Users

{

get

{

return context.Users;

}

}

public void CreateUser(User aUser)

{

context.Users.Add(aUser);

context.SaveChanges();

}

public User GetUser(int aID)

{

return context.Users.FirstOrDefault(x => x.ID == aID);

}

public User GetUser(string aEmail)

{

return context.Users.FirstOrDefault(x => x.EMail == aEmail);

}

public bool UpdateUser(User aUser)

{

context.Users.Attach(aUser);

var entry = context.Entry(aUser);

entry.State = System.Data.Entity.EntityState.Modified;

return Convert.ToBoolean(context.SaveChanges());

}

public bool DeleteUser(User aUser)

{

aUser.IsDelete = true;

context.Users.Attach(aUser);

var entry = context.Entry(aUser);

entry.State = System.Data.Entity.EntityState.Modified;

return Convert.ToBoolean(context.SaveChanges());

}

public User Login(string aUserName, string aPassword)

{

return context.Users.FirstOrDefault(x => x.EMail == aUserName && x.Password == aPassword);

}

}

public interface IRequestRepository

{

IQueryable<Request> Requests { get; }

void CreateRequest(Request aRequest);

Request GetRequest(int aID);

bool UpdateRequest(Request aRequest);

bool DeleteRequest(Request aRequest);

}

public class RequestRepository : IRequestRepository

{

[Inject]

public SignAppContext context { get; set; }

public IQueryable<Request> Requests

{

get

{

return context.Requests;

}

}

public void CreateRequest(Request aRequest)

{

context.Requests.Add(aRequest);

context.SaveChanges();

}

public Request GetRequest(int aID)

{

return context.Requests.FirstOrDefault(x => x.ID == aID);

}

public bool UpdateRequest(Request aRequest)

{

context.Requests.Attach(aRequest);

var entry = context.Entry(aRequest);

entry.State = System.Data.Entity.EntityState.Modified;

return Convert.ToBoolean(context.SaveChanges());

}

public bool DeleteRequest(Request aRequest)

{

aRequest.IsDelete = true;

context.Requests.Attach(aRequest);

var entry = context.Entry(aRequest);

entry.State = System.Data.Entity.EntityState.Modified;

return Convert.ToBoolean(context.SaveChanges());

}

}

# Приложение 2. Таблица системных состояний

Системная таблица SystemList, используемая в программе для определения действий и состояний, содержит следующие группы:

1. Состояние документа.
2. Состояние запроса.
3. Тип действия.
4. Группа загруженного файла.
5. Тип письма.

В системной таблице SystemListValue хранятся значения для каждой из описанных выше групп, они выглядят следующим образом:

1. Состояние документа:
2. «Загружен».
3. «Редактируется».
4. «Отправлен».
5. «Готов к отправке».
6. Состояние запроса:
7. «Запрос отправлен».
8. «Запрос отменен».
9. «Документ просмотрен».
10. «Документ подписан».
11. Тип действия:
12. «Запись добавлена».
13. «Запись изменена».
14. «Запись удалена».
15. «Запрос отправлен».
16. Группа загруженного файла:
17. «Исходный документ».
18. «Страница документа».
19. «Страница документа в низком качестве».
20. «Подпись».
21. «Окончательный документ».
22. Тип письма:
23. «Регистрация».
24. «Запрос на подпись».
25. «Подписанный документ для подписавшего».
26. «Подписанный документ для отправителя запроса».

# Приложение 3. Пример представления страниц логина и регистрации

<div class="container" ng-show="vm.isShowLogin">

<form class="form-signin" action="Login/Login" method="POST">

<h2 class="form-signin-heading">Вход</h2>

<label for="txtEmail" class="sr-only">

Email address

</label>

<input type="email" id="txtEmail" name="txtEmail" class="form-control" placeholder="Email адрес" required="" autofocus="">

<label for="txtPassword" class="sr-only">

Password

</label>

<input type="password" id="txtPassword" name="txtPassword" class="form-control" placeholder="Пароль" required="">

<div class="checkbox">

<label>

<input type="checkbox" id="chkRememberMe" name="chkRememberMe" value="rememberMe"> Запомнить меня

</label>

</div>

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

<div>

<a class="pull-left" ng-click="vm.showForgotPassword()" href="#">Забыли пароль?</a>

<a class="pull-right" ng-click="vm.showRegistration()" href="#">Регистрация</a>

</div>

</form>

</div>

<div class="container registration" ng-show="vm.isShowRegistration">

<form class="form-signin" action="Login/Register" method="POST">

<h2 class="form-signin-heading">Регистрация</h2>

<div class="row">

<a class="pull-left" ng-click="vm.showLogin()" href="#">< Назад</a>

</div>

<div class="row">

<label for="txtFirstName" class="sr-only">

Имя

</label>

<input type="text" id="txtFirstName" name="txtFirstName" class="form-control" placeholder="Введите ваше имя" required="" autofocus="">

</div>

<div class="row">

<label for="txtSecondName" class="sr-only">

Фамилия

</label>

<input type="text" id="txtSecondName" name="txtSecondName" class="form-control" placeholder="Введите вашу фамилию" required="" autofocus="">

</div>

<div class="row">

<label for="txtLastName" class="sr-only">

Отчество

</label>

<input type="text" id="txtLastName" name="txtLastName" class="form-control" placeholder="Введите ваше отчество" required="" autofocus="">

</div>

<div class="row">

<label for="txtEmail" class="sr-only">

Email

</label>

<input type="text" id="txtEmail" name="txtEmail" class="form-control" placeholder="Введите ваш email" required="" autofocus="">

</div>

<div class="row">

<div class="g-recaptcha" data-sitekey="6Lec3AQTAAAAAC\_oNumvw7SEuuBXWRbIBhThUNgg"></div>

</div>

<div class="row">

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

</div>

</form>

</div>

<div class="container" ng-show="vm.isShowForgotPassword">

<form class="form-signin" action="Login/Login" method="POST">

<h2 class="form-signin-heading">Забыли пароль?</h2>

<div class="row">

<a class="pull-left" ng-click="vm.showLogin()" href="#">< Назад</a>

</div>

</form>

</div>