Загрузка и конвертация файлов

Пользователь, зарегистрированный в системе, может загружать необходимые ему для подписи документы. Для этого нужно в меню «Документы», выбрать раздел «Загрузить документ» (рис. \*):

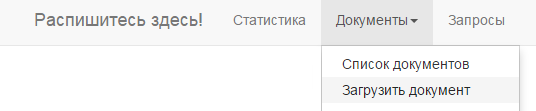
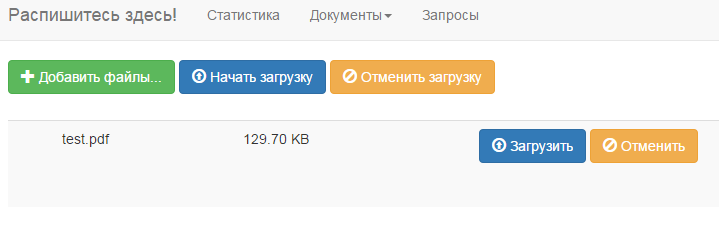


Рис. \*. Меню загрузки документов

Перейдя на страницу загрузки документов и выбрав необходимые файлы, пользователь нажимает на кнопку «Загрузить» или «Начать загрузку» (рис. \*): Рис. \*. Загрузка документа

Далее, происходит процесс загрузки файла в систему и его конвертация. Условно, данный процесс можно разделить на следующие части:

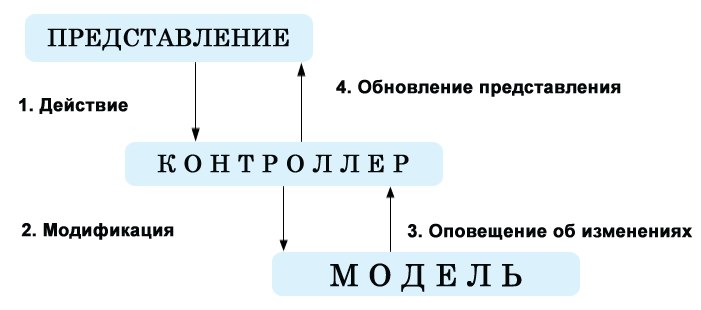
1. Конвертация исходного файла постранично в файлы формата «.art». Файлы данного формата представляют собой векторное изображение, поэтому конвертация происходит без потери качества изображения.
2. Конвертация исходного файла постранично в файлы формата «.jpg». Данные файлы конвертируются в плохом качестве и будут использоваться только для превью документа.

2. Анализ выбранных технологий

## 2.1. Среда разработки Visual Studio 2012

Microsoft .NET Framework - это платформа для создания, развертывания и запуска Web-сервисов и приложений. Она предоставляет высокопроизводительную, основанную на стандартах, многоязыковую среду, которая позволяет интегрировать существующие приложения с приложениями и сервисами следующего поколения, а также решать задачи развертывания и использования интернет-приложений. .NET Framework состоит из трех основных частей - общеязыковой среды выполнения (common language runtime), иерархического множества унифицированных библиотек классов и компонентной версии ASP, называемую ASP.NET.

## 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. Стандартная схема архитектуры «Модель-Вид-Контроллер» изображена на следующем рис. \*:  Рис. 2.3. 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).

## 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.

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. \*).

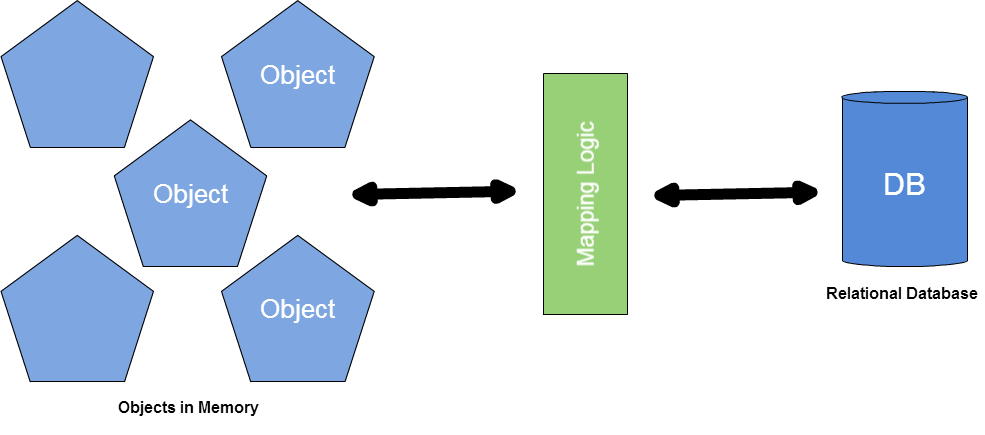


Рис. 2.1. Функционирование ORM системы

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).

### 2.1.1. Шаблоны разработки Entity Framework

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

* 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.

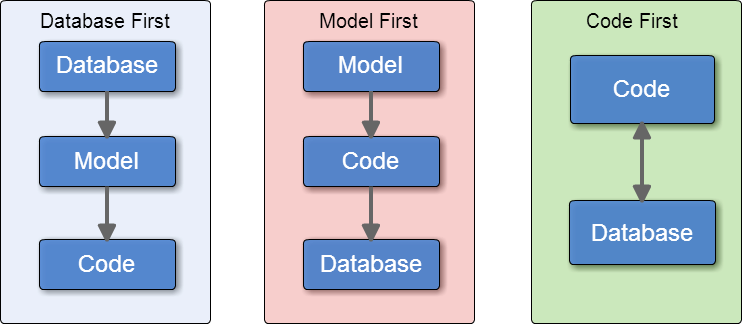


Рис. 2.2. Шаблоны разработки, предлагаемые Entity Framework

### 2.1.2. Методика Code First

## 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.

Схема работы AngularJS представлена на рис. 2.4:

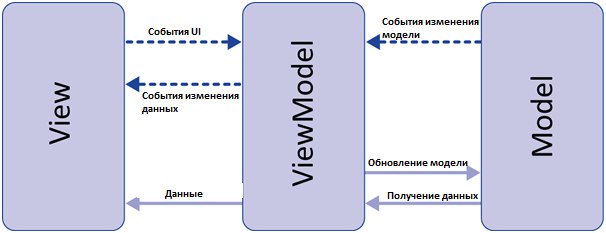


Рис. 2.4. Структурная схема работы AngularJS

# 4. Проектирование баз данных

Процесс, в ходе которого решается, какой вид будет у вновь создаваемой БД, называется проектированием базы данных. На этапе проектирования необходимо предусмотреть все возможные действия, которые могут возникнуть на различных этапах жизненного цикла БД.

## 4.1. Анализ предметной области и запросов к БД

На данном этапе необходимо проанализировать запросы пользователей, выбрать информационные объекты и их характеристики и на основе анализа структурировать предметную область.

Анализ предметной области целесообразно разбить на три фазы:

* Анализ концептуальных требований и информационных потребностей;
* Выявление информационных объектов и связей между ними;
* Построение концептуальной модели предметной области и проектирование концептуальной схемы БД.

## 4.2. Анализ предметной области и запросов к БД

На этапе анализа концептуальных требований и информационных потребностей необходимо решить следующие задачи:

* Анализ требований пользователей к БД (концептуальных требований);
* Выявление имеющихся задач по обработке информации, которая должна быть представлена в БД (анализ приложений);
* Выявление перспективных задач (перспективных приложений);
* Документирование результатов анализа.

Требования пользователей к разрабатываемой БД представляют собой список запросов с указанием их интенсивности и объемов данных. Эти сведения разработчики получают в диалоге с будущими пользователями БД.

Здесь же выясняются требования к вводу, обновлению и корректировке информации. Требования пользователей уточняются и дополняются при анализе имеющихся и перспективных приложений [4].

## 4.3. Выявление информационных объектов и связей между ними

Вторая фаза анализа предметной области состоит в выборе информационных объектов, задании необходимых свойств для каждого объекта, выявлении связей между объектами, определении ограничений, накладываемых на информационные объекты, типы связей между ними, характеристики информационных объектов.

При выборе информационных объектов необходимо ответить на ряд вопросов:

1. На какие таблицы можно разбить данные, подлежащие хранению в БД?
2. Какое имя можно присвоить каждой таблице?
3. Какие наиболее интересные характеристики (с точки зрения пользователя) можно выделить?
4. Какие имена можно присвоить выбранным характеристикам?

В ходе процесса выделения связи между информационными объектами необходимо ответить на следующие вопросы:

1. Какие типы связей между информационными объектами?
2. Какое имя можно присвоить каждому типу связей?
3. Каковы возможные типы связей, которые могут быть использованы впоследствии?

Попытка задать ограничения на объекты, их характеристики и связи приводит к необходимости ответа на следующие вопросы:

1. Какова область значений для числовых характеристик?
2. Каковы функциональные зависимости между характеристиками одного информационного объекта?
3. Какой тип отображения соответствует каждому типу связей? При проектировании БД существуют взаимосвязи между информационными объектами трех типов: «один к одному», «один ко многим», «многие ко многим» (рис. \*). Например:



Рис. \*. Типы взаимосвязей между информационными объектами

Каждая связь может иметь одну из двух модальностей связи (рис. \*):

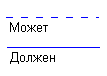


Рис. \*. Типы модальности связей

Модальность "может" означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром.

Модальность "должен" означает, что экземпляр одной сущности обязан быть связан не менее чем с одним экземпляром другой сущности.

Связь может иметь разную модальность с разных концов.

## 4.4. 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. \*):

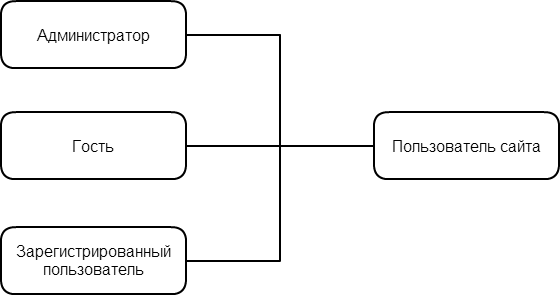


Рис. \*. Обобщение информационных объектов в родовой объект

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.

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 \* изображена концептуальная модель “Сущность-связь”, которая представлена в виде графической схемы. Необходимо доделать схему сделать может-должен связи

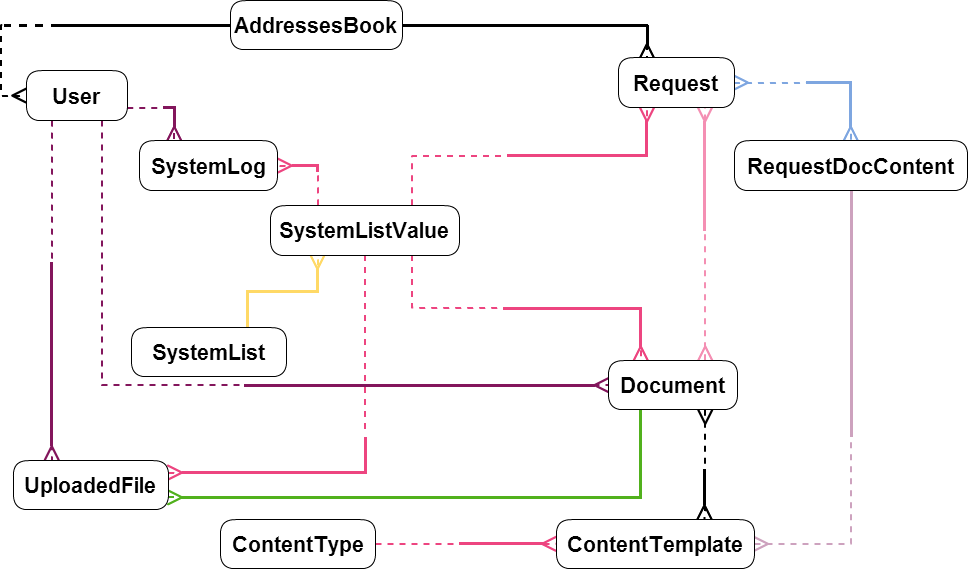


Рис. \* Модель “Сущность-связь”

## 4.5. Выбор конкретной СУБД

Одним из основных критериев выбора СУБД 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.

## 4.6. Отображение концептуальной схемы на логическую схему

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, 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.

Разработанная выше диаграмма является концептуальной (рис. \*). Это означает, что диаграмма не учитывает особенности конкретной СУБД. По данной концептуальной диаграмме можно построить физическую диаграмму, которая 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. \*. Переделать модель

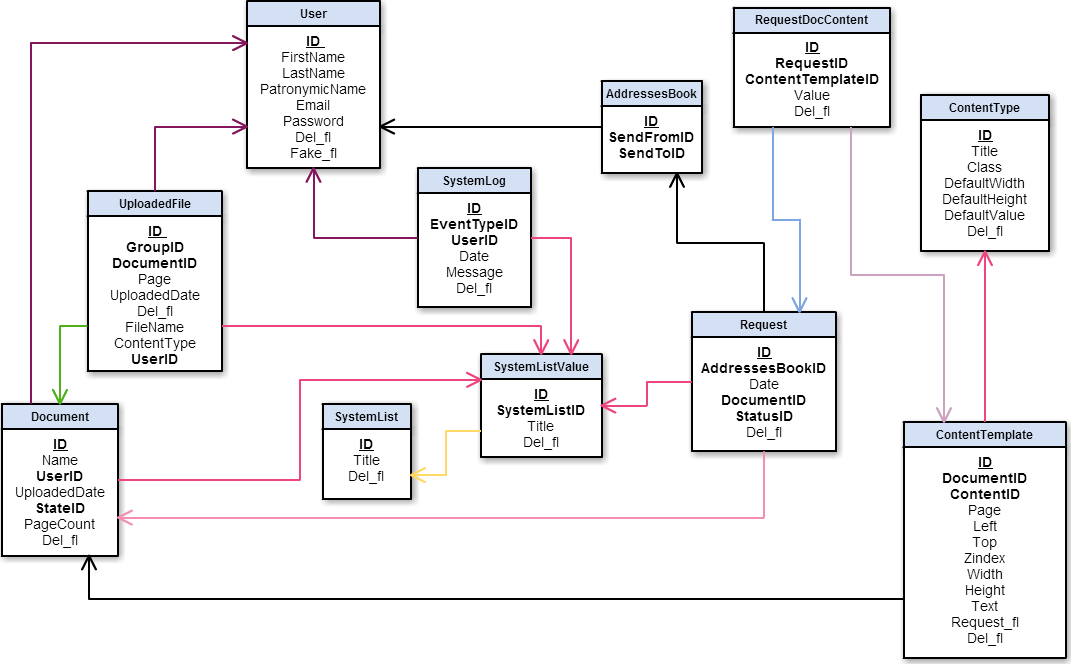


Рис. \* Концептуальная схема базы данных web-приложения

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.

## 4.7. Разработка базы данных web-приложения

Проектирование базы данных обычно начинается с построения ее концептуальной схемы, на которой реляционные таблицы представлены в виде классов (сущностей), а логические соединения между таблицами изображены линиями, имитирующими двунаправленные или однонаправленные отношения между сущностями. Данная схема представлена на рисунке %выше%.

База данных насчитывает %???? Число таблиц % реляционную таблицу. Ниже приведено описание таблиц, которое включает имена таблиц, название и описание полей каждой таблицы.

Таблица User содержит информацию обо всех пользователях (табл. 4.1). В поле Fake\_fl заносится показатель: зарегистрирован пользователь в системе или нет.

Таблица 4.1. 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. 4.2). В поле ID заносится идентификатор записи.

Таблица 4.2. 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. 4.3). В поле UserID заносится идентификатор пользователя, загрузившего файл. Поле StateID содержит системное значение о текущем состоянии документа.

Таблица 4.3. 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. 4.4). В поле SendFromID заносится идентификатор пользователя, отправлявшего запрос. Поле SendToID содержит идентификатор получателя запроса.

Таблица 4.4. Структура таблицы AddressesBook

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

В таблице System List хранятся системные таблицы, используемые в программе (табл. 4.5). В поле Title заносится название таблицы.

Таблица 4.5. 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. 4.6). 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 4.6. Структура таблицы UploadedFile

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Содержание** |
| ID | int | Идентификатор файла |
| GroupID | int | Идентификатор типа документа |
| DocumentID | int | Идентификатор документа |
| Page | int | Номер страницы |
| 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. 4.7). В поле SystemListID заносится идентификатор одной из системных таблиц (например, состояние документа, состояние заявки и пр.).

Таблица 4.7. 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. 4.8). В поле StatusID записывается идентификатор системного значения, показывающего статус запроса (например, запрос отправлен, запрос отменен).

Таблица 4.8. 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. 4.9). Например, элемент подписи или элемент для написания текста.

Таблица 4.9. 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. 4.10).

Таблица 4.10. 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, прикрепляемых к документам, и их значения по умолчанию (табл. 4.11).

Таблица 4.11. Структура таблицы ContentType

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