БУТКИНА А. А., ТРЕМАСКИН К. Д., ШАМАЕВ А. В.

РАЗРАБОТКА ПРОГРАММНО-ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ АВТОМАТИЗАЦИИ ВЕДЕНИЯ АРХИВА ЛИЧНЫХ ДОКУМЕНТОВ С ИСПОЛЬЗОВАНИЕМ СИСТЕМЫ ТЕГОВ

Аннотация. В статье описана разработанная авторами программно-информационная система, предназначенная для автоматизации процесса ведения архива личных документов с использованием системы тегов, повышающих эффективность фильтрации данных. Описана реализация интеграции разработанного веб-приложения с Яндекс Диском.

Ключевые слова: веб-приложение, архив, документы, автоматизация, система тегов, PHP, Laravel, базы данных, СУБД MySQL, облачные сервисы, Яндекс Диск.

BUTKINA A. A., TREMASKIN K. D., SHAMAEV A. V.

DEVELOPMENT OF SOFTWARE INFORMATION SYSTEM TO AUTOMATE MAINTENANCE OF PERSONAL DOCUMENTS ARCHIVE USING TAG SYSTEM

Abstract. The article presents the software information system developed by the authors and designed to automate the process of maintaining an archive of personal documents using a tag system to improve the efficiency of data filtering. The implementation of the integration of the developed web application with Yandex Disk is described.

Keywords: web application, archive, documents, automation, tag system, PHP, Laravel, database, MySQL DBMS, cloud services, Yandex Disk.

Введение. Каждый человек в повседневной жизни человек рано или поздно сталкивается с проблемой удобного ведения архива личных документов. К документам любого личного архива относятся: биографические данные (свидетельства, удостоверения, справки, результаты исследований и т.п.), документы служебной и общественной деятельности, переписка, фотографии, изобразительные материалы. При этом зачастую происходят события (например, оформление наследства), связанные с оформлением, хранением и обработкой большого количества документов. Не запутаться в таком множестве разнообразных документов без использования специального инструментария для их упорядоченного хранения крайне тяжело.

Ведение архива личных документов только в физическом виде является крайне нерациональным, так как в любой момент может возникнуть необходимость в получении копии определённого документа или проверке наличия искомого документа в архиве. Использование в качестве такого архива существующих облачных хранилищ в том виде, в каком они представлены сейчас, так же представляет собой неидеальный вариант, так как с

ростом количества документов, будет все сложнее выполнять добавление новых файлов и поиск существующих. Об эффективном хранении локальных электронных копий документов на домашнем компьютере или смартфоне тем более говорить не приходится.

Данное исследование посвящено созданию программно-информационной системы в виде веб-приложения, предназначенного для автоматизации процесса ведения архива личных документов пользователя с использованием системы тегов, обеспечивающих удобную и эффективную фильтрацию данных. Данное решение позволит пользователям упростить ориентирование в архиве своих документов благодаря наличию системы тегов и грамотно организованному пользовательскому интерфейсу. При этом пользователь сможет буквально в несколько кликов находить именно то, что ему необходимо, что позволит сэкономить время и усилия, затраченные на поиск, по сравнению, например, с использованием того же физического архива.

Актуальность работы состоит в том, что существующие сервисы, предназначенные для хранения данных, неудобны при работе с большим количеством файлов, а их системы фильтрации неэффективны, что доставляет массу неудобств пользователям. Также следует отметить, что сильная вовлеченность людей в использование интернет-технологий позволит пользователям иметь оперативный доступ к своим документам в любой момент времени.

Для достижения указанной цели были поставлены и решены следующие ***задачи***:

- выполнить анализ предметной области;
- выполнить анализ и выбор инструментов разработки;
- описать входные данные и построить базу данных системы;
- разработать архитектуру системы с помощью UML-диаграмм;
- реализовать автоматизированное заполнение базы данных (БД) файлами, размещенными в облачном хранилище пользователя;
- реализовать систему на выбранных языках программирования;
- выполнить проверку работоспособности разработанной системы.

Анализ предметной области. Первоочередным вопросом, вставшим перед авторами, являлся выбор облачного хранилища, предназначенного для интеграции с разрабатываемой программно-информационной системой. Были изучены научные статьи, посвященные указанной тематике, в частности, в статье [1] рассмотрены облачные хранилища, которые являются наиболее популярными на сегодняшний день – Dropbox, Яндекс Диск, GoogleDrive, MicrosoftSkyDrive и SugarSync, выполнено сравнение их характеристик и предлагаемого ими функционала: тарифные планы, предоставляемые объемы памяти, скорость соединения и стабильность работы серверов. Авторами также было выполнено собственное исследование описанных сервисов, по результатам которого можно сформулировать следующие выводы.

На сегодняшний день отсутствуют по-настоящему удобные и эффективные сервисы для ведения личного архива документов. Рассмотренные облачные хранилища либо не имеют систем фильтрации, либо имеют их весьма неудачные реализации. Отметим, что на данный момент не существует сервисов, специализирующихся на хранении документов. В качестве оптимального решения для интеграции разрабатываемой системы с облачным хранилищем был выбран отечественный сервис Яндекс Диск.

Выбор технологии реализации. Так как разрабатываемое приложение включает клиентскую и серверные части, далее опишем инструменты, выбранные для их разработки.

Для реализации серверной части могут быть использованы различные инструменты и языки программирования. Наиболее популярными языками программирования для backend разработки в настоящее время являются Java, Ruby, Golang, PHP, Python. У каждого из них есть свои преимущества и недостатки, начиная с распространенности и заканчивая специфичностью работы. В техническом плане для большинства проектов отсутствуют какие-либо ограничения на выбор языка – практически любой функционал может быть успешно реализован на любом серверном языке, поэтому выбор языка не накладывает никаких ограничений на проект. Однако разница существует с экономической точки зрения. Одним из наиболее распространенных языков программирования в России на сегодня является PHP, поэтому при его использовании достаточно просто найти как отдельных программистов, так и аутсорсинговые компании, которые смогут работать над проектом. Из минусов — порог вхождения в сферу PHP-программирования невысок, поэтому на рынке присутствует немало «непрофессионалов». Если рассматривать в качестве альтернативы другие языки из представленного выше списка, то средний уровень специалистов на кадровом рынке, как правило, выше, но найти их уже не так просто и их услуги стоят несколько дороже. Наш проект, как и любой другой, необходимо сопровождать в ходе эксплуатации, поэтому экономическая составляющая вопроса занимает не последнее место.

С точки зрения разработчиков наилучшим вариантом также будет PHP. Для него существует хорошо описанная документация, существует большое сообщество программистов. Кроме того, PHP-приложение будет относительно просто адаптировать при реализации сервера для мобильного приложения, что несомненно является большим преимуществом. Резюмируя выбор языка программирования, можно сказать, что ***PHP*** является наиболее оптимальным вариантом для реализации нашего проекта.

В качестве фреймворка для разработки был выбран ***Laravel***, так как в настоящее время ему не существует значимой альтернативы, поскольку остальные фреймворки существенно уступают ему по предоставляемому функционалу. В качестве отличительных особенностей Laravel можно выделить его простоту и наличие большого количества модулей.

Далее был выполнен выбор системы управления базами данных (СУБД) с учетом того ограничения, что в качестве основного языка программирования будет использоваться PHP. Были проанализированы такие СУБД как MySQL, MariaDB, PostgreSQL, SQLite, MongoDB, Redis, CouchDB. В результате проведенного анализа была выбрана СУБД ***MySQL***.

В статье [2] проводится анализ двух популярных механизмов хранения данных (движков) в MySQL – InnoDB и MyISAM. Первый является механизмом по умолчанию в MySQL 5.1 и не поддерживает механизмы транзакций и внешних ключей. Его преимущество заключается в том, что он имеет высокую скорость доступа. Второй, InnoDB, более распространен на практике – он поддерживает транзакции, внешние ключи, блокировку на уровне строк, однако является более медленным в работе, по сравнению с MyISAM. Поэтому, было решено использовать оба движка: InnoDB – для всей информации проекта, а MyISAM – для хранения информации, к которой необходим постоянный доступ на чтение.

При разработке frontend части веб-приложения использовался фреймворк ***Bootstrap.***

В качестве дополнительных инструментов разработки использовались интегрированная среда разработки PhpStorm, редакторы Visual Studio Code и Notepad++ (для работы с конфигурационными файлами PHP), система контроля версий Git, инструмент для тестирования API Postman, препроцессор SASS**.**

Построение базы данных. Эффективность любого приложения, использующего в своей работе базу данных, зависит от того, насколько оптимально спроектирована ее схема.

Поскольку разрабатываемая система должна хранить информацию о документах, пользователях, тегах, а также связи между всеми сущностями системы, то для этого необходимо создать базу данных с подходящей структурой, которая будет готова к масштабированию в ходе развития проекта. Опишем подробнее ключевые понятия:

– Тег – метка (дескриптор, описание), которая создается администраторами (подготавливаются базовые заготовки) или самими пользователями для её последующего присваивания документу или группе документов для облегчения поиска документов;

– Документ – сущность, которая содержит в себе данные о загружаемом файле (название, описание, тип и т.д.) и его расположении в хранилище на сервере или в облаке;

– Группа документов – документы, которые пользователь объединил в единое целое, например, через присваивание им тегов;

– Пользователь – объект, который представляет действующее лицо системы, это может быть как рядовой пользователь системы, так и её администратор.

В процессе разработки основного модуля веб-приложения, реализующего базовый функционал (авторизация, работа с документами, присваивание им тегов, фильтрация данных) была спроектирована схема базы данных, состоящая из семи таблиц (рис. 1).

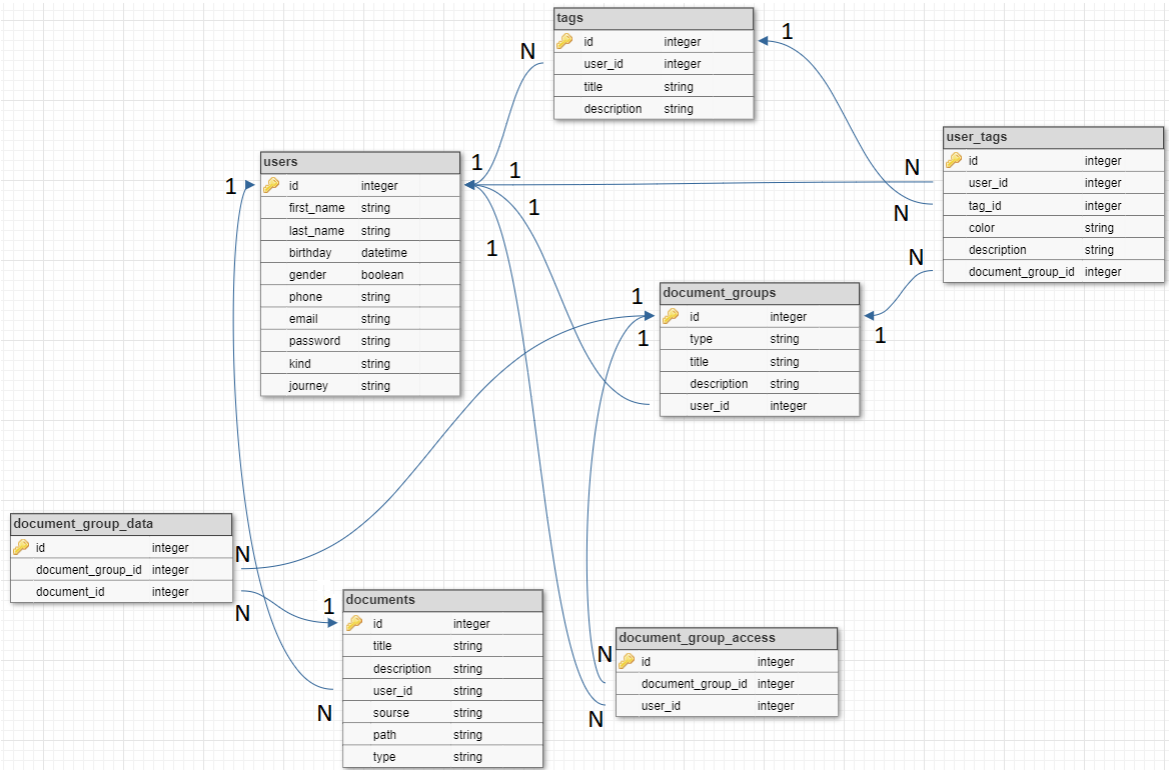


Рис. 1. Схема основного модуля базы данных.

Опишем назначение каждой из таблиц, приведенной на рисунке 1:

– Таблица «users» хранит данные обо всех пользователя и содержит поля с личной информацией, поля для аутентификации и служебные поля (тип пользователя, особые отметки). Эта таблица имеет следующие связи с другими таблицами: documents (один ко многим) – один пользователь может загрузить много документов, tags (один ко многим) – один пользователь может создать множество тегов, user_tags (один ко многим) – один пользователь может присвоить много тегов разным группам, document_group (один ко многим) – один пользователь может создать много групп документов, document_group_access (один ко многим) – один пользователь может иметь доступ ко многим группам документов;

– Таблица «documents» содержит информацию о документах, загруженных пользователями: название и описание документа, его тип, принадлежность конкретному пользователю, способ загрузки документа и полный путь к его расположению. Эта таблица имеет связь с таблицей document_group_data (один ко многим) – один документ может принадлежать ко многих группам документов одновременно;

– Таблица «documents_groups» хранит данные о группе документов, созданной пользователем, в качестве основных характеристик которой задаются её тип, название и описание. Эта таблица имеет такие связи с другими таблицами: document_group_data (один ко многим) – одна группа документов может содержать много документов, user_tags (один ко многим) – одной группе может быть присвоено несколько тегов, document_group_access (один ко многим) – доступ к одной группе могут иметь много пользователей;

– Таблица «documents_group_data» содержит информацию о группах – это связи группы и конкретных документов, где каждая запись содержит id документа и группы, к которой он отнесен. Документ может быть отнесен сразу к нескольким группам;

– Таблица «documents_group_access» хранит данные о пользователях, которые имеют доступ к этой группе документов. Каждая запись в данной таблице содержит id группы документов и id пользователя, к которому она относится;

– Таблица «tags» хранит данные о базовых и публичных тегах, которые может использовать каждый пользователь, а также основную модель тега, которая включает название, описание и автора тега. Эта таблица имеет связь с таблицей user_tags (один ко многим) – один базовый тег может быть использован многими пользователями;

– Таблица «user_tags» содержит теги, которые использует только определенный пользователь или группа документов. Данная модель может существовать как независимо, так и базироваться на модели из таблицы «tags», «наследуя» от нее различные свойства. В таблице также задаются дополнительные параметры тега (например, цвет отображения тега).

Создание таблиц осуществлялось с помощью ***механизма миграций***. Миграции позволяют определять схемы базы данных приложения и совместно использовать их. Чтобы сгенерировать новую миграцию базы данных, использовалась команда `php artisan make:migration`. Эта команда помещает новый класс миграции в каталог приложения `database/migrations`. Каждое имя файла миграции содержит временную метку, которая позволит Laravel определять порядок применения миграций. Дополнительно в каждую таблицу были добавлены timestamps-поля `created_at`, `updated_at` и `deleted_at`, которые необходимы для корректной работы Eloquent ORM Laravel.

После создания классов миграций для каждой из таблиц, приведенных на рисунке 1, был запущен механизм миграции консольной командой `php artisan migrate`. В результате в базе данных были созданы описанные таблицы (рисунок 2).

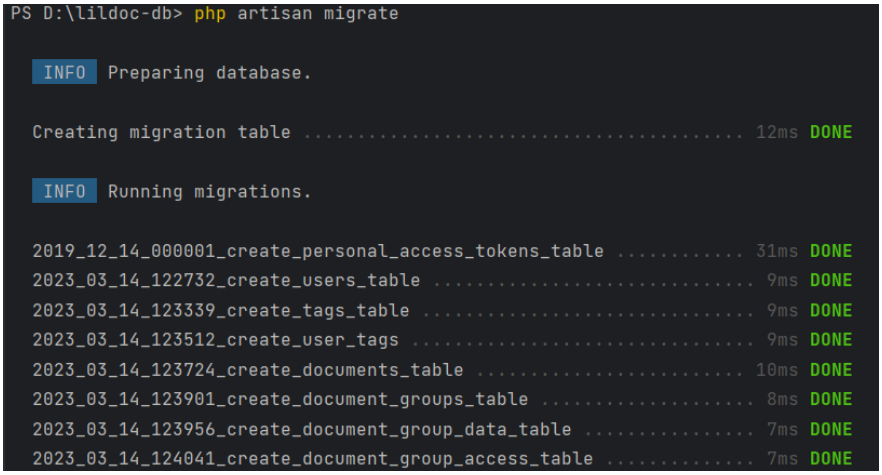


Рис. 2. Вызов консольной команды для запуска механизма миграций.

После формирования таблиц для каждой из них были созданы Eloquent модели, используемые для взаимодействия с родительскими таблицами. Помимо получения записей из таблицы БД модели Eloquent позволяют вставлять, обновлять и удалять их из таблицы. Модели расширяют класс Illuminate\Database\Eloquent\Model. Для генерации новой модели Eloquent использовалась команда `php artisan make:model`. Она помещает новый класс модели в каталог приложения app/Models. Также внутри класса модели были установлены зависимости между таблицами для организации удобного взаимодействия с данными из таблиц. Пример вызова консольной команды для создания модели представлен на рисунке 3.

```
PS D:\lildoc-db> php artisan make:model User

  INFO  Model [D:\lildoc-db\app/Models/User.php] created successfully.
```

Рис. 3. Вызов консольной команды для создания модели для таблицы «users».

После создания Eloquent моделей можно обращаться к данным таблицы, создавая экземпляр класса, которой относится к требуемой таблице.

Проектирование. Далее авторами была разработана архитектура системы с помощью следующих UML-диаграмм: диаграммы вариантов использования (рис. 4), диаграммы размещения и диаграммы компонентов. Рассмотрим их более подробно.

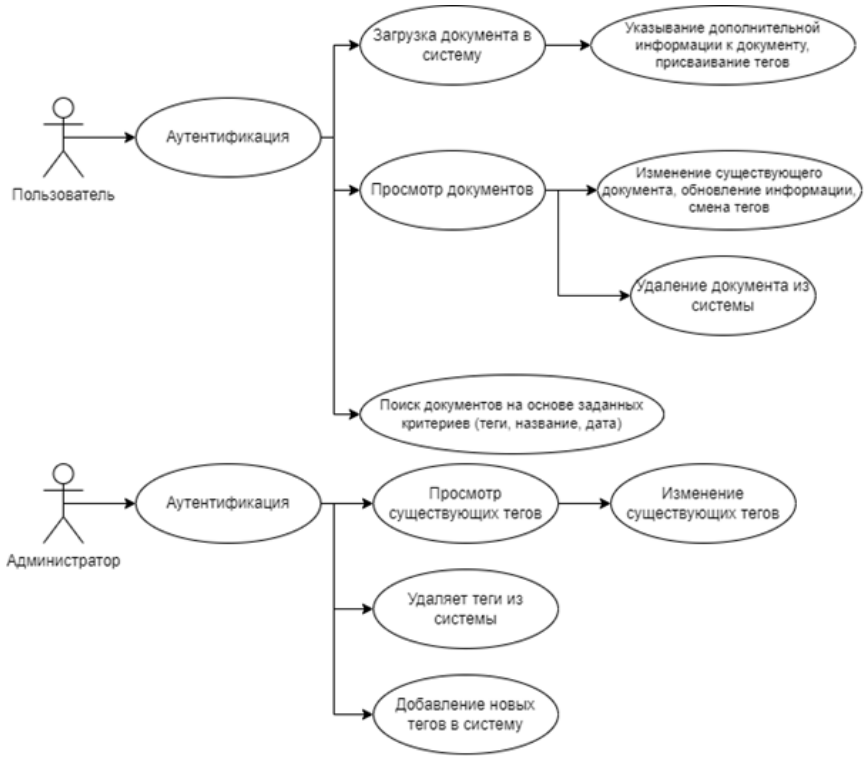


Рис. 4. Диаграмма вариантов использования.

При построении диаграммы вариантов использования были выделены два актера: администратор – актер, который имеет возможность управления базовыми тегами и фильтрами, а также пользователь – актер, использующий функционал приложения по его

назначению. Следует отметить, что доступ ко всему функционалу пользователь получает только после успешного прохождения процедуры регистрации и аутентификации. Основные сценарии для обоих актеров показаны на рисунке 4.

Диаграмма размещения является важным инструментом при проектировании и анализе системы, помогая понять её структуру и взаимодействие компонентов (рис. 5).

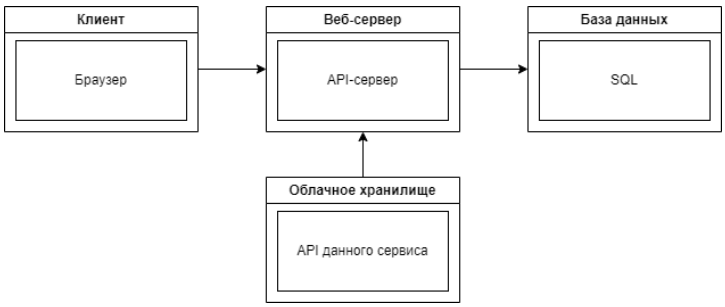


Рис. 5. Диаграмма размещения.

Связь между компонентами осуществляется путем взаимодействия через сеть интернет. В данном случае видно, что на сервере разворачивается веб-приложение, которое имеет связь с сервером базы данных. При этом пользователи обращаются к веб-серверу с различных устройств через браузеры. Опишем схему работы системы подробнее:

1. ***Клиентское приложение***. Это программное обеспечение, запущенное на устройстве пользователя и обеспечивающее интерфейс для взаимодействия с системой. С его помощью пользователь отправляет запросы на веб-сервер для обработки и получения данных. Оно также отображает результаты выполненных запросов;

2. ***Веб-сервер***. Этот компонент принимает и обрабатывает запросы от клиентского приложения, служит промежуточным звеном между клиентским приложением и БД;

3. ***База данных***. База данных является центральным хранилищем информации о личных документах. В ней хранятся данные, такие как документы, метаданные и система тегов. База данных позволяет сохранять, извлекать, обновлять и удалять данные в архиве;

4. ***Облачное хранилище***. Данный компонент позволяет получить доступ к файлам пользователя, хранящимся на стороннем сервисе (в нашем случае Яндекс Диск). Подключение к нему осуществляется через соответствующий API и используется с целью переноса файлов пользователя в разрабатываемую систему.

Опишем диаграмму компонентов для разрабатываемого веб-приложения (рис. 6):

1. ***Компоненты управления состоянием*** **(*StateManagement*)**. Используются для централизованного хранения и управления состоянием приложения.

2. ***Сетевой слой*** (***API***). Взаимодействует с серверным API для обмена данными.

3. ***Управление маршрутизацией*** **(*Router*)**. Отвечает за навигацию веб-приложения на основе текущего URL или действий пользователя.

4. ***Клиентский интерфейс*** (***UI***). Отвечает за отображение пользовательского интерфейса веб-приложения. Включает страницы, компоненты и элементы интерфейса.

5. ***Бизнес-логика и обработка данных*** (***business logic***). Выполняет бизнес-логику и обрабатывает данные приложения, например, поиск документов по тегам.

6. ***Серверная часть*** (***Backend***). Обеспечивает функциональность, требующую обработки на сервере, такую как аутентификация, хранение и обработка данных.

7. ***База данных*** (***SQL DataBase***). Хранит данные приложения, такие как информация о пользователях, документах и тегах.

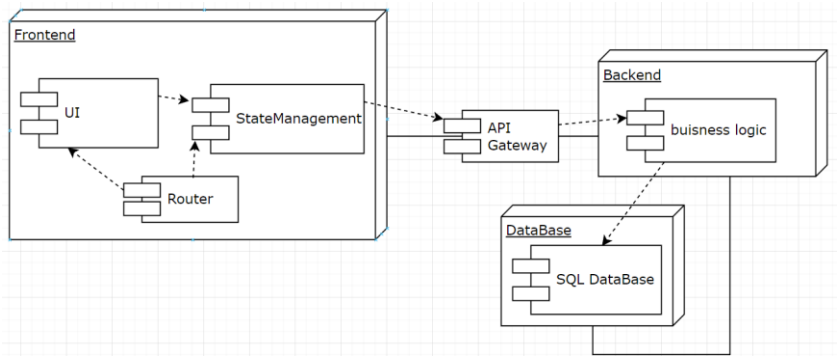


Рис. 6. Диаграмма компонентов.

Между компонентами организованы следующие связи (рис. 6):

1. Клиентский интерфейс взаимодействует с компонентами управления состоянием для отображения данных и реагирования на действия пользователей;

2. Компоненты управления состоянием: обращаются к сетевому слою для обмена данными с сервером, взаимодействуют с компонентами управления маршрутизацией для изменения отображаемого контента на основе текущего URL или действий пользователя и выполняют бизнес-логику и обработку данных, связанных с обучением и заданиями;

3. Сетевой слой взаимодействует с серверной частью для обработки запросов, аутентификации и обработки данных;

4. Серверная часть взаимодействует с БД для чтения и записи данных приложения.

Реализация приложения. Одной из основных задач данного исследования является настройка соединения с облачным сервисом (в частности рассматривается интеграция с Яндекс Диском) для получения доступа к файлам пользователя, хранящимся в облаке. Для установления стабильного соединения с Яндексом необходимо пройти несколько этапов:

1. **Регистрация приложения** в личном кабинете разработчика в Яндекс. На данном этапе были указаны запрашиваемые у пользователя права доступа к его личной информации: дате рождения, адресу электронной почты, логину, имени и фамилии, полу, номеру телефона, и права доступа к информации Яндекс Диска: доступ к папке приложения на Диске, чтение всего Диска, запись в любом месте на Диске, доступ к информации о Диске.

После регистрации приложения Яндекс сгенерировал ClientID – ключ приложения, требуемый для выполнения авторизации в приложении, и Client Secret – специальный ключ, с помощью которого можно расшифровывать данные, полученные от API Яндекса (рис.7).

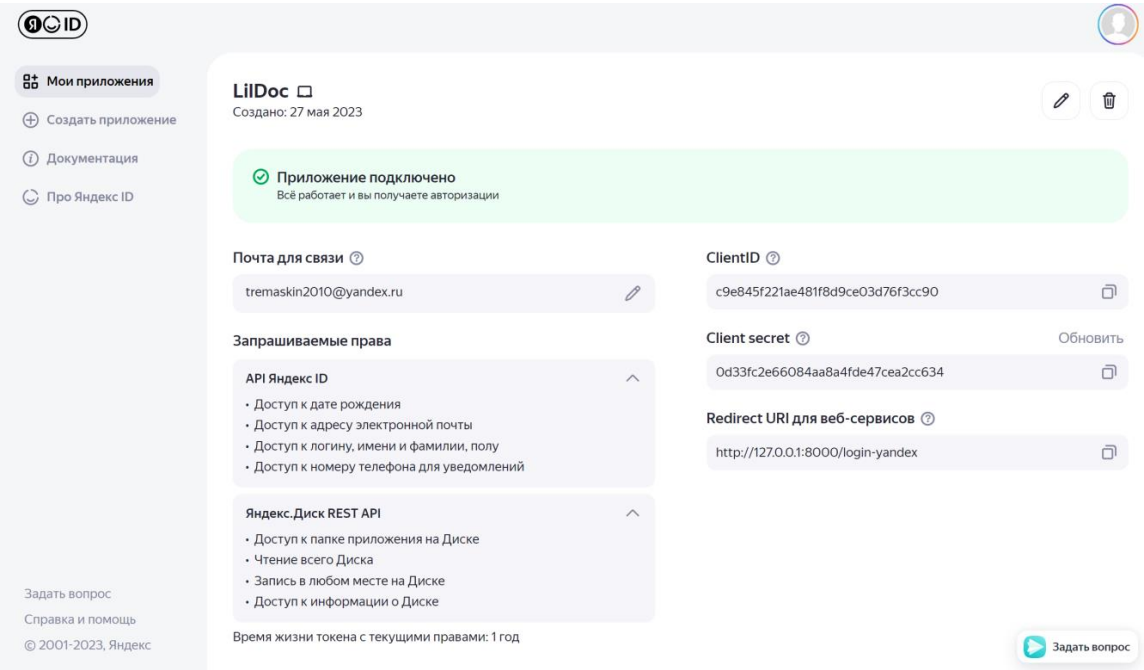


Рис. 7. Личный кабинет разработчика при регистрации приложения в Яндекс.

2. Получение OAuth-токена. Для того чтобы пользователь мог войти в разрабатываемое приложение через сервисы Яндекс, нужно настроить работу с протоколом OAuth. Для этого требуется получить OAuth-токен, который предоставляет возможность обращаться к сервисам Яндекса от имени конкретного пользователя, предоставившего соответствующие разрешения. Токен необходимо запрашивать для каждого нового пользователя, входящего через сервис Яндекс. Получить его можно несколькими способами: с помощью виджета LoginSDK (на текущий момент работает только для мобильных устройств) или используя отладочный токен. В работе был выбран вариант с использованием виджета от Яндекса. Для этого в html код страницы, содержащий виджет, добавляется код на языке JavaScript, загружающий кнопку с помощью конструкции iFrame. Данный код также позволяет управлять внешним видом кнопки и настройками авторизации в приложении.

Затем, на странице, принимающей токен, был добавлен скрипт, который «подтверждает» получение токена и загружает токен разрабатываемое приложение:

```
window.onload = function() {
    window.YaSendSuggestToken("http://127.0.0.1:8000");};
```

3. Получение пользовательской информации. После авторизации пользователя в системе с помощью Яндекса, появляется возможность отправлять запросы к API на: получение информации о пользователе, о конкретном файле, получение списка всех файлов на Диске, скачивание конкретного файла. Данные запросы осуществляются с помощью

встроенного в Laravel класса `Illuminate\Support\Facades\Http`. Он позволяет удобно конструировать запросы, задавать заголовки и передавать параметры.

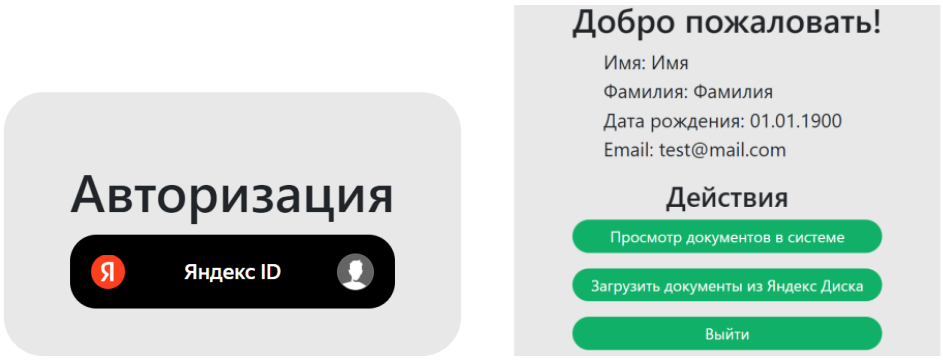
Далее была выполнена реализация основного функционала системы, заключающаяся в извлечении необходимых данных из Яндекс Диска в созданную БД, оптимизации работы с ней и организации взаимодействия между серверной частью и интерфейсом пользователя.

При разработке системы использовалась архитектуры MVC (Model-View-Controller), которая зарекомендовала себя как решение по созданию эффективной структуры приложений, позволяющее разделить бизнес-логику, работу с БД и визуальную составляющую проекта друг от друга. Это позволяет сделать код читабельным, а процесс разработки более комфортным, разграничивая работу frontend и backend разработчиков.

Для работы системы с целью аутентификации пользователя необходимо хранить и передавать с каждым запросом его ID. Также для удобства и уменьшения нагрузки на БД выполняется хранение и передача информации о токене Яндекса во время сессии.

Для реализации процесса загрузки файлов из Яндекс Диска, при получении OAuth-токена одновременно оформляется учетная запись пользователя в разрабатываемой системе. При этом выполняется проверка того, был ли пользователь авторизирован в системе ранее, и если это так – выполняется обновление информации о нем и запись данных в сессию.

Опишем алгоритм работы приложения со стороны пользователя. Работа с приложением начинается со страницы авторизации (рис. 8а), поскольку весь функционал приложения в соответствии с рисунком 4 доступен только зарегистрированным пользователям. После успешной авторизации открывается главная страница приложения, содержащая информацию о пользователе и кнопки с доступными действиями: просмотр документов в системе, загрузка документов из Яндекс Диска и выход (рис. 8б).



а) б)

Рис. 8. Страница приложения а) до авторизации и б) после успешной авторизации.

Далее рассмотрим страницу загрузки документов в систему из Яндекс Диска, где выводится список документов, кнопки для загрузки в систему всех файлов и выбранного документа (рис. 9). Для вывода списка файлов использовалась Blade директива @foreach.

а) б)

Рис. 9. Страницы приложения а) со списком документов из Яндекс Диска и б) с уведомлением об успешной загрузке документа.

Опишем алгоритм добавления тега документу. Его основная сложность заключается в определении того, как создавать тег: есть ли у пользователя нужные теги, подходят ли они, и как определить требуемый. Сначала выполняется запрос об отборе всех тегов, доступных пользователю. Он весьма ресурсоемок и деление его на части может замедлить систему, но при нормализации БД он будет выполняться быстрее. После обработки полученных данных определяется, нужно ли создавать новый тег или можно добавить тэг из имеющейся группы.

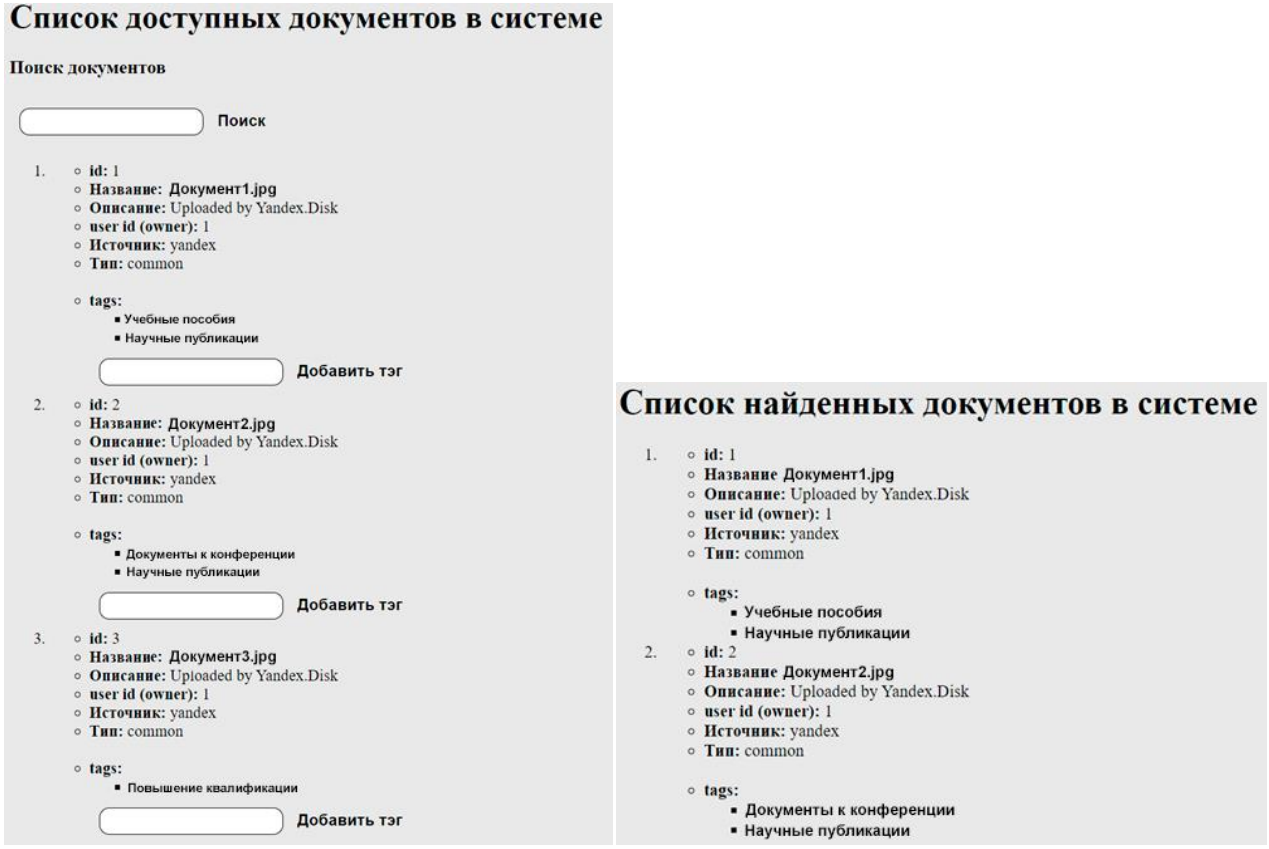
Поясним суть алгоритма поиска документов в системе: сначала выполняется проверка того, к каким группам документов есть доступ у пользователя, затем просматриваются теги, привязанные к этим группам и документов из этой группы. Можно было бы сформировать единый запрос, выполняющий описанные выше действия, однако он будет неэффективен при большом количестве данных, так как при этом потребуется выполнить обращение ко множеству таблиц. Поэтому для повышения эффективности запрос был разделен на 3 части:

1. Выбор групп документов, к которым пользователь имеет доступ. Этот запрос будет выполняться быстро, так как нет обращения к другим таблицам, и задается одно условие.

2. Поиск в описаниях документов. Выполняется обращение к таблицам document_group_data и documents, в которых ищется совпадение в заглавии и описании документа с использованием оператора like, позволяющего осуществлять поиск по шаблону.

3. Поиск по тегам. В данном запросе выполняется поиск совпадения в названиях и описаниях тегов по таблицам tags и user_tags с использованием оператора like.

Перейдем к странице с загруженными в систему документами, где отображается поле поиска документов, список документов с данными о каждом из них, и кнопки добавления тегов (рис.10а). Последняя созданная в приложении страница отображает список найденных документов. На рисунке 10б показан результат поиска по тегам «Научные публикации» и «Документы к конференции» для списка документов, отображенных на рисунке 10а.



а) б)

Рис. 10. Страницы со списком документов: а) доступных и б) найденных по тегам «Научные публикации» и «Документы к конференции».

Заключение. Все задачи, поставленные в рамках данного исследования, были решены. Результатом работы является разработанное веб-приложение, предназначенное для автоматизации процесса ведения архива личных документов пользователя с использованием системы тегов, обеспечивающих удобную и эффективную фильтрацию данных. Особенностью данного приложения, помимо использования системы тегов для организации поиска, является возможность хранения документов не только на устройствах пользователя, но и в облачном хранилище сервиса Яндекс Диск, с которым выполнена интеграция по API.

СПИСОК ЛИТЕРАТУРЫ

1. Онуфриенко С. Г., Хоняк Ю. А. Облачные хранилища данных // Материалы докладов 51-й Международной научно-технической конференции преподавателей и студентов в двух томах (г. Витебск, 25 апреля 2018 года): Том 2. – Витебск: ВГТУ, 2018. – С. 63–66.
2. Хромушкин Р. Р. Оптимизация работы MYSQL (innodb и myisam) // ADVANCED SCIENCE: сборник статей X Международной научно-практической конференции (г. Пенза, 12 ноября 2019 года). – Пенза: "Наука и Просвещение" (ИП Гуляев Г.Ю.), 2019. – С. 54–56.