Оглавление

[Анализ 3](#_Toc290133951)

[Концепция веб-сервисов 3](#_Toc290133952)

[Введение 3](#_Toc290133953)

[Основные положения модели веб-сервисов 4](#_Toc290133954)

[Серверная часть 7](#_Toc290133955)

[Анализ вариантов использования 7](#_Toc290133956)

[Выбор технологии реализации 9](#_Toc290133957)

[XML RPC в PHP 11](#_Toc290133958)

[Выбор схемы обеспечения безопасности 12](#_Toc290133959)

[CMS Joomla 13](#_Toc290133960)

[Реализация бизнес-логики на базе Joomla API 15](#_Toc290133961)

[Клиентская часть 21](#_Toc290133962)

[Анализ вариантов использования 21](#_Toc290133963)

[Выбор технологии реализации 22](#_Toc290133964)

[Обзор Adobe Lightroom 23](#_Toc290133965)

[Средства автоматизации Lightroom 24](#_Toc290133966)

[Реализация бизнес-логики на базе Lightroom API 25](#_Toc290133967)

[Взаимодействие клиента и сервера 27](#_Toc290133968)

[Общая схема взаимодействия 27](#_Toc290133969)

[Диаграмма последовательности 28](#_Toc290133970)

[Описание методов веб-сервиса 30](#_Toc290133971)

[Формат XML сообщений 31](#_Toc290133972)

[Реализация 33](#_Toc290133973)

[Серверная часть 33](#_Toc290133974)

[Классы модели 33](#_Toc290133975)

[Классы контроллеров 38](#_Toc290133976)

[Классы представлений 43](#_Toc290133977)

[Обеспечение безопасности 47](#_Toc290133978)

[Интерфейс пользователя 49](#_Toc290133979)

[Клиентская часть 51](#_Toc290133980)

[Ключевые файлы 51](#_Toc290133981)

[Интерфейс пользователя 58](#_Toc290133982)

[Механизм публикации и обратной связи 61](#_Toc290133983)

# Анализ

## Концепция веб-сервисов

### Введение

Сеть Интернет стала общепризнанным фактором деловой и общественной жизни. Широкая распространенность и возросшая пропускная способность создают условия, при которых выгодно решать многие задачи при помощи интернет-технологий.

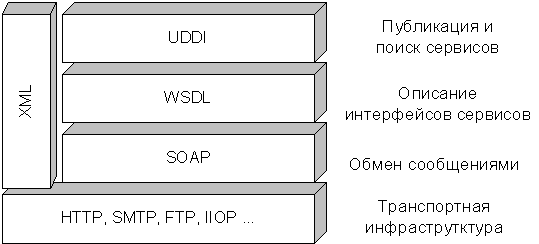
Однако Интернет объединяет в себе много различных платформ, а информация содержится в разнообразных источниках данных. Поэтому актуальна проблема связи таких разнородных данных, а также создания способа, который позволяет получать их в виде удобном для дальнейшей обработки.

Концепция веб-сервисов (Web Services) призвана решить эту задачу объединения, интеграции разнородных систем на основе открытых стандартов. Данная работа посвящена веб-сервисам, в ней кратко рассмотрены основные положения модели веб-сервисов, а также компоненты этой модели и технологии, используемые для их реализации. Практическая часть работы содержит небольшой пример, демонстрирующий разработку веб-сервиса и приложений использующих его. Пример основывается на реализации концепции веб-сервисов в рамках Java-технологий. Для понимания примера достаточно базовых знаний Java.

### Основные положения модели веб-сервисов

Веб-сервисы являются концепцией создания таких приложений, функции которых можно использовать при помощи стандартных протоколов Интернет. В настоящее время эту концепцию применяют и развивают многие ведущие компании в IT-области. Концепция веб-сервисов реализуется при помощи ряда технологий, которые стандартизованы World Wide Web Consortium (W3C).

Взаимосвязь этих технологий можно условно представить следующим образом.



Веб-сервисы являются одним из вариантов реализации компонентной архитектуры.

XML является фундаментом для создания большинства технологий, связанных с веб-сервисами.

Для удаленного взаимодействия с веб-сервисами используется Simple Object Access Protocol (SOAP). SOAP обеспечивает взаимодействие распределенных систем, независимо от объектной модели, операционной системы или языка программирования. Данные передаются в виде особых XML документов особого формата.

Согласно определению W3C, веб-сервисы это приложения, которые доступны по протоколам, которые являются стандартными для Интернет. Нет требования, чтобы веб-сервисы использовали какой-то определенный транспортный протокол. Спецификация SOAP определяет, каким образом связываются сообщения SOAP и транспортный протокол.

Наиболее часто реализуется передача SOAP сообщений по протоколу HTTP. Также широко распространено использование в качестве транспортного протокола SMTP, FTP, TCP.

Согласно определению W3C, "WSDL - формат XML для описания сетевых сервисов как набора конечных операций, работающих при помощи сообщений, содержащих документно-ориентированную или процедурно-ориентированную информацию". Документ WSDL полностью описывает интерфейс веб-сервиса с внешним миром. Он предоставляет информацию об услугах, которые можно получить, воспользовавшись методами сервиса, и способах обращения к этим методам.

Технология Universal Description, Discovery and Integration (UDDI) предполагает ведения реестра веб-сервисов. Подключившись к этому реестру, потребитель сможет найти веб-сервисы, которые наилучшим образом удовлетворяют его потребностям. Технология UDDI дает возможность поиска и публикации нужного сервиса, как человеком, так и программой-клиентом. Поиск и публикация в реестре предоставляется программе-клиенту как набор веб-сервисов реестра UDDI.

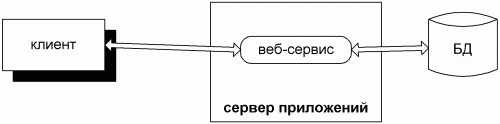
Веб-сервисы позиционируются как программное обеспечение промежуточного слоя. Использовать веб-сервисы могут как клиентские приложения, непосредственно работающие с пользователем, так и другие приложения (в том числе и другие веб-сервисы).

Веб-сервисы размещаются на серверах приложений.

Разработчики концепции веб-сервисов предлагают следующие сценарии применения веб-сервисов:

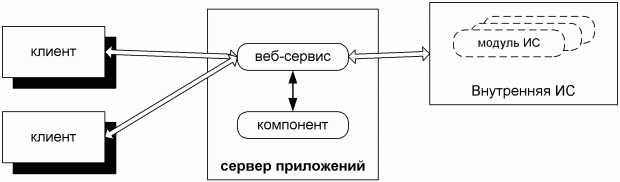
* Веб-сервисы как реализация логики приложения (бизнес-логики).

То есть, создание нового приложения бизнес-логика, которого реализуется в веб-сервисе



* Веб-сервисы как средство интеграции.

То есть, использование веб-сервиса как способа доступа удаленных клиентов к внутренней ИС компании, или для организации взаимодействия компонента (например, EJB, COM-компонента) с различными удаленными клиентами.



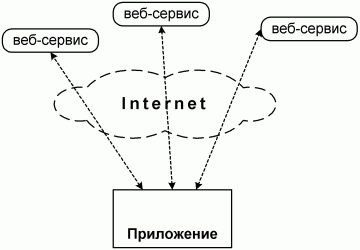
* Создание альтернативного сервиса.

В этом случае, при разработке нового веб-сервиса используется описание интерфейса уже существующего веб-сервиса. Таким образом, сервис имеет много потенциальных клиентов сразу с момента начала работы, а подключение к нему не требует существенных изменений на стороне клиента.

Как было сказано выше, концепция веб-сервисов включает в себя возможность ведения реестра веб-сервисов. Описание интерфейса может быть получено из такого реестра. После создания и внедрения нового веб-сервиса, имеет смысл зарегистрировать его в реестре. Тогда клиенты при поиске сервисов, реализующих исходный интерфейс, получат указание и на новый веб-сервис.

* Использование веб-сервиса как строительного блока при создании приложения.

Приложение может использовать веб-сервисы как удаленные компоненты, которые предоставляют определенную функциональность. Существуют различные сервисы, которые предоставляют качественное решение таких задач как аутентификация, ведение календаря, отправка сообщений, поиск, перевод и т. П

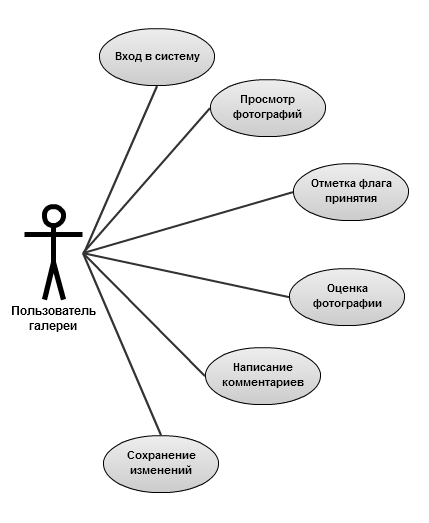


Кроме этого, и возможны другие варианты использования веб-сервисов. Например, существуют исследования по использованию веб-сервисов для построения распределенных вычислительных и информационных систем и одноранговых и со сложной иерархической структурой.

## Серверная часть

### Анализ вариантов использования

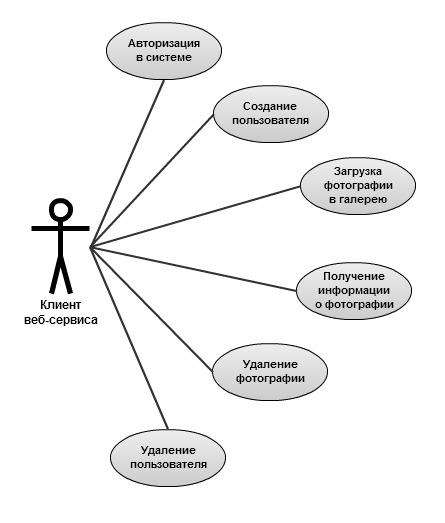
Серверная часть представляет собой объект для действий двух категорий объектов. Во-первых, это пользователи онлайн фотогалереи – именно она является фронтендом серверного программного обеспечения. Во-вторых, это клиентские системы, которые используют эту часть как веб-сервис. Исходя из этого, необходимо проанализировать две группы вариантов использования. На первой UseCase диаграмме изображены варианты использования системы пользователем фотогалереи.



Они включают в себя:

* вход в систему – пользователь проходит авторизацию и аутентификацию для определения списка фотографий, подлежащих просмотру и редактированию;
* просмотр фотографий – пользователь имеет возможность просмотра фотографий, опубликованных для него;
* отметка флага принятия – каждая фотография может быть принята / не принята пользователем, либо отметка может быть не проставлена. Этот признак позволяет фотографу отсеять ненужные снимки
* оценка фотографии – клиент проставляет оценку по 5-балльной шкале каждой фотографии. Это делается для того, чтобы выделить наиболее желаемые для дальнейшей обработки снимки;
* написание комментариев – здесь пользователь может оставить какие-либо пожелания для дальнейшей обработки фотографий;
* сохранение изменений – после того, как клиент проделал все необходимые манипуляции с опубликованными снимками, он сохраняет свои изменения, тем самым уведомляя фотографа об окончании процесса приёмки.

Вторая UseCase диаграмма иллюстрирует варианты использования серверной части системы в качестве веб-сервиса различными клиентами.



Она включает в себя:

* авторизация в системе – для совершения операций, клиентской системе необходимо авторизоваться при помощи пары логин-пароль;
* создание пользователя – в процессе публикации фотографий есть возможность создания нового пользователя фотогалереи с заданным именем и паролем;
* загрузка фотографии в галерею – публикация фотографий осуществляется по одной, при этом необходимо помимо самого графического файла указать пользователя, для которого выполняется публикация;
* получение информации о фотографии – запрос и получение флага принятия, рейтинга и комментариев для конкретной фотографии определенного пользователя;
* удаление фотографии – удаление опубликованной фотографии с сервера;

удаление пользователя – удаление пользователя и всех его фотографий с сервера.

### Выбор технологии реализации

#### PHP или ASP

При определении технологии реализации систем, работающих в веб, традиционно выбор останавливается на одной из двух технологий: APS и PHP.

**Active Server Pages**(ASP) - это технология Microsoft, позволяющая создавать динамические web страницы. ASP поддерживает несколько языков программирования, наиболее распространенным является VBScript (опять же разработка Microsoft). Соответственно, ASP - это технология для Windows систем.

**PHP** - аналог ASP, который может работать как в среде UNIX так и в Windows. PHP - это система разработки скриптов, включающая в себя CGI - интерфейс, интерпретатор языка и набор функций для доступа к базам данных и различным объектам WWW.

По мнению автора данной работы, PHP выигрывает у ASP как в плане технических (функциональных) возможностей, так и по критерию популярности/общедоступности.

* ASP технология основана на модели COM (COM - Component Object Model), то есть работать вы будете с объектами и компонентами. Например, для вывода информации по запросу клиента задействуется объект Response, при обращении к базе данных, задействуется еще один объект, для работы с файлами вызывается третий объект и т.д. Таким образом, подобное нагромождение замедляет работу ASP приложения. При использовании модулей PHP, работа идет в едином пространстве памяти PHP, и это несколько быстрее, чем использование разрозненных объектов, задействованных в различных процессах.
* PHP использует более оптимизированный механизм управления памятью, благодаря чему снижается нагрузка на веб-сервер
* Процесс разработки небольших приложений при помощи PHP занимает гораздо меньше времени
* ASP требует лицензирования веб-сервера, ОС и СУБД, PHP ;

Из этого можно сделать вывод, что для данного проекта целесообразно применять технологию PHP.

#### SOAP или XML RPC

В классической концепции веб-сервисов для передачи сообщений серверу используется SOAP – Simple Object Access Protocol — простой протокол доступа к объектам. Однако он используется не только для вызова удаленных процедур, но и для передачи произвольных сообщений в формате XML. XML-RPC (сокр. от [англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Extensible Markup Language Remote Procedure Call — XML-вызов удалённых процедур) — стандарт/[протокол](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [вызова удалённых процедур](http://ru.wikipedia.org/wiki/Remote_Procedure_Call), основанный на [XML](http://ru.wikipedia.org/wiki/XML), является прародителем [SOAP](http://ru.wikipedia.org/wiki/SOAP), отличается исключительной простотой применения. XML-RPC, как и любой другой интерфейс [RPC](http://ru.wikipedia.org/wiki/Remote_Procedure_Call), определяет набор стандартных типов данных и команд, которые программист может использовать для доступа к функциональности другой программы, находящейся на другом [компьютере](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) в [сети](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C). Следовательно, для вызова удаленных процедур на сервере без необходимости обмена произвольными сообщениями, XML RPC подходит лучше, благодаря простоте описания и реализации.

### XML RPC в PHP

Существует множество проектов Open Source, которые могут облегчить жизнь программиста, избавив от необходимости писать свой код для генерирования XML-RPC-запросов и ответов.

* *XMLRPC-EPI*, <http://xmlrpc-epi.sourceforge.net/>

XMLRPC-EPI изначально было разработан для внутреннего пользования и оказалось настолько успешным, что в настоящее время снабжает сам PHP экспериментальными XML-RPC функциями. По сути XMLRPC-EPI -это базовый класс, написанный в C++ (остальные написаны в PHP), а поэтому без пользователя root установить на сервер его не удастся: потребуется перекомпилировать PHP.

* *eZ xmlrpc*, <http://developer.ez.no/article/static/53/>

Разработка Бэрда Фастарда. Это XML-RPC класс, используемый в приложении ezPublish для получения запросов с локального клиента. Дизайн класса превосходен интуитивностью управления, кроме того, прекрасно описан в техническом руководстве, и позволяет проводить интроспекцию.

Всё прекрасно, но в одном класс нехорош: для его работы требуется, чтобы в конфигурации PHP был доступен и установлен [**xml parser**](http://doc.trolltech.com/2.3/xml-dom.html%20qt-dom) (--with-qtdom). Для большинства инсталляций PHP это не типично, чаще встречается [**expat XML parser**](http://sourceforge.net/projects/expat/) (--with-xml).

* *Fase 4 XML-RPC*, <http://www.fase4.com/xmlrpc/>
* *phpRPC*, <http://sourceforge.net/projects/phprpc/>

Класс с большим потенциалом. Разработчики не остановились на простой интерпретации и управлении запросами/ответами и предоставили пользователю более широкую функциональность, как, например, подключение к "абстрактной" базе данных. Последняя доступная версия - альфа 0.9,

* *phpxmlrpc*, <http://phpxmlrpc.sourceforge.net/>

Эта разработка корпорации Useful, самого создателя стандарта XML-RPC, очевидно содержит много полезного. По существу, приложение полностью поддерживает стандарт XML-RPC и, кроме того, предоставляет возможность отладки, которая (как мы увидим позже) может стать самой большой вашей проблемой при разработке web-сервиса.

Таким образом, существует множество реализаций технологии XML RPC в PHP. В используемой в данном проекте системе применяется разработка phpxmlrpc, о ней речь пойдет ниже.

### Выбор схемы обеспечения безопасности

При анализе схем обеспечения безопасности в данной системе определяющим фактором является сбалансированное соотношение стойкости и простоты реализации. В этой связи, были рассмотрены простые схемы парольной аутентификации.

* Классическая схема – клиентская программа посылает аутентификационные данные при каждой операции, серверная часть проверяет эти данные и, в случае их корректности, выполняет требуемую операцию.

Эта схема проста в реализации, но имеет недостаток: в том случае, если логин и пароль становятся известными третьей стороне, серверная часть не имеет возможности отличить подлинного клиента от фальшивого.

* Классическая схема с добавлением токена. При таком подходе при входе в систему сервер выдает клиентской системе предварительно сгенерированную последовательность символов – «токен». Эта строка хранится на сервере в течение всей сессии работы. При каждом удаленном вызове процедур клиентской программой передаётся этот токен, зашифрованный по алгоритму, известному только серверу и подлинным клиентам. Сервер производит шифрование токена, хранящегося в текущей сессии и сравнивает его с переданным. Если результаты совпадают, операция производится.

Эта схема обеспечивает возможность удаленного вызова процедур веб-сервиса только с клиентов, которым известен алгоритм шифрования токенов. Таким образом, при обеспечении сохранности этого алгоритма, обеспечивается безопасность веб-сервиса.

### CMS Joomla

Большинство веб-проектов не создаются полностью с нуля, а используют уже существующие системы в качестве каркаса. Одним из видов таких систем является Система управления содержимым (контентом)([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Content management system, CMS). Главной целью такой системы является возможность собирать в единое целое и объединить на основе ролей и задач все разнотипные источники знаний и информации, доступные как внутри организации, так и за ее пределами, а также возможность обеспечения взаимодействия сотрудников, рабочих групп и проектов с созданными ими базами знаний, информацией и данными так, чтобы их легко можно было найти, извлечь и повторно использовать привычным для пользователя образом. Однако, помимо функций по управлению содержимым, многие из таких систем предоставляют также развитое API для облегчения процесса разработки. Одной из таких CMS является Joomla – [система управления содержимым](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B8%D0%BC%D1%8B%D0%BC), написанная на языках [PHP](http://ru.wikipedia.org/wiki/PHP) и [JavaScript](http://ru.wikipedia.org/wiki/JavaScript), использующая в качестве хранилища базу данных [MySQL](http://ru.wikipedia.org/wiki/MySQL). Является [свободным программным обеспечением](http://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), распространяемым под лицензией [GNU GPL](http://ru.wikipedia.org/wiki/GNU_General_Public_License). Её возможности:

* Функциональность можно расширять с помощью дополнительных расширений ([компонентов](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82), модулей и [плагинов](http://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD)).
* Модуль безопасности для многоуровневой аутентификации пользователей и администраторов (используется собственный алгоритм аутентификации и "ведения" сессий).
* Система шаблонов позволяет легко изменять внешний вид сайта или создать свой уникальный.
* Настраиваемые схемы расположения модулей, включая левый, правый, центральный и любое другое произвольное положения блока (при желании содержимое модуля можно включить в содержимое материала. например выражение {loadmodule mod\_fpslideshow} (вместе с фигурными скобками) введенное в произвольное место в статье выведет содержимое модуля, которому задана позиция вывода как "mod\_fpslideshow").
* К преимуществам системы можно отнести то, что все компоненты, модули, плагины и шаблоны можно написать самому, разместить их в структурированном каталоге расширений или отредактировать существующее расширение по своему усмотрению.
* Для каждой динамической страницы можно создать своё описание и ключевые слова в целях повышения рейтинга в поисковых системах;
* Начало и окончание публикации любых материалов можно запрограммировать по календарю;
* Возможность ограничить доступ к определённым разделам сайта только для зарегистрированных пользователей а с выходом Joomla 1.6 доступ как к разделу так и к определённому материалу с точностью до конкретной связи материал<->пользователь;
* Настраиваемые схемы расположения элементов по областям шаблона
* Различные модули (последние новости, счётчик посещений, подробная статистика посещений, гостевая книга, форум и другие.);
* Возможность создания не одной, а нескольких форм обратной связи для каждого контакта;
* Модуль приёма от удалённых авторов новостей, статей и ссылок;
* Иерархия объектов;
* Менеджер рассылки новостей. Поддержка более чем 360 служб рассылки новостей по всему миру;
* Встроенный визуальный редактор [TinyMCE](http://ru.wikipedia.org/wiki/TinyMCE);
* [ЧПУ](http://ru.wikipedia.org/wiki/%D0%A7%D0%9F%D0%A3_(%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82)) — «человекопонятный URL».

Именно возможность расширять стандартную функциональность при помощи подключаемых программных модулей, а также развитый API послужили причинами для выбора этой CMS в качестве основы для построения веб-сервиса.

### Реализация бизнес-логики на базе Joomla API

#### Программные модули

#### Поддержка MVC

#### Работа с БД

#### Отображение интерфейса пользователя

#### JavaScript и AJAX

#### Joomla и XML RPC

CMS Joomla использует проект phpxmlrpc в качестве базы при работе с XML RPC. При этом запросы отправляются по адресу http://site.ru/service/publish. Обработчик сообщений разбирает XML сообщения и

## Клиентская часть

### C:\Documents and Settings\Артем\Рабочий стол\Реализация\Пользователь lightroom.pngАнализ вариантов использования

### Выбор технологии реализации

### Обзор Adobe Lightroom

### Средства автоматизации Lightroom

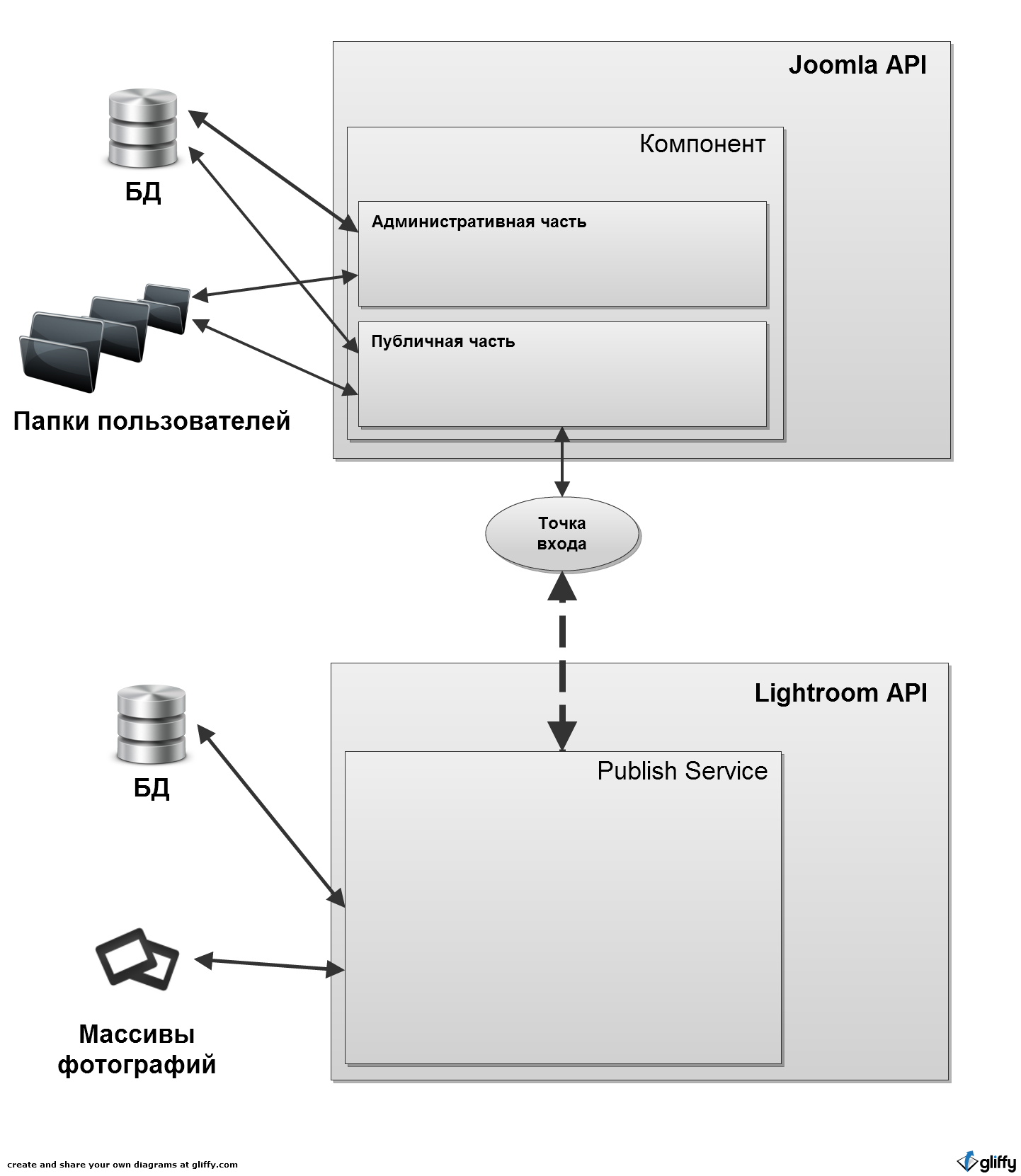
### Реализация бизнес-логики на базе Lightroom API

#### Программные модули

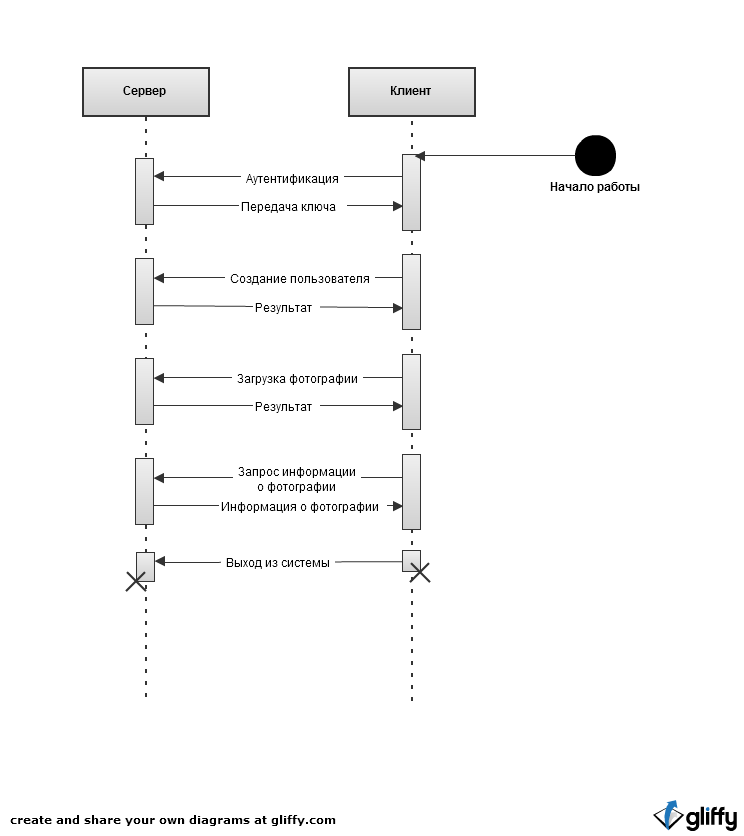
#### Работа с массивами фотографий

## Взаимодействие клиента и сервера

### Общая схема взаимодействия



### Диаграмма последовательности



1. Пользователь запускает клиентское приложение.
2. Клиент посылает серверу запрос на аутентификацию с передачей имени пользователя и пароля. Формат запроса описан в главе «Формат XML сообщений».
3. Сервер отвечает на запрос передачей ключа (токена), который действителен на время сессии. Все последующие операции передают этот ключ для подтверждения подлинности клиента.
4. Если требуется создать нового пользователя системы, клиент посылает запрос на создание пользователя, передавая имя и пароль для него.
5. Сервер возвращает результат операции. Формат результата описан в главе «Формат XML сообщений».
6. При загрузке фотографий на сервер клиент посылает по одному запросу для каждой фотографии, передавая имя пользователя-владельца этой фотографии, параметры фотографии и файл с изображением
7. Сервер возвращает результат операции.
8. При необходимости обновления метаданных фотографии клиент делает запрос на получение этой информации, передавая идентификатор фотографии.
9. Сервер возвращает результат операции, передавая запрошенную информацию о фотографии.
10. При окончании работы клиент посылает на сервер запрос на выход из системы. Ключ (токен) сеанса удаляется сервером и становится непригодным для дальнейших операций.

### Описание методов веб-сервиса

В соответствии с диаграммой вариантов использования, веб-сервис публикации фотографий имеет следующие методы:

* login(login, password)

Метод выполняет вход в систему с использованием имени пользователя и пароля. Метод возвращает секретный токен, используемый для дальнейших операций с сервером и сообщение об ошибке (если таковая произошла);

* createUser(username, password, token)

Метод выполняет создание пользователя с указанным именем и паролем. Также на сервере создается папка с именем, равным имени пользователя, для хранения фотографий. Метод возвращает статус операции создания и сообщение об ошибке (если таковая произошла);

* uploadPhoto(username, photo\_name, photo\_file, token)

Метод выполняет публикацию фотографии для указанного пользователя; указывается название фотографии и передаётся файл с изображением. Метод возвращает идентификатор фотографии на сервере, а также статус операции и сообщение об ошибке (если таковая произошла);

* getPhotoInfo(username, photo\_name, token)

getPhotoInfo(photo\_id, token)

Методы выполняют запрос на получение информации о фотографии, которая определяется по серверному идентификатору, либо по комбинации имени пользователя и названия фотографии. Метод возвращает информацию о запрошенном изображении, а также статус операции и сообщение об ошибке (если таковая произошла);

* deletePhoto(username, photo\_name, token)

deletePhoto(photo\_id, token)

Методы удаляют фотографию на сервере, которая определяется по серверному идентификатору, либо по комбинации имени пользователя и названия фотографии. Метод возвращает статус операции удаления и сообщение об ошибке (если таковая произошла);

* deleteUser(username, token)

Метод удаляет указанного пользователя на сервере, а также все его фотографии. Метод возвращает статус операции и сообщение об ошибке (если таковая произошла);

* logout(token)

Метод выполняет выход из системы. Дальнейшая работа с указанным токеном становится невозможной; для повторных запросов необходимо заново авторизоваться на веб-сервисе.

### Формат XML сообщений

XML сообщения, используемые для удаленного вызова процедур сервиса, имеют следующий формат:

<?xml version="1.0"?>

<methodCall>

<methodName>getPhotoInfo</methodName>

<params>

<param>

<value>

<username>ivanov</username>

</value>

</param>

<param>

<value>

<photo\_name>Portrait001</photo\_name>

</value>

</param>

<param>

<value>

<token>djk32i38dsjdk</token>

</value>

</param>

</params>

</methodCall>

XML сообщения, используемые для ответа сервера на вызов процедуры, имеют следующий формат:

<?xml version="1.0"?>

<methodResponse>

<params>

<param>

<value>

<accepted\_flag>true</accepted\_flag>

</value>

</param>

<param>

<value>

<rating>5</rating>

</value>

</param>

<param>

<value>

<comment>bla bla bla</comment>

</value>

</param>

</params>

<errors>

<error>

<code>0</code>

<message>OK</message>

</error>

</errors>

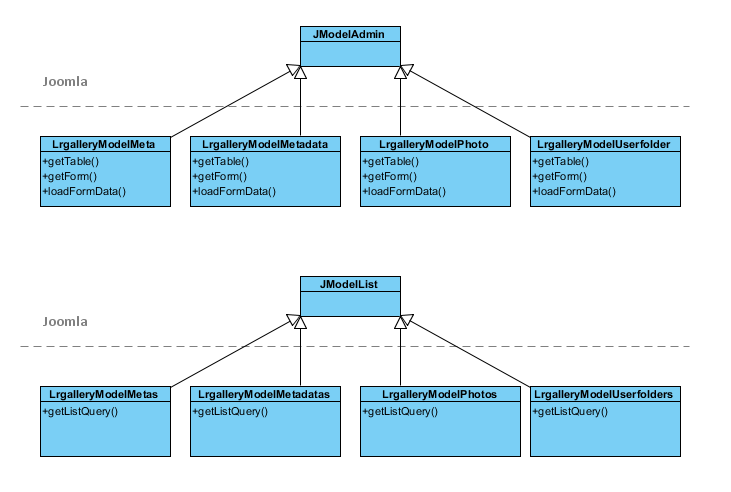
</methodResponse>

# Реализация

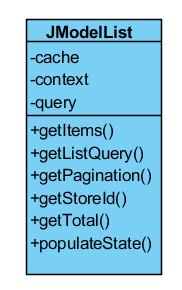
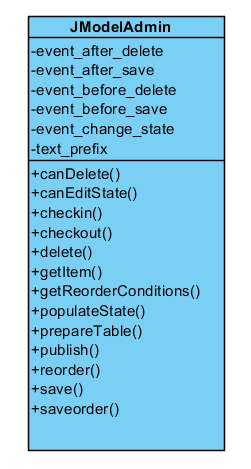
## Серверная часть

### Классы модели

Административная часть:



Все классы моделей административной части наследуются от базовых классов JModelAdmin и JModelList:



Большинство операций выполняются при помощи методов базовых классов и суперклассов. Есть всего несколько методов, которые требуют определения в наследуемых классах.

* LrgalleryModelMeta

Класс модели для работы с полями метаданных.

Методы:

* public function getTable($type = 'meta', $prefix = 'lrgalleryTable',

$config = array())

Возвращает экземпляр класса JTable для хранения данных о полях метаданных

* public function getForm($data = array(), $loadData = true)

Возвращает экземпляр класса JForm для хранения шаблона отображения полей данных о полях метаданных

* protected function loadFormData()

Выполняет заполнение полей формы данными

* LrgalleryModelMetadata

Класс модели для работы со значениями полей метаданных.

Методы:

* public function getTable($type = 'meta', $prefix = 'lrgalleryTable',

$config = array())

Возвращает экземпляр класса JTable для хранения данных о значениях полей метаданных

* public function getForm($data = array(), $loadData = true)

Возвращает экземпляр класса JForm для хранения шаблона отображения полей данных о значениях полей метаданных

* protected function loadFormData()

Выполняет заполнение полей формы данными

* LrgalleryModelPhoto

Класс модели для работы с фотографиями.

Методы:

* public function getTable($type = 'meta', $prefix = 'lrgalleryTable',

$config = array())

Возвращает экземпляр класса JTable для хранения данных о фотографиях

* public function getForm($data = array(), $loadData = true)

Возвращает экземпляр класса JForm для хранения шаблона отображения полей данных о фотографиях

* protected function loadFormData()

Выполняет заполнение полей формы данными

* LrgalleryModelUserfolder

Класс модели для работы с папками пользователей.

Методы:

* public function getTable($type = 'meta', $prefix = 'lrgalleryTable',

$config = array())

Возвращает экземпляр класса JTable для хранения данных о папках пользователей

* public function getForm($data = array(), $loadData = true)

Возвращает экземпляр класса JForm для хранения шаблона отображения полей данных о папках пользователей

* protected function loadFormData()

Выполняет заполнение полей формы данными

* LrgalleryModelMetas

Класс модели для работы со списком полей метаданных.

Методы:

* protected function getListQuery()

Возвращает экземпляр класса JQuery для выполнения запроса для получения списка полей метаданных

* LrgalleryModelMetadatas

Класс модели для работы со списком значений полей метаданных.

Методы:

* protected function getListQuery()

Возвращает экземпляр класса JQuery для выполнения запроса для получения списка значений полей метаданных

* LrgalleryModelPhotos

Класс модели для работы со списком фотографий.

Методы:

* protected function getListQuery()

Возвращает экземпляр класса JQuery для выполнения запроса для получения списка фотографий

* LrgalleryModelUserfoldes

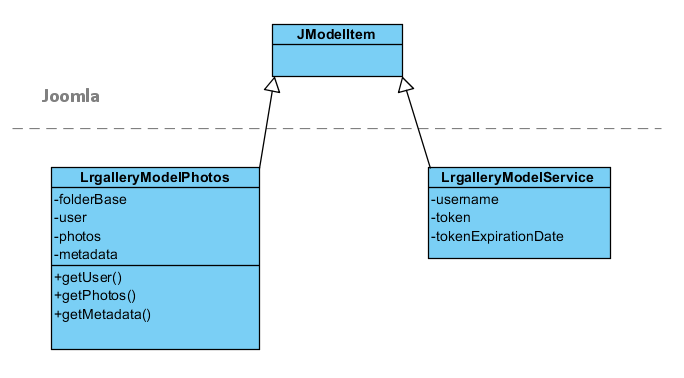
Класс модели для работы со списком папок пользователей.

Методы:

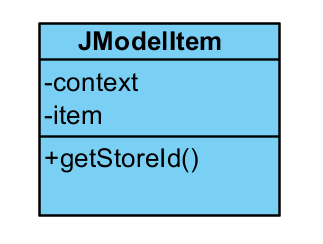
* protected function getListQuery()

Возвращает экземпляр класса JQuery для выполнения запроса для получения списка папок пользователей

Публичная часть:



Все классы моделей публичной части наследуются от базового класса JModelItem:



Некоторые операции выполняются при помощи методов базовых классов и суперклассов. Однако большинство методов требуют определения в наследуемых классах.

* LrgalleryModelPhotos

Класс модели для основной работы с интерфейсом пользователя, отображения фотографий и данных о них.

Поля:

* const folderBase = 'media/user\_folders'

Базовый путь для папок с пользовательскими фотографиями

* protected $user

Текущий пользователь

* protected $photos

Фотографии текущего пользователя

* protected $metadata

Значения полей метаданных для фотографий текущего пользователя

Методы:

* public function getUser()

Возвращает текущего пользователя

* public function getPhotos()

Возвращает фотографии текущего пользователя

* public function getMetadata()

Возвращает значения полей метаданных для фотографий текущего пользователя

* LrgalleryModelService

Класс модели для работы с обработкой запросов клиентов веб-сервиса.

Поля:

* protected $username

Хранит текущего пользователя

* protected $token

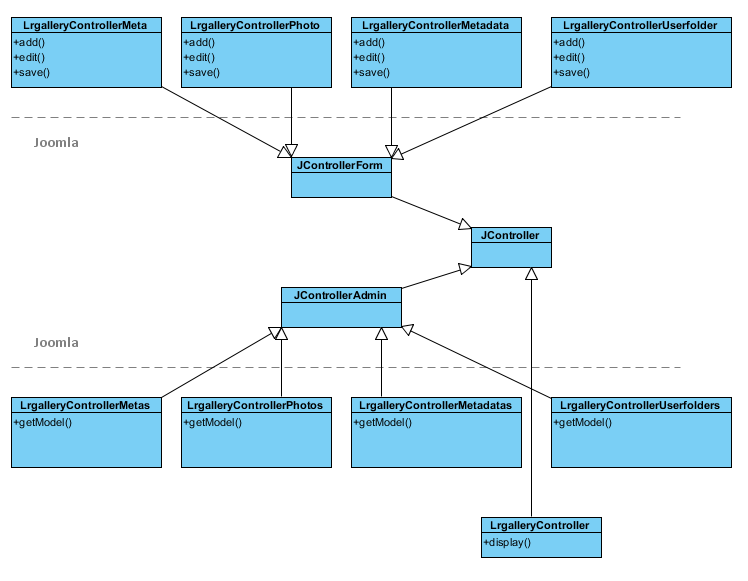
Хранит токен текущей аутентификации

* protected $tokenExpirationDate

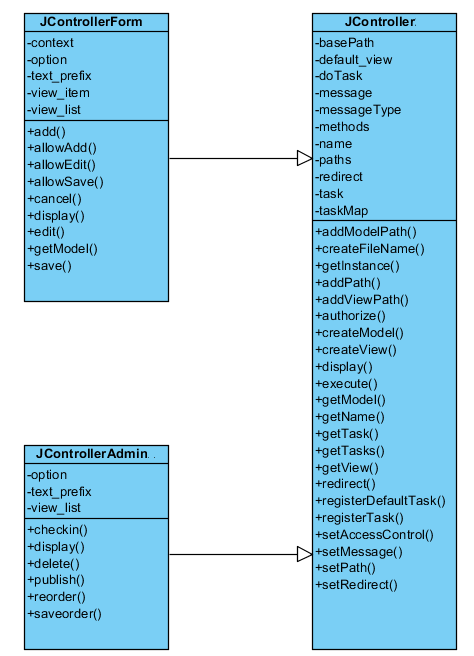
Хранит дату окончания срока действия токена

### Классы контроллеров

Административная часть:



Все пользовательские классы контроллеров наследуются от базовых классов JController, JControllerForm и JControllerAdmin:



Большинство операций выполняются при помощи методов базовых классов и суперклассов. Есть всего несколько методов, которые требуют переопределения в наследуемых классах.

* LrgalleryController

Класс основного контроллера для координации показа представлений и создания моделей.

Методы:

* public function display($cachable = false)

Управляет начальным отображением view и координирует действия остальных контроллеров

* LrgalleryControllerMeta

Класс контроллера для работы с полями метаданных.

Методы:

* public function add()

Метод для добавления нового поля метаданных

* public function edit($key = null, $urlVar = null)

Метод для редактирования поля метаданных

* protected function save($key = null, $urlVar = null)
* Метод для сохранения поля метаданных
* LrgalleryControllerMetadata

Класс контроллера для работы со значениями полей метаданных.

Методы:

* public function add()

Метод для добавления нового значения поля метаданных

* public function edit($key = null, $urlVar = null)

Метод для редактирования значения поля метаданных

* protected function save($key = null, $urlVar = null)
* Метод для сохранения значения поля метаданных
* LrgalleryControllerPhoto

Класс контроллера для работы с фотографиями.

Методы:

* public function add()

Метод для добавления новой фотографии

* public function edit($key = null, $urlVar = null)

Метод для редактирования фотографии

* protected function save($key = null, $urlVar = null)
* Метод для сохранения фотографии
* LrgalleryControllerUserfolder

Класс контроллера для работы с папками пользователей.

Методы:

* public function add()

Метод для добавления новой папки пользователя

* public function edit($key = null, $urlVar = null)

Метод для редактирования папки пользователя

* protected function save($key = null, $urlVar = null)
* Метод для сохранения значения папки пользователя
* LrgalleryControllerMetas

Класс контроллера для работы со списком полей метаданных.

Методы:

* public function getModel()

Возвращает экземпляр класса JModel, используемый при работе с данными

* LrgalleryControllerMetadatas

Класс контроллера для работы со списком значений полей метаданных.

Методы:

* public function getModel()

Возвращает экземпляр класса JModel, используемый при работе с данными

* LrgalleryControllerPhotos

Класс контроллера для работы со списком фотографий.

Методы:

* public function getModel()

Возвращает экземпляр класса JModel, используемый при работе с данными

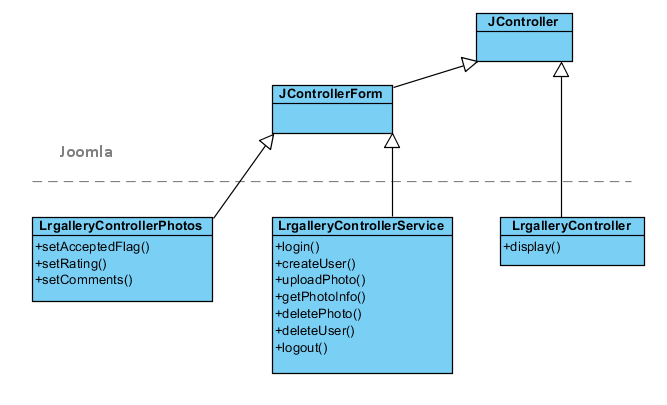
* LrgalleryControllerUserfolders

Класс контроллера для работы со списком папок пользователей.

Методы:

* public function getModel()

Возвращает экземпляр класса JModel, используемый при работе с данными

Публичная часть:

Некоторые операции выполняются при помощи методов базовых классов и суперклассов. Однако большинство методов требуют определения в наследуемых классах.

* LrgalleryController

Класс основного контроллера для координации показа представлений и создания моделей.

Методы:

* public function display($cachable = false)

Управляет начальным отображением view и координирует действия остальных контроллеров

* LrgalleryControllerPhotos

Класс контроллера для основной работы с интерфейсом пользователя, отображения фотографий и данных о них.

Методы:

* public function setAcceptedFlag($flag)

Устанавливает флаг принятия фотографии

* public function setRating($rating)

Устанавливает рейтинг фотографии

* public function setComments($comments)

Устанавливает комментарии к фотографии

* LrgalleryControllerService

Класс контроллера для обработки удаленных запросов клиентов веб-сервиса.

Методы:

* public function login($username, $password)

Метод выполняет вход в систему с использованием имени пользователя и пароля. Метод возвращает секретный токен, используемый для дальнейших операций с сервером и сообщение об ошибке (если таковая произошла);

* public function createUser($username, $password, $token)

Метод выполняет создание пользователя с указанным именем и паролем. Также на сервере создается папка с именем, равным имени пользователя, для хранения фотографий. Метод возвращает статус операции создания и сообщение об ошибке (если таковая произошла)

* public function uploadPhoto($username, $photo\_name, $photo\_file,

$token)

Метод выполняет публикацию фотографии для указанного пользователя; указывается название фотографии и передаётся файл с изображением. Метод возвращает идентификатор фотографии на сервере, а также статус операции и сообщение об ошибке (если таковая произошла);

* public function getPhotoInfo($username, $photo\_name, $token)

public function getPhotoInfo($photo\_id, $token)

Методы обрабатывают запрос на получение информации о фотографии, которая определяется по серверному идентификатору, либо по комбинации имени пользователя и названия фотографии. Метод возвращает информацию о запрошенном изображении, а также статус операции и сообщение об ошибке (если таковая произошла)

* public function deletePhoto($username, $photo\_name, $token)

public function deletePhoto($photo\_id, $token)

Методы удаляют фотографию на сервере, которая определяется по серверному идентификатору, либо по комбинации имени пользователя и названия фотографии. Метод возвращает статус операции удаления и сообщение об ошибке (если таковая произошла)

* public function deleteUser($username, $token)

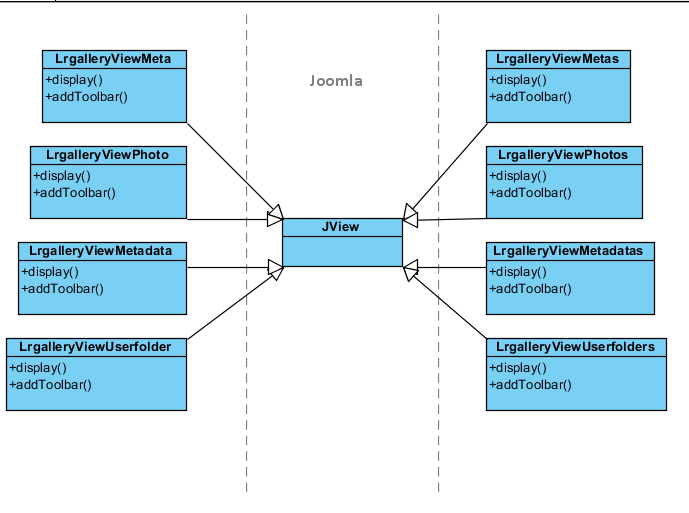
Метод удаляет указанного пользователя на сервере, а также все его фотографии. Метод возвращает статус операции и сообщение об ошибке (если таковая произошла)

* public function logout($token)

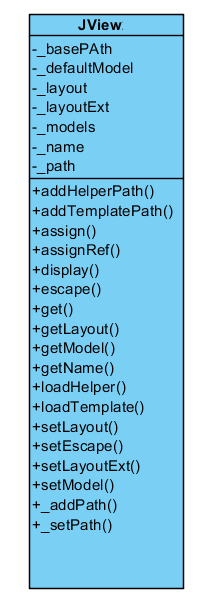
Метод выполняет выход из системы. Дальнейшая работа с указанным токеном становится невозможной; для повторных запросов необходимо заново авторизоваться на веб-сервисе

### Классы представлений

Административная часть:



Все классы представлений административной части наследуются от базового класса JView:



Большинство операций выполняются при помощи методов базовых классов и суперклассов. Есть всего несколько методов, которые требуют определения в наследуемых классах.

* LrgalleryViewMeta

Класс представления для работы с полями метаданных.

Методы:

* public function function display($tpl = null)

Метод для отображения view

* public function addToolbar()

Метод для добавления панели инструментов в административном интерфейсе

* LrgalleryViewMetadata

Класс представления для работы со значениями полей метаданных.

Методы:

* public function function display($tpl = null)

Метод для отображения view

* public function addToolbar()

Метод для добавления панели инструментов в административном интерфейсе

* LrgalleryViewPhoto

Класс представления для работы с фотографиями.

Методы:

* public function function display($tpl = null)

Метод для отображения view

* public function addToolbar()

Метод для добавления панели инструментов в административном интерфейсе

* LrgalleryViewUserfolder

Класс представления для работы с папками пользователей.

Методы:

* public function function display($tpl = null)

Метод для отображения view

* public function addToolbar()

Метод для добавления панели инструментов в административном интерфейсе

* LrgalleryViewMetas

Класс представления для работы со списками полей метаданных.

Методы:

* public function function display($tpl = null)

Метод для отображения view

* public function addToolbar()

Метод для добавления панели инструментов в административном интерфейсе

* LrgalleryViewMetadatas

Класс представления для работы со списками значений полей метаданных.

Методы:

* public function function display($tpl = null)

Метод для отображения view

* public function addToolbar()

Метод для добавления панели инструментов в административном интерфейсе

* LrgalleryViewPhotos

Класс представления для работы со списками фотографий.

Методы:

* public function function display($tpl = null)

Метод для отображения view

* public function addToolbar()

Метод для добавления панели инструментов в административном интерфейсе

* LrgalleryViewUserfolders

Класс представления для работы со списками папок пользователей.

Методы:

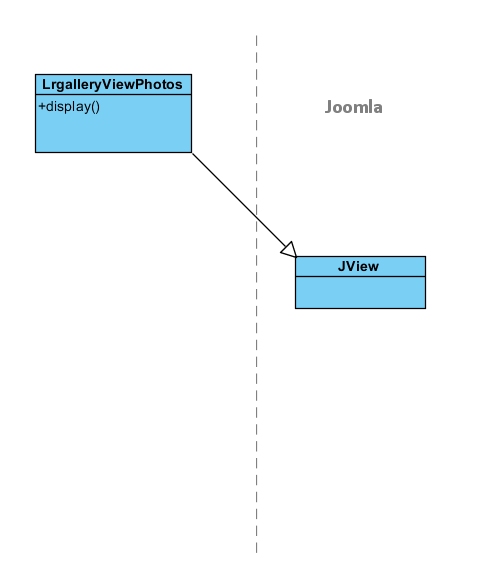
* public function function display($tpl = null)

Метод для отображения view

* public function addToolbar()

Метод для добавления панели инструментов в административном интерфейсе

Публичная часть:



Все классы представлений публичной части наследуются от базового класса JView, описание которого приведено выше.

* LrgalleryViewPhotos

Класс представления для основной работы с интерфейсом пользователя, отображения фотографий и данных о них.

Методы:

* public function function display($tpl = null)

Метод для отображения шаблона view

### Обеспечение безопасности

#### Защита коммуникаций между сервером и клиентом на основе токенов

При совершении операции входа в систему клиентом, в ответном сообщении сервер сообщает токен, соответствующий текущей сессии. Эта строка является необходимой для аутентификации всех последующих действий клиента и должна передаваться в качестве параметры с каждым сообщением XML RPC. Выданный токен хранится в базе данных сервера в течение всего периода общения с клиентом и используется для его авторизации.Время действия токена ограничено; каждый удаленный вызов процедуры с использованием токена продлевает его действие. Таким образом, если клиент в течение определенного времени не производит никаких действий по отношению к веб-сервису, срок действия токена истекает, и следующая операция требует повторной аутентификации.

#### Парольная защита папок пользователей на сервере

Для защиты папок пользователей на сервере используется механизм .htaccess.

1. Подготовка файла с паролями

Для подготовки используется утилита htpasswd. Для генерации паролей в командной строке даются следующие команды:

* + Для первого пароля (логин - yyy, пароль - zzz):   
    htpasswd.exe -bc .htpasswd yyy zzz  
    При этом будет создан файл .htpasswd.
  + Для добавления паролей в файл:  
    htpasswd.exe -b .htpasswd xxx zzz  
    При этом в файл .htpasswd будут дописаны пароли.

После этого файл можно открыть в текстовом редакторе и убедиться, что там есть строчки, соответствующие пользователям, которые были добавлены. Пример файла:

yyy:$apr1$XE1.....$U22NqaiWxdRURmPcwEAmS.  
xxx:$apr1$2K1.....$x.wbKD7PCD.3EgbcsXA8d/

1. Защита папки

Необходимо создать файл .htaccess примерно следующего содержания:

AuthUserFile /home/username/www/users/ivanov/ .htpasswd    
AuthName Admin  
AuthType Basic  
Require valid-user

Здесь первая строка - путь до файла .htpasswd. Часть /ivanov/ соответствует папке, которую нужно закрывать. Путь перед ней - это абсолютный путь к сайту.

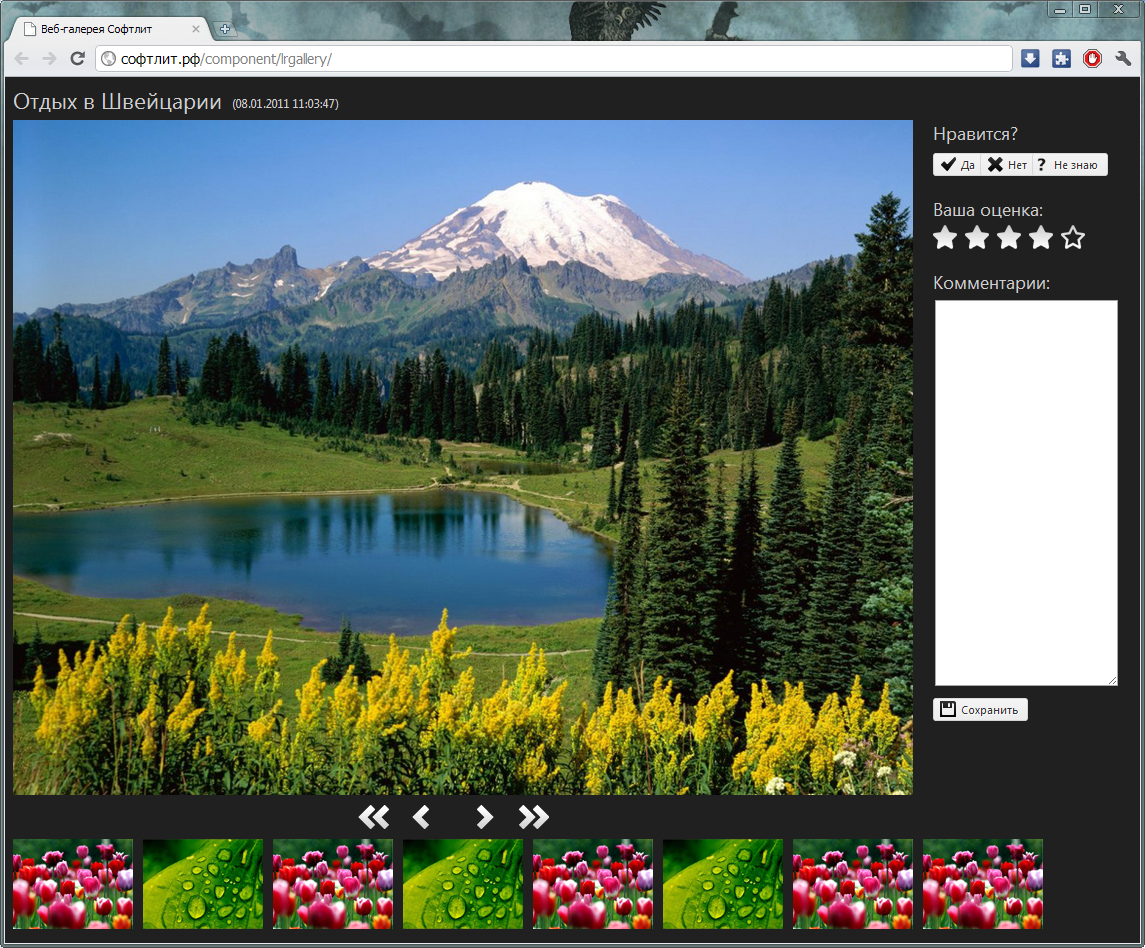
Вторая строчка - имя блока авторизации, тут нужно ввести имя пользователя.После этого файлы .htpasswd и .htaccess необходимо поместить в папку ivanov.

1. Проверка авторизации

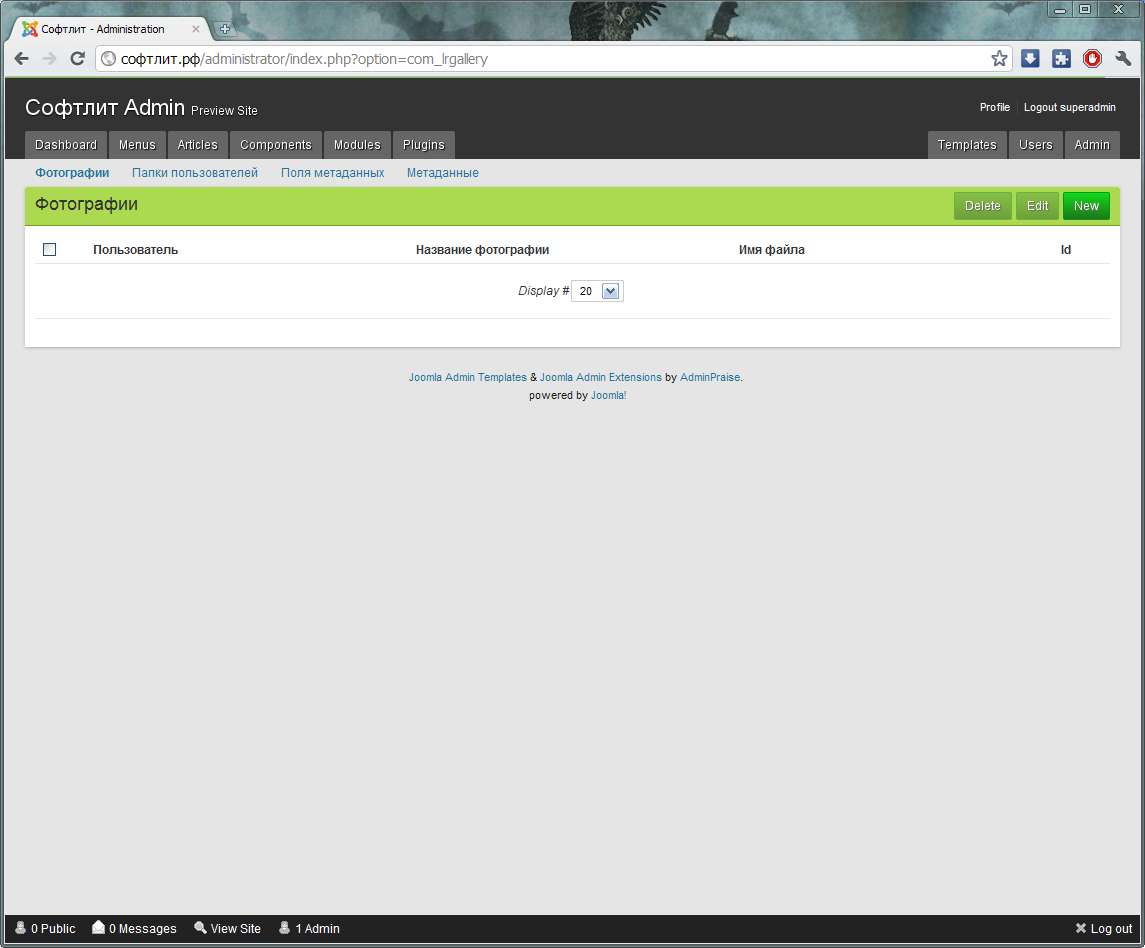
Здесь важно не путать отказ в просмотре каталога из-за отсутствия индексного файла и отказ авторизации - сообщения выдаются похожие, но всё же немного разные.

### Интерфейс пользователя

#### Публичная часть



#### Административная часть



## Клиентская часть

### Ключевые файлы

Клиентская часть построена на базе программного интерфейса Adobe Lightroom. Она представляет собой плагин для этого приложения. Из вариантов, предоставляемых API Lightroom, этот программный модуль реализует функциональность т.н. publush service – сервиса публикаций. Эта функциональность реализуется путем включения в файлы проекта определённых директив, а также наличием определенных для этого типа плагинов программных файлов. Для описания реализации плагина рассмотрим эти файлы подробнее.

#### info.lua:

return {

LrSdkVersion = 3.0,

LrSdkMinimumVersion = 3.0,

LrToolkitIdentifier = 'ru.softlit.export.lrgallery',

LrPluginName = LOC "$$$/Lrgallery/PluginName=Lrgallery",

LrExportServiceProvider = {

title = LOC "$$$/Lrgallery/Lrgallery-title=Lrgallery",

file = 'LrgalleryExportServiceProvider.lua',

},

VERSION = { major=1, minor=0, revision=0, build=000000, },

}

Этот файл используется для декларирования содержимого плагина, его функциональности и версии.

* LrSdkVersion – предпочитаемая версия Ligthroom SDK для плагина
* LrSdkMinimumVersion – минимальная версия SDK, необходимая для корректной работы плагина. В случае с сервисом публикации, она должна быть не ниже 3.0
* LrToolkitIdentifier – уникальный идентификатор программного модуля. В данном случае удобно использовать доменное имя в обратной последовательности
* LrPluginName – название плагина
* LrExportServiceProvider – заголовок пункта назначения экспорта и имя файла, содержащего определение сервиса публикации, исполняемого при выборе пункта назначения экспорта
* LrMetadataProvider – имя файла, содержащего определения полей метаданных, доступных только в Lightroom
* VERSION – версия плагина

Функция LOC используется для возможности локализации модуля. На текущий момент среда Lightroom не поддерживает русский язык, однако, при появлении такой возможности, достаточно включить файлы локализации в плагин для отображения его интерфейса на русском языке.

#### Файл определения сервиса публикации

Этот файл должен возвращать таблицу, содержащую:

* конструктор и деструктор плагина – пара функций, определяющих его поведение при создании и уничтожении. Для них существуют зарезервированные названия – startDialog и endDialog
* параметры, определяемые на уровне модуля. Для данного плагина в них хранятся имя текущего пользователя, токен аутентификации и прочие параметры, необходимые для работы с сервером публикации
* элементы кастомизации диалога экспорта интерфейса приложения Lightroom. Следущие типы кастомизаций могут быть использованы:
  + ограничение уже имеющихся функций, предоставляемых диалогом экспорта
  + добавление новых элементов в этот диалог

В данном проекте добавляются новые секции диалога, позволяющие производить аутентификацию на сервере публикации, а также устанавливать различные настройки процесса экспорта, такие как степень сжатия изображений, выходной формат и прочее

* функцию, определяющую операцию экспорта, применяемую к обработанным фотографиям. Она определяется в файле под названием processRenderedPhotos и отвечает за передачу обработанного изоюражения в место назначения, то есть на сервер публикации.

LrgalleryExportServiceProvider.lua (в сокращенном виде):

local LrBinding = import 'LrBinding'

local LrDialogs = import 'LrDialogs'

local LrFileUtils = import 'LrFileUtils'

local LrPathUtils = import 'LrPathUtils'

local LrView = import 'LrView'

local bind = LrView.bind

local share = LrView.share

require 'LrgalleryAPI'

require 'LrgalleryPublishSupport'

local exportServiceProvider = {}

for name, value in pairs( LrgalleryPublishSupport ) do

exportServiceProvider[ name ] = value

end

exportServiceProvider.supportsIncrementalPublish = 'only'

exportServiceProvider.exportPresetFields = {

{ key = 'username', default = "" },

{ key = 'fullname', default = "" },

{ key = 'auth\_token', default = '' },

{ key = 'addToPhotoset', default = false },

{ key = 'photoset', default = '' }

}

function exportServiceProvider.startDialog( propertyTable )

if not propertyTable.LR\_editingExistingPublishConnection then

propertyTable.username = nil

propertyTable.nsid = nil

propertyTable.auth\_token = nil

end

propertyTable:addObserver( 'validAccount', function()

updateCantExportBecause( propertyTable ) end )

updateCantExportBecause( propertyTable )

require 'LrgalleryUser'

LrgalleryUser.verifyLogin( propertyTable )

End

function exportServiceProvider.sectionsForTopOfDialog( f, propertyTable )

...

end

function exportServiceProvider.sectionsForBottomOfDialog( f, propertyTable )

...

end

function exportServiceProvider.processRenderedPhotos(functionContext, exportContext)

...

end

exportServiceProvider.hideSections = { 'exportLocation' }

exportServiceProvider.allowFileFormats = { 'JPEG' }

exportServiceProvider.allowColorSpaces = { 'sRGB' }

exportServiceProvider.hidePrintResolution = true

exportServiceProvider.canExportVideo = false

* startDialog – функция, вызываемая Lightroom при выборе пользователем сервиса публикации в диалоге экспорта или публикации, либо когда пункт назначения публикации уже выбран, и вызывается диалоговое окно
* sectionsForTopOfDialog – создание секций верхней части диалогового окна интерфейса пользователя
* sectionsForBottomOfDialog – создание секций нижней части диалогового окна интерфейса пользователя
* processRenderedPhotos – основная функция для обработки фотографий в процессе публикации. В этой функции происходит непосредственная отправка изображений на сервер публикации, заполнение и обновление метаданных.

#### LrgalleryUser.lua

Это вспомогательный файл, содержащий функции для авторизации и аутентификации на сервере публикации.

local LrDialogs = import 'LrDialogs'

local LrFunctionContext = import 'LrFunctionContext'

local LrTasks = import 'LrTasks'

local logger = import 'LrLogger'( 'LrgalleryAPI' )

require 'LrgalleryAPI'

--============================================================================--

LrgalleryUser = {}

--------------------------------------------------------------------------------

local function storedCredentialsAreValid( propertyTable )

...

end

--------------------------------------------------------------------------------

local function notLoggedIn( propertyTable )

...

end

--------------------------------------------------------------------------------

local doingLogin = false

function LruserUser.login( propertyTable )

...

end

--------------------------------------------------------------------------------

function FlickrUser.verifyLogin( propertyTable )

...

end

--------------------------------------------------------------------------------

* storedCredentialsAreValid – проверяет на корректность сохраненные учетные данные
* notLoggedIn – проверяет, выполнен ли вход на сервер
* login – выполняет вход на сервер
* verifyLogin – проверяет состояние аутентификации

#### LrgalleryAPI.lua

Файл содержит функции для работы с API сервиса публикации. Они определяются списком методов сервиса, которые подробно описаны в соответствующей главе. Каждая из этих функций вызывает соответствующую удаленную процедуру сервера.

LrgalleryAPI.lua:

local LrBinding = import 'LrBinding'

local LrDate = import 'LrDate'

local LrDialogs = import 'LrDialogs'

local LrErrors = import 'LrErrors'

local LrFunctionContext = import 'LrFunctionContext'

local LrHttp = import 'LrHttp'

local LrMD5 = import 'LrMD5'

local LrPathUtils = import 'LrPathUtils'

local LrView = import 'LrView'

local LrXml = import 'LrXml'

local prefs = import 'LrPrefs'.prefsForPlugin()

local bind = LrView.bind

local share = LrView.share

local logger = import 'LrLogger'( 'LrgalleryAPI' )

function login()

...

end

function createUser()

...

end

function uploadPhoto()

...

end

function getPhotoInfo()

...

end

function deleteUser()

...

end

function deleteUser()

...

end

function logout()

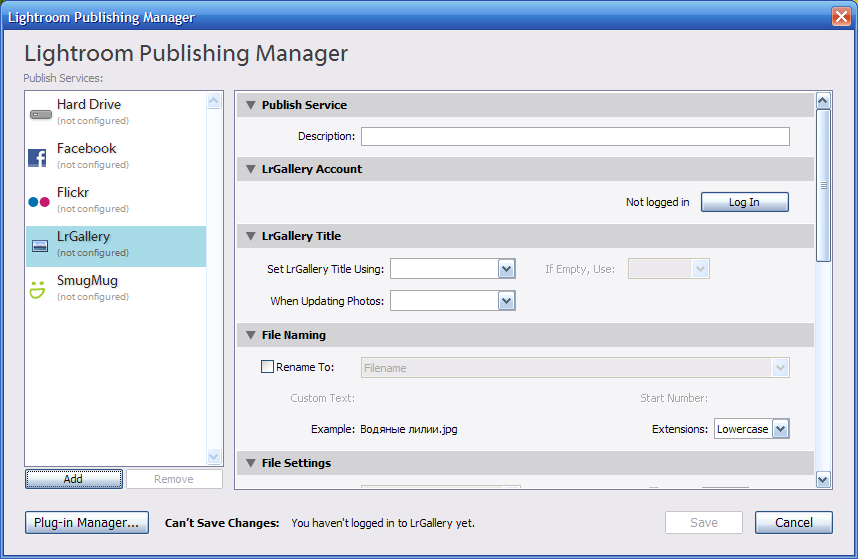
...

end

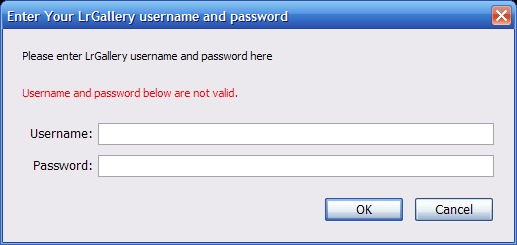
...

### Интерфейс пользователя

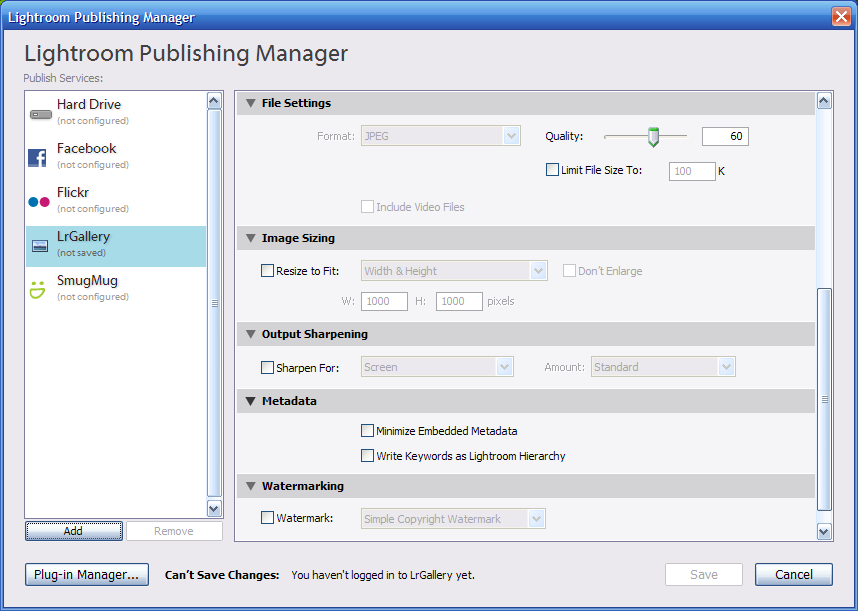
При установке плагина в раздел сервисов публикации добавляется пункт “LrGallery”.

Интерфейс пользователя представляет собой диалоговое окно с выбором настроек сервиса публикации. Элементы графического интерфейса определяются в функциях sectionsForTopOfDialog и sectionsForBottomOfDialog файла LrGalleryExportServiceProvider.lua

* Description – альтернативное описание сервиса публикации, которое может быть задано пользователем
* LrGallery Account – секция для ввода учетных данных пользователя, под которым происходит публикация. При нажатии на кнопку Log In появляется диалоговое окно, в котором необходимо ввести имя пользователя и пароль



* LrGallery Title – секция, где можно задать правило наименования фотографий в галерее (по заголовку фотографии или имени файла), правило замены фотографий при повторной публикации
* FileNaming – секция для задания правила наименования фотографий с возможностью использования даты, серийной последовательности и др.



* FileSettings – секция для указания формата выходных файлов, качества и размера итогового изображения
* Image Sizing – секция для задания требуемых параметров линейных размеров выходных фотографий
* Output Sharpening – настройки оптимизации изображений при изменении размера
* Metadata – настройки сохранения метаданных в файле фотографии
* Watermarking – секция для включения режима нанесения водяных знаков на итоговые ихображения

### Механизм публикации и обратной связи

Процесс публикации фотографий представляет собой их преобразование из оригинала в легковесный формат, подходящий для просмотра в интернете, с последующим переносом файлов из локальной файловой системы на сервер публикации. При этом возникает три ключевых момента:

* обработка оригиналов и экспорт этих изображений в заранее заданный формат;
* доставка конечного файла на сервер публикации и фиксация этого факта в базе данных Adobe Lightroom;
* осуществление обратной связи.

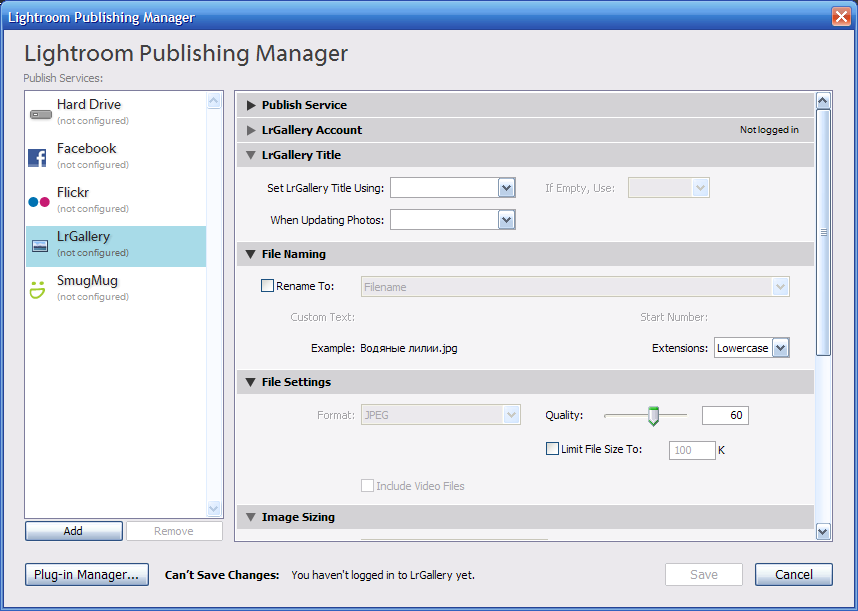
Все пункты полностью реализуются при помощи стандартного API приложения и концепции процессов и данных, которые оно предоставляет. Рассмотрим сначала первый пункт.

#### Обработка оригиналов и экспорт этих изображений в заранее заданный формат

В процессе операции экспорта изображения в конечный формат данные фотографии передаются по цепочке специальных обработчиков от функции Export Service Provider до встроенного в Lightroom механизма формирования изображений, а затем построенные изображения возвращаются назад. Эта очередь обработок проделывается над каждой фотографией из публикуемого списка. Последовательность, с которой та или иная операция применяется к изображению, определяется порядком, в котором операции добавляются в очередь обработки, а он, в свою очередь, определяется порядком появления секция в диалоговом окне. На следующем примере показана очередь выполнения операция, применяемых при публикации фотографии на сервисе LrGallery:

* переименование файла
* формат сжатия и настройки его качества
* изменения размеров фотографии
* метод оптимизации сжатия
* обработка метаданных
* наложение водяных знаков.

Все эти пункты предоставляются Lightroom API для сервисов публикации и предназначены для предварительной обработки изображения перед отправкой в конечное назначение.



Если рассмотреть подробно контроль процесса обработки фотографий плагином, можно выделить пять этапов.

1. **Определение, как будет строиться изображение**

Если в плагине определена функция updateExportSettings(), данный этап будет выполнен. Эта функция принимает один аргумент – таблицу установок экспорта – и позволяет сервису публикации самостоятельно переопределить некоторые параметры построения изображения, такие как размеры. Например, следующий фрагмент кода устанавливает максимальные размеры изображения:

updateExportSettings = function( exportSettings )

exportSettings.LR\_size\_maxHeight = 400

exportSettings.LR\_size\_maxWidth = 400

end

1. **Определение, какие именно изображения будут построены**

На данном этапе функция shouldRenderPhoto() вызывается для каждой фотографии из очереди обработки. Если она возвращает true, изображение остаётся в очереди и передается дальше по цепочке обработчиков. Если же она возвращает false, обработка фотографии прекращается. Таким образом, это позволяет фильтровать обрабатываемые файлы. Если функция shouldRenderPhoto() не определена в сервисе публикации, фильтрации изображения не происходит, и оно передаётся дальше по цепочке.

1. **Запрос на построение изображений**
   1. Для каждой фотографии, оставшейся в очереди, сессия операции экспорта LrExportSession генерирует объект запроса на построение фото (LrExportRendition). При этом непосредственное построение изображения еще не начато. На этом этапе Lightroom может показать сообщение в том случае, если фотография уже существует в целевой местоположении. Это поведение контролируется установкой параметра LR\_collisionHandling плагина.
   2. Вызывается функция сервиса processRenderedPhotos(). Если она не определена в плагине, вызывается стандартная заглушка. Здесь важно понимать, что каждый из обработчиков фотографий выполняется в своем собственном потоке, созданном при помощи класса LrTasks, поэтому несколько фотографий могут обрабатываться одновременно разными обработчиками. Это значительно повышает общую скорость процесса по сравнению с последовательным вариантом.
   3. Функция processRenderedPhotos() вызывает exportContext:renditions() и затем ожидает завершения каждого выполнения построения изображения на rendition:waitForRender().
   4. Запрос на построение отправляется по цепочке далее к фильтрам изображений. Если они отсутствуют, запрос направляется сразу во встроенный механизм построения изображений.
   5. Для каждого фильтра вызывается функция postProcessRenderedPhotos(), если она определена. Она вызывается только один раз для всех фотографий в очереди. В ней должен выполняться цикл по типу

for sourceRendition, renditionToSatisfy in filterContext:renditions() do

...

end

* 1. После прохождения всей цепочки фильтров запрос окончательно передается во встроенный механизм построения ихображений Lightroom.

1. **Обработка построенных изображений**

Как только Lightroom завершает обработку каждого запроса на построение, он сигнализирует об этом снятием блокировки и возвращением из вызова функции rendition:waitForRender(). Цикл, описанный на третьем этапе, завершается, и для каждого фильтра выполняется последовательность шагов:

* 1. waitForRender() завершается. Если предыдущее построение в цепочке было выполнено успешно, путь к образованному при этом файлу присутствует в пути sourceRendition
  2. Фильтр выполняет свои действия над этой фотографией, обычно это вызов стороннего приложения при помощи LrTasks.execute().
  3. Когда обработка завершается, фильтр обязан сообщить об этом вызовом функции renditionToSatisfy:renditionIsDone( success, message). Обычно эта функция вызывается автоматически итератором filterContext:renditions() и должна вызываться вручную только в случае ошибки обработки.
  4. Вызов renditionToSatisfy:renditionIsDone() снимает блокировку с последующего waitForRender().
  5. Когда се фильтры завершают обработку, цикл в processRenderedPhotos завершается. Сервис далее осуществляет конечную отправку изображения в целевое расположение и ожидает следующего.

1. **Сообщение об ошибках и финальная очистка**

Как только цикл функции processRenderedPhotos() завершается, Lightroom осуществляе следующие действия по очистке процессов:

* Если фотографии были построены во временном каталоге, удаляется этот каиалог и его содержимое
* Если экспорт был произведен из диалога экспорта, то выполняются следующие действия:
  + Проигрывается звук завершения операции
  + Если в процессе публикации возникали ошибки, отображается диалоговое окно со всеми сообщениями, возникшими при экспорте
  + Создается временная коллекция «Предыдущий экспорт» с исходными фотографиями, которые были опубликованы

#### Доставка конечного файла на сервер публикации

Доставка фотографии на сервер публикации выполняется на двух уровнях. В процессе работы функции processRenderedPhotos() после принятия решения о публикации, обновлении или удалении снимка осуществляется вызов соответствующего метода из пространства имён LrGalleryAPI. Например, чтобы выполнить публикацию фотографии, вызывается метод LrGalleryAPI.uploadPhoto(), куда передаются метаданные и путь к файлу изображения. Этот метод возвращает уделённый ID фотографии, назначенный ей на сервере публикации. Он используется в дальнейшем для связи фотографий из внутреннего каталога и опубликованных на сервисе изображений, это подробно описано в следующем пункте.

Метод из модуля LrGalleryAPI конструирует запрос в формате XML на основе выбранных параметров. Функция constructXml() , выполняющие эти действия, приведена на листинге

--[[ Construct xml message of format

<?xml version="1.0"?>

<methodCall>

<methodName>getPhotoInfo</methodName>

<params>

<param>

<value>

<username>ivanov</username>

</value>

</param>

<param>

<value>

<token>djk32i38dsjdk</token>

</value>

</param>

</params>

</methodCall>

]]--

local function constructXml(params)

local xmlBuilder = LrXml.createXmlBuilder()

xmlBuilder:beginBlock('methodCall')

xmlBuilder:beginBlock('methodName')

xmlBuilder:text(method)

xmlBuilder:endBlock('methodName')

xmlBuilder:beginBlock('params')

for param, value in pairs(params) do

xmlBuilder:beginBlock('param')

xmlBuilder:beginBlock('value')

xmlBuilder:beginBlock(param)

xmlBuilder:text(value)

xmlBuilder:endBlock(param)

xmlBuilder:endBlock('value')

xmlBuilder:endBlock('param')

end

xmlBuilder:endBlock('params')

xmlBuilder:endBlock('methodCall')

return xmlBuilder:serialize()

end

Для включения данных изображения в xml файл используется кодирование Base64, при помощи которого бинарные данные сериализуются в последовательность ASCII символов. Полученная таким образом строка занимает примерно на 30% больше объема памяти, чем исходное изображение, но этот способ лучшим образом подходит для передачи фотографии в XML формате.

Далее, на сервер публикации отправляется HTTP-запрос при помощи пакета LrHttp, полученный ответ в формате XML разбирается и обрабатывается. Ниже приведен листинг функции xml2table, используемой для разбора ответного XML сообщения в стандартную таблицу языка Lua:

local function xml2table(node)

local type = string.lower( node:type() )

if type == 'element' then

local element = setmetatable( {}, simpleXmlMetatable )

element.\_name = node:name()

element.\_value = node:text()

local count = node:childCount()

for i = 1, count do

local name, value = traverse( node:childAtIndex( i ) )

if name and value then

element[ name ] = value

end

end

if type == 'element' then

for k, v in pairs( node:attributes() ) do

element[ k ] = v.value

end

end

return element.\_name, element

end

end

Таким образом, вызов удаленного метода сводится к последовательности действий, показанных ниже на примере метода uploadPhoto():

function LrGalleryAPI.uploadPhoto(propertyTable, params)

-- Encode image in Base64

local rawPhotoData = LrFileUtils.readFile(params.photoFile)

local base64photo = LrStringUtils.encodeBase64(rawPhotoData)

-- Set request params

local callParams = {

title = params.title,

photo = base64photo,

}

params.params = callParams

-- Call uploadPhoto method

local result, xmlResponse = LrGalleryAPI.callXmlMethod(propertyTable, params)

-- Return result

return result

end

#### Осуществление обратной связи

Для осуществления обратной связи (получение комментариев, рейтинга и др. фотографии), используется привязка к ID фотографии. При публикации, удаленный сервер назначает полученной фотографии свой идентификатор, remoteId в терминах Lightroom. Этот ID возвращается методом публикации, и Lightroom API позволяет сохранять его в собственной базе данных. Это делается при помощи вызова метода recordPublishedPhotoId() объекта rendition, доступного внутри цикла в функции processRenderedPhotos():

lrgalleryPhotoId = LrGalleryAPI.uploadPhoto( exportSettings, params)

rendition:recordPublishedPhotoId(lrgalleryPhotoId )

Таким образом, стандартное API Lirgtroom предоставляет инструмент для хранения назначенных сервисом идентификаторов фотографий. В дальнейшем, при обновлении метаданных фотографии, этот ID используется для получения данных о изображении при помощи метода getPhotoInfo(). Метаданные фотографий из внутреннего каталога обновляются полученными таким путём данными от сервиса публикации.