Software Design Document

Version 1.0

October 21, 2014

Digital Photo Organizing Software

Prepared by:
Ranon Martin
rm1905@nova.edu

Prepared for:
CISC 680 – Software Engineering
Instructor: Frank J. Mitropoulos, Ph.D.
Fall 2014

Nova Southeastern University
Graduate School of Computer and Information Sciences
Master of Computer Science

i. Revision History

Date	Description	Author	Comments

ii. Document Approval

The following Software Design Document has been accepted and approved by the following:

Signature	Printed Name	Title	Date
	Ranon Martin	Lead Software Eng.	
	Dr. Frank J. Mitropoulos	Professor, CISC 680	

Table of Contents

1 Introduction

This section gives a description of the project scope, and the purpose of this Software Design Document (SDD).

1.1 Purpose

This software design document aims to provide a low-level description of PixBox, the digital photo organizing software, by providing insight into its structure and design. This document covers the architectural design, class structure and interactions, user interface and data persistence among others.

This document consists of the software design specifications for PixBox. This SDD does not provide design specification for the Graphic User Interface (GUI) classes.

In summary, this document is meant to provide the reader with a solid understanding of how the internals of PixBox works.

1.2 Scope

The scope of this project is to develop a single user, standalone digital photo organizing software that should give the user the ability to accomplish tasks relating to:

- Photo organization
    - Create photo albums
    - Move one or more photo albums
    - Delete one or more photo albums
    - Rename a photo album
    - Add a photo to an existing photo album
    - Move one or more photos
    - Copy one or more photos
    - Delete one or more photos
- Photo viewing
    - View a single photo
    - View an entire album of photos as a slideshow
- Meta-data management

  - Add one or more keywords to a photo
  - Delete a keyword from a photo
  - Change a keyword
- Photo sharing
  - Send a photo to a specified recipient via email without the use of a third-party solution.
  - Send an entire album of photos to a specified recipient via email without the use of a third-party solution.
- Preference configuration
  - Setup email settings
  - Modify email settings
  - Modify slideshow transition duration
  - Modify slideshow replay loop count
- Search
  - Search for a photo or photos by using a keyword or a group of comma-separated keywords.
- Basic photo editing
  - Convert a photo's image to black and white
  - Change the contrast of a photo's image
  - Change the brightness of a photo's image
  - Rotate a photo's image by a specified degree

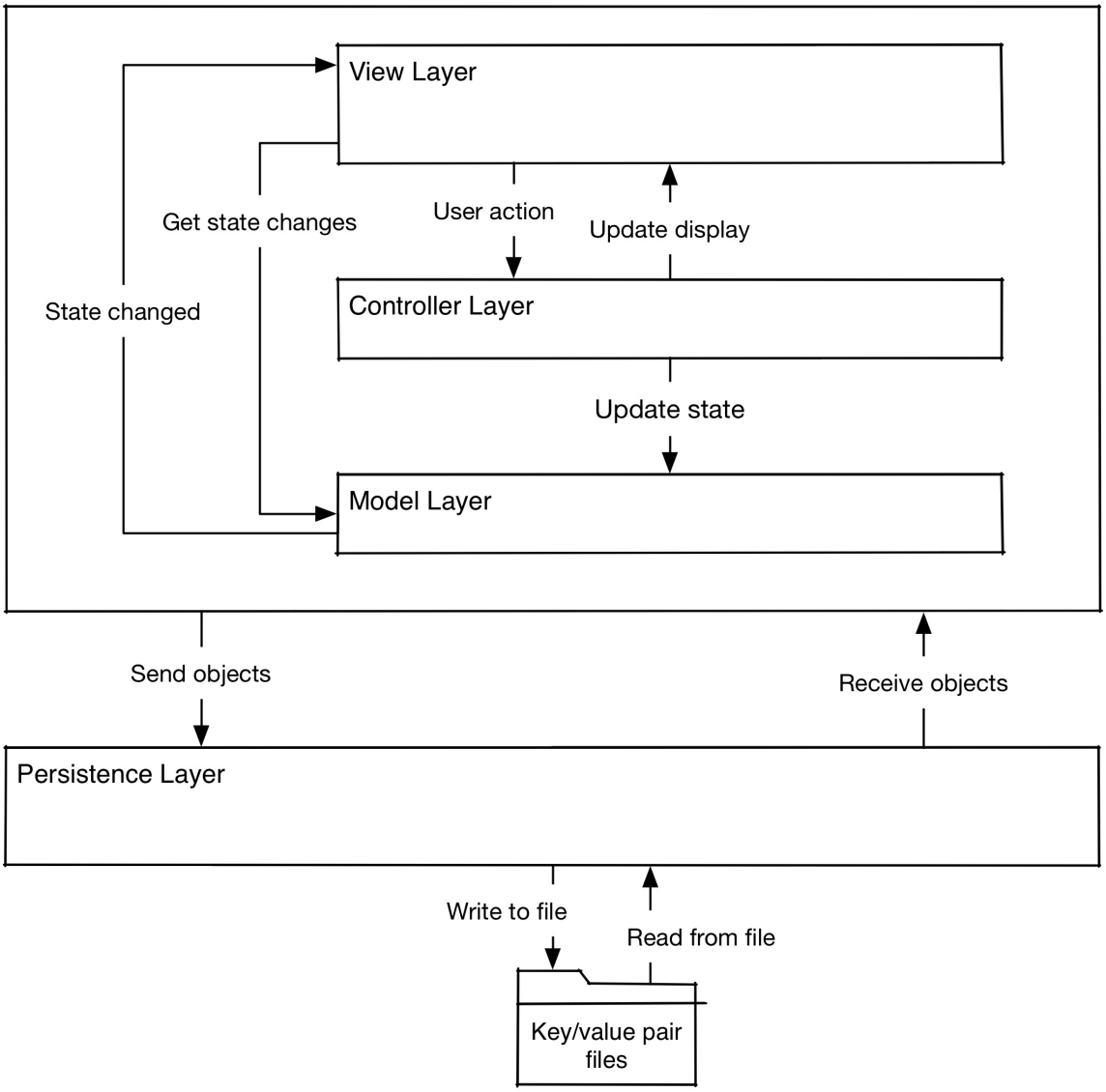
1.3 Related document

This software design document refers to the PixBox digital photo organizer Software Requirements Specification document. This document consists of the detailed requirements and the glossary of terms

2 System Overview

This section provides an overview of the system, with a high-level view of the architectural design inclusive of details outlining the different layers.

2.1 High-level Architectural Design



2.2 Description

PixBox digital photo organizer's architectural design is based on the MVC (Model, View, Controller) architectural pattern built on top of a persistence layer. The MVC architectural pattern divides the internals of the software into three interconnected parts, separating the presentation layer from the internal representations of the data so as to promote reusability.

The *View* layer of the architecture serves as the presentation layer that presents the Graphical User Interface (GUI) to the software's user to interact with the

software. When the user interacts with the software via its *View* layer, the user actions are communicated to the requisite controller in the *Controller layer* to update the state of one or more data representations existing in the *Model layer*.

Hence, the *View* layer provides the user interface to display the data that the model layer has, to the user. The *Controller* layer provides the avenue to manipulate the software and the data representations of the *Model* layer. The core of the software is at the Model layer, which maintains the state and the data represented. The Model layer updates the View layer when there are significant changes.

To separate persistence and its logic from the core of the software, a *Persistence* layer is added. The persistence layer has the responsibility for the logic of retrieving and saving the data represented in the Model layer to a data store. The persistence layer is designed using the repository pattern so that the flexibility exists to change the data store without hassle.

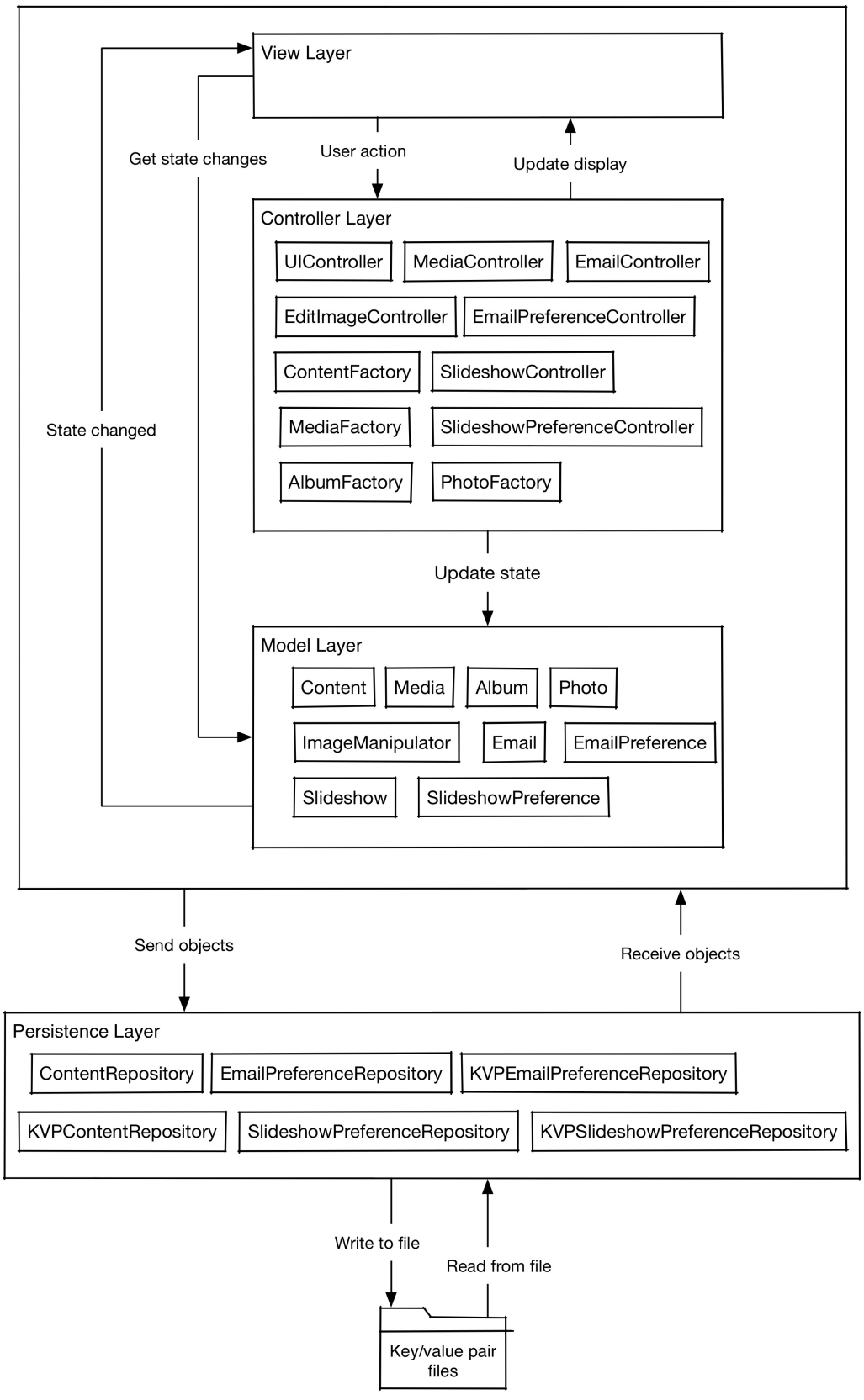
3 Design Description

This section presents the architectural design of the system in detail, identifying the where the different classes fall within the architecture in section 3.1 Architectural design.

A high-level description of the class diagram for the different classes in this project is presented in section 3.2.1 with details of the classes presented in sections 3.2.2, 3.2.3, 3.2.4 broken down according to the architectural layers: controller layer, model layer and the persistence layer respectively.

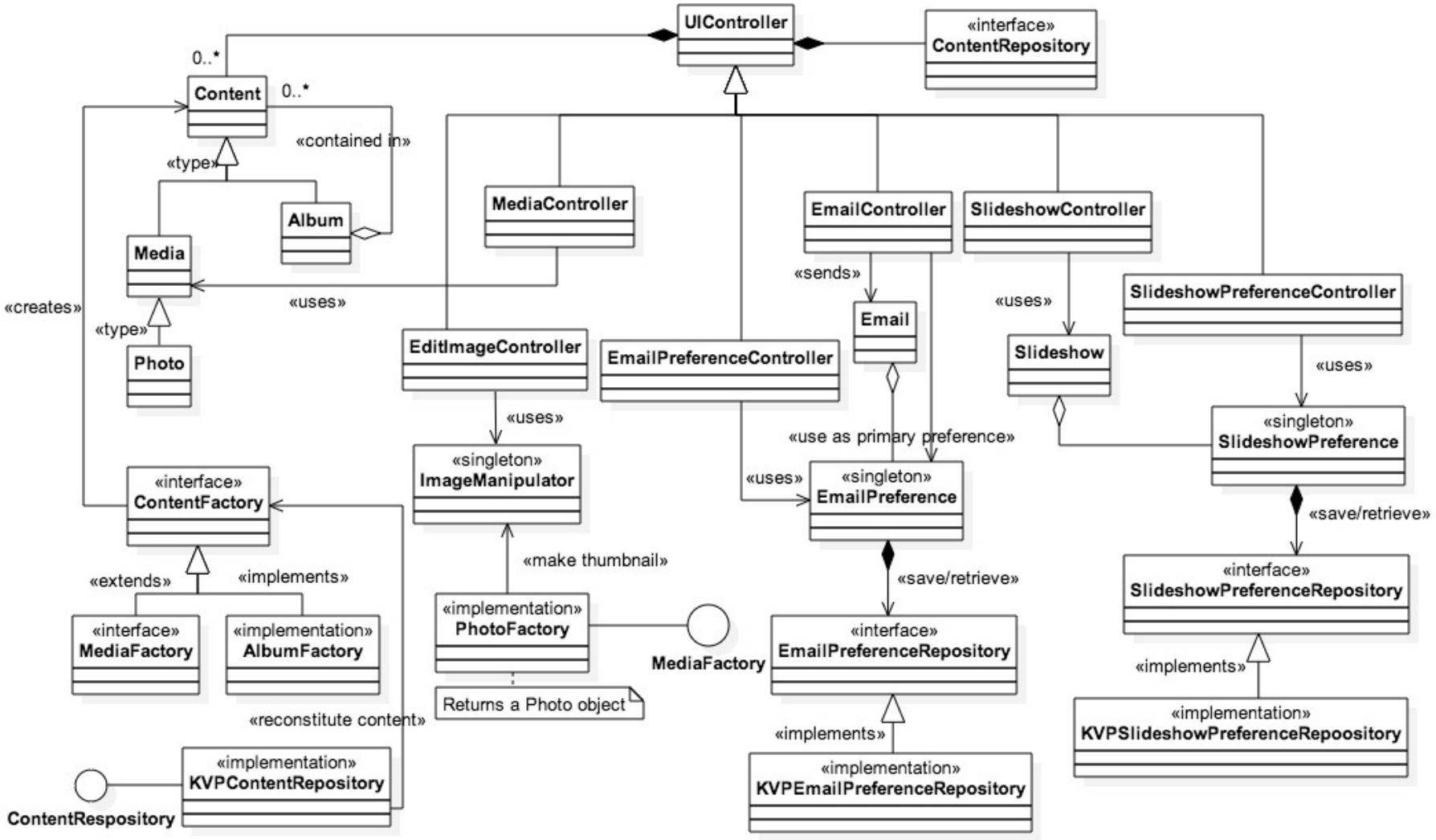
A sequence diagram detailing the interaction among the required classes when the search feature is used, is presented in section 3.3.

3.1 Architectural Design

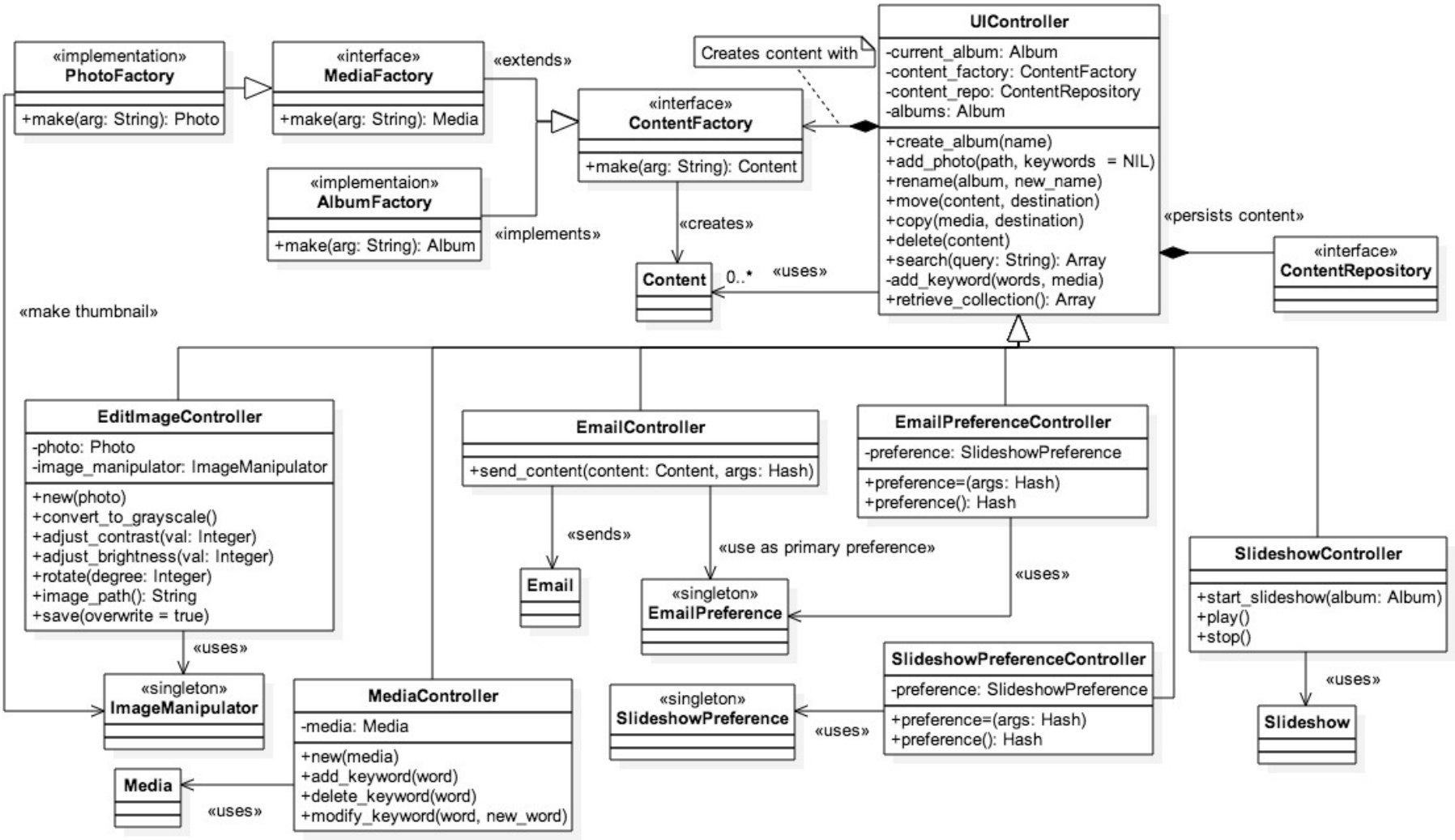


3.2 Class Diagram

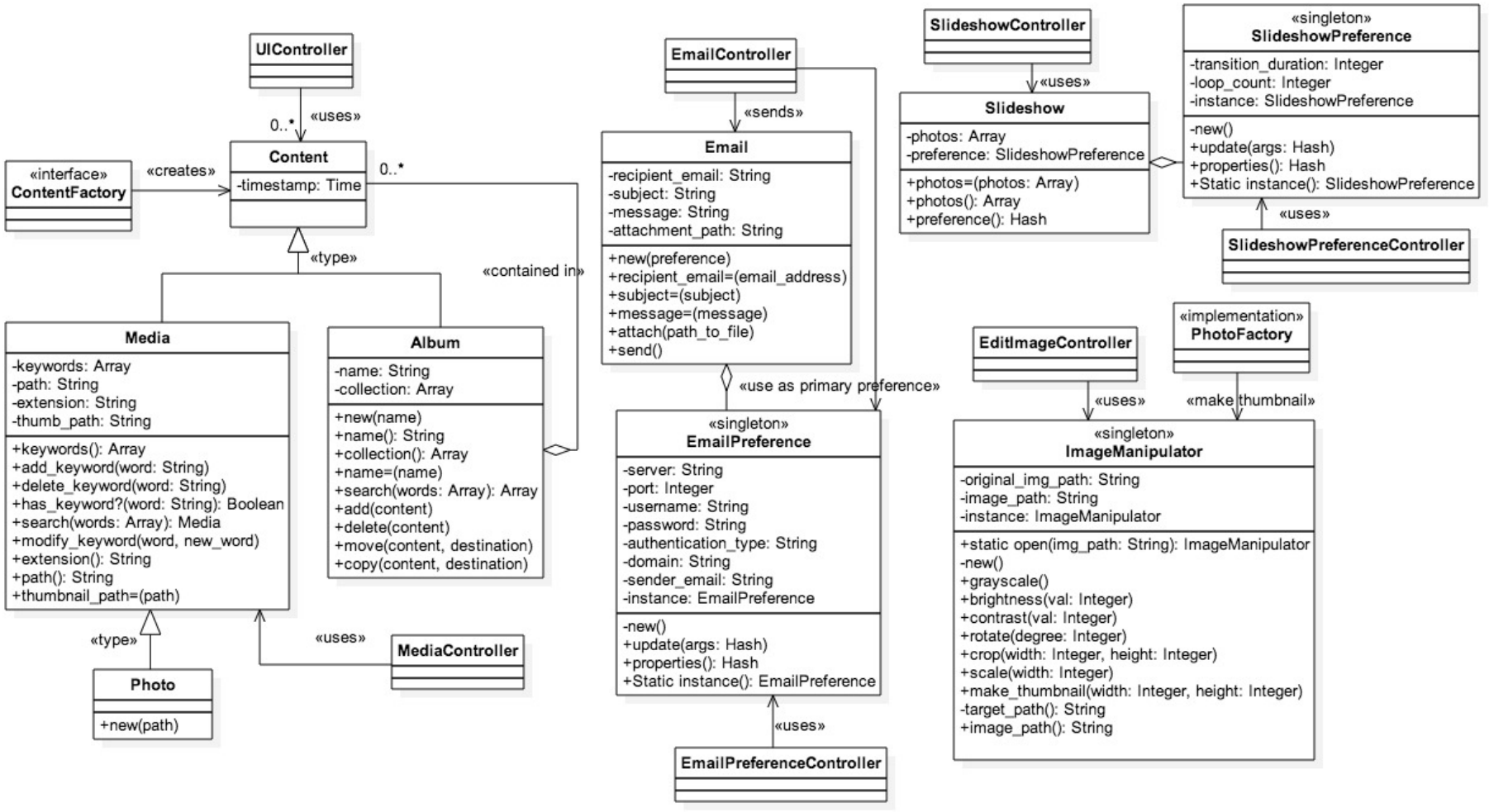
3.2.1 High-level class diagram



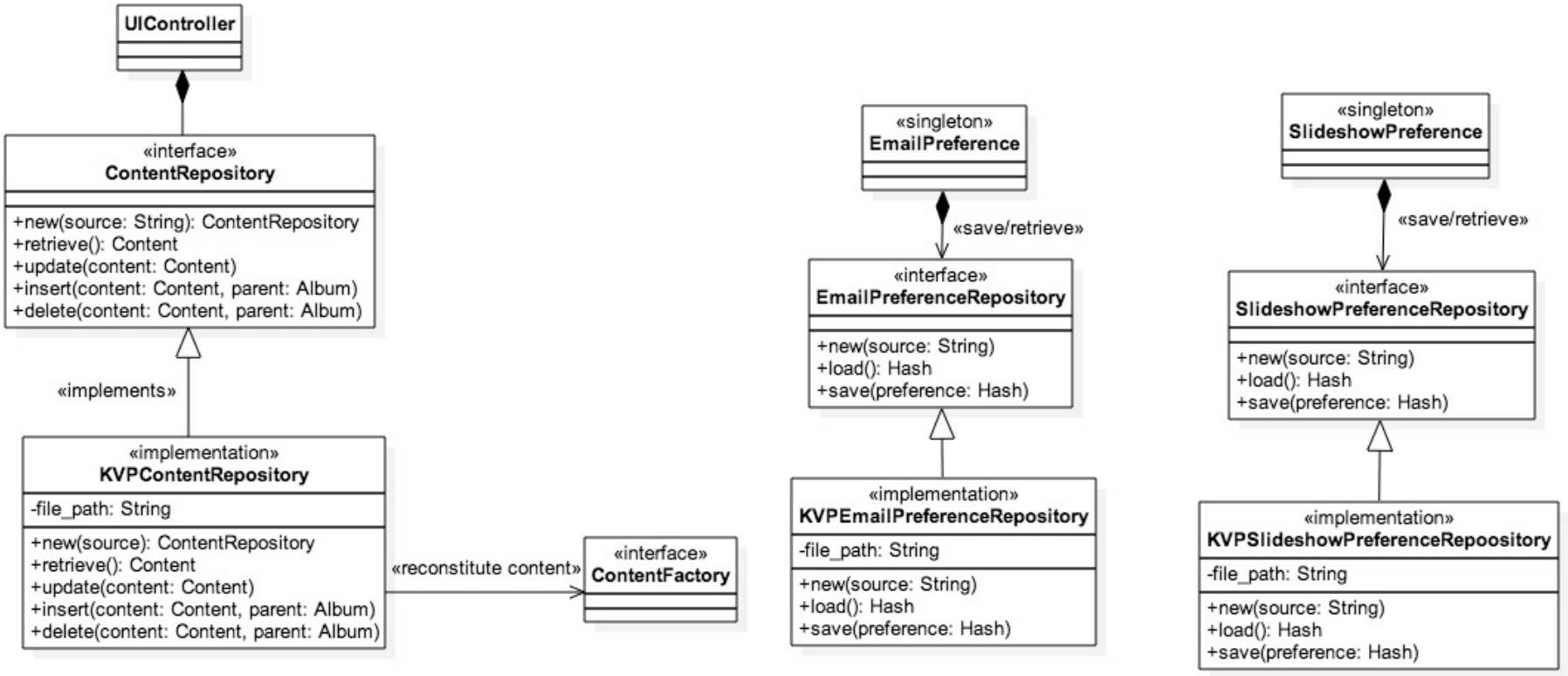
3.2.2 Controller layer



3.2.3 Model layer

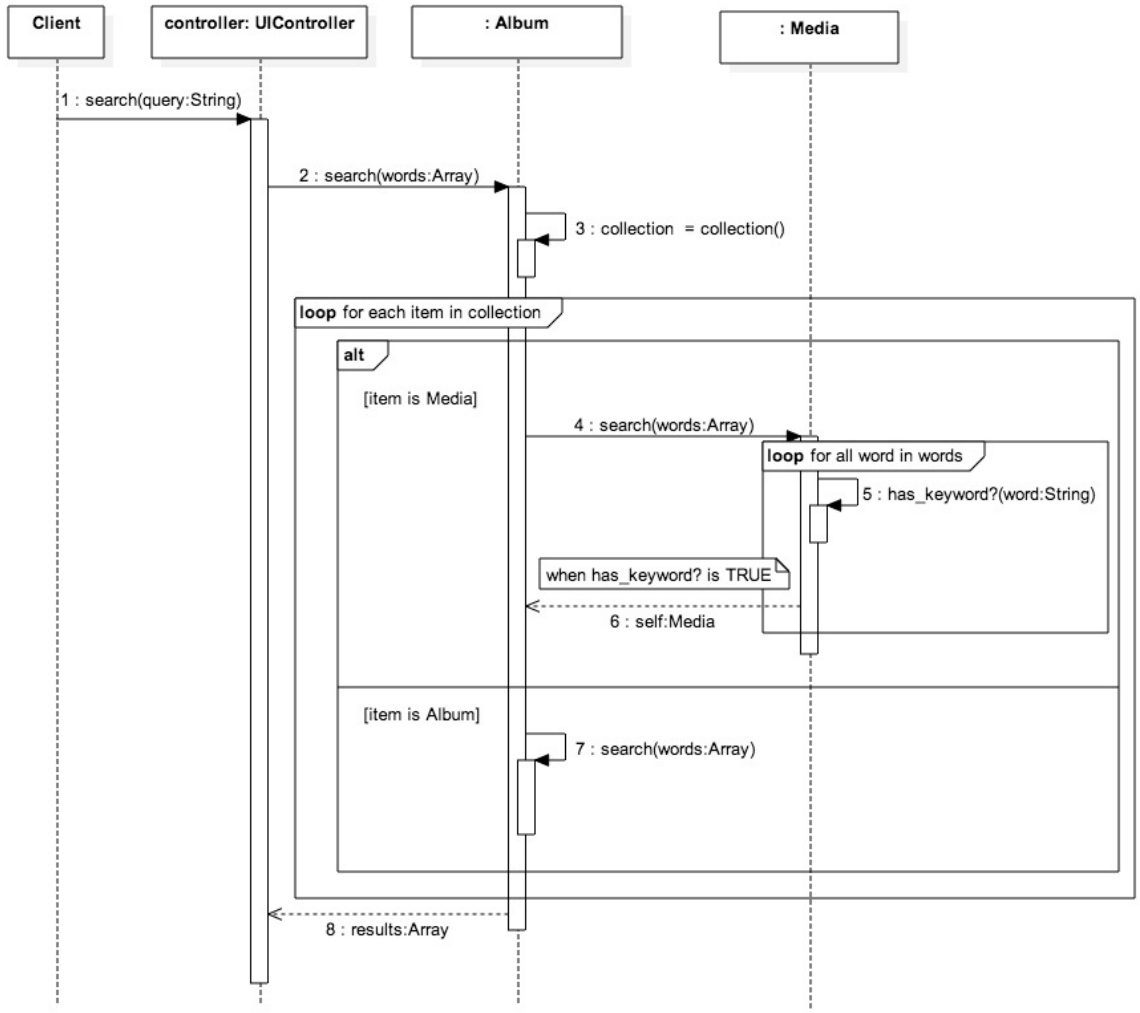


3.2.4 Persistence layer



3.3 Sequence Diagrams

The following sequence diagram outlines the interaction among the UIController, Album and Media classes when the search feature is used to find a photo or a group of photos that has one or more of the keyword(s) specified.



4 Class Specifications

This section provides the design specification for each class that is included in this system inclusive of the functions, attributes and data types. The purpose of each class, attribute, and method is clearly documented. This section also specifies the relationships between the different classes and provides their interface details.

High-level pseudo code is used to describe each method of each class.

4.1 Content

The Content class is the generalization for the Album and Media classes. This class has a timestamp attribute and no methods.

4.1.1 Attributes of Content class

4.1.1.1 timestamp

Purpose: This attribute holds the current system-generated timestamp

Data type: Time

Visibility: Private

4.1.2 Methods available in Content class

None

4.2 Album

The Album class serves as a container for Media objects and other Album objects (Content objects). This class has a name attribute and a collection attribute, and provides methods to add, delete, move, and copy Content objects that exists within. Album inherits from the Content class.

4.2.1 Attributes of Album class

4.2.1.1 name

Purpose: This attribute holds the name of the album.

Data type: String

Visibility: Private

4.2.1.2 collection

Purpose: This attribute holds all the Content objects an Album object.

Data type: Array

Visibility: Private

4.2.2 Methods available in Album class

4.2.2.1 new (String name)

Purpose: Creates a new instance of Album with the specified name in parameter: *name*.

Precondition: N/A

Post condition: Returns an instance of Album

Description:
`SET attribute: name to the parameter: name`
`SET attribute: timestamp to the current date/time`
`SET attribute: collection to an empty array`

4.2.2.2 name ( )

Purpose: Returns the name of the current instance of Album.

Precondition: The attribute: *name* has a value

Post condition: The name of the current instance of Album has been returned.

Description:
`RETURN attribute: name`

4.2.2.3 name= (String name)

Purpose: To change the name of the current instance of Album to the value specified in the parameter.

Precondition: Parameter: *name* has a value.

Post condition: The attribute: *name* is changed to the value passed by the parameter: *name*.

Description:
`SET attribute: name to parameter: name`

4.2.2.4 collection ( )

Purpose: Retrieve and return all the Content objects belonging to the current instance of Album.

Precondition: The attribute: *collection is* not empty.

Post condition: All the Content objects in attribute: *collection* has been returned.

Description:

```
RETURN attribute: collection
```

4.2.2.5 search (Array words)

Purpose: Find all the Media objects in the current instance of Album and descending Albums that has keyword(s) that match any of the query strings in the parameter: *words*.

Precondition:

- Parameter: *words* has an Array of Strings.
- One or more Media objects in attribute: *collection,* have one or more keywords associated with it.

Post condition: An array of Media objects matching any of the specified query strings has been returned.

Description:

```
DECLARE search_results as Array
FOR each content in attribute: collection
   CALL content.search with words RETURNING
   media
   APPEND media to search_results
ENDFOR
RETURN search_results
```

4.2.2.6 add (Content content)

Purpose: Adds an Album or Media object (Content object) to the current instance of Album.

Precondition: The content parameter has a Content object.

Post condition: The current instance of Album has the Content object

passed via parameter: *content* in its attribute: *collection*.

Description: `APPEND parameter: content to attribute: collection`

4.2.2.7 delete (Content content)

Purpose: Deletes the specified Content object from the current instance of Album.

Precondition:
- Parameter: *content* has a Content object
- The Content object in parameter: *content* exists in attribute: *collection*.

Post condition: The Content object specified via parameter: *content* is deleted from the current instance of Album.

Description: `DELETE parameter: content from the attribute: collection`

4.2.2.8 move (Content content, Album destination)

Purpose: Moves the specified Content object from the current instance of Album to another instance of Album that is specified.

Precondition:
- Parameter: *content* has a Content object
- Parameter: *destination* has an Album object
- The Content object in parameter: *content* exists in attribute: *collection*.

Post condition: The specified Content object is removed from the current instance of Album and is added to the instance of the Album specified in parameter: *destination*.

Description: `CALL copy with parameter: content and parameter: destination`

`CALL delete with parameter: content`

4.2.2.9 copy (Content content, Album destination)

Purpose: Make a copy of the specified Content object from the current instance of Album to another instance of Album that is specified.

Precondition:
- Parameter: *content* has a Content object
- Parameter: *destination* has an Album object
- The Content object in parameter: *content* exists in attribute: *collection*.

Post condition: A copy of the specified Content object is added to the instance of Album specified in parameter: *destination*.

Description: `CALL parameter: destination.add with parameter: content`

4.3 Media

The Media class serves as the generalization for the Photo class. This class consists of the keywords attribute and other attributes that describe the properties of a media object. This class consists of methods to manage and search its keywords. The Media class inherits from the Content class and serves as the generalization for the Photo class.

4.3.1 Attributes of Media class

4.3.1.1 keywords

Purpose: This attribute is a collection that holds all the keywords for a Media object.

Data type: Array

Visibility: Private

4.3.1.2 path

Purpose: This attribute holds the path to the media file added for a Media object.

Data type: String

Visibility: Private

4.3.1.3 extension

Purpose: This attribute holds the file extension of the media file of a Media object.

Data type: String

Visibility: Private

4.3.1.4 thumb_path

Purpose: This attribute holds the path to the image thumbnail for a Media object.

Data type: String

Visibility: Private

4.3.2 Methods available in Media class

4.3.2.1 keywords ()

Purpose: Retrieves and returns all the keywords associated with the current instance of Media.

Precondition: The current instance of Media has one or more keywords.

Post condition: All the keywords associated with the current instance of Media are returned.

Description: `RETURN attribute: keywords`

4.3.2.2 add_keyword (word)

Purpose: Adds a keyword to the current instance of Media.

Precondition: Parameter: *word* is not empty

Post condition: The current instance of Media has the keyword specified in parameter: *word*.

Description: `APPEND parameter: word to attribute: keywords`

4.3.2.3 delete_keyword (word)

Purpose: Deletes the specified keyword from the current instance of Media.

Precondition:
- Parameter: *word* is not empty
- The value of parameter: *word* exists in attribute: *keywords*

Post condition: The keyword specified is removed from the current instance of Media.

Description: `DELETE parameter: word from attribute: keywords`

4.3.2.4 has_keyword? (word)

Purpose: Checks if the current instance of Media has the specified keyword.

Precondition: Parameter: *word* is not empty

Post condition: Returns a Boolean value indicating whether or not the specified word exists in the current instance of Media.

Description:
```
IF parameter: word exists in attribute: keywords
      RETURN TRUE
ELSE
      RETURN FALSE
ENDIF
```

4.3.2.5 search (Array words)

Purpose: Returns the current instance of Media if it has one or more of the words specified in the parameter: *words* as a keyword.

Precondition: Parameter: *words* is not empty

Post condition: Returns the current instance of Media if keyword exists.

Description:

```
FOR each word in parameter: words
      IF CALL has_keyword? with word returns
      TRUE
            RETURN SELF
ENDFOR
```

4.3.2.6 modify_keyword (String word, String new_word)

Purpose: Changes an existing keyword of the current instance of Media to a new word.

Precondition:

- Parameter: *word* exists in attribute: *keywords*
- Parameter: *new_word* is not empty

Post condition: The specified keyword for the current instance of Media is replaced with the new keyword.

Description:

```
FIND the index of parameter: word in attribute:
keywords RETURNING position

SET the value of attribute: keywords at index
position to parameter: new_word
```

4.3.2.7 extension ()

Purpose: Retrieves and returns the file extension of the current instance of Media.

Precondition: N/A

Post condition: The file extension for the current instance of Media is returned.

Description: `RETURN attribute: extension`

4.3.2.8 path ()

Purpose: Retrieves and returns the file path of the current instance of Media.

Precondition: Attribute: *path* has a value

Post condition: The file path for the current instance of Media is returned.

Description: `RETURN attribute: path`

4.3.2.9 thumbnail_path ()

Purpose: Retrieves and returns the file path for the thumbnail image of the current instance of Media.

Precondition: N/A

Post condition: The file path for the thumbnail image belonging to the current instance of Media is returned.

Description: `RETURN attribute: thumb_path`

4.4 Photo

The Photo class serves as an encapsulation for a photo Media type. This class only has a method to create a new instance. Photo inherits from the Media class.

4.4.1 Attributes of Photo class

None

4.4.2 Methods available in Photo class

4.4.2.1 new (String path)

Purpose: Creates a new instance of Photo with the specified image file path in parameter: *path*.

Precondition:
- An image for this Photo instance was uploaded and processed.
- Parameter: *path* is the path to the image uploaded.

Post condition: Returns an instance of Photo

Description: `SET attribute: path to parameter: path`

4.5 ImageManipulator

The ImageManipulator class is a singleton that uses a third-party image manipulation code library to manipulate any image specified to it. This class provides methods to perform the following image manipulation techniques:

- Convert to black and white
- Adjust brightness
- Adjust contrast
- Rotate
- Crop
- Scale
- Thumbnail generation

4.5.1 Attributes of ImageManipulator class

4.5.1.1 original_image_path

Purpose: This attribute holds the path to the original image that is to be manipulated.

Data type: String

Visibility: Private

4.5.1.2 image_path

Purpose: This attribute holds the path to the image that has been manipulated.

Data type: String

Visibility: Private

4.5.1.3 instance

Purpose: This attribute holds the only instance of the ImageManipulator class.

Data type: ImageManipulator

Visibility: Private

4.5.2 Methods available in ImageManipulator class

4.5.2.1 open (String img_path)

Purpose: This method is a Static method that opens the image at img_path for manipulation and returns the attribute: *instance*.

Precondition: Parameter: *img_path* is not empty.

Post condition: Returns an instance of ImageManipulator

Description:

```
IF attribute: instance is NULL
      CALL ImageManipulator.new RETURNING
      instance
      SET attribute: instance to instance
      Initialize the image manipulation library
ENDIF
SET attribute: original_image_path to parameter:
img_path
RETURN attribute: instance
```

4.5.2.2 new ()

Purpose: This method is a ***private*** method that creates a new instance of ImageManipulator.

Precondition: No instance of ImageManipulator exists.

Post condition: Returns an instance of ImageManipulator

Description: N/A

4.5.2.3 gray_scale ()

Purpose: This method uses the Image manipulation library to convert the target image to black and white.

Precondition: The target image exists.

Post condition: The specified image is converted to black and white and its path is set in attribute: *image_path*.

Description:

```
CALL target_image RETURNING target_path

Convert image at target_path to black and white
using image manipulation library

SET attribute: image_path to the path of the
converted black and white image
```

4.5.2.4 brightness (Integer val)

Purpose: This method uses the Image manipulation library to adjust the brightness of the target image.

Precondition: The target image exists.

Post condition: The brightness of the target image is adjusted based on the value of parameter: *val* and its path is set in attribute: *image_path*.

Description:

```
CALL target_image RETURNING target_path
```

Adjust the brightness of the image at *target_path* base on parameter: *val* using image manipulation library

SET attribute: *image_path* to the path of the adjusted image

4.5.2.5 contrast (Integer val)

Purpose: This method uses the Image manipulation library to adjust the contrast of the target image.

Precondition: The target image exists.

Post condition: The contrast of the target image is adjusted based on the value of parameter: *val* and its path is set in attribute: *image_path*.

Description: CALL target_image RETURNING *target_path*

Adjust the contrast of the image at *target_path* based on parameter: *val* using image manipulation library

SET attribute: *image_path* to the path of the adjusted image

4.5.2.6 rotate (Integer degree)

Purpose: This method uses the Image manipulation library to rotate the target image by the degree value specified.

Precondition: The target image exists.

Post condition: The target image is rotate by the specified degrees and its

path is set in attribute: *image_path*.

Description: `CALL target_image RETURNING target_path`

`Rotate the image at target_path based on parameter: degree using image manipulation library`

`SET attribute: image_path to the path of the rotated image`

4.5.2.7 crop (Integer width, Integer height)

Purpose: This method uses the Image manipulation library to crop the target image using the width and height specified.

Precondition: The target image exists.

Post condition: The target image is cropped by the specified width and height and its path is set in attribute: *image_path*.

Description: `CALL target_image RETURNING target_path`

`Crop the image at target_path based on parameter: width and parameter: height using image manipulation library`

`SET attribute: image_path to the path of the cropped image`

4.5.2.8 scale (Integer width)

Purpose: This method uses the Image manipulation library to scale the target image proportionally by the width specified.

Precondition:

- The target image exists.

- Paramter: *scale* has a value.

Post condition: The target image is scaled proportionally by the specified width and its path is set in attribute: *image_path*.

Description: `CALL target_image RETURNING target_path`

`Scale the image at target_path proportionally by parameter: width specified, using image manipulation library`

`SET attribute: image_path to the path of the scaled image`

4.5.2.9 make_thumbnail (Integer width, Integer height)

Purpose: Generates a thumbnail image based on the specified width and height.

Precondition:
- The target image exists.
- Parameter: *width* has a value.
- Parameter: *height* has a value.

Post condition: A thumbnail based on the specified parameter: *width* and parameter: *height* is generated from the target image and its path is set in attribute: *image_path*.

Description:
```
CALL scale with parameter: width
CALL crop with parameter: width and parameter: height
```

4.5.2.10 target_path ()

Purpose: Determines and returns the target path for the image to be manipulated.

Precondition: Either attribute: *image_path* or attribute:

original_image_path has an image path

Post condition: The targeted image path is determined and returned.

Description:

```
IF attribute: image_path is empty
      RETURN attribute: original_image_path
ELSE
      RETURN attribute: image_path
END
```

4.6 Email

The Email class is responsible for composing and sending an email that has an attachment. This class consists of attributes and methods to setup an Email object with the required information. This class uses a third-party code library to send emails.

4.6.1 Attributes of Email class

4.6.1.1 recipient_email

Purpose: This attribute holds the email address of the intended recipient of an Email object.

Data type: String

Visibility: Private

4.6.1.2 subject

Purpose: This attribute holds the subject of an Email object.

Data type: String

Visibility: Private

4.6.1.3 message

Purpose: This attribute holds the message of an Email object.

Data type: String

Visibility: Private

4.6.1.4 attachment_path

Purpose: This attribute holds the path for the file that is to be sent in an Email object.

Data type: String

Visibility: Private

4.6.2 Methods available in Email class

4.6.2.1 new (EmailPreference preference)

Purpose: Creates a new instance of Email with the specified email preference from parameter: *preference*.

Precondition: An EmailPreference instance exists in parameter: *preference*.

Post condition:
- A new instance of Email is created.
- The email library is initialized with the values of parameter: *preference.properties*.

Description: `Initialize email library with parameter: preference.properties`

4.6.2.2 recipient_email= (String email_address)

Purpose: Assigns the intended recipient's email address to the current instance of Email.

Precondition:
- Parameter: email_address has an email address.
- The current instance of Email exists.

Post condition: The attribute: *recipient_email* has the email address passed via parameter: *email_address.*

Description: `SET attribute: recipient_email to parameter:`

email_address

4.6.2.3 subject= (String subject)

Purpose: Assigns the subject to the current instance of Email

Precondition:
- Parameter: subject has a value
- The current instance of Email exists

Post condition: The attribute: *subject* holds the value from parameter: *subject*

Description: `SET attribute: subject to parameter: subject`

4.6.2.4 message= (String message)

Purpose: Assigns a message to the current instance of Email

Precondition:
- Parameter: *message* has a value
- The current instance of Email exists

Post condition: The attribute: *message* has the value passed via parameter: *message*

Description: `SET attribute: message to parameter: message`

4.6.2.5 attach (String path_to_file)

Purpose: Assigns the path of the file that is to be sent as an attachment to the current instance of Email

Precondition:
- Parameter: *path_to_file* has a file path
- The current instance of Email exists

Post condition: The attribute: *attachment_path* has the file path passed via parameter: *path_to_file*

Description: `SET attribute: attachment_path to parameter: path_to_file`

4.6.2.6 send ()

Purpose: Sends an email with the attached file to the intended recipient using the values from the attributes of the current instance of Email.

Precondition:
- Attribute: *recipient_email* has an email address
- Attribute: *subject* has a value
- Attribute: *message* has a value
- Attribute: *attachment_path* has a path to the target file
- The current instance of Email exists

Post condition: An email with the subject of attribute: *subject*, message of attribute: *message* and attachment of attribute: *attachment_path* is sent to the specified intended recipient of attribute: *recipient_email*.

Description:

```
Populate the email library's message options
with attribute: recipient_email, attribute:
subject, attribute: message and attribute:
attachment_path

SEND email using the initialized email library
```

4.7 EmailPreference

The EmailPreference is a singleton class that has the responsibility of providing the necessary configuration values to the integrated email library via the Email class. This class uses the EmailPreferenceRepository interface class to use the appropriate persistence implementation to populate it self and to persist values when updated.

4.7.1 Attributes of EmailPreference class

4.7.1.1 server

Purpose: This attribute holds the outgoing mail server's address.

Data type: String

Visibility: Private

4.7.1.2 port

Purpose: This attribute holds the port number for the outgoing mail address.

Data type: Integer

Visibility: Private

4.7.1.3 username

Purpose: This attribute holds the username for the email account corresponding to the user's email address.

Data type: String

Visibility: Private

4.7.1.4 password

Purpose: This attribute holds the password for the email account corresponding to the user's email address.

Data type: String

Visibility: Private

4.7.1.5 authentication_type

Purpose: This attribute holds the authentication type required for the email account corresponding to the user's email account.

Data type: String

Visibility: Private

4.7.1.6 domain

Purpose: This attribute is optional and can be used to hold the domain for the email account corresponding to the user's email account.

Data type: String

Visibility: Private

4.7.1.7 sender_email

Purpose: This attribute holds the user's email address.

Data type: String

Visibility: Private

4.7.1.8 preference_repo

Purpose: This attribute holds an instance of the EmailPreferenceRepository interface.

Data type: EmailPreferenceRepository

Visibility: Private

4.7.1.9 instance

Purpose: This attribute holds the current instance of EmailPreference.

Data type: EmailPreference

Visibility: Private

4.7.2 Methods available in EmailPreference class

4.7.2.1 new ( )

Purpose: This method is a private method that creates a new instance of EmailPreference.

Precondition: There is no instance of EmailPreference available, i.e. attribute: *instance* is NULL.

Post condition:
- Returns an instance of EmailPreference.
- Initialized the EmailPreferenceRepository interface class

Description: `CALL EmailPreferenceRepository.new with` *`filepath`* `RETURNING` *`preference_repository`*

`SET attribute:` *`preference_repo`* `to` *`preference_repository`*

4.7.2.2 update (Hash args)

Purpose: To update the attributes of the instance of EmailPreference with the values of parameter: *args*.

Precondition:
- An instance of EmailPreference exists.
- Parameter: *args* has values.

Post condition:
- The values of the attributes of the EmailPreference instance are updated.
- The updated EmailPreference attributes were persisted.

Description: `SET attribute:` *`server`* `to parameter:` *`args'`*`s server value`
`SET attribute:` *`port`* `to parameter:` *`args'`*`s port value`
`SET attribute:` *`username`* `to parameter:` *`args'`*`s`

```
username value
SET attribute: password to parameter: args's
password value
SET attribute: authentication_type to parameter:
args's authentication_type value
SET attribute: domain to parameter: args's
domain value
SET attribute: sender_email to parameter: args's
sender_email value

CALL properties RETURNING properties
CALL attribute: preference_repo.save with
properties
```

4.7.2.3 properties ( )

Purpose: Returns a Hash of the attributes belonging to the instance of EmailPreference.

Precondition:
- An instance of EmailPreference exists.
- Each attribute of the EmailPreference instance has a value.

Post condition: The value of each attribute of the EmailPreference instance is used to build a Hash of values and is returned.

Description:

```
Initialize new Hash RETURNING properties
SET server value of properties to attribute:
server
SET port value of properties to attribute: port
SET username value of properties to attribute:
username
SET password value of properties to attribute:
password
SET authentication_type value of properties to
```

```
attribute: authentication_type
SET domain value of properties to attribute:
domain
SET sender_email value of properties to
attribute: sender_email

RETURN properties
```

4.7.2.4 instance ( )

Purpose: This method is a ***Static method*** that returns the instance of EmailPreference.

Precondition: N/A

Post condition: Returns an instance of EmailPreference

Description:

```
IF attribute: instance is NULL
      CALL EmailPreference.new RETURNING
instance
      SET attribute: instance to instance
      CALL attribute: preference_repo.load
RETURNING values as Hash
      CALL update with values
ENDIF
RETURN attribute: instance
```

4.8 Slideshow

The Slideshow class serves as an aggregate of the slideshow preference and a collection of the selected Photos that are to be used by the slideshow library and slideshow GUI to start a slideshow.

4.8.1 Attributes of Slideshow class

4.8.1.1 photos

Purpose: This attribute holds the collection of selected Photos to be displayed via slideshow.

Data type: Array

Visibility: Private

4.8.1.2 preference

Purpose: This attribute holds all the slideshow preference.

Data type: SlideshowPreference

Visibility: Private

4.8.2 Methods available in Slideshow class

4.8.2.1 new ( )

Purpose: Creates a new instance of Slideshow

Precondition: SlideshowPreference has values or access to the required values.

Post condition:
- Retrieves and sets the slideshow preference.
- Returns an instance of Slideshow.

Description:
```
CALL Static SlideshowPreference.instance
RETURNING preference
SET attribute:preference to preference
```

4.8.2.2 photos= (Array photos)

Purpose: Populate the instance of Slideshow with the selected Photo objects passed via parameter: *photos*

Precondition: Parameter: *photos* has an array of Photo objects

Post condition: Attribute: *photos* has an array of Photo objects passed via

parameter: *photos*

Description: `SET attribute: photos to parameter: photos`

4.8.2.3 photos ()

Purpose: Retrieves and returns an Array of Photo objects that were selected for the slideshow.

Precondition: Attribute: *photos* has a collection of Photo objects.

Post condition: The Photo objects of attribute: *photos* are returned as an Array.

Description: `RETURN attribute: photos`

4.8.2.4 preference ()

Purpose: Retrieves and returns the slideshow preference of the instance of Slideshow.

Precondition: Attribute: *preference* has a SlideshowPreference instance.

Post condition: The SlideshowPreference instance in attribute: *preference* is returned.

Description: `RETURN attribute: preference`

4.9 SlideshowPreference

The SlideshowPreference is a singleton class that has the responsibility of providing the necessary configuration values to the integrated slideshow library via the Slideshow class. This class uses the SlideshowPreferenceRepository interface class to use the appropriate persistence implementation to populate it self and to persist values when updated.

4.9.1 Attributes of SlideshowPreference class

4.9.1.1 transition_duration

Purpose: This attribute holds the transition duration for each slide in seconds.

Data type: Integer

Visibility: Private

4.9.1.2 loop_count

Purpose: This attribute holds the number of complete loops the slideshow should take when it plays.

Data type: Integer

Visibility: Private

4.9.1.3 preference_repo

Purpose: This attribute holds an instance of the SlideshowPreferenceRepository interface.

Data type: SlideshowPreferenceRepository

Visibility: Private

4.9.1.4 instance

Purpose: This attribute holds the current instance of SlideshowPreference.

Data type: SlideshowPreference

Visibility: Private

4.9.2 Methods available in SlideshowPreference class

4.9.2.1 new ( )

Purpose: This method is a ***private method*** that creates a new

instance of SlideshowPreference.

Precondition: There is no instance of SlideshowPreference available, i.e. attribute: *instance* is NULL.

Post condition:
- Returns an instance of SlideshowPreference.
- Initialized the SlideshowPreferenceRepository interface class

Description: `CALL SlideshowPreferenceRepository.new with filepath RETURNING preference_repository`

`SET attribute: preference_repo to preference_repository`

4.9.2.2 update (Hash args)

Purpose: Updates the attributes of the instance of SlideshowPreference with the specified values of parameter: *args*.

Precondition:
- An instance of SlideshowPreference exists.
- Parameter: *args* has values.

Post condition:
- The values of the attributes of the SlideshowPreference instance are updated.
- The updated SlideshowPreference attributes were persisted.

Description: `SET attribute: transition_duration to parameter: args's transition_duration value`

`SET attribute: loop_count to parameter: args's loop_count value`

`CALL properties RETURNING properties`

```
CALL attribute: preference_repo.save with
properties
```

4.9.2.3 properties ( )

Purpose: Returns a Hash of the attributes belonging to the instance of SlideshowPreference.

Precondition:
- An instance of SlideshowPreference exists.
- Each attribute of the SlideshowPreference instance has a value.

Post condition: The value of each attribute of the SlideshowPreference instance is returned as a Hash.

Description:

```
Initialize new Hash RETURNING properties

SET transition_duration value of properties to
attribute: transition_duration

SET loop_count value of properties to attribute:
loop_count

RETURN properties
```

4.9.2.4 instance ( )

Purpose: This method is a ***Static method*** that returns the instance of SlideshowPreference.

Precondition: N/A

Post condition: Returns an instance of SlideshowPreference

Description:

```
IF attribute: instance is NULL
     CALL SlideshowPreference.new RETURNING
     instance
```

```
            SET attribute: instance to instance

            CALL attribute: preference_repo.load
            RETURNING values as Hash

            CALL update with values
        ENDIF
        RETURN attribute: instance
```

4.10 UIController

The UIController is the generalized controller class for all the other controller classes within the system. The main responsibility of this class is to keep communication flowing between the view layer (GUI) and the model layer. Therefore, this class is the main contact for the GUI to the main parts of the system.

4.10.1 Attributes of UIController class

4.10.1.1 current_album

Purpose: This attribute holds the current instance of Album that is to be interfaced with.

Data type: Album

Visibility: Private

4.10.1.2 content_factory

Purpose: This attribute holds the instance of ContentFactory which will be used create Content objects.

Data type: ContentFactory

Visibility: Private

4.10.1.3 content_repo

Purpose: This attribute holds an instance of the ContentRepository interface.

Data type: ContentRepository

Visibility: Private

4.10.1.4 albums

Purpose: This attribute holds all the entire collection Album objects.

Data type: Album

Visibility: Private

4.10.2 Methods available in UIController class

4.10.2.1 new ( )

Purpose: Creates a new instance of UIController and retrieve existing content from store.

Precondition: The application was started.

Post condition:

- Returns an instance of UIController.
- Initialized the ContentRepository interface class
- Existing content is retrieved from store and assigned to attribute: *albums*.
- Attribute: *albums* is assigned to attribute: *current_album*

Description:

```
CALL ContentFactory.new RETURNING
content_factory
SET attribute: content_factory to
content_factory
```

CALL ContentRepository.new RETURNING *content_repo*
SET attribute: *content_repo* to *content_repo*

CALL attribute: *content_repo.retrieve* RETURNING *albums*
SET attribute: *albums* to *albums*

SET attribute: *current_album* to *albums*

4.10.2.2 create_album (String name)

Purpose: Creates an instance of Album, and adds it to the current album.

Precondition: Attribute: *current_album* is not NULL.

Post condition:
- A new instance of Album is added to attribute: *current_album*.
- The new Album is persist to disk

Description: CALL album_factory.make with parameter: *name* RETURNING *album*

CALL attribute: *current_album.add* with *album*

CALL attribute: *content_repo.insert* with *album* and attribute: *current_album*

4.10.2.3 add_photo (String path, Array keywords = NIL)

Purpose: Creates an instance of Photo and adds it to the current album.

Precondition:
- Attribute: *current_album* has an instance of Album.

- Parameter: *path* has a path to an image file

Post condition:

- A new instance of Photo is added to attribute: *current_album*.
- If parameter: *keywords* is not NIL, its values are added to Photo.
- The new Photo is persist to disk

Description:

```
CALL photo_factory.make with parameter: path
RETURNING photo

If parameter: keywords not equal to NIL
     CALL add_keywords with photo and
     parameter: keywords RETURNING photo
CALL attribute: current_album.add with photo
CALL attribute: content_repo.insert with photo
and attribute: current_album
```

4.10.2.4 add_keywords (Array keywords, Media media)

Purpose: This is a ***private method*** that adds one or more keywords to the specified Media object.

Precondition:

- The Media object in parameter: *media* exists.
- Parameter: *keywords* has an Array of Strings.

Post condition: The specified keywords are added to the target Media object.

Description:

```
FOR each word in parameter: keywords
     CALL parameter: media.add_keyword with
     word
ENDFOR
RETURN parameter: media
```

4.10.2.5 rename (Album album, String new_name)

Purpose: Changes the name of the target album to a specified new name.

Precondition:
- The Album object in parameter: *album* exists.
- Parameter: *new_name* has a value.

Post condition: The specified Album has the specified new name.

Description: `CALL attribute: album.name= with parameter: new_name`

4.10.2.6 move (Content content, Album destination)

Purpose: Moves a Content object from the current album to the specified Album object.

Precondition:
- The Album object in parameter: *destination* exists.
- The Content object in parameter: *content* exists in attribute: *current_album*.

Post condition: The specified Content object is moved to the specified Album.

Description: `CALL attribute: current_album.move with parameter: album and parameter: destination`

4.10.2.7 copy (Media media, Album destination)

Purpose: Copies a Media object from the current album to the specified Album object.

Precondition:
- The Album object in parameter: *destination* exists.
- The Media object in parameter: *media* exists in attribute: *current_album*.

Post condition: The Album object in parameter: *destination* has a copy of the Media object specified in parameter: *media*.

Description: `CALL attribute: current_album.copy with`

parameter: *media* and parameter: *destination*

4.10.2.8 delete (Content content)

Purpose: Deletes the specified Content object from the current album.

Precondition: The Content object in parameter: *content* exists in attribute: *current_album*.

Post condition: The specified Content object in parameter: *content* has been removed from attribute: *current_album*.

Description:
CALL attribute: *current_album.delete* with parameter: *content*

4.10.2.9 search (String query)

Purpose: Find the Media objects in the current album or descendant album(s) that has one or more keywords that matches any of the words specified.

Precondition:
- The current album has Content objects
- Parameter: *query* has a value

Post condition: The Media objects in the current album or descendant album(s) are found and returned.

Description:
PARSE parameter: *query* delimited by comma RETURNING Array of strings as *words*

CALL attribute: *current_album.search* with words RETURNING *results*

RETURN *results*

4.10.2.10 retrieve_collection ()

Purpose: Retrive and return all the Content objects that exists in the current album.

Precondition: The current album has Content objects

Post condition: All the Content objects that exist in attribute: current_album are returned.

Description:
```
CALL attribute: current_album.contents
RETURNING contents
SORT contents by timestamp in descending order
RETURN contents
```

4.11 MediaController

The MediaController class inherits from the UIController class and serves as the main contact between the GUI that needs full access to a Media object and the Media class.

4.11.1 Attributes of MediaController class

4.11.1.1 media

Purpose: This attribute holds the current instance of Media that is to be interfaced with.

Data type: Media

Visibility: Private

4.11.2 Methods available in MediaController class

4.11.2.1 new (Media media)

Purpose: Creates a new instance of MediaController

Precondition: The GUI to execute a single Media and display its properties is chosen.

Post condition:
- Initializes attribute: *media*

- Returns an instance of MediaController.

Description: `SET attribute: media to parameter: media`

4.11.2.2 add_keyword (String word)

Purpose: Adds a keyword to the current instance of Media.

Precondition: Parameter: *word* has a value.

Post condition: The specified word is added to the keyword of the current instance of Media.

Description: CALL attribute: *media.add_keyword* with parameter: *word*

4.11.2.3 delete_keyword (String word)

Purpose: Deleted the specified keyword from the current instance of Media.

Precondition:
- Parameter: *word* has a value.
- Parameter: *word* exists in the keyword list of the current instance of Media

Post condition: The specified keyword is deleted from the keyword list of the current instance of Media.

Description: `CALL attribute: media.delete_keyword with parameter: word`

4.11.2.4 modify_keyword (String word, String new_word)

Purpose: Changes an existing keyword in the current instance of Media to a new keyword.

Precondition:
- Parameter: *word* has a value.
- Parameter: *new_word* has a value.
- Parameter: *word* exists in the keyword list of the

current instance of Media

Post condition: The specified keyword is replaced with the new keyword in the keyword list of the current instance of Media.

Description: `CALL attribute: media.modify_keyword with parameter: word and parameter: new_word`

4.12 EmailController

The EmailController class inherits the UIController class and serves as the main contact between the email form GUI and the Email class.

4.12.1 Attributes of EmailController class

None

4.12.2 Methods available in EmailController class

4.12.2.1 Send_content (Content content, Hash args)

Purpose: Sends the specified Content object as an email message to the recipient specified.

Precondition:

- The command to send a Content object as email has been chosen.
- Parameter: *content* has a Content object.
- Parameter: *args* has values for email recipient, subject and message.
- The necessary email preference properties are configured.
- The necessary email library is configured and is functioning properly.

Post condition: An email message with the specified content as an attachment has been sent to the intended recipient with the specified subject and message.

Description: `CALL Static EmailPreference.instance returning preference`

```
CALL Email.new with preference RETURNING email

CALL email.recipient_email with recipient_email
value of parameter: args

CALL email.subject with subject value of
parameter: args

CALL email.message with message value of
parameter: args

IF parameter: content is an Album object
     DECLARE media_files as Array

      FOR each media in parameter: content &
      Albums within
           APPEND media.path to media_files
     ENDFOR

      COMPRESS files in media_files RETURNING
      compressed
     CALL email.attach with compressed
ELSE
     CALL email.attach with parameter:
     content.path
ENDIF

CALL email.send
```

4.13 EditImageController

The EditImageController class inherits from the UIController class and serves as the main contact between the image editing GUI and the ImageManipulator class.

4.13.1 Attributes of EditImageController class

4.13.1.1 photo

Purpose: This attribute holds the current instance of Photo that is to be interfaced with.

Data type: Photo

Visibility: Private

4.13.1.2 image_manipulator

Purpose: This attribute holds the instance ImageManipulator that is going to be used to manipulate the image of Photo.

Data type: ImageManipulator

Visibility: Private

4.13.2 Methods available in EditImageController class

4.13.2.1 new (Photo photo)

Purpose: Creates a new instance of EditImageController and make preparations for manipulating the image of the specified Photo.

Precondition:

- The command to edit the image of a Photo is chosen.
- The parameter: *photo* has a Photo object.

Post condition:

- ImageManipulator is setup with the specified image of Photo

- Returns an instance of EditImageController.

Description: `SET attribute: photo to parameter: photo`

`CALL Static ImageManipulator.open with attribute: photo.path RETURNING image_manipulator`

`SET attribute: image_manipulator to image_manipulator`

4.13.2.2 convert_to_grayscale ( )

Purpose: Converts the target image to black and white.

Precondition:
- The attribute: *photo* has a Photo object.
- The attribute: *image_manipulator* has an instance of ImageManipulator that has a path to the image of the specified Photo.

Post condition: A black and white copy of the target image is in the application's temporary folder.

Description: `CALL attribute: image_manipulator.grayscale`

4.13.2.3 Adjust_contrast (Integer val)

Purpose: Changes the contrast of the target image by the specified value.

Precondition:
- Parameter: *val* has an Integer value.
- The attribute: *photo* has a Photo object.
- The attribute: *image_manipulator* has an instance of ImageManipulator that has a path to the image of the specified Photo.

Post condition: A copy of the target image with adjusted contrast is in the application's temporary folder.

Description: `CALL attribute: image_manipulator.contrast with parameter: val`

4.13.2.4 Adjust_brightness (Integer val)

Purpose: Changes the brightness of the target image by the specified value.

Precondition:
- Parameter: *val* has an Integer value.
- The attribute: *photo* has a Photo object.
- The attribute: *image_manipulator* has an instance of ImageManipulator that has a path to the image of the specified Photo.

Post condition: A copy of the target image with adjusted contrast is in the application's temporary folder.

Description: `CALL attribute: image_manipulator.brightness with parameter: val`

4.13.2.5 rotate (Integer degree)

Purpose: Rotates the target image by the specified degree.

Precondition:
- Parameter: *degree* has an Integer value.
- The attribute: *photo* has a Photo object.
- The attribute: *image_manipulator* has an instance of ImageManipulator that has a path to the image of the specified Photo.

Post condition: A copy of the rotated target image is in the application's temporary folder.

Description: `CALL attribute: image_manipulator.rotate with parameter: degree`

4.13.2.6 image_path ()

Purpose: Returns the image path of the manipulated image that exists in the application's temporary folder.

Precondition:
- The image of the Photo object in attribute: *photo* has to be manipulated.
- The attribute: *image_manipulator* has an instance of ImageManipulator that has a path to the image of the specified Photo.

Post condition: The path to the manipulated image is returned.

Description:
```
CALL attribute: image_manipulator.image_path
RETURNING path

RETURN path
```

4.13.2.7 save (overwrite = TRUE)

Purpose: Binds the manipulated image with a Photo object.

Precondition:
- The image of the Photo object in attribute: photo has to be manipulated.
- The attribute: image_manipulator has an instance of ImageManipulator that has a path to the image of the specified Photo.

Post condition: The manipulated image has been binded with either the Photo object in attribute: *photo* or a new Photo object.

Description:
```
IF parameter: overwrite is TRUE
    Move manipulated image at attribute:
    image_manipulator.image_path to the
```

```
        application's image folder, replacing
        image at attribute: photo.path
    ELSE
        CALL add_photo with attribute:
        image_manipulator.image_path
    ENDIF
```

4.14 EmailPreferenceController

The EmailPreferenceController class inherits from the UIController class and serves as the main contact between the email preference configuration GUI and the EmailPreference class.

4.14.1 Attributes of EmailPreferenceController class

4.14.1.1 preference

Purpose: This attribute holds the instance of EmailPreference.

Data type: EmailPreference

Visibility: Private

4.14.2 Methods available in EmailPreferenceController class

4.14.2.1 new ()

Purpose: Creates a new instance of EmailPreferenceController and retrieves the instance of EmailPreference.

Precondition: The GUI to modify the email preference is chosen.

Post condition:
- Returns an instance of EmailPreferenceController.
- Retrieved the instance of EmailPreference

Description: `CALL Static EmailPreference.instance RETURNING preference`

```
SET attribute: preference to preference
```

4.14.2.2 preference= (Hash args)

Purpose: Updates the attributes of EmailPreference.

Precondition:
- Attribute: *preference* has the instance of EmailPreference.
- Parameter: *args* has a Hash of values for the keys: server, port, username, password, authentication_type, domain and sender_email.

Post condition: The instance of EmailPreference has the new values supplied via parameter: *args*.

Description:
```
CALL attribute: preference.update with
parameter: args
```

4.14.2.3 preference ( )

Purpose: Retrieves the properties of EmailPreference.

Precondition: Attribute: *preference* has the instance of EmailPreference.

Post condition: The properties of the instance of EmailPreference are returned as a Hash.

Description:
```
CALL attribute: preference.properties RETURNING
properties

RETURN properties
```

4.15 SlideshowPreferenceController

The SlideshowPreferenceController class inherits from the UIController to serve as the main contact between the slideshow preference configuration GUI and the SlideshowPreference class.

4.15.1 Attributes of SlideshowPreferenceController class

4.15.1.1 preference

Purpose: This attribute holds the instance of SlideshowPreference.

Data type: SlideshowPreference

Visibility: Private

4.15.2 Methods available in SlideshowPreferenceController class

4.15.2.1 new ()

Purpose: Creates a new instance of SlideshowPreferenceController and retrieves the instance of SlideshowPreference.

Precondition: The command to modify the slideshow preference is chosen.

Post condition:

- Returns an instance of SlideshowPreferenceController.
- Retrieved the instance of SlideshowPreference

Description: CALL Static *SlideshowPreference.instance* RETURNING *preference*

SET attribute: *preference* to *preference*

4.15.2.2 preference= (Hash args)

Purpose: Updates the attributes of SlideshowPreference.

Precondition:
- Attribute: *preference* has the instance of SlideshowPreference.
- Parameter: *args* has a Hash of values for the keys: transition_duration and loop_count.

Post condition: The instance of SlideshowPreference has the new values supplied via parameter: args.

Description: `CALL attribute: preference.update with parameter: args`

4.15.2.3 preference ( )

Purpose: Retrieves the properties of SlideshowPreference.

Precondition: Attribute: *preference* has the instance of SlideshowPreference.

Post condition: The properties of the instance of SlideshowPreference are returned as a Hash.

Description:
```
CALL attribute: preference.properties RETURNING properties

RETURN properties
```

4.16 SlideshowController

The SlideshowController class inherits from the UIController class, serves as the main contact between the slideshow GUI and the Slideshow class in collaboration with the slideshow library.

4.16.1 Attributes of SlideshowController class

None

4.16.2 Methods available in SlideshowController class

4.16.2.1 start_slideshow (Album target)

Purpose: Initializes the slideshow library with Slideshow and plays the slideshow Photo objects from the target Album and its sub Albums.

Precondition:
- Parameter: *target* has an Album object.
- An instance of Slideshow exists.
- The necessary slideshow preference properties are configured.
- The necessary slideshow library is configured and is functioning properly.

Post condition: A slideshow with the Photo objects of specified Album and its sub-Albums has started.

Description:

```
DECLARE photos as Array

FOR each Photo object as photo in parameter:
target and its sub-Albums
     APPEND photo.path to photos
ENDFOR

CALL Slideshow.new RETURNING slideshow
CALL slideshow.photos= with photos

Initialize the slideshow library with slideshow
CALL play
```

4.16.2.2 play ( )

Purpose: Commands the slideshow GUI to play the initialized

slideshow

Precondition: The slideshow library has been initialized with an instance of slideshow.

Post condition: The initialized slideshow has started to play.

Description: `COMMAND slideshow GUI to play slideshow`

4.16.2.3 stop ( )

Purpose: Commands the slideshow GUI to stop the current slideshow

Precondition: The initialized slideshow has been playing.

Post condition: Slideshow is not playing.

Description: `COMMAND slideshow GUI to stop slideshow`

4.17 ContentFactory

The ContentFactory interface serves as a contract for the Factory class(es) that implements it to create a type of Content. This interface defines the method that the implementation class(es) must have.

4.17.1 Methods defined by the ContentFactory interface

4.17.1.1 make (String arg)

Purpose: This is the method definition for the method that creates a Content object.

Return type: Content

Visibility: Private

4.18 MediaFactory

The MediaFactory interface that inherits from the ContentFactory interface serves as a contract for the Factory class(es) that implements it to create a type of Media. This interface defines the method that the implementation class(es) must have.

4.18.1 Methods defined by the MediaFactory interface

4.18.1.1 make (String arg)

Purpose: This is the method definition for the method that creates a Media object.

Return type: Media

Visibility: Private

4.19 AlbumFactory

The AlbumFactory class implements the ContentFactory interface to serve as the creator of Album objects.

4.19.1 Attributes of AlbumFactory class

None

4.19.2 Methods available in AlbumFactory class

4.19.2.1 make (String arg)

Purpose: Creates a new instance of Album that has the name specified at parameter: *arg*.

Precondition: Parameter: *arg* has a value.

Post condition:

- Returns an instance of Album.
- The instance of Album returned has the name attribute as specified in parameter: *arg*.

Description: `CALL Album.new with parameter: arg RETURNING`

```
album
RETURN album
```

4.20 PhotoFactory

The PhotoFactory class implements the MediaFactory interface to serve as the creator of Photo objects.

4.20.1 Attributes of PhotoFactory class

None

4.20.2 Methods available in PhotoFactory class

4.20.2.1 make (String arg)

Purpose: Creates a new instance of Photo that has an image at the path specified in parameter: *arg*.

Precondition: Parameter: *arg* has a value.

Post condition:

- Returns an instance of Photo.
- The instance of Photo returned has the path attribute as specified in parameter: *arg*.
- The instance of Photo returned has the thumbnail path for the generated image thumbnail.

Description:

```
RENAME the image at parameter: arg to the
current timestamp

MOVE image to the application's image folder
RETURNING path

CALL Photo.new with path RETURNING photo

CALL ImageManipulator.open with photo.path
```

```
RETURNING image

CALL image.make_thumbnail with width and height

CALL image.image_path RETURNING thumbnail_path

CALL photo.thumbnail_path= with thumbnail_path
RETURN photo
```

4.21 ContentRepository

The ContentRepository interface defines the methods that must be used in any of its implementations to serve in the persistent layer of the application to deal with operations relating obtaining from and persisting Content objects to the preferred data store.

4.21.1 Methods defined by the ContentRepository interface

4.21.1.1 new (String source)

Purpose: This is the method definition for the method that creates a new instance of the class that implements this interface.

Return type: ContentRepository

Visibility: Private

4.21.1.2 retrieve ( )

Purpose: This is the method definition for the method that retrieves all the content from storage and reconstitutes them in Content objects to be returned.

Return type: Content

Visibility: Private

4.21.1.3 update (Content content)

Purpose: This is the method definition for the method that updates the values corresponding to the specified content in storage.

Return type: Void

Visibility: Private

4.21.1.4 insert (Content content, Album parent)

Purpose: This is the method definition for the method that persist a Content object and associates it with a specified Album object.

Return type: Void

Visibility: Private

4.21.1.5 delete (Content content, Album parent)

Purpose: This is the method definition for the method that deletes the target Content object from a specified Album object.

Return type: Void

Visibility: Private

4.22 EmailPreferenceRepository

The EmailPreferenceRepository interface defines the methods that must be used in any of its implementations to serve in the persistent layer of the application to deal with operations relating to retrieving and persisting email preference values to the preferred data store.

4.22.1 Methods defined by the EmailPreferenceRepository interface

4.22.1.1 new (String source)

Purpose: This is the method definition for the method that creates a new instance of the class that implements this interface.

Return type: EmailPreferenceRepository

Visibility: Private

4.22.1.2 load ( )

Purpose: This is the method definition for the method that retrieves the email preference properties from storage and returns them as a Hash.

Return type: Hash

Visibility: Private

4.22.1.3 save (Hash preference)

Purpose: This is the method definition for the method that accepts a Hash of values corresponding to the keys: server, port, username, password, authentication_type, domain and sender_email, and persist them to storage.

Return type: Void

Visibility: Private

4.23 SlideshowPreferenceRepository

The SlideshowPreferenceRepository interface defines the methods that must be used in any of its implementations to serve in the persistent layer of the application to deal with operations relating retrieving and persisting slideshow values to the preferred data store.

4.23.1 Methods defined by the SlideshowPreferenceRepository interface

4.23.1.1 new (String source)

Purpose: This is the method definition for the method that creates a new instance of the class that implements this interface.

Return type: SlideshowPreferenceRepository

Visibility: Private

4.23.1.2 load ( )

Purpose: This is the method definition for the method that retrieves the slideshow preference properties from storage and returns them as a Hash.

Return type: Hash

Visibility: Private

4.23.1.3 save (Hash preference)

Purpose: This is the method definition for the method that accepts a Hash of values corresponding to the keys: transition_duration and loop_count, and persist them to storage.

Return type: Void

Visibility: Private

4.24 KVPContentRepository

The KVPContentRepository class implements the ContentRepository interface to serve in the persistent layer of the application to deal with operations relating retrieving and/or persisting Content objects to a key-value pair file. This class utilizes the power of the ContentFactory class to aid in the process of reconstituting the values retrieved from the key-value pair file into Content

objects. This class also has the capability to destruct Content objects into key-value pair when it is time to persist to file.

4.24.1 Attributes of KVPContentRepository class

4.24.1.1 file

Purpose: This attribute holds the path to the key-value pair file.

Data type: String

Visibility: Private

4.24.2 Methods available in KVPContentRepository class

4.24.2.1 new (String source)

Purpose: Creates a new instance of KVPControllerRepository and configure the file source for the data file.

Precondition: N/A

Post condition:
- An instance of KVPControllerRepository is returned.
- Attribute: *file* has the data file path received via parameter: *source*.

Description:

```
SET attribute: file to parameter: source
IF attribute: file does not exist
      CREATE data file at path attribute: file
```

4.24.2.2 retrieve ( )

Purpose: Extract and return all the Content objects from the key-value pair file.

Precondition: The data file specified at attribute: *file* exists and has values.

Post condition: The Content objects existing in the file are extracted and

returned.

Description: `Retrieve hash values starting from the root`

`Use ContentFactory to recursively reconstitute each object with the hashes retrieved to form album`

`RETURN album`

4.24.2.3 update (Content content)

Purpose: Find the specified Content object within the key-value pair file and update it with its new values.

Precondition:
- The data file specified at attribute: *file* exists and values.
- Parameter: *content* has a Content object.
- Parameter: *content* already exists in the key-value pair file.

Post condition: The specified Content has been updated in the key-value pair with the new values.

Description: `Find parameter: content in attribute: file`

`Update parameter: content with its new values in attribute: file`

4.24.2.4 insert (Content content, Album parent)

Purpose: Persist the specified Content object in parameter: content to the key-value pair file at attribute: *file* and associate it with the specified Album object in parameter: *parent*.

Precondition:
- The data file specified at attribute: *file* exists.

- Parameter: *content* has a Content object.
- Parameter: *parent* already exists in the key-value pair file.

Post condition:

- The specified Content object in parameter: *content* has been persisted to the key-value pair file at attribute: *file*.
- The association for the newly persisted Content object and parameter: *parent* has been persisted to file.

Description: `ADD parameter: content into parameter: parent's contents collection in attribute: file`

4.24.2.5 delete (Content content, Album parent)

Purpose: Deletes the specified Content object in parameter: *content* that is in the Album parameter: *parent* from the key-value pair file at attribute: *file*.

Precondition:

- The data file specified at attribute: *file* exists.
- Parameter: *content* has a Content object.
- Parameter: *content* exists in the key-value pair file.
- Parameter: *parent* already exists in the key-value pair file.

Post condition: The specified Content object in parameter: *content* is not in parameter: *parent* in the key-value pair file at attribute: *file*.

Description: `DELETE parameter: content from parameter: parent's contents collection in the key-value pair file at attribute: file`

4.25 KVPEmailPreferenceRepository

The KVPEmailPreferenceRepository class implements the EmailPreferenceRepository interface to serve in the persistent layer of the application to deal with operations relating retrieving and persisting EmailPreference object values to and from a key-value pair file.

4.25.1 Attributes of KVPEmailPreferenceRepository class

4.25.1.1 file

Purpose: This attribute holds the path to the key-value pair file.

Data type: String

Visibility: Private

4.25.2 Methods available in KVPEmailPreferenceRepository class

4.25.2.1 new (String source)

Purpose: Creates a new instance of KVPEmailPreferenceRepository and configure the file source for the data file.

Precondition: N/A

Post condition:

- An instance of KVPEmailPreferenceRepository is returned.
- Attribute: *file* has the data file path received via parameter: *source*.

Description:

```
SET attribute: file to parameter: source
IF attribute: file does not exist
      CREATE data file at path attribute: file
```

4.25.2.2 load ( )

Purpose: Extract and return a Hash consisting of email preference

values corresponding to the keys: server, port, username, password, authentication, domain and sender_email from the key-value pair file.

Precondition: The data file specified at attribute: *file* exists and has values.

Post condition: The values existing in the file are extracted and returned.

Description:

```
Retrieve values corresponding to: server, port,
username, password, authentication_type, domain
and sender_email keys from the file at
attribute: file

COMPOSE a properties Hash with the values
retrieved

RETURN properties
```

4.25.2.3 save (Hash preference)

Purpose: Persist the email preference values from parameter: *preference* to the key-value pair file.

Precondition:

- The data file specified at attribute: *file* exists.
- Parameter: *preference* has a Hash of values corresponding to the keys: server, port, username, password, authentication_type, domain, and sender_email.

Post condition: The values specified are persisted to the key-value pair file at parameter: *file*.

Description:

```
UPDATE the values of the key-value pair file at
attribute: file with the values from parameter:
preference
```

4.26 KVPSlideshowPreferenceRepository

The KVPSlideshowPreferenceRepository class implements the SlideshowPreferenceRepository interface to serve in the persistent layer of the application to deal with operations relating retrieving and persisting SlideshowPreference object values to and from a key-value pair file.

4.26.1 Attributes of KVPSlideshowPreferenceRepository class

4.26.1.1 file

Purpose: This attribute holds the path to the key-value pair file.

Data type: String

Visibility: Private

4.26.2 Methods available in KVPSlideshowPreferenceRepository class

4.26.2.1 new (String source)

Purpose: Creates a new instance of KVPSlideshowPreferenceRepository and configure the file source for the data file.

Precondition: N/A

Post condition:

- An instance of KVPSlideshowPreferenceRepository is returned.
- Attribute: *file* has the data file path received via parameter: *source*.

Description:

```
SET attribute: file to parameter: source
IF attribute: file does not exist
      CREATE data file at path attribute: file
```

4.26.2.2 load ( )

Purpose: Extract and return a Hash consisting of slideshow preference values corresponding to the keys: transition_duration and loop_count from the key-value pair file.

Precondition: The data file specified at attribute: *file* exists and has values.

Post condition: The values existing in the file are extracted and returned.

Description:

```
RETRIEVE values corresponding to:
transition_duration and loop_count keys from the
file at attribute: file

COMPOSE a properties Hash with the values
retrieved

RETURN properties
```

4.26.2.3 save (Hash preference)

Purpose: Persist the slideshow preference values from parameter: *preference* to the key-value pair file.

Precondition:

- The data file specified at attribute: *file* exists.
- Parameter: *preference* has a Hash of values corresponding to the keys: transition_duration and loop_count.

Post condition: The values specified are persisted to the key-value pair file at parameter: *file*.

Description:

```
UPDATE the values of the key-value pair file at
attribute: file with the values from parameter:
preference
```

5 Data Persistence and File Structure

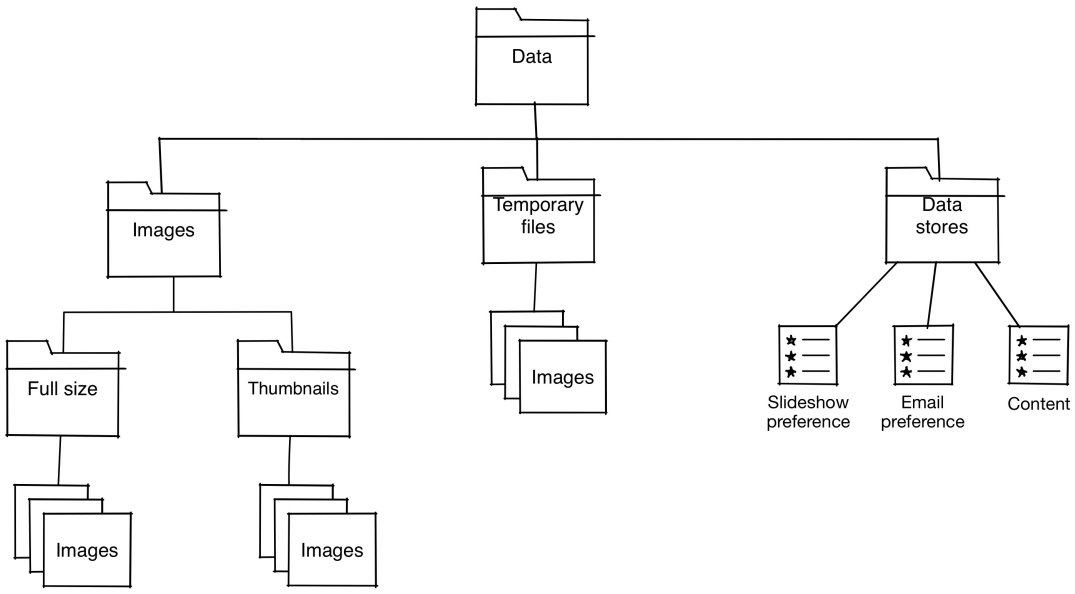
This section describes data persistence in detail, specifying how data is managed in relation to persistence in section 5.2 Persistence data management and a description of the data that is persisted and its format in section 5.2.2 Data dictionary.

This section also covers how the data and consumable files are organized on disk within the file structure in section 5.1 File structure.

Special consideration for persisting sensitive data is outlined in section 5.3 Special considerations.

5.1 File structure

To organize the data stored on disk, the software will use the following file structure:



The Data directory consists of three sub directories:

- Images
- Temporary files
- Data stores

The **Images** directory consists of two sub directories:

- Full size
- Thumbnails

The **Full size** directory serves as the repository for all images that has a Photo object within the software. Each image that is stored within this directory is named with the timestamp of the date/time it was added to the software.

The **Thumbnails** directory serves as the repository for the thumbnail images generated from its corresponding image in the **Images** directory.

The **Temporary files** directory serves as the temporary holding area for images that the software is carrying out image manipulation activities on. An image is moved from this directory to the Images directory when it is bind with a Photo object.

The **Data stores** directory serves as the repository for the key-value pair data store files that the software will retrieve data from and persist data to. See section 5.2 Persistence data management for more details.

5.2 Persistence data management

5.2.1 Overview

PixBox will depend on three key-value pair data stores for data persistence. Each data store will be responsible for storing a specific object type.

- Content key-value pair data store – responsible for storing data that represents Content objects.
- Slideshow preference key-value pair data store – responsible for storing values from the slideshowPreference object.
- Email preference key-value pair data store – responsible for storing values from the EmailPreference object.

Each key-value pair data store is file based and is located in the Data stores directory on disk.

PixBox's architectural design includes a persistence layer that manages the communication between the domain level objects and the data source. The persistence layer is designed using the repository design pattern to manage the

separation of the persistence layer from the layers that mange the domain objects, see 3.1 Architectural Design for more details.

Separate objects from the persistence layer manage each data store:

- The KVPContentRepository object manages the Content key-value pair data store.
- The KVPEmailPreferenceRepository object manages the Email preference key-value pair data store.
- The KVPSlideshowPreferenceRepository object manages the Slideshow preference key-value pair data store.

5.2.2 Data dictionary

5.2.2.1 Content key-value pair data store

The Content key-value pair data store supports the persistence of two types of Content objects, namely: Album and Photo.

Album and Photo are two different specializations of the Content class that consists of different types of values with only timestamp in common.

The key-value pair data store is flexible enough to allow for the persistence of the Album and Photo objects despite their differences.

The timestamp should be used as the key for each object that will be persisted to the key-value pair data store.

Each entry of an Album object should have the following properties in the key-value pair data store:

Property	Description
Type	The Class the object belongs to. In the case of the Album object, this property will have the value of 'Album'.
Name	Corresponds to the name attribute of the Album object.
Collection	A collection of timestamps that corresponds to the key-value Photo or Album objects that are a part of the Album object in question.

Example entry	1416166567: {
	"type" => "Album",
	"name" => "A day at the beach",
	"collection" => [1416166843,
	1416166852, 1416166866]
	}

Each entry of a Photo object should have the following properties in the key-value pair data store:

Property	Description
Type	The Class the object belongs to. In the case of the Photo object, this property will have the value of 'Photo'.
Path	Corresponds to the path attribute of the Photo object.
Extension	Corresponds to the extension attribute of the Photo object.
Keywords	Corresponds to the keywords attribute of the Photo object.
Thumbnail path	Corresponds to the thumb_path attribute of the Photo

	object.

Example entry	1416166852 : {
	“type” => “Photo”,
	“path” => “/path/to/file/photo.jpg”,
	“extension” => “jpg”,
	“Keywords” => [“sample”,
	“sunshine”],
	“thumb_path” => “/path/to/t/thumb.jpg”
	}

5.2.2.2 Email preference key-value pair data store

Since there is only one Email preference, there should be a key for each email preference value from the EmailPreference object.

The key-value pair data store for the email preference should have the following keys:

Keys	Description
Server	Corresponds to the server attribute of the EmailPreference object.
Port	Corresponds to the port attribute of the EmailPreference object.
Username	Corresponds to the username attribute of the EmailPreference object.
Password	Corresponds to the password attribute of the EmailPreference object.
Authentication_type	Corresponds to the authentication_type attribute of the EmailPreference object.

Domain	Corresponds to the domain attribute of the EmailPreference object.
Sender_email	Corresponds to the sender_email attribute of the EmailPreference object.
Example entry	server : mail.gmail.com port : 465 username : someemail@gmail.com password : 232jnjdsmds* authentication_type : STARTTLS domain : NULL sender_email : someemail@gmail.com

5.2.2.3 Slideshow preference key-value pair data store

Since there is only one Slideshow preference, there should be a key for each email preference value from the SlideshowPreference object.

The key-value pair data store for the slideshow preference should have the following keys:

Keys	**Description**
Transition duration	Corresponds to the transition_duration attribute of the SlidePreference object.
Loop count	Corresponds to the loop_count attribute of the SlideshowPreference object.
Example entry	transition_duration : 3 loop_count : 1

5.3 Special considerations

Password encryption:

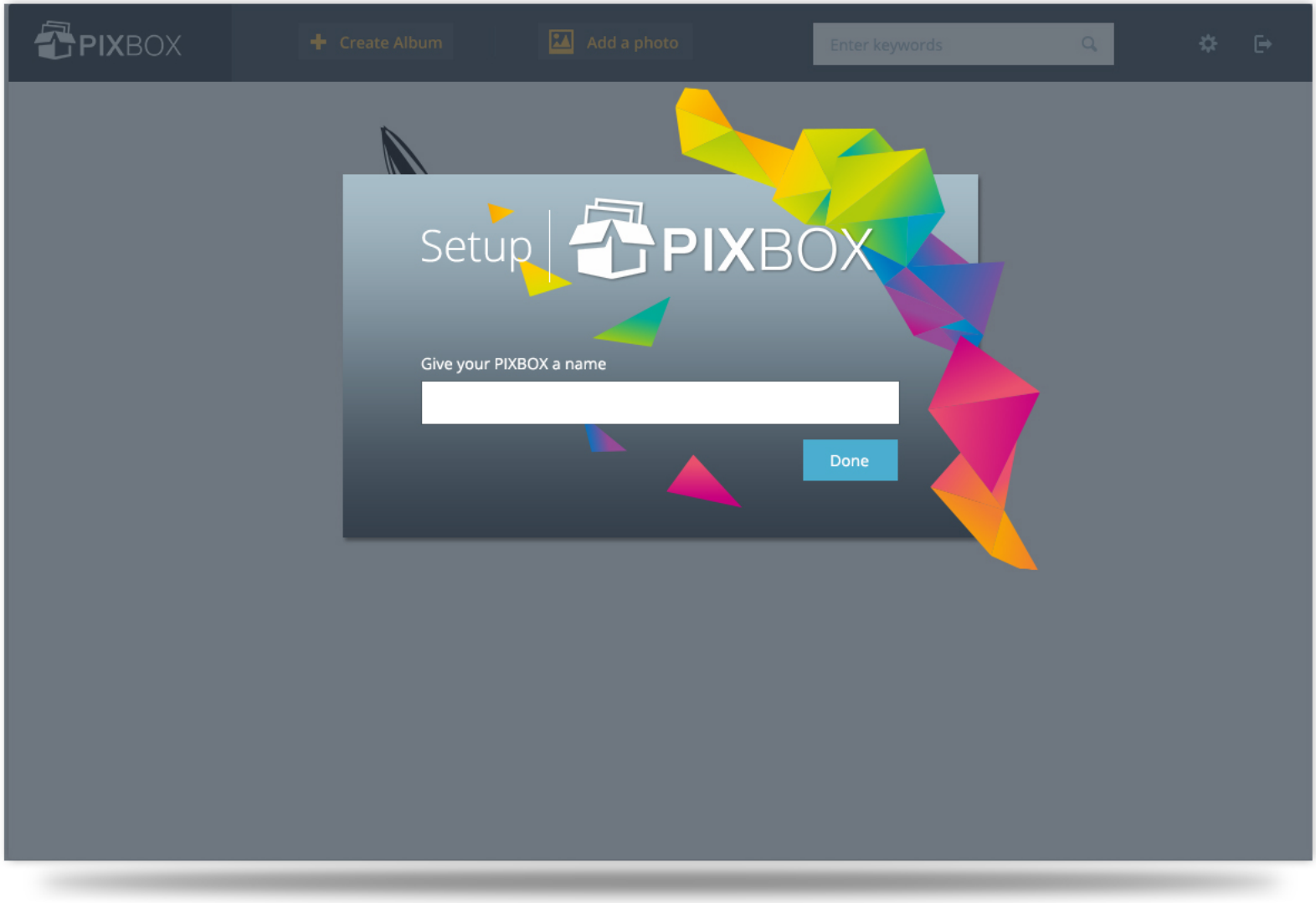
The value of the password attribute of the EmailPreference class needs to be encrypted before it is persisted to the data store, and decrypted when retrieved.

Password encryption is necessary to protect the password from prying eyes.

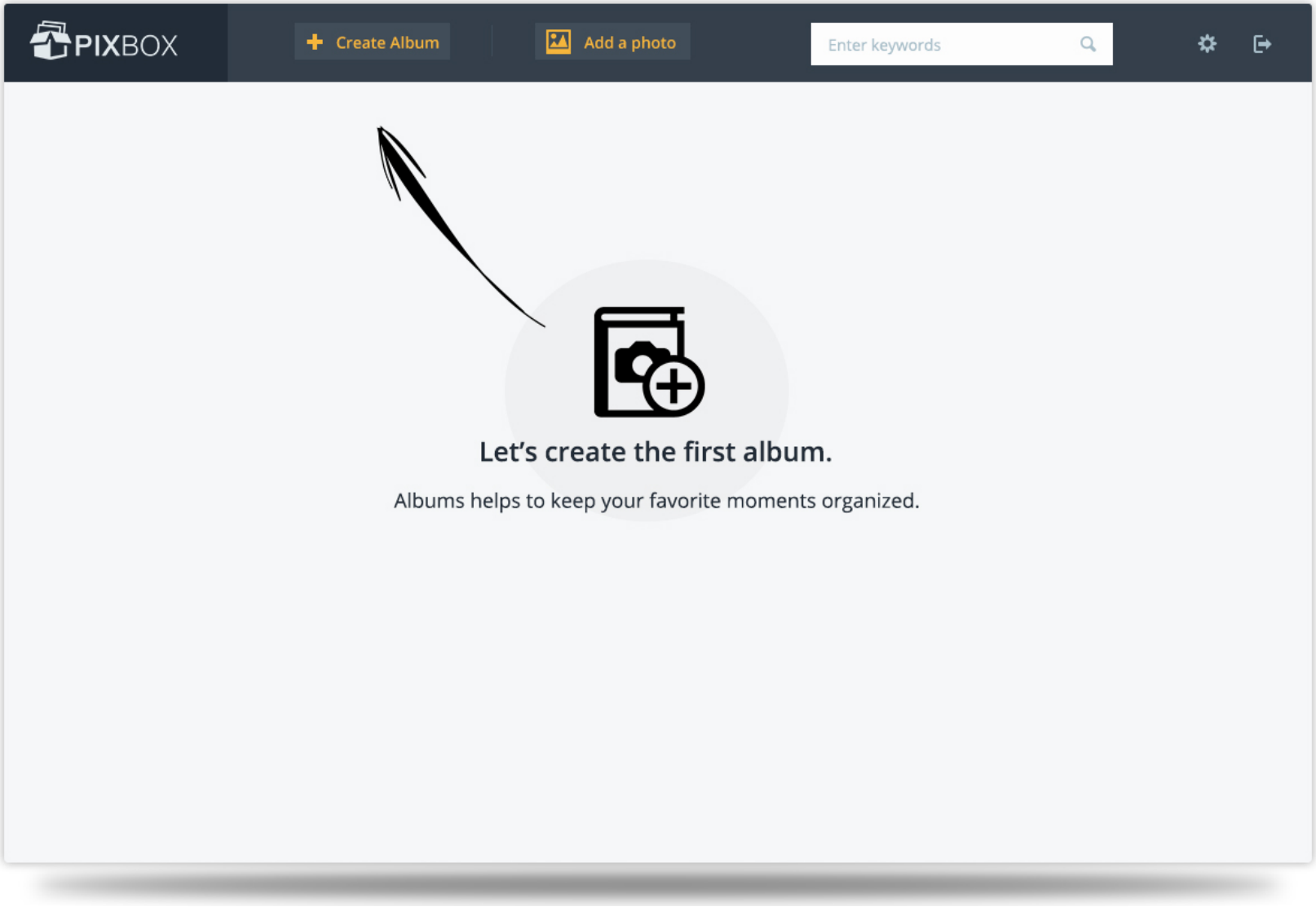
6 Interface Design

This section covers the details related to the structure of the graphical user interface (GUI), including preliminary mockups of PixBox.

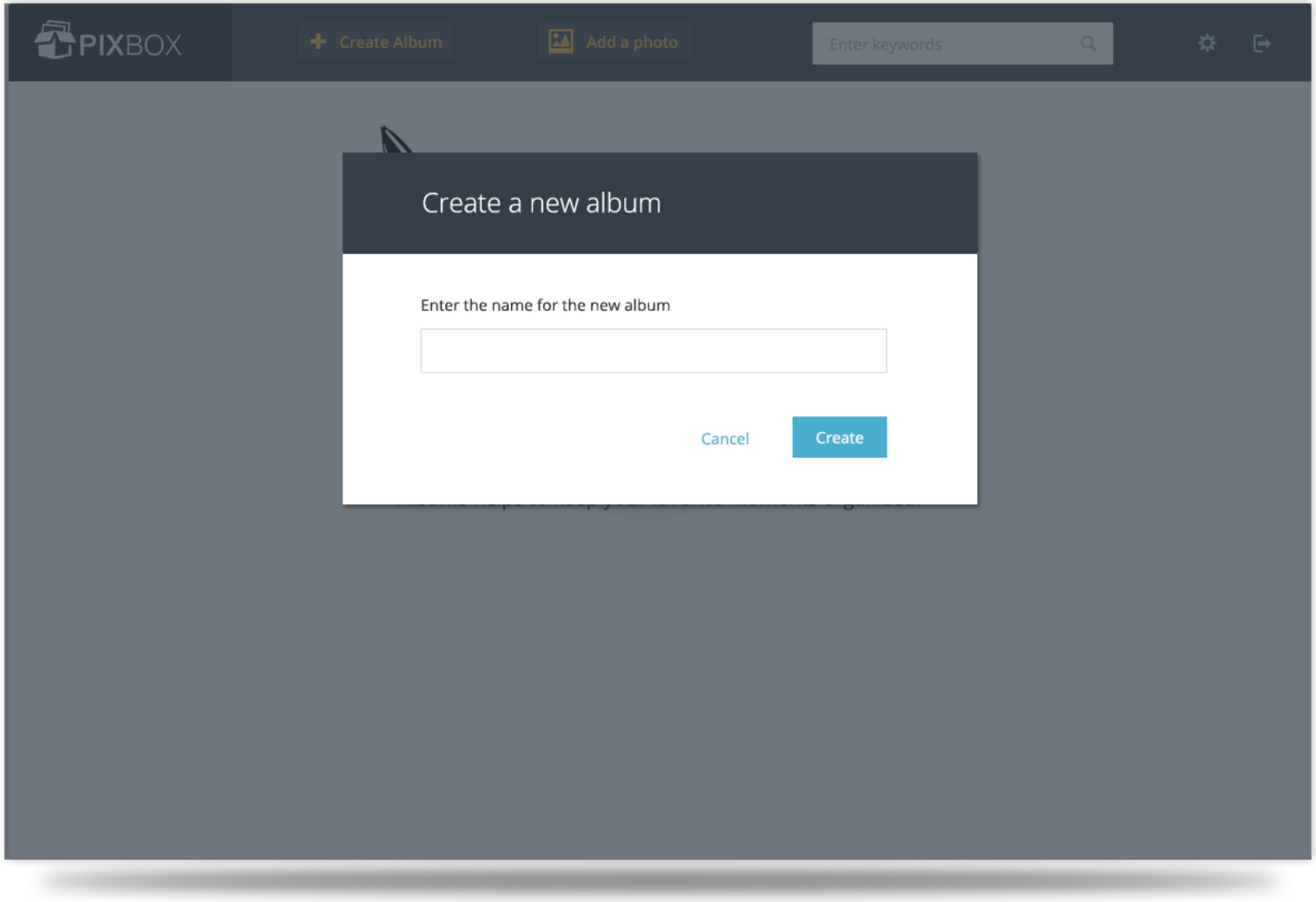
6.1 User interface 1.1: Create root photo album



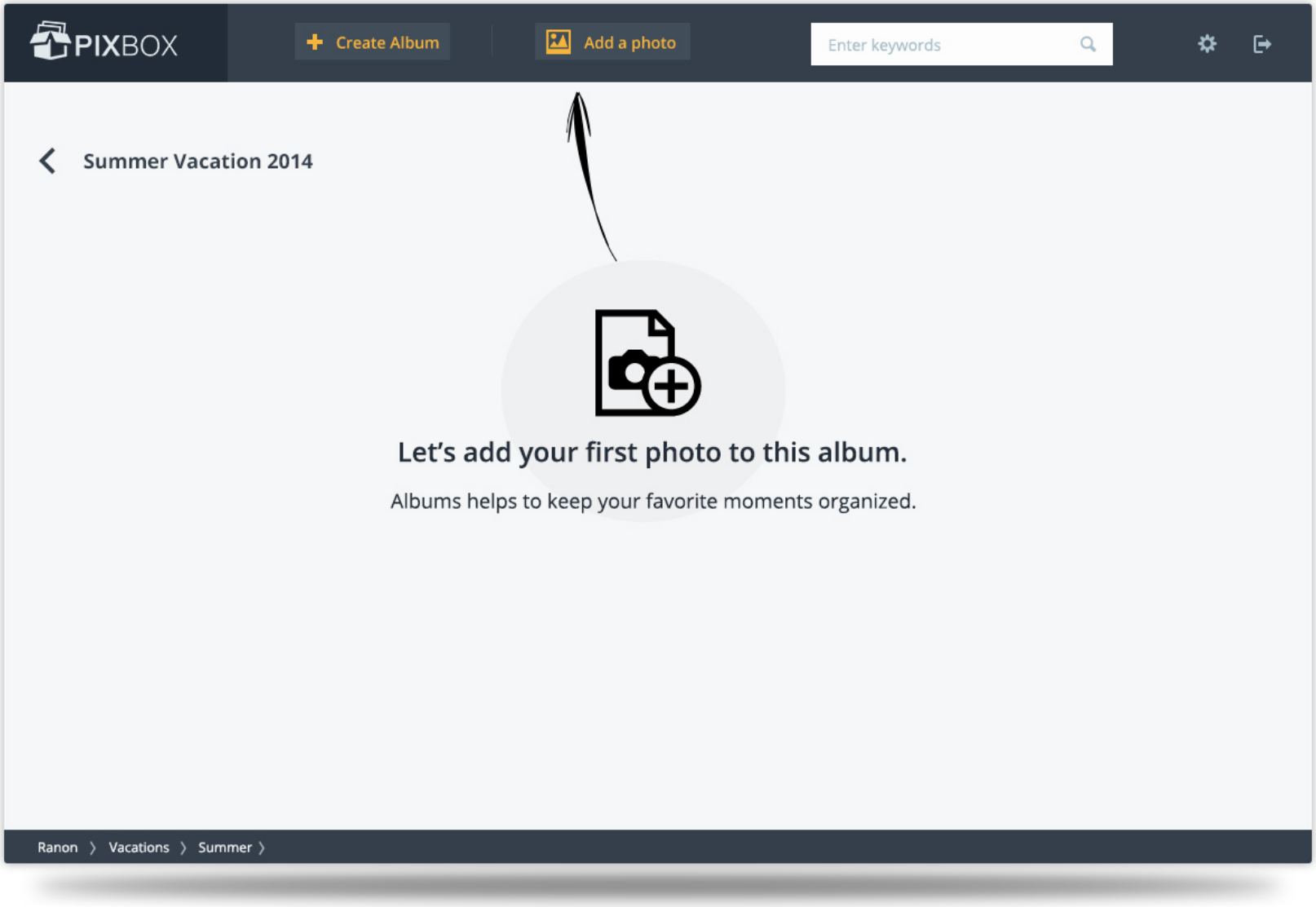
6.2 User interface 1.2: Empty slate – no photo album exists



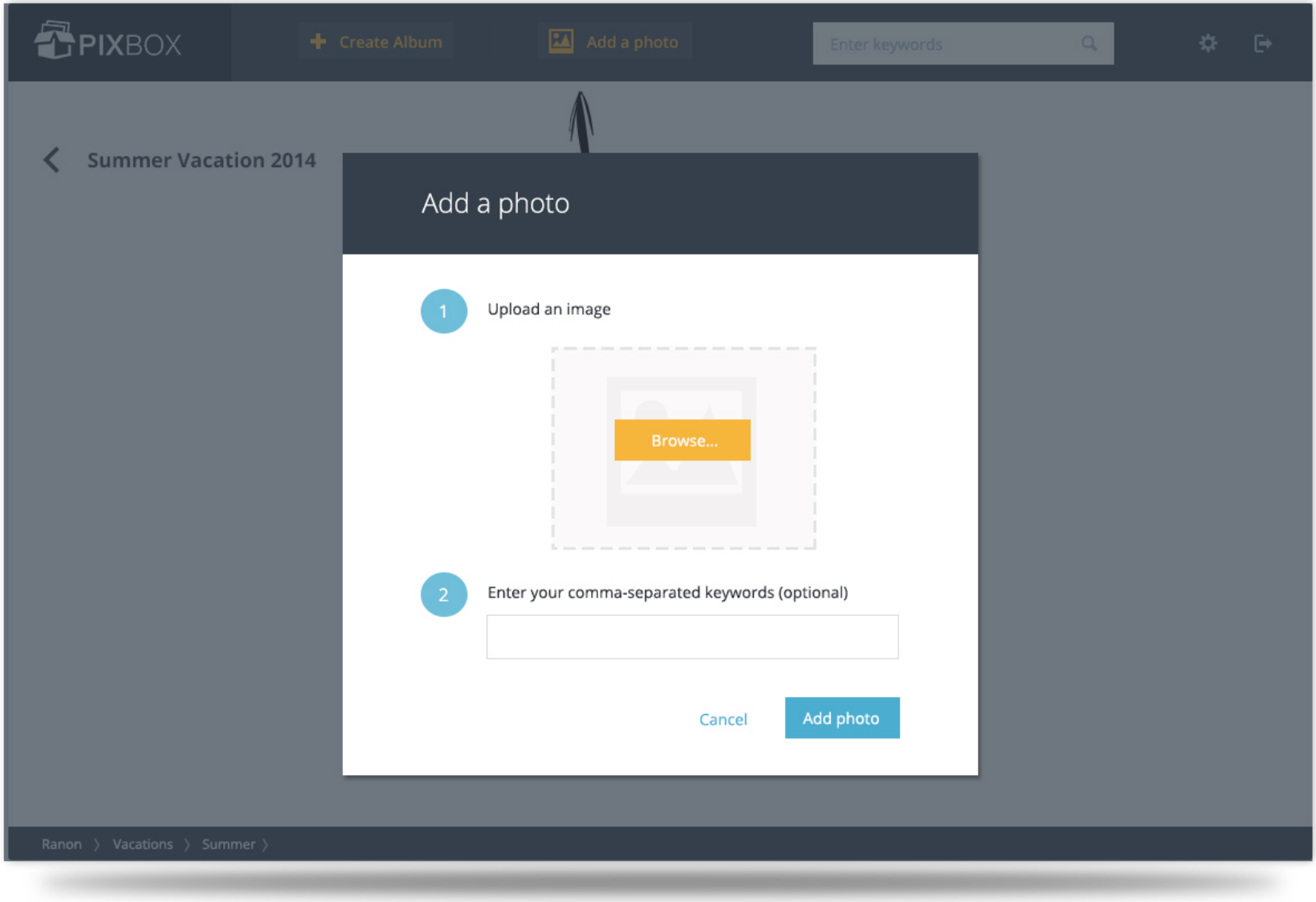
6.3 User interface 1.3: Create new photo album



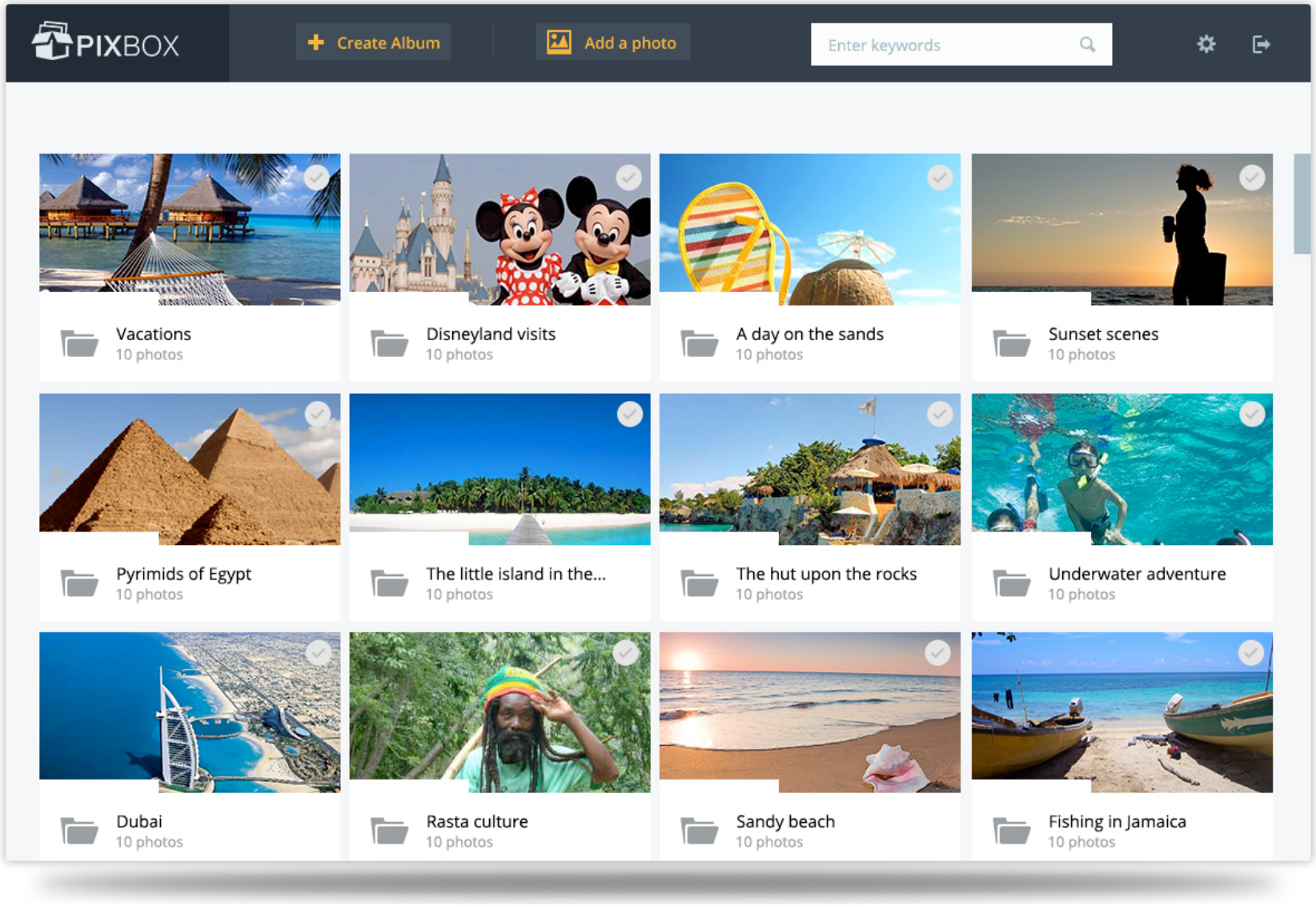
6.4 User interface 1.4: Empty slate – no photo exists



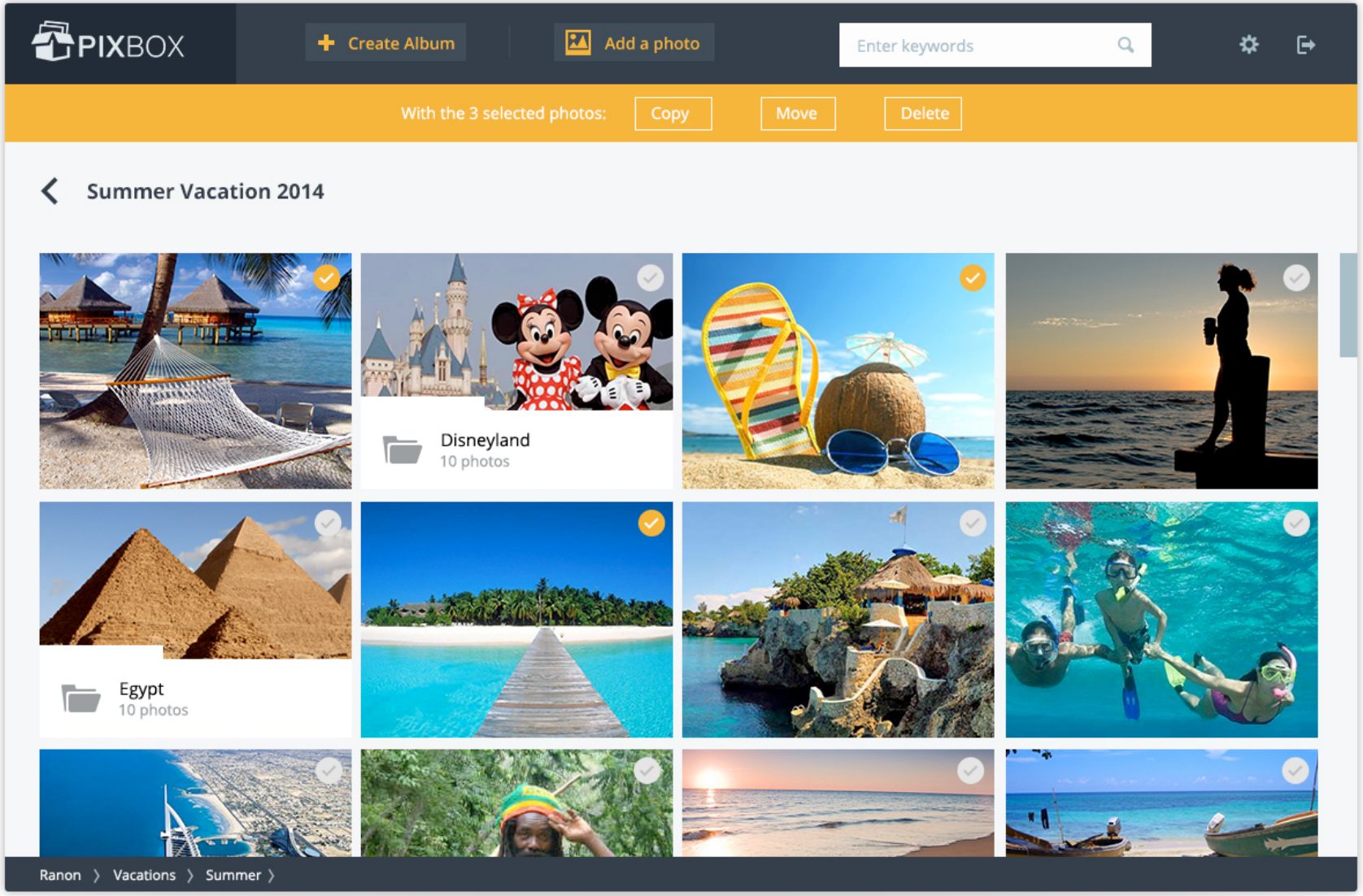
6.5 User interface 1.5: Add a photo to an existing photo album



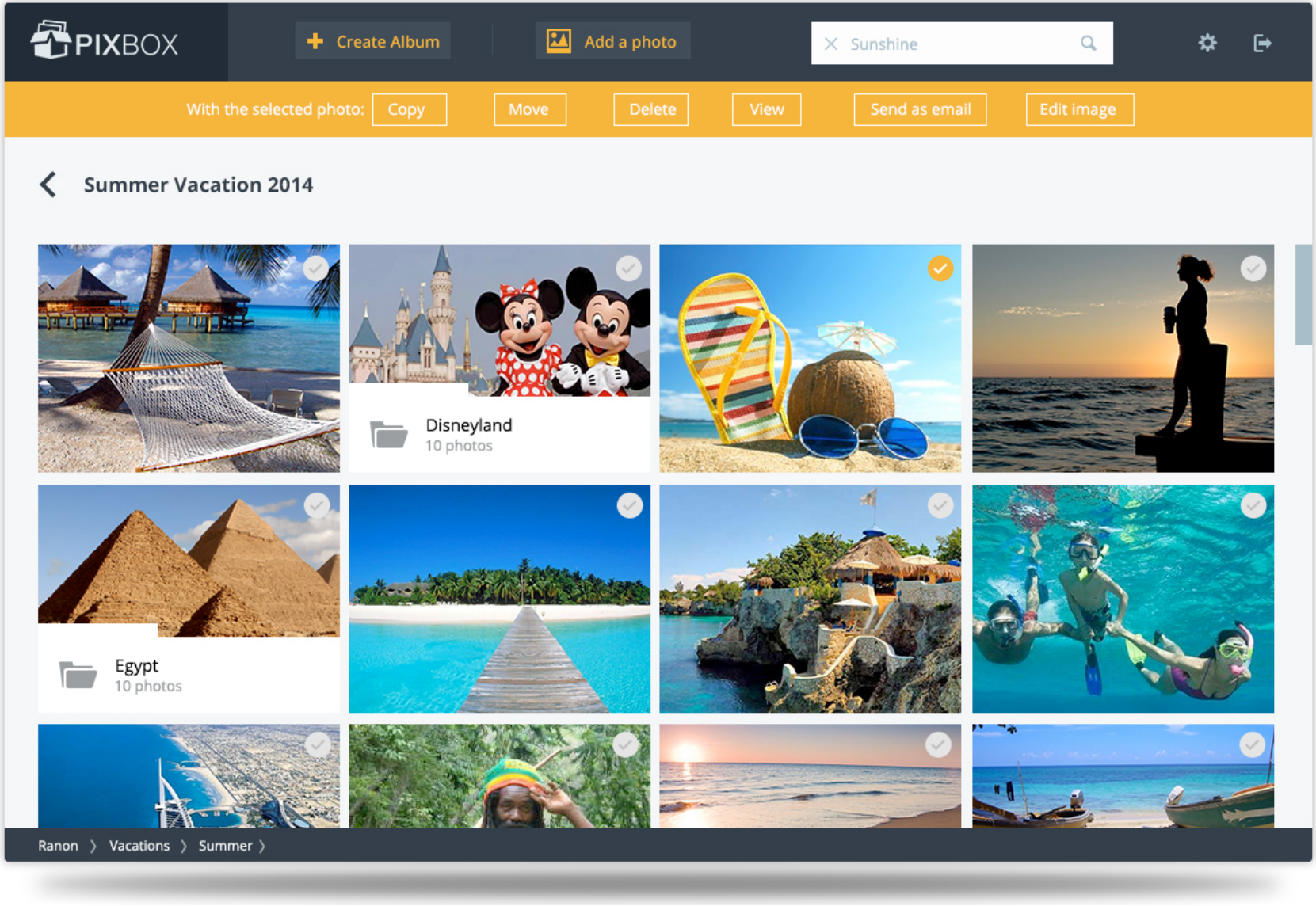
6.6 User interface 1.6: Root photo album with content



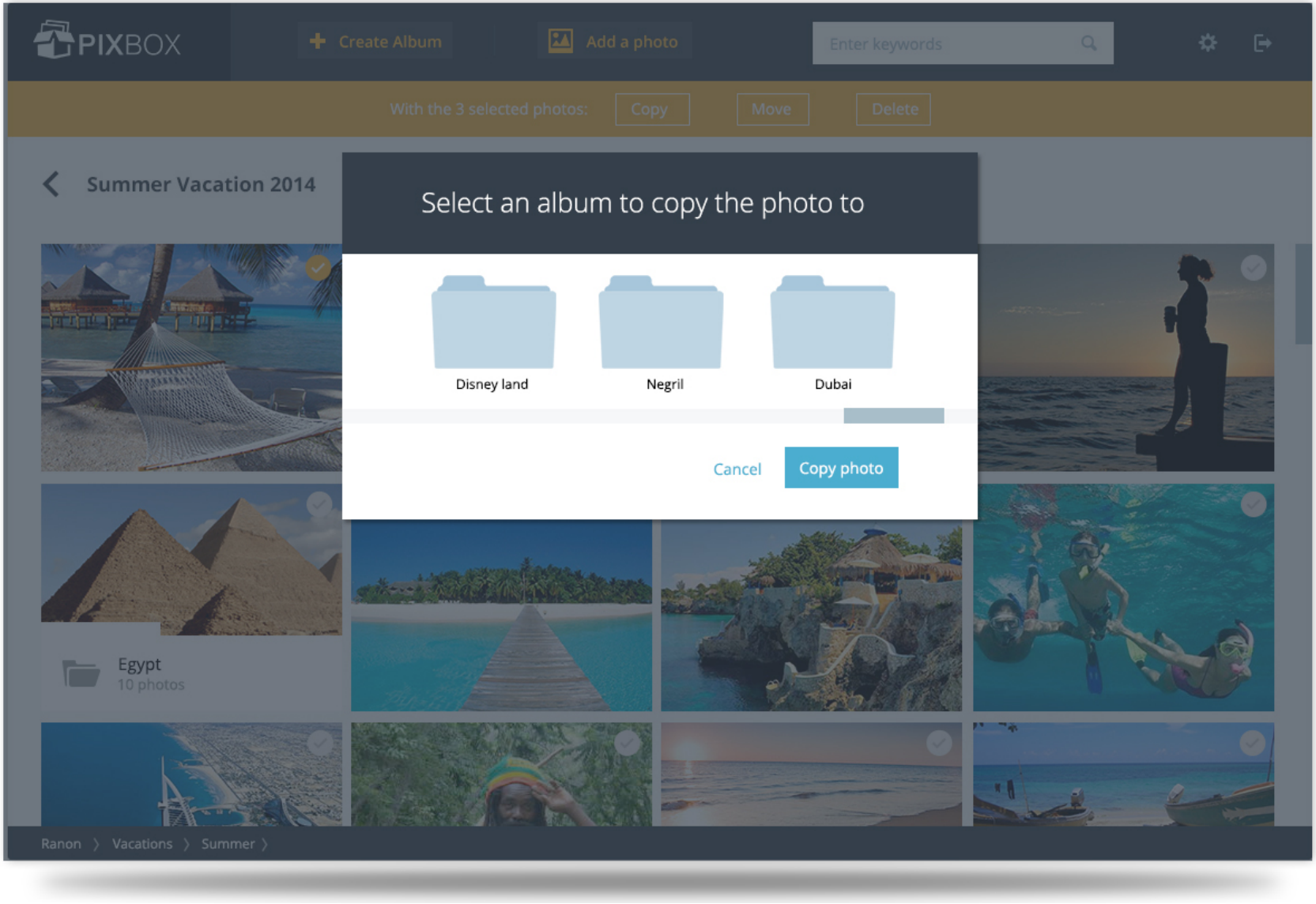
6.7 User interface 1.7: Photo album with content and multiple photo selection



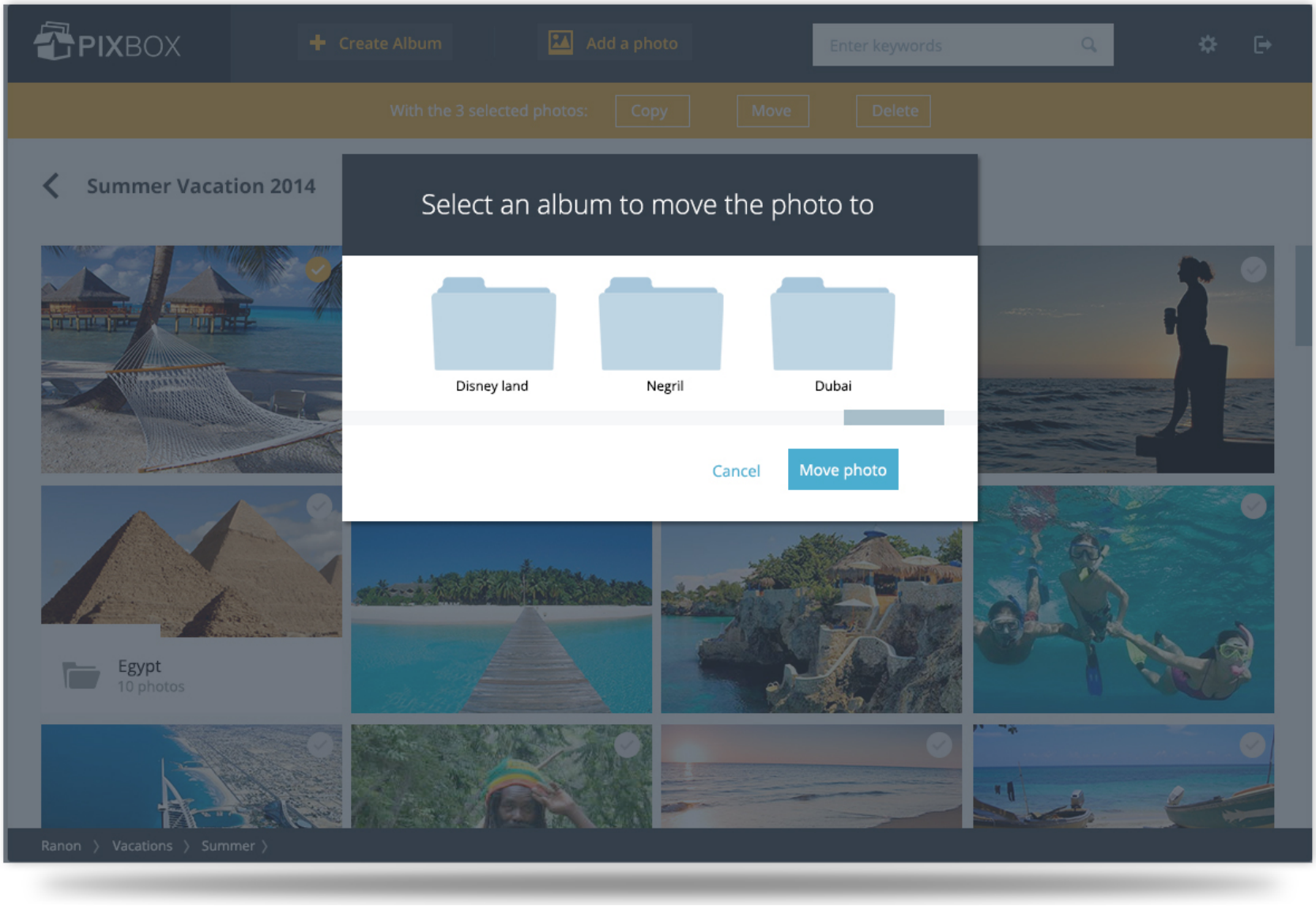
6.8 User interface 1.8: Photo album with content and single photo selection



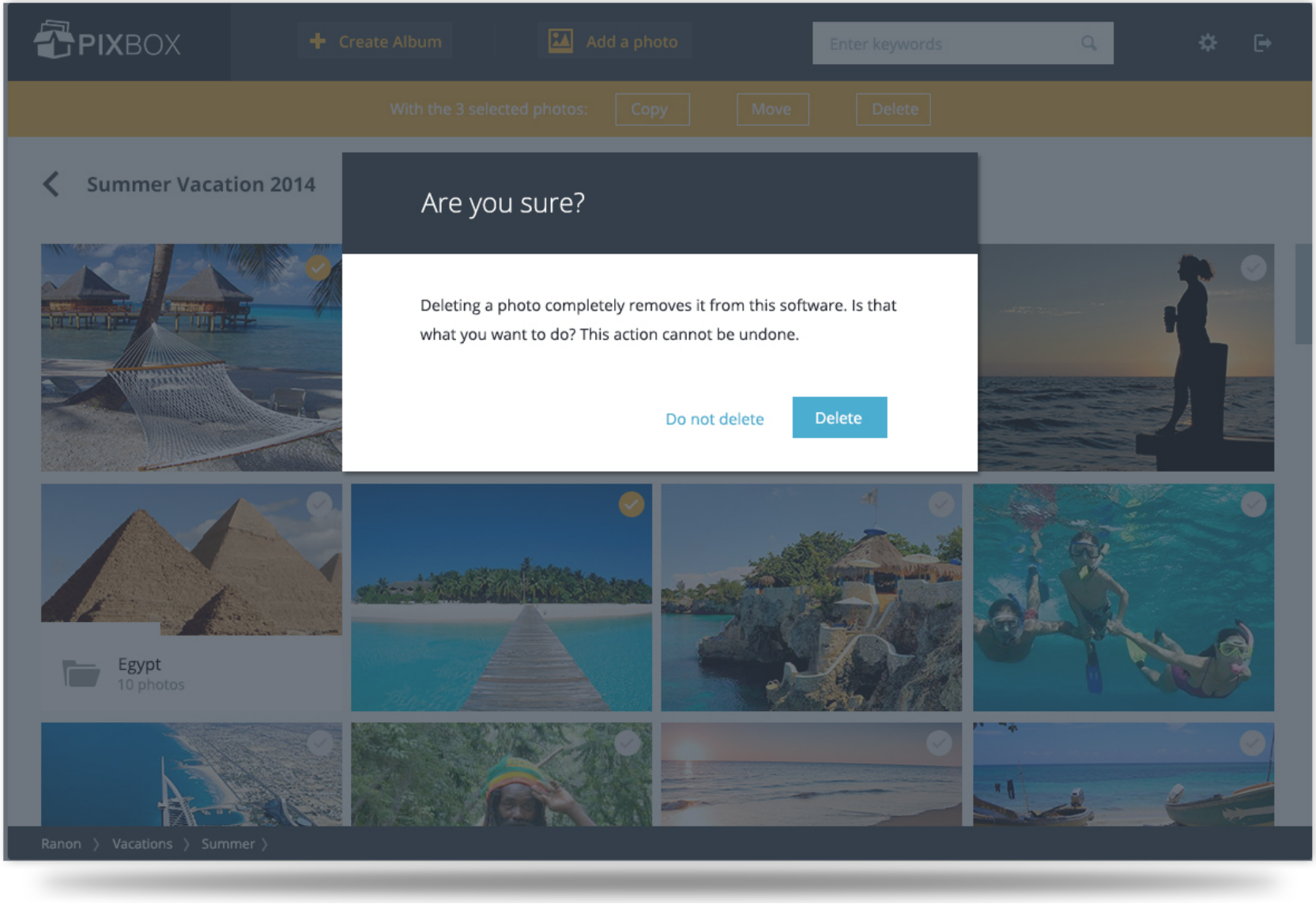
6.9 User interface 1.9: Copying a photo to an album



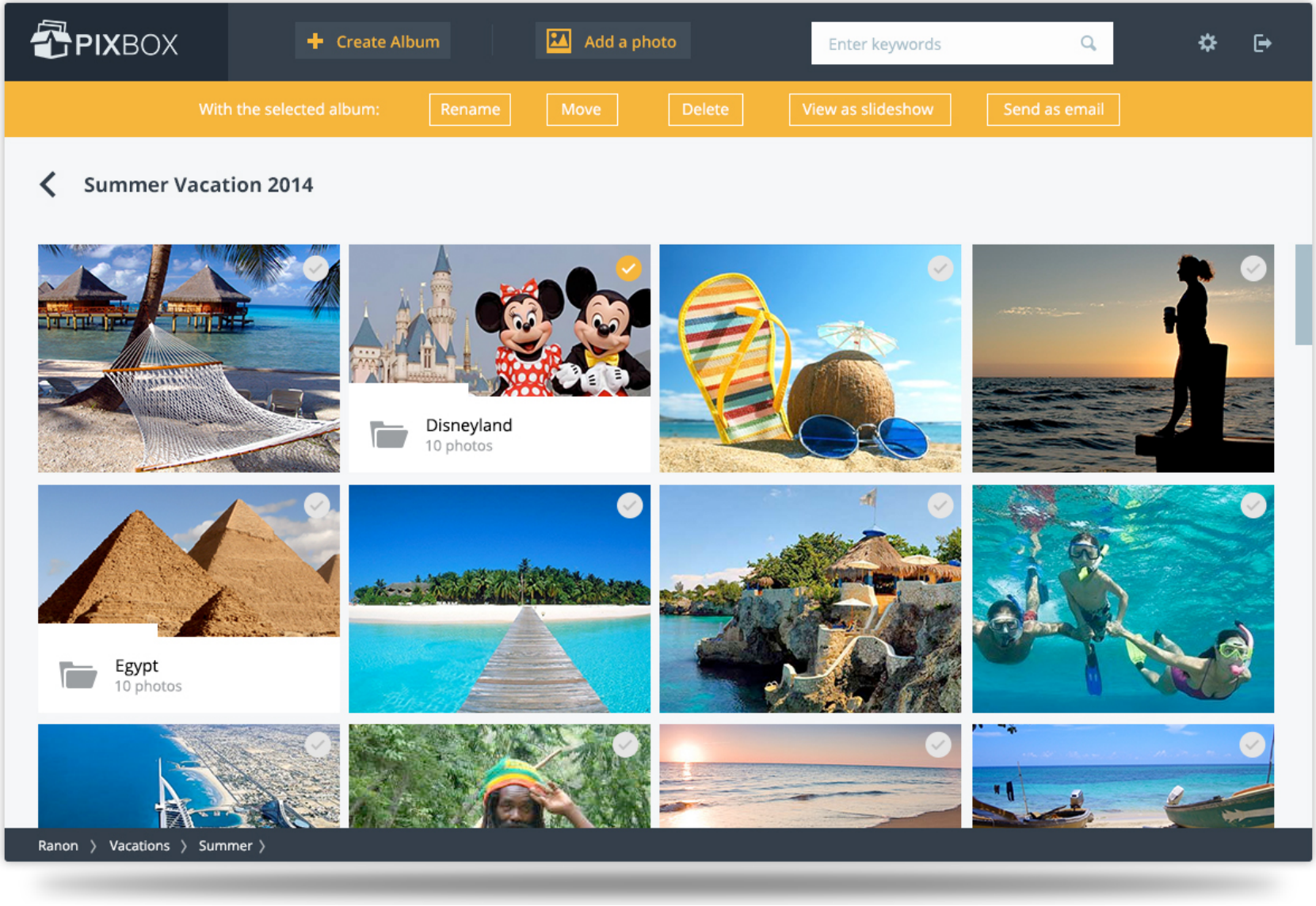
6.10 User interface 1.10: Moving a photo from one album to another



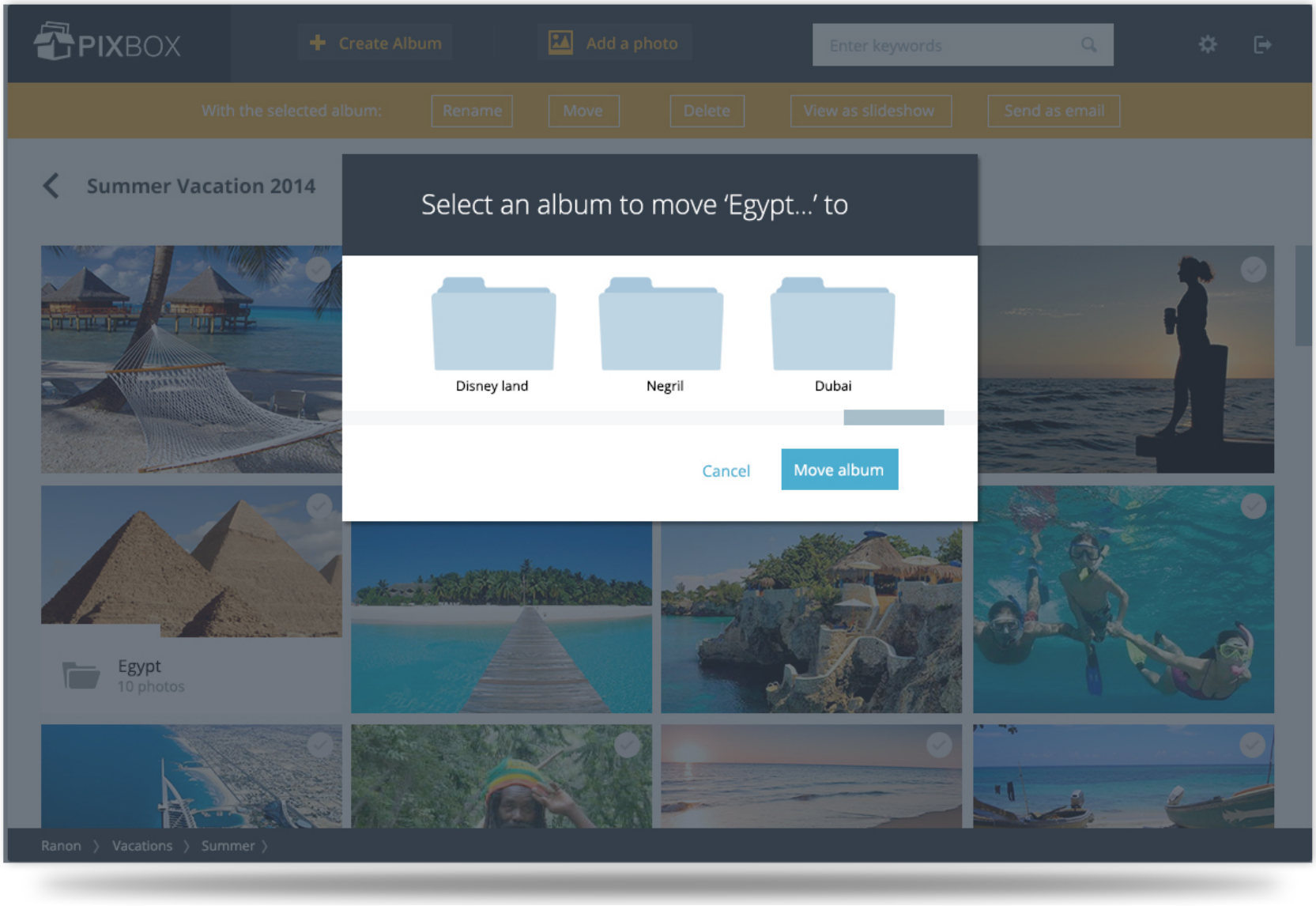
6.11 User interface 1.11: Deleting a photo



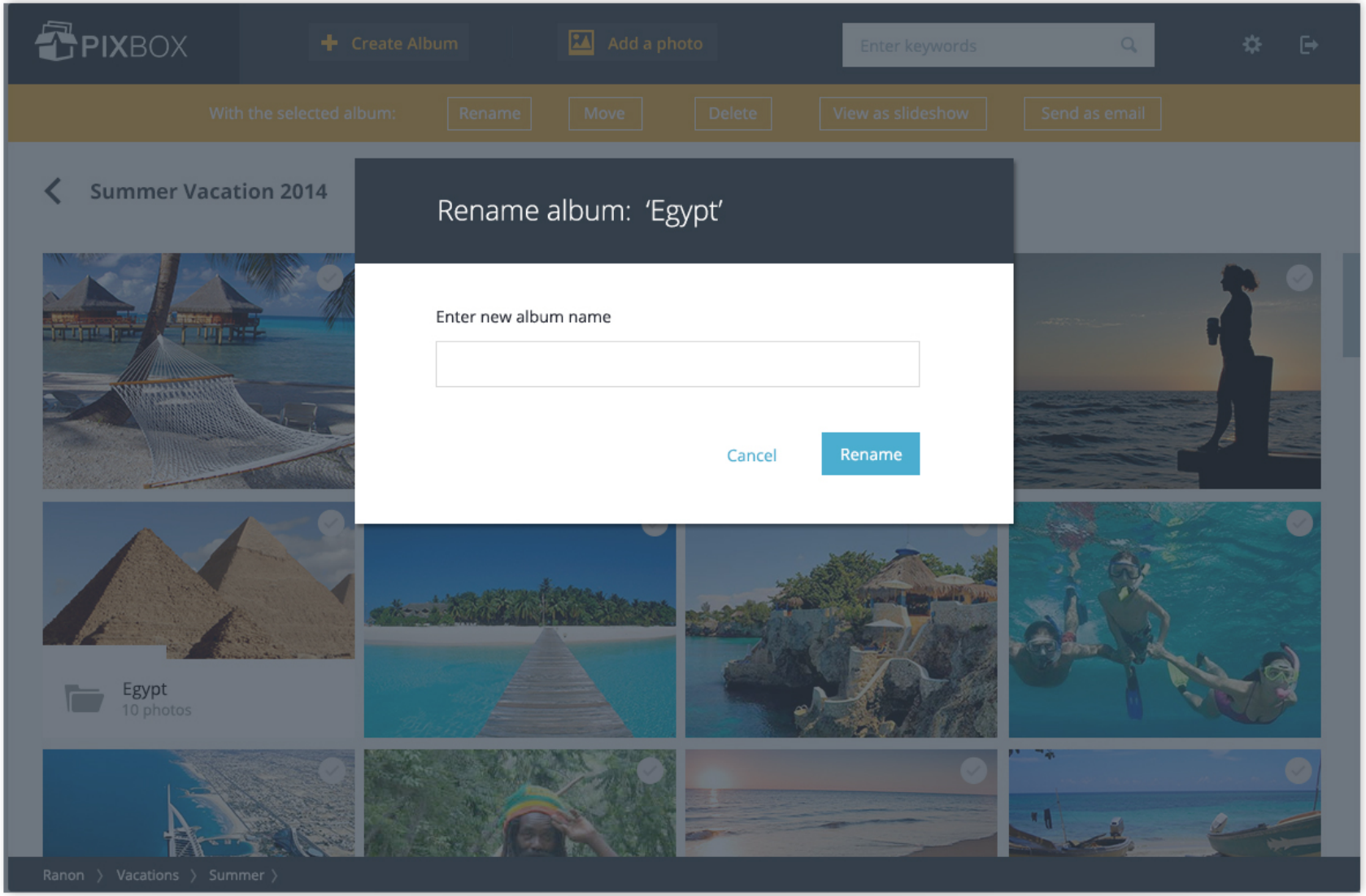
6.12 User interface 1.12: Photo album with content and single photo album selection



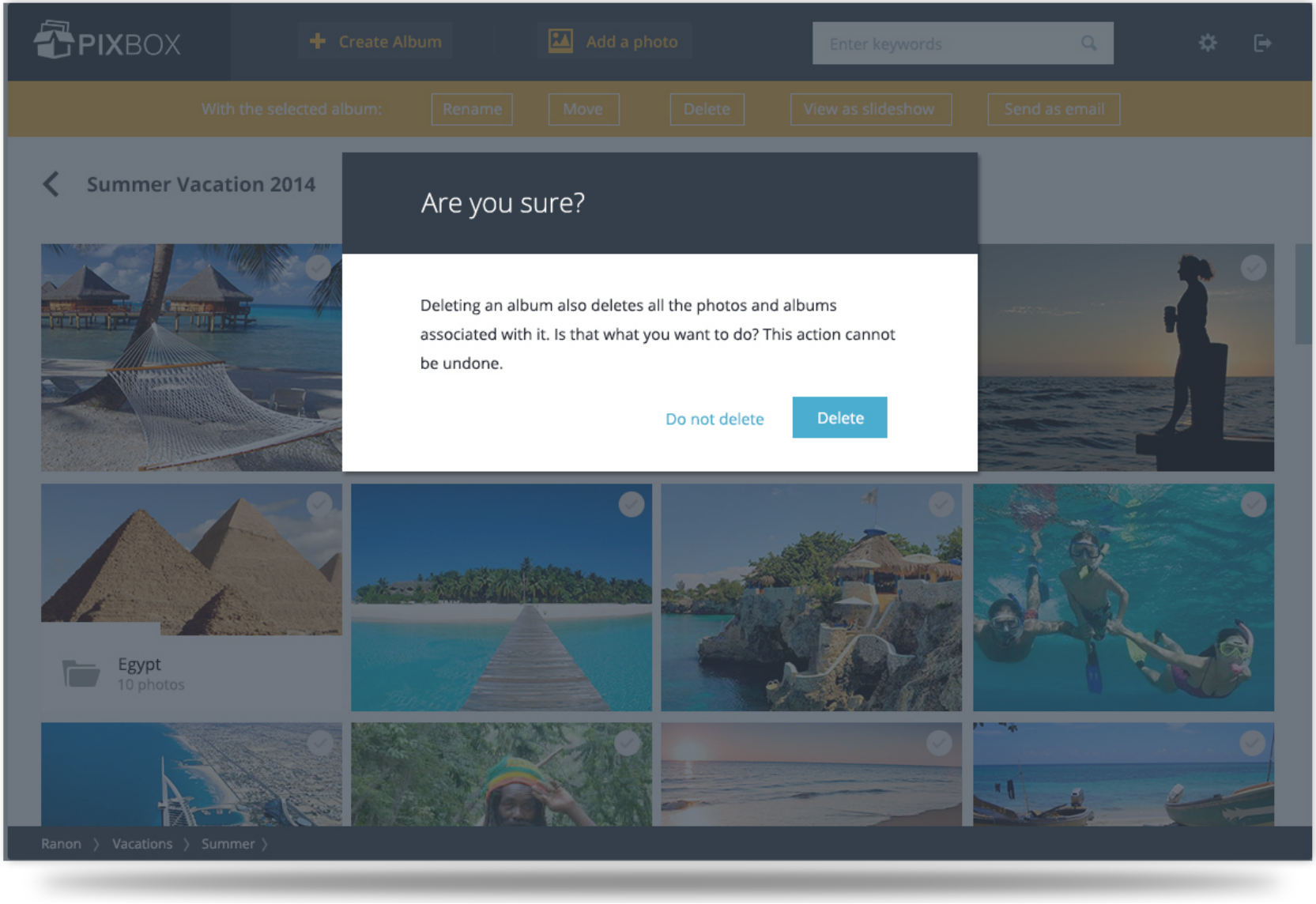
6.13 User interface 1.13: Moving a photo album from one photo album to another



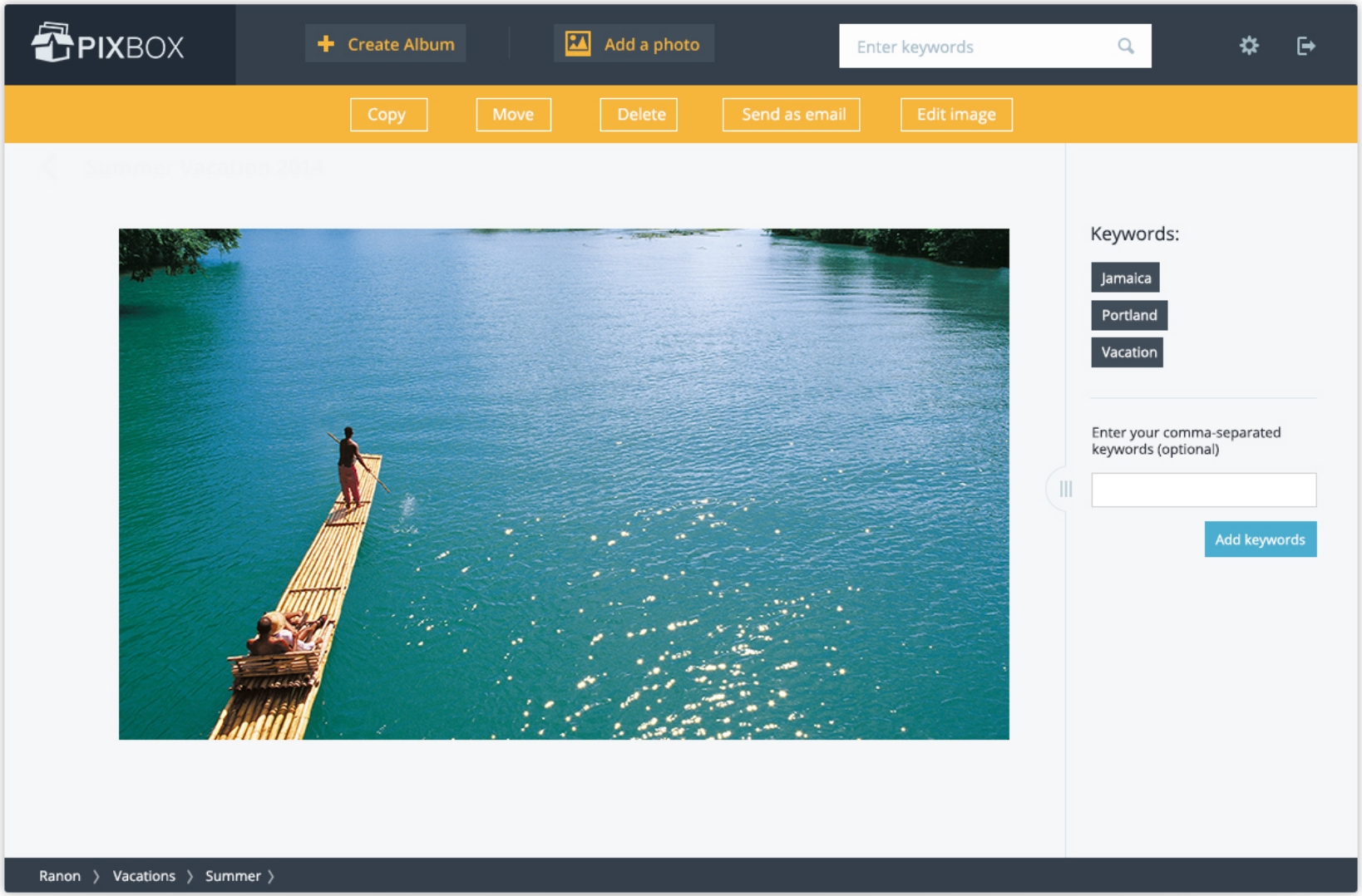
6.14 User interface 1.14: Rename a photo album



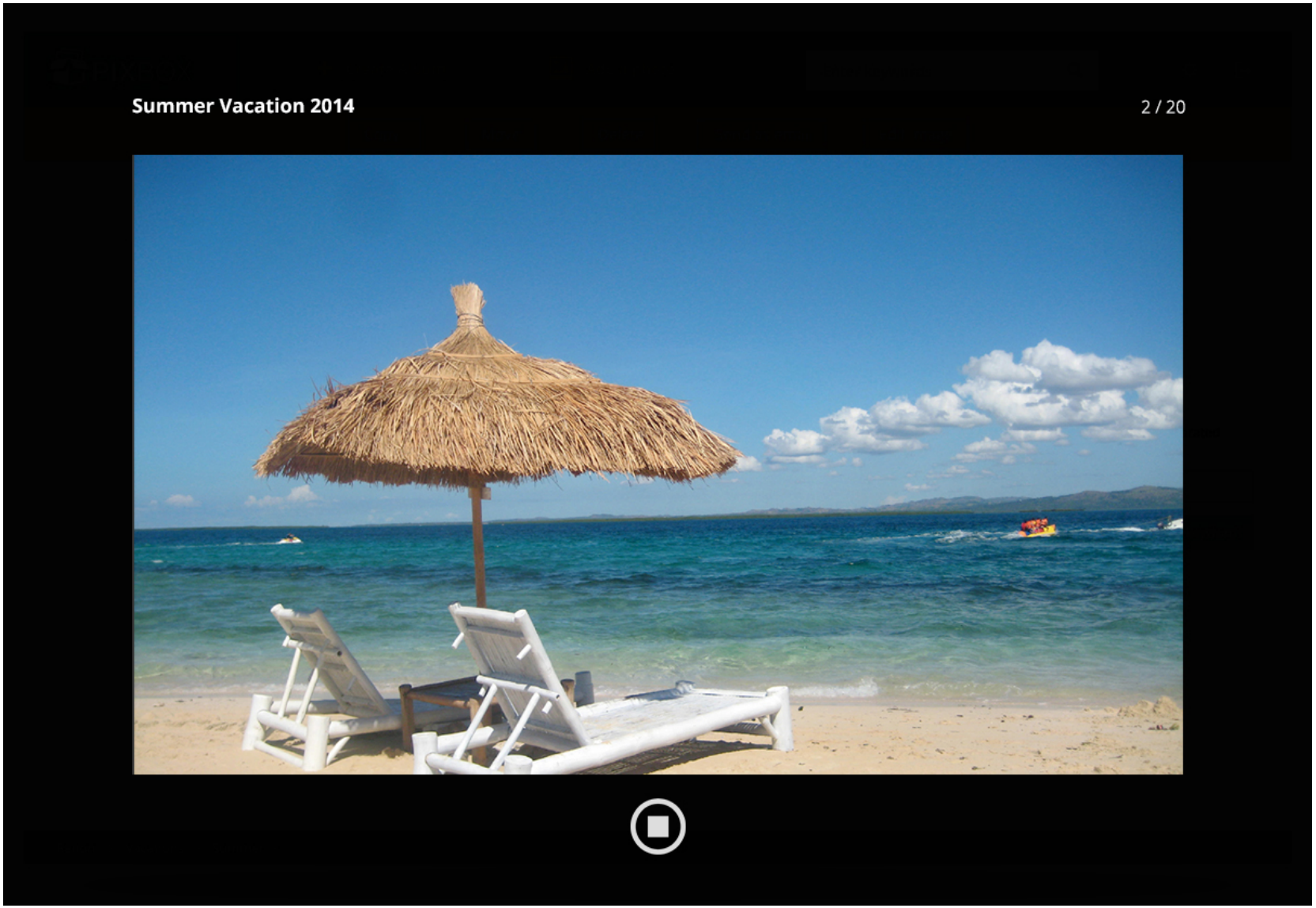
6.15 User interface 1.15: Deleting a photo album



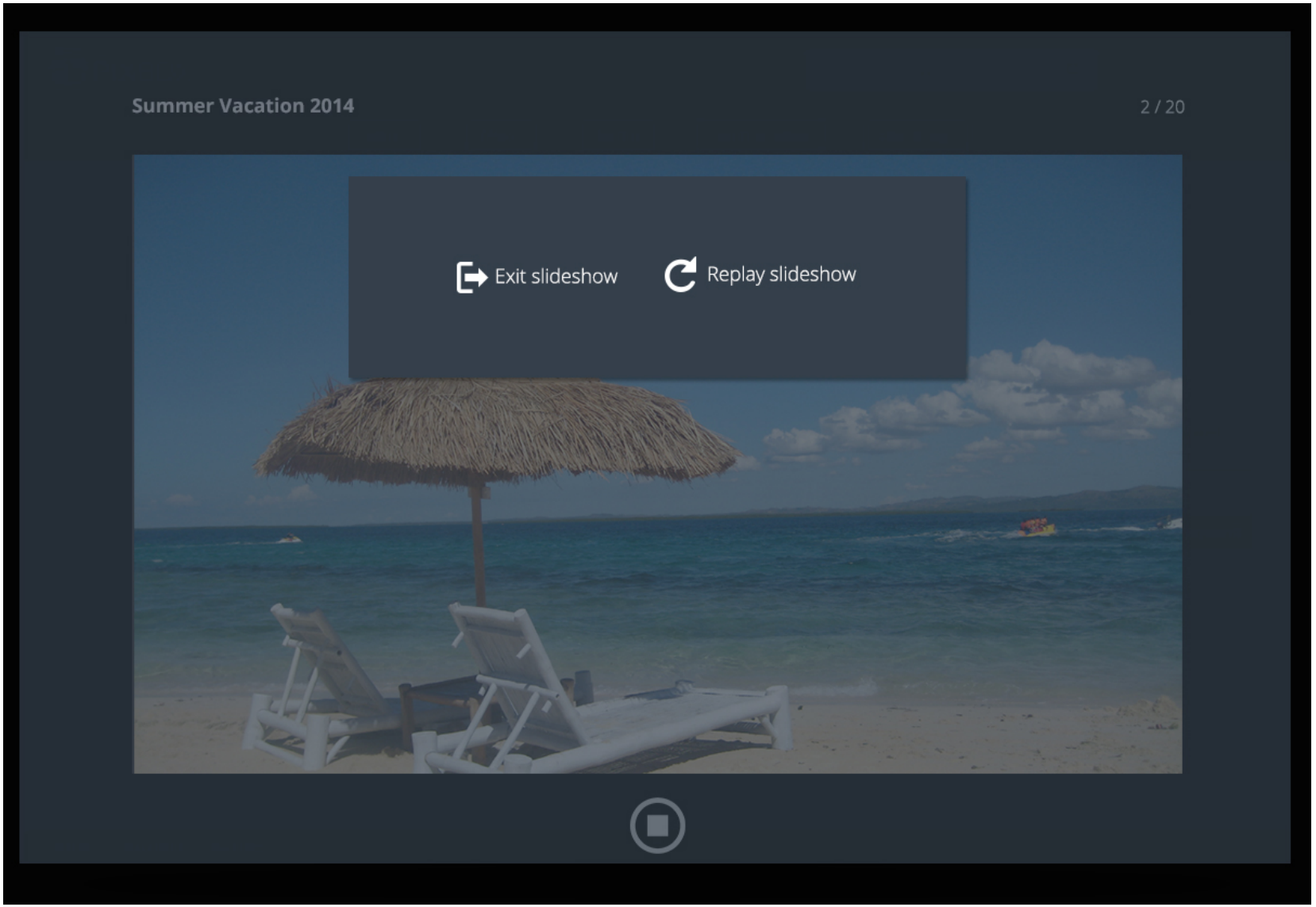
6.16 User interface 2.1: View a single photo



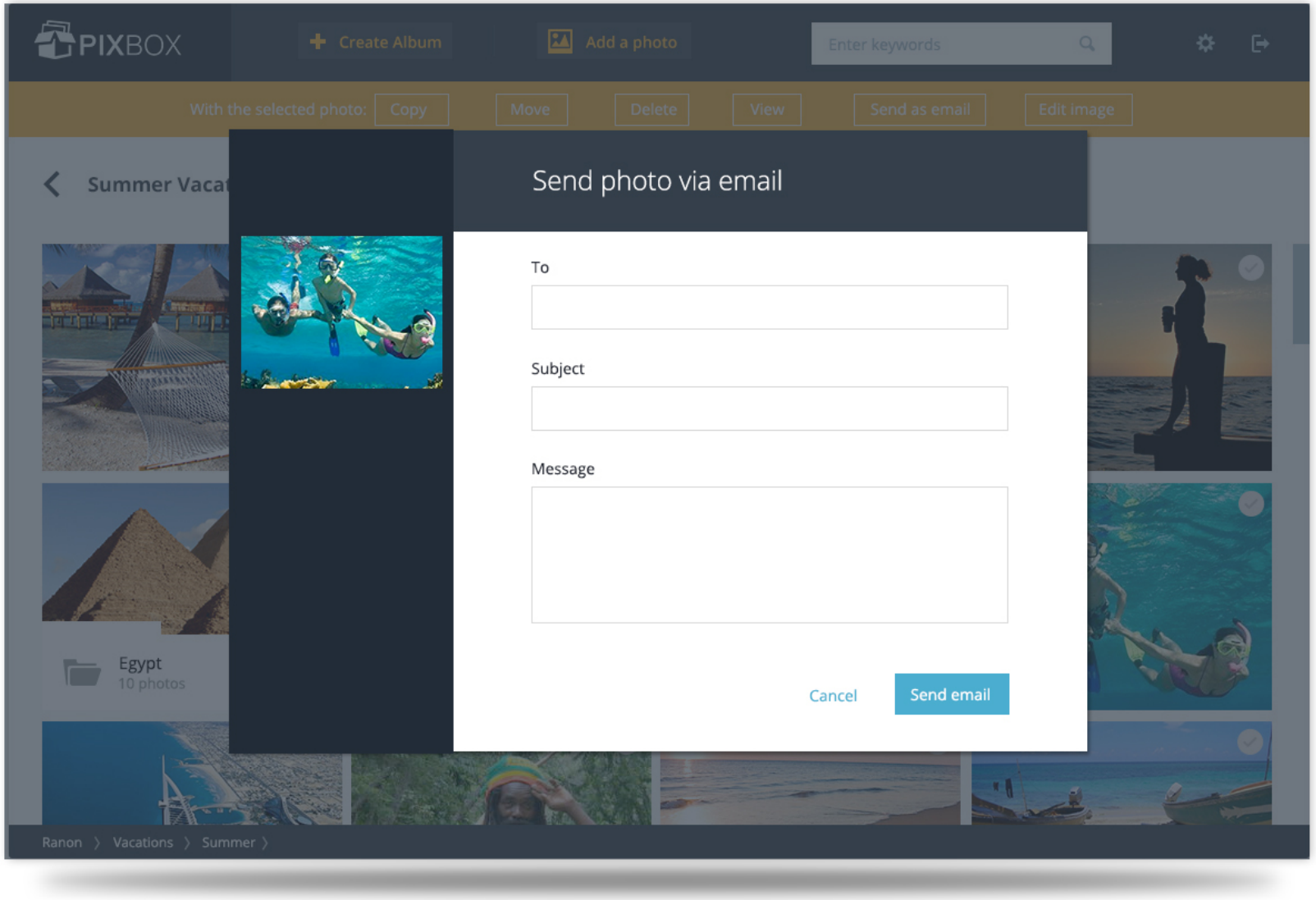
6.17 User interface 2.2: View an entire photo album of photos as a slideshow



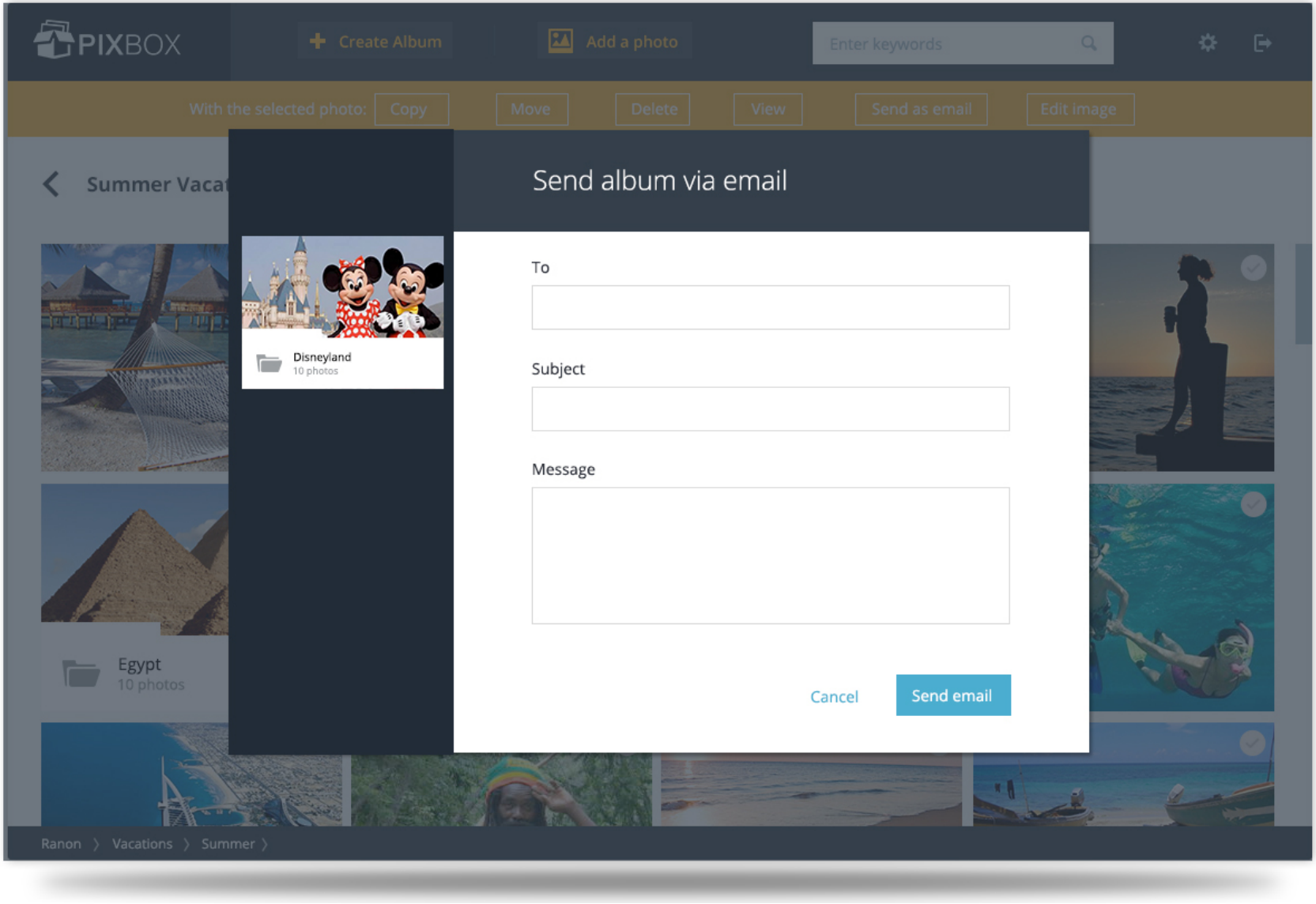
6.18 User interface 2.3: Slideshow dialog



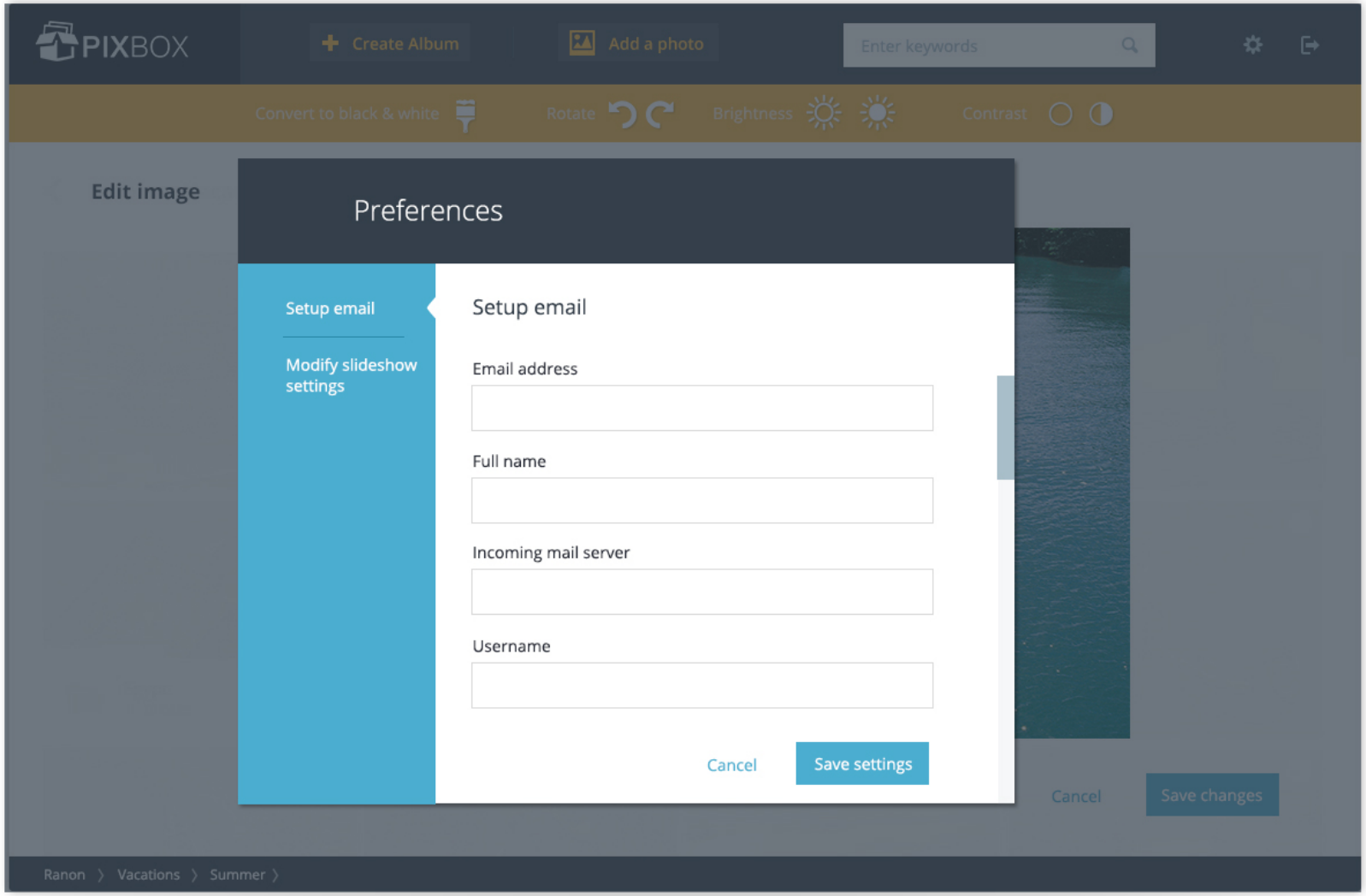
6.19 User interface 3.1: Send a single photo as an email



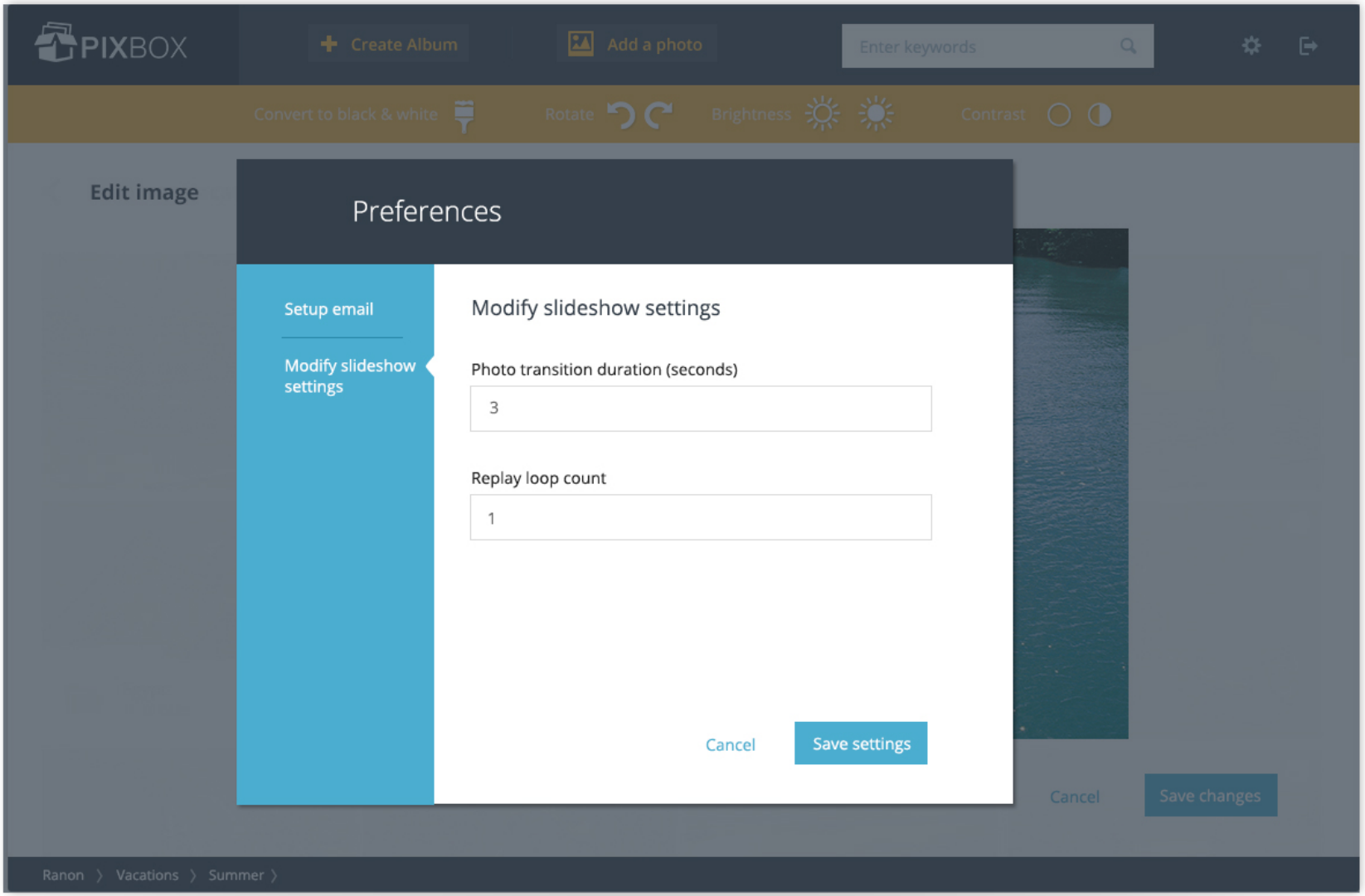
6.20 User interface 3.2: Send an entire photo album of photos as an email



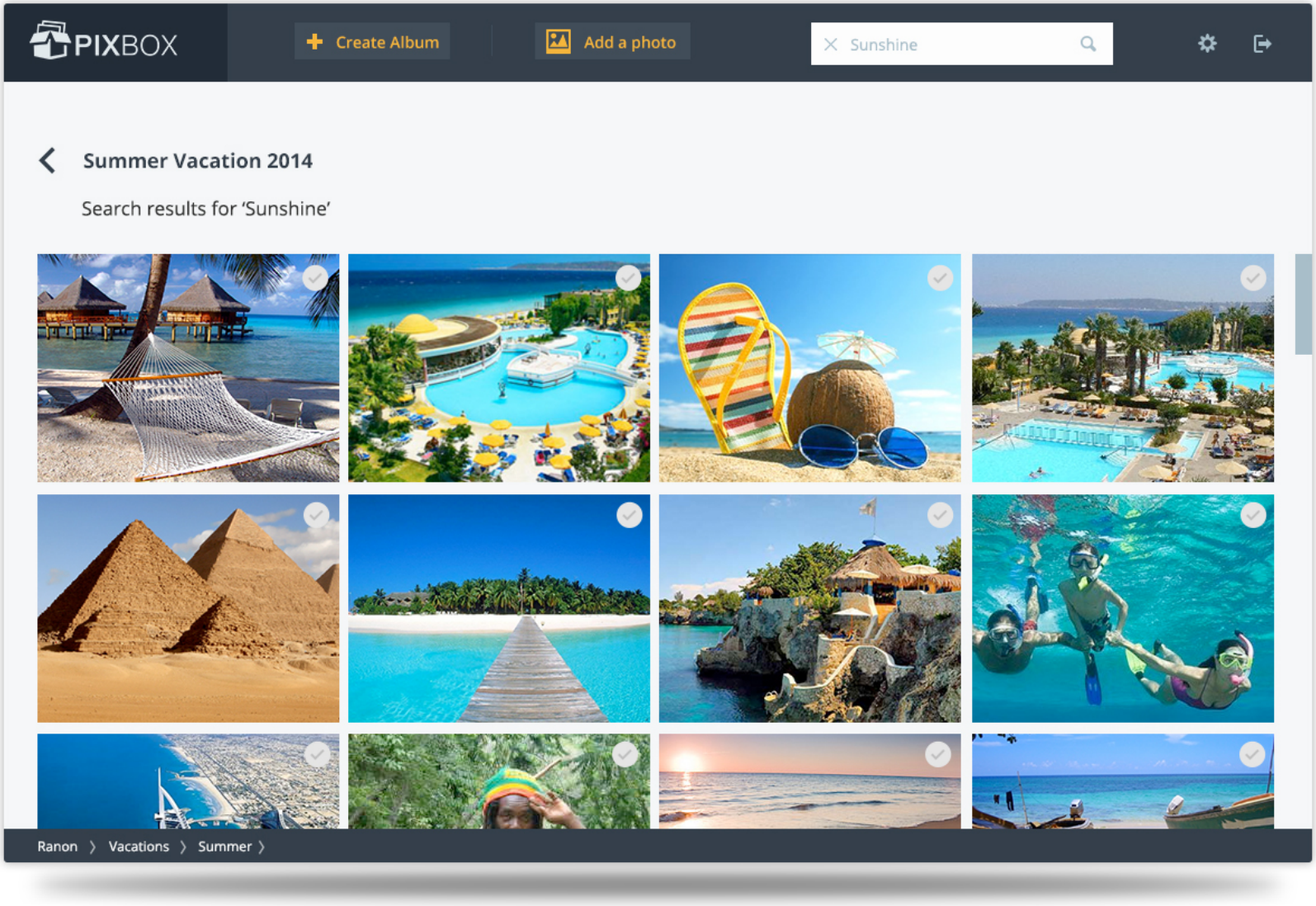
6.21 User interface 4.1: Email preference setup



6.22 User interface 4.2: Modify slideshow preference



6.23 User interface 5.1 Search results



6.24 User interface 6.1 Edit photo image