Project Plan: Photo Album

CSE-C3210 Web Software Development

Markus Isomäki, 41870R
markus.isomaki@nokia.com

Helena Kola, 58320L
helena.kola@aalto.fi

Heikki Moilanen, 69221E
heikki.moilanen@aalto.fi

1. Introduction
This is a project plan for a photo album, which is done as an assignment for course CSE-C3210 Web Software Development in Aalto University during winter 2013-2014. In this project plan we will present a general description of the project, how we are planning to implement the project in terms of views and models used, how they relate to each other, our order of implementation and timetable. In this project we will utilize valid HTML/CSS, JavaScript and jQuery, Django web framework, Ajax and Python. We shall also use Github for version control, Flowdock for communication and Heroku for web hosting.

In Chapter 2 we present the requirements of this project in order of their planned implementation and provide a general description of the implementation. In Chapter 3 we present the planned implementation of these requirements in terms of used models (Chapter 3.1) and views (Chapter 3.2), and briefly touch on the intended layout. Chapter 4 describes the timetable of the project.

2. Requirements

In this chapter we present the requirements for this project in the order of their planned implementation. The requirements provided by the client - the course - are presented in italic followed by a scenario which explains how these requirements will be fulfilled from the viewpoint of the user. Models and views concerned by these features are listed below the scenario and are further elaborated on in chapter 4.

1. *Authentication*
    a. *This should include login, logout and register to the service*
    b. *By default, only owner of an album should be able to view or edit his or her albums*
    c. *Use django.auth*

When the user hits the landing page, he will be able to register or login. Logout button is provided in all user pages except album view. The user is also provided with a way to reset his password. When the user logs in, he is directed to user main page. In providing admin functionality we will utilize Django's automatic admin interface. Also, the albums are viewable and editable only by the user who created them, except for the public view-only version accessed by the public link, which is uneditable.

Models: User
Views: Login, Logout, Register, RecoverPassword, UserMain, UpdateUserInfo.

Possible extensions to be implemented later:
    Customizing admin pages.

2. *Basic album functionalities*
    a. *Create albums (one user can have multiple albums)*

When the user is logged in, he is able to create an album or edit an existing one. When creating an album, the user will be provided with an album template including the first page, one spread, and the last page.

Models: Album and User
Views: CreateAlbum, ViewAlbum, EditAlbum, DeleteAlbum, ShareAlbum

b. *Add pages to albums and select the layout of the newly created page - a predefined set of page-layouts is enough. There should be layout options with different numbers of images/captions on the page.*

The user will then be able to add spreads to the album as he wishes. The layout is described in chapter 3.3. When the user clicks "create page", the navigation bar changes to possible layouts, which is a tab which was normally hidden.

Models: Page and Layout
Views: CreatePage,UpdatePage, AddLayout, ViewSpread, EditSpread, AddImage

c. *Modify existing albums (add and remove pages, change images and captions)*

When an album is open, the user is able to add or remove pages from the side bar on the left-hand side of the pages. Images linked by the user can be found on a bar below the page and dragged and dropped to their place in the album. Links to new images can be added by clicking an empty image spot or a button next to the lower bar. The user will be prompted for an url where the image can be found. Same image can be used in multiple albums.To protect order info an album is locked from editing if it's already been ordered.

Models; Page, Album, Image, ImagesOnPage
Views: CreatePage, DeletePage, UpdatePage, AddLayout, ViewSpread, EditSpread, AddImage

Possible features to be implemented later:
Delete album. Only allowed if a physical copy hasn't been ordered.

3. *Order albums*
   a. *Allow users to create orders from albums and to use our internal payment service to pay orders before accepting them. See http://payments.webcourse.niksula.hut.fi/ to find more about the payments service. Optionally order handling etc.*

The user can select multiple albums in the albumsite and click "order these albums". The user will be guided through the order process.

Models: Order, PaymentInfo

Views: CreateOrder, ViewOrder, ProcessPaymentResult

4. *Use of Ajax*
    a. *Use Ajax somewhere where it is meaningful in your application. For example, to browse an album so that a whole page is not loaded when a user "flips a page". Or the user can swap images via drag and drop and the change is sent to server in the background.*

Ajax will be used in the album edit stage when the user flips a page, adds an image link or adds images to a page from image list. It is also used to create a new page and choose its layout. Basically most of the functionality necessary to do any changes to an existing album uses ajax.

Models: Album, Page, Image, ImagesOnPage, Layout
Views: AddImage, UpdatePage, AddLayout, CreatePage

5. *Public link to photo albums*
    a. *Owners of an albums should be able to share a public links to their albums. This can be done by copy-and-pasting the URL from the service.*
    b. *These links should not require login to your service and should also be difficult to guess.*
    c. *Public links should not allow editing of albums*

Each album will have a “share link” -button which generates a unique hard-to-guess share ID and uses it to build an url to a public view-only version of the album. The shared album template shares some features and functionality with the editable album, but naturally all features that have anything to do with editing the album or viewing user information are missing. Not even a link to user page is included.

Models: Album
Views: ShareAlbum

6. *Share albums*
    a. *Allow users to easily post a public link to a photo album to Twitter, Google+, or other similar services.*

The implementation of this requirement will be familiarized with at a later date.

7. *Integrate with an image service API*
    a. *Use the Flickr API or some other image service API to allow users to search images when adding images to pages.*

The implementation of this requirement will be familiarized with at a later date.

8. *3rd party login*
    a. *Allow OpenID, Gmail or Facebook login to your system. Hint: Think what information you get from third party login services and if some extra information is also needed. This is the only feature where you are supposed to use third party Django apps in your service.*

The implementation of this requirement will be familiarized with at a later date.

Should we have time after the features presented in this chapter, we will possibly implement some extra features. We may also implement some smaller tasks that we feel necessary even if all the tasks mentioned earlier are not finished. These features can include, but are not limited to:

- Recovering a lost password
- Being able to change layout without remaking the page
- Staying logged in with cookies

3. Implementation

In this chapter we will present models (Chapter 3.1) and views (Chapter 3.2) we plan to use in our project. Comments on the implementation are marked with #.

3.1 Models

This section gives an overview of the Models to be used.

Figure 1 shows the the models and their relationships (1-N = one-to-many; N-N = many-to-many; note that ImagesOnPage is defined as a special intermediary model using Django's "through" relationship).

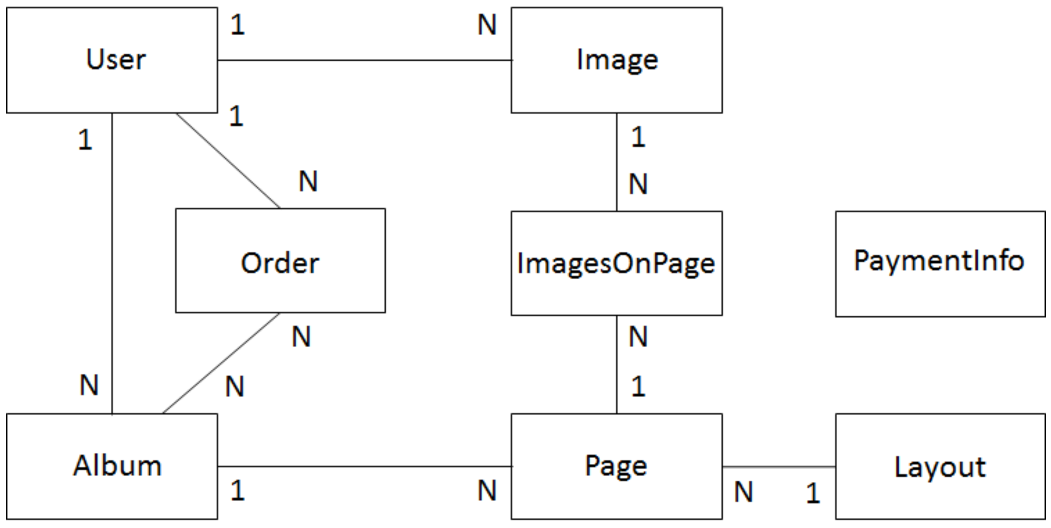

Figure 1: Models and their relationships.

The following describes the fields and their main purpose within each model.

```
User
        # imported from django.contrib.auth.models
        unique username
        password
        first name
        last name
        street address
        postal code
        city
        country # list options Finland or Other
        e-mail

Album
        # If shareId exists, an Album has a public URL based on the shareId
        shareId
        name
        description
        dateCreated
        # each Album is owned by a single User
        user = models.ForeignKey(User)
        # If album has been ordered, it's marked locked. Default is false.
        locked

Page
        heading
        # Each Page belongs to a single Album
        album = models.ForeignKey(Album)
        layout = models.ForeignKey(Layout)

Layout
        # Thumbnail of the layout
        linkURL

Image
        linkURL
        # Each Image belongs to a single User
        user = models.ForeignKey(User)
        # Image can be on multiple Pages and Page can have multiple Images in a specific order
        pages = models.ManyToManyField(Page, through=”ImagesOnPage”)

ImagesOnPage
        page = models.ForeignKey(Page)
        image = models.ForeignKey(Image)
        # Specific spot within the layout for a Image on a Page
        index
        # Specific caption for a Image on a Page
        caption

Order
        # Order status: ordered, paid, or delivered.
        status
        dateCreated
        # Each Order belongs to a single User
        user = models.ForeignKey(User)
        # Each Order can contain multiple Albums and each Album can belong to multiple Orders
        albums = models.ManyToManyField(Album)

PaymentInfo
        sid (seller id)
        success_url
        cancel_url
        error_url
        baseRate
        pageRate
        # The model will include a function to calculate the price for an album.
```

3.2 Views and templates

The base for the layout is main.html, which includes headers and a main layout common to all pages. It's inherited by majority of the other templates. There will also be simple pages dealing with static information, such as about, pricing etc.

These are the basic Views:

Registration:

Register

Contains functionality to register new user: username, email, password. Checks that the requested username isn't already taken. Links to template register.html. If all ok, creates new user.

RecoverPassword

Contains form functionality to request a new password by entering email address. Template recoverpassword.html.

Start page / Login

Standard login functionality. If login successful, directs to UserMain. Template login.html.

Logout

Standard logout functionality. Logs out and directs to start page.

The following views are only accessible if user is logged in except for the views accessed via public url.

User specific views:

UserMain

The user's central hub that lists user info, created albums and orders made. Also these lists provide links to relevant functionalities, i.e. CreateAlbum, EditAlbum, ShareAlbum, Logout, UpdateUserInfo etc. Uses template user_main.html.

UpdateUserInfo

Gives possibility for the user to change their address, email or password if they need to. Default country is Finland, and only other option provided is "Other". Uses template change_info.html. If given info is valid, updates the user information.

Album-related views:

CreateAlbum

This view initializes a new album. It creates a new Album object and adds front page, last page and one spread.

ViewAlbum

ViewAlbum renders album for viewing. Most of the work to look through the album is done in JavaScript using Ajax functionality. This view is meant to be accessed via the public link. The album to be viewed is selected based on the shareid attribute included in the link

EditAlbum

EditAlbum opens the album in edit mode as shown in chapter 3.3. It shares many aspects in common with ViewAlbum in that it uses the same functionality to show and flip through the album, but it also enables adding and removing pages, images and captions. Editing is only available for albums that aren't included in any orders.

DeleteAlbum

DeleteAlbum deletes an album along with all data associated with it except images used in it. This action cannot be undone, and isn't allowed if an order has been placed with the album included.

ShareAlbum

ShareAlbum creates a shareId attribute for an existing album. That attribute is used to form a public link (URL) to the album.

Views dealing with pages and images; RESTful views via JSON:

CreatePage
: This view creates a new page to the photo album. As this is done asynchronously, the request and reply will be in JSON format. Side bar is set to display all layout options.

AddLayout
: Adds a chosen layout to page. This sets the number of images on the page, and can't be changed later.

EditSpread
: Fetches (next, previous or as chosen from sidebar) two consecutive pages (i.e., a spread) and renders them for viewing. This includes functionality necessary for editing. If only one page is chosen, the other side will be empty.

ViewSpread
: Similar view to EditSpread, but without editing functionality. Possibly merged with EditSpread later, but is for now as reference as a separate view.

DeletePage
: Deletes a given page and updates page numbers for later pages.

UpdatePage
: Deals with requests associated with changes in page content. Adding or changing images to their slots in the page, image captions or page caption.

AddImage
: Creating new image objects. Will be displayed on the bottom image list right away.

Views dealing with the ordering system:

CreateOrder
: CreateOrder deals with building an order form for user to fill in and processes its submission. User information, such as address, is pre-filled from the user profile. If the form data is valid, it creates a new Order object. If user chooses to pay, the Niksula payment service is called.

ViewOrder
: Shows more detailed information about an order.

ProcessPaymentResult
: Handles the requests to success_url, cancel_url and error_url from the Niksula payment service after the payment transaction is completed. The corresponding information is shown to the user. If payment was successful, the status of the order is changed to “paid”.

3.3 Layout

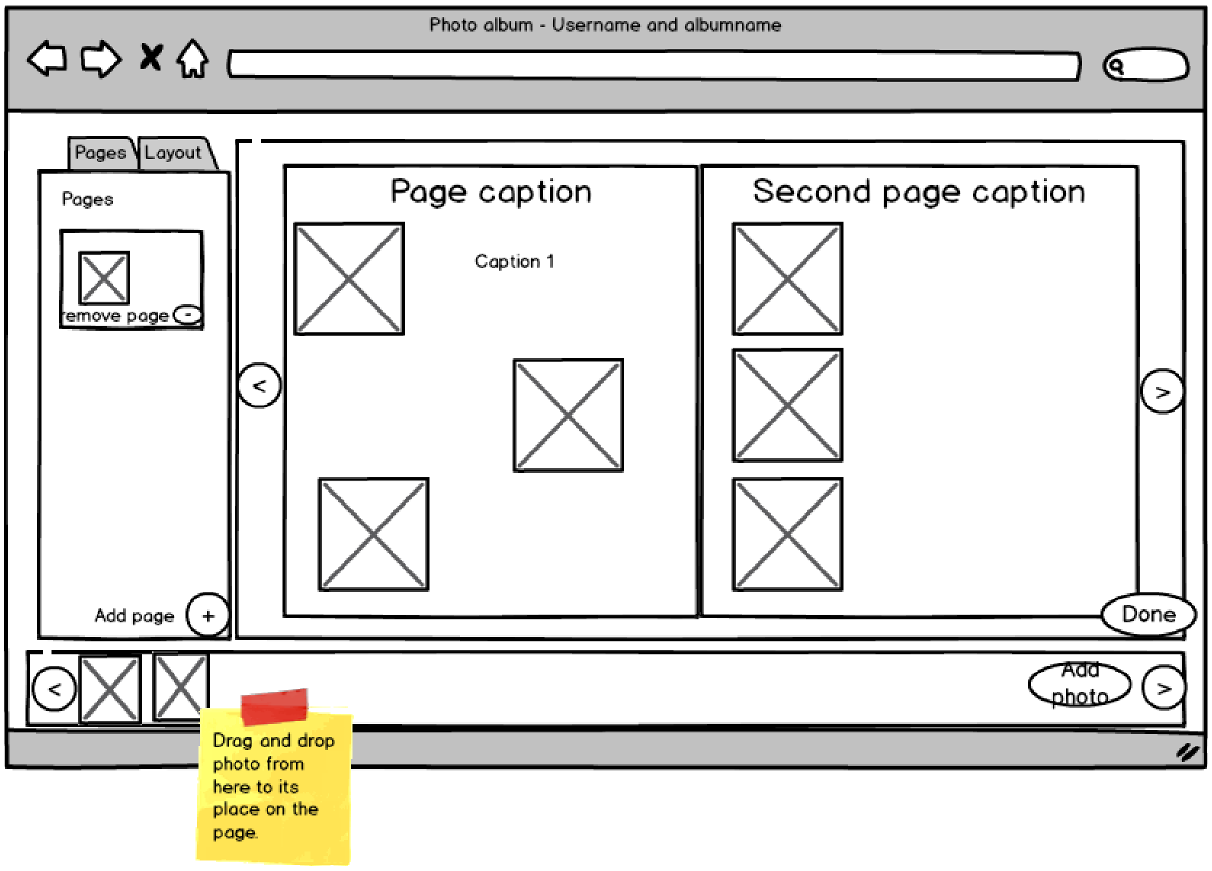
Photo album - Username and albumname
Pages
Layout
Pages
remove page
Add page
Page caption
Caption 1
Second page caption
Done
Add photo
Drag and drop photo from here to its place on the page.

Figure 2: Layout of one album spread including 2 pages

Figure 2 shows the basic layout of the album edit page. When a new album is created a new album with front and last page and one spread is generated, and the user can start working on his album. The bottom section of the layout shows all images the user has ever linked to the service. The left section shows blueprints of the pages in the album, their delete buttons and the possibility to add a new page. The layout options are shown in this area only when creating a new page.

Other pages on this site will be more "traditional" in terms of layout and functionality. They are mainly concerned with data that isn't changed as intensely.

4. Timetable

This project is divided into three sprints. Each sprint is planned at the beginning of the sprint.

Sprint 0: December 2013
Sprint 1: 1.1.2014 - 15.1.2014
Sprint 2: 16.1.2014 - 31.1.2014

Sprint 3: 1.2.2014 - 14.2.2014

Sprint 0 deals with making the project plan, setting up development environments and once the plan is accepted, building a base for the project.

Sprint 1 is meant for working on the main functionality and making sure that it roughly works. Also rough layouts are formed at this stage.

Sprint 2 is for fine-tuning functionality and layout, and possibly adding non-essential features. This can spill to sprint 3. The main product should be pretty much done at the end of the sprint.

Sprint 3 is reserved for testing, fine-tuning and finalizing documents. Also if there is time, we can add smaller non-essential features at this time. This is in an effort to get everything done on time.