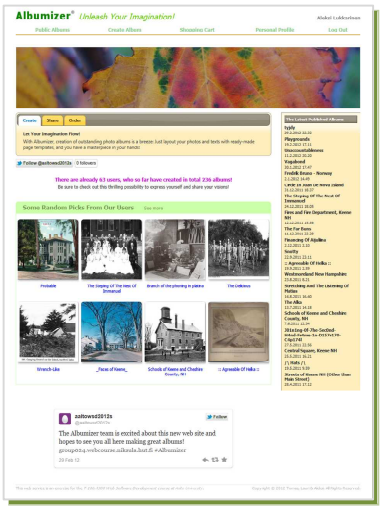
T-106.4300 Web Software Development, Aalto University

Project Documentation

Basic Information



Date	Spring 2012		
Project Size	5 ECTS Credits		
Group Members	Tomas Heiskanen	64843T	tomas.heiskanen@gmail.com
	Lauri Helkkula	62820H	lauri.helkkula@aalto.fi
	Aleksi Lukkarinen	270908	aleksi.lukkarinen@aalto.fi
Group ID	group024		
Website URL	http://group024.webcourse.niksula.hut.fi/		
GitHub URL	https://github.com/lhelkkul/PhotoAlbum/		

Implemented Features And Tasks

The following table shows the main tasks done during the project. Every task has a responsible person. The responsible person has done all or most of the task. Everyone has participated actively in the project.

Feature / Task	Required by Course	Responsible Person
A welcome page describing the application and listing some public albums.	No	Aleksi
Customer can log in, log out and register.	Yes	Aleksi
Customer can list all public albums and his/her private albums.	Yes	Tomas
Customer can create new albums.	Yes	Aleksi
Customer can modify existing albums which he/she owns.	Yes	Divided into smaller tasks
New pages can be added. Layout of every page can be selected from a predefined set of layouts.	Yes	Tomas
Existing pages can be removed.	Yes	Lauri
Captions can be changed on each page.	Yes	Tomas
Images can be changed on each page. Customer is able to upload images to the server from his/her workstation.	Yes	Tomas
Each of the albums can be declared as either public or private.	Yes	Aleksi
Customer can browse existing public and his/her own albums.	Yes	Tomas
User can view an album with a browser (without ajax).	Yes	Lauri
User can view an album with ajax.	Yes	Lauri
Slideshow for viewing albums.	No	Lauri
Customer can put albums to a shopping cart and finally make an order containing the selected items. The payment is made via the Simple Payments service.	Yes	Aleksi

Customer can view and modify his/her personal information (address etc.) on his/her profile page.	No	Lauri
Customer can post a link to a photo album to Twitter.	Yes	Aleksi
For each album he/she owns, customer can generate a public link that can be used to link to the album when sharing online.	Yes	Aleksi
Customer can login to the service with his/her Facebook account.	Yes	Lauri
Thumbnail generation for album covers + hashing, permissions and folder structure of uploaded files	No	Aleksi
Initial implementation of models	(implied)	Lauri
Configuring the Django's admin site and admindocs	No	Aleksi
Configuring Django Debug Toolbar	No	Aleksi
Django-admin command for populating database	No	Aleksi
Caching control on view, template and low level	No	Aleksi
Final HTML5 and CSS3 validation testing	No	Lauri
Deployment to course server	Yes	Everyone
GitHub research, initial repository	No	Lauri
Initial Django project creation	(implied)	Aleksi
Final UI tweaking	No	Aleksi & Lauri
Final document	Yes	Aleksi & Lauri
Project planning	Yes	Everyone

We have tracked time usage here:
https://docs.google.com/spreadsheet/ccc?key=0Ard1Vur7AcqTdEZUdkowczFxczNHQkVmVklidXdRckE

Project Functionality

How to setup the software

1. Required libraries
    a. Django Debug Toolbar (http://pypi.python.org/pypi/django-debug-toolbar/)
    b. Django South (http://south.aeracode.org/)
    c. Python Imaging Library (http://www.pythonware.com/products/pil/)
    d. docutils (http://docutils.sf.net/)
2. Most important environment-specific settings
    a. We have used an environment-specific settings file, which is determined from the environment variable "`USER`". Although this didn't work in the course server, it was great during development phase for using version controlled settings file and still allowing different settings for different environments
    b. Set database settings (`DATABASES`)
    c. Set the folder for uploaded images (`MEDIA_ROOT`)
    d. If not using the Django development server
        i. Set the folder for static files (`STATIC_ROOT`)
        ii. Set `STATIC_URL`
        iii. Set `ADMIN_MEDIA_PREFIX`
    e. Set logging parameters (`LOGGING`)
        i. At minimum please set the "filename" settings to point to a suitable location

3. Optional settings related to 3rd-party social media services:
    a. Twitter settings
        i. `TWITTER_HASHTAG`
        ii. `TWITTER_ACCOUNT`
    b. Facebook settings *(these are empty in the source code we gave in but do exist on the server)*
        i. in settings_facebook.py. settings.py is publicly accessible via github, and this file contains secret information of an application in Facebook. Misuse might result in trouble for a group member. Therefore this is a separate file which is not version controlled.
        ii. `FACEBOOK_APP_ID`
        iii. `FACEBOOK_SECRET`
4. Execute the following commands:
    a. `python manage.py syncdb`
        i. Don't create an admin user when prompted
    b. `python manage.py migrate albumizer`
5. If not using the Django development server, collect static files by executing
   `python manage.py collectstatic`
6. If you want some test data to be generated
    a. Put some jpg and/or png images in a folder
    b. Get help for the test data generation command by executing
       `python manage.py help insert_albumizer_test_data`
    c. Perform the actual data base population by execute e.g.
       `python manage.py insert_albumizer_test_data --verbosity=2 --users=100 --albumsmax=20 --ordersmax=15 --image-base-path=/PATH/TO/IMAGES/`
    d. The test data generator can always be stopped gracefully by pressing Ctrl+C once.

Default fixtures

- Two layouts for album pages
- One admin user for each group member and one for general testing purposes
- Order statuses
- States of USA, Australia and Brazil
- Countries

How to use the software

Here is a list of most important pages in the website. All pages in the top level of the list are accessible using the navigation in the top of each page (some pages' visibility depends on whether the user is logged in or not). Other pages are listed as sub-items and are accessible through the "parent" page.

- Front page (url `/`)
    - Short description of the site
    - Random picture list (from public albums)
        - If JavaScript is enabled, updated dynamically every 20 seconds
    - A list of latest public albums
    - Twitter plugin
    - Login for JavaScript users via a jQuery dialog
        - Username & password or Facebook
- Login page (`/accounts/login/`)
    - Login for non-JavaScript users and for giving error messages related to login
- Registration page (`/accounts/register/`)
- Profile information (`/accounts/profile/`)
    - Basic profile information, order history, own albums
    - Editing profile information (`/accounts/information/`)
    - Order information (`/order/<order_id>/`)
- Public album list (`/album/`)
    - Adding album to shopping cart
    - Viewing a single album (`/album/<album_id>/`)
        - Twitter plugin, "tweet this album"
        - Editing a single album (`/album/<album_id>/edit/`)

            - The public link for the album is at the bottom of this page
                - If the album is public, the url is the same for everyone
                - If the album is private, the url is suffixed with a secret hash
        - View a single page (`/album/<album_id>/<page_number>/`)
            - Supports browsing pages without JavaScript
            - Ajax-browsing if JavaScript enabled. Note also history management.
            - Edit a single page (`/album/<album_id>/<page_number>/edit_page/`)
        - Viewing a private album via a secret link, which is given at the bottom of private albums' pages
            - Same url:s as for an album, a page or a slideshow, but suffixed with `/s/<secret_hash>`
        - View a slide show of the album (`/album/slideshow/<album_id>/`)
            - Javascript required
- Album creation (`album/create/`)
- Shopping cart (`/cart/`)
    - Cart contents
    - Delivery addresses (`/order/addresses/?<validation_hash>`)
        - Each item in the cart can be ordered to a different address
        - Order summary (`/order/summary/?<validation_hash>`)
            - Order created, go to payment (`/order/<order_id>/successful/`)
            - Payment successful (`/payment/sps/successful/?<payment_data>`)

There are no differences between admin and non-admin accounts except the possibility to access the admin site. To test the functionality of the parts visible to regular users, any user account present in the database can be utilized—the functionality (exceluding the admin site) is the same for all of them. For example, one can pick an album from the list of public albums and check its creator's username to log in.

The following admin user accounts are created by the database fixtures: aleksi, lauri, tomas, testuser.
Default password of all accounts created by the fixtures or generated by the test data generator is "salasana".

The admin site is located at `/albumizer_manager/`.
Admindocs is available at `/albumizer_manager/doc/`.

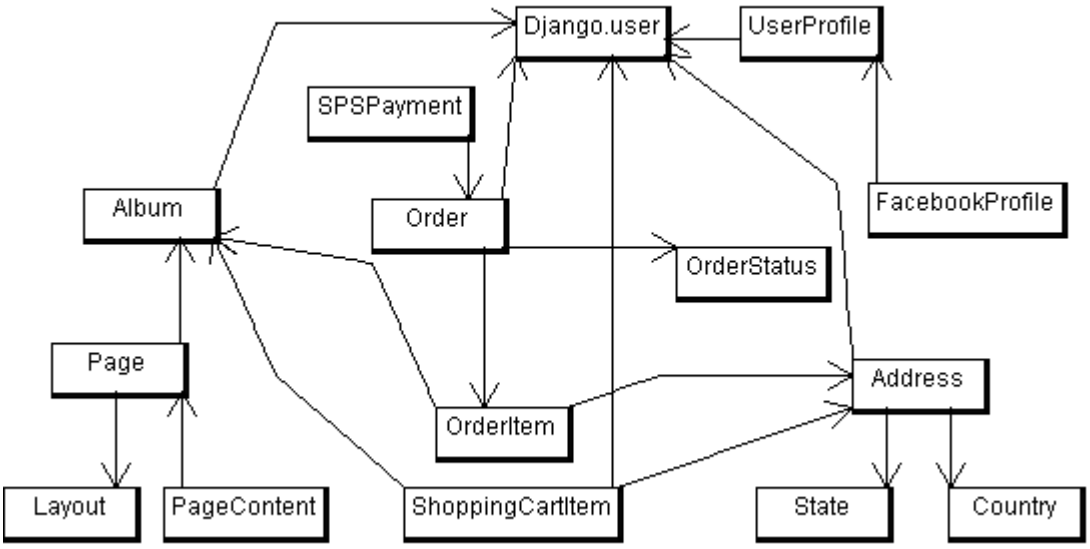
Program Structure

- The root folder
    - Settings
    - Urls (included from albumizer, admin and admindocs)
- albumizer
    - All site functionality
    - context_processors.py
        - some custom variables for templates
    - facebook_api.py
        - Facebook API, has one method to get user data from Facebook with the provided token
    - facebook_backend.py
        - A custom Django authentication backend for Facebook authentication
    - forms.py, models.py, urls.py
    - utils.py
        - Common functions
    - fixtures
        - initial data
    - management
        - test data generation
    - migrations
        - south migration files
    - static
        - static files: css, images and javascript

- templates
  - described by the folder structure and filenames
  - base-template.html is the base template for all other templates (except the slideshow)
  - basicform.html is used to render almost all forms in the site
  - the custom-tags folder includes some general snippets that are used by templatetags
- templatetags
  - common tags that are used in templates

Models

All model classes are shown in the following figure. Fields and functions are not shown to improve readability.



Ajax

Ajax is used to update the random album list and usage counters, and in browsing album pages. Note also management of browser's history when browsing album pages with ajax.

Known Faults And Missing Features

- The user can only edit one address. Multiple addresses are supported by the domain model and the test data generator generates multiple addresses for users.
- In a single order it's not possible to send copies of a single album to multiple addresses. If that is desired, at the moment one has to make as many orders as there are desired delivery addresses.
- State of albums is not saved or freezed in any way at the moment of making an order, so content of an album can change or the album can be deleted altogether after an order is made.
- An existing user account can't be linked to a Facebook account (Facebook authentication creates a new user account which can't be used with username+password authentication)
- When authenticating via Facebook, the user's name and email address are updated with data from Facebook. However, the user is able to edit that data in the profile page. The changes are overridden if the users authenticates via Facebook again.
- Logging out of the website logs the user out of Facebook. We wanted to provide the user the option to log out, but we could not find a way to disconnect the user's Facebook account from our application without logging out of Facebook as well.
- Some places of the HTML code try to utilize some microformats. However, testing of their functionality was done just before project deadline, and their usage seems to be incorrect, although the way to utilize them is the same than in some examples found in their specifications. The W3C Semantic Data Extractor cannot found all the data that is intended to be found, but there wasn't enough time to fix it.
- Nothing has been done to prevent concurrency issues related to database handling.
- For a real applicaiton, the domain model should be redesigned to rectify some problems (like images and captions currently in a single table) and to include lots of functionality needed in the real world but missing from this prototype version.

- Money calculations in the code should be implemented correctly in a real application.
- Logging is done only just enough to demonstrate its usage possibilities.
- User interface and user experience level varies between different parts of the application. Also, it would've been great to have a graphical talent to design the application. However, we didn't have one.
- No link from the album slideshow to the page showing all of the pages of that album.
- The W3C HTML validator is reporting some HTML5 validation errors, which we chose to ignore.
    - x-moz-error-message is not a standard attribute, but is available in Firefox. Other browsers ignore the attribute.
- The W3C CSS validator is reporting some CSS3 validation errors, which we chose to ignore.
    - Some non-standard attributes and attributes supposed to be valid in CSS3 are used. The validator complains about them, but they work in the browsers which understand them while others just ignore them.
    - Some of the reported errors do not make any sense and at least Firefox doesn't show any warnings about our CSS file.
    - "word-wrap:break-word;" not accepted, although it is a part CSS3.
- Mixed coding styles: tabs vs. spaces, camel casing vs. underscore, try-except vs. checking conditions, using vs. not using a specific Django's feature or a model method someone has already implemented etc.
- There isn't any kind of automated testing performed. There should be at least unit tests and functional tests realized with e.g. Django's testing facility and Selenium.
- There is no JavaScript-based form validation, and the HTML5-based form validation checks just basics. Also, some things, like availability of a username for the registration form, could be checked via ajax.
- When there are errors in form fields and the form is redisplayed, the default focus should be in the first field causing the errors. Currently this is not implemented.
- The buttons of the Simple Payments service seem not to work with the Opera browser.
- There is no pagination of the order history on users' profile pages.

The Top Three Parts of the Code

Best

- Versatile usage of Django's features, e.g.
    - Forms based on ModelForm—an easy and safe way to create forms
    - Admin site—an easy way to provide a powerful admin interface
    - Server-side low-level and template-level caching
    - Custom template tags, filters and admin command for populating the database
- Usage of HTML5 features, like forms validation and new elements (e.g. nav, article, section)
- Aiming for generic code: a general purpose form renderer (basicform.html), custom tags, base classes, base templates

Worst

- Forms.py, views.py and models.py are pretty big; they should be divided into smaller sections if development would be continued.
- CSS definitions could use more generic selectors
- Some parts of the code is copy-pasted from place to place with only slight modifications (needs to be refactored), some lacks proper documentation, and the code base could be more streamlined and uniform.

Diversions From the Original Plan

The most important changes to the initial model:

- The `UserProfile` model replaced the corresponding old table after we learned how to save additional user data in Django's way. After that, the foreign keys referencing to the old user profile table were changed to point to the Django's own `Auth.User` table.
- The `Layout` model was added during the implementation of album pages.
- A custom ImageField was added to the `PageContent` model.
- The `OrderStatus` table was created to model different states which an order can be in.
- `Country` and `State` models added for addresses instead if plaintext fields.
- Delivery address link moved from `Order` to `OrderItem`.
- `SPSPayment` was needed when implementing payment services.
- `FacebookProfile` created for Facebook-related user data.

The initial implementation order had one major change: The Facebook authentication was implemented earlier than planned due to a group member's inability to attend group coding sessions due to an illness. The initial timetable differed from actual timetable by couple days.

References

- For Facebook authentication, the django-facebook (https://github.com/tschellenbach/Django-facebook) was tried to put in use without success, after which an own implementation was coded with help of the django-facebook.
- Slideshow uses JavaScript code directly from http://code.google.com/p/html5slides/, licenced under the Apache License 2.0. HTML markup generation is implemented by us by modifying the template provided by the authors.
- A read-only form field found here: http://lazypython.blogspot.com/2008/12/building-read-only-field-in-django.html.
- The idea how to implement thumbnailing functionality is got from a book called *Python Web Development with Django (*by Jeff Forcier & Paul Bissex & Wesley Chun) and developed further.
- Other major sources for small pieces of code include Django's documentation and the StackOverflow website. Nothing was ever directly copied, but the sources were used to get ideas for implementing some specific piece of functionality.
- List of USA's, Australia's and Brazil's states is created based on Wikipedia article about federated states (http://en.wikipedia.org/wiki/Federated_state).
- List of countries is created by processing the list given in the ISO 3166 (http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm).
- Lists of source information for the test data generator are combined from many miscellaneous sources both during this project and in the past.