Project Plan

1. Process Selection

We will use Kanban as the process tool. The following elements of Kanban is put into place

1. Event-driven: We trigger a planning meeting whenever we start running out of stuff to do.
2. We will limit the WIP per Sprint
3. We will limit the WIP in Kanban
4. We will try to keep weekly Sprints, each sprint is supposed to produce a release, which contains:
    a. Unit testing/module testing/UI testing with high code code & branch coverage
    b. Updated Architecture Status
    c. Deployed to Heroku with full testing and functionality of the Sprint content

At the beginning of each sprint, we will do the following:

1. (Retrospective)
2. Updating the backlog based on previous Sprint results (i.e. new user stories, new todo's, errors, blockers etc)
3. Re-Prioritizing the backlog
4. Review of the current architecture
5. Selecting the backlog items (i.e. EPIC/User Story) for the Sprint
6. Breaking the backlog items into user stories and tasks (only for the selected backlog items for the Sprint)
7. Freezing the Sprint content
8. Reviewing the target architecture after the target Sprint

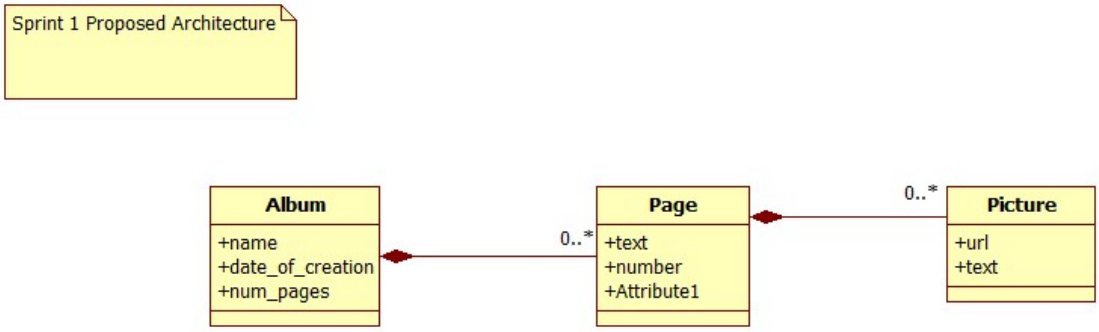
We will try to implement all the requirements if resources suffice. Here is a **proposal** for the requirements **in green as "must have"** and **grey in "optional"**

Requirement	EPIC Description
Basic album functionalities	Create albums
	Add pages 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.
	Modify existing albums (add and remove pages, change images and captions)
Authentication	This should include login, logout and register to the service (django.auth)
	By default, only owner of an album should be able to view or edit his or her albums
Integrate with an image service API	Use the Flickr API or some other image service API to allow users to search images when adding images to pages
Use of Ajax	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".
	By default, only owner of an album should be able to view or edit his or her albums
Public link to photo albums	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.
	These links should not require login to your service and should also be difficult to guess.
	Public links should not allow editing of albums
Share albums	Allow users to easily post a public link to a photo album to Twitter, Google+, or other similar services.
Order albums	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.
3rd party login	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

Below is the agreed content for Sprint 1 content broken down into detailed Tasks

Basic album functionalities	BAF.1	Create albums	
	BAF.2	Add pages 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.	
	BAF.3	Modify existing albums (add and remove pages, change images and captions)	
User Story ID	**User Story Description**		
BAF1.1	Create an Album which contains Pages, each page containing Pictures and texts. Refer to the schema.		
Sprint 1			
Task ID	**Task Description**		**Expected**
BAF1.1.1	Picture model with valid url and text is functional		unit testing, coverage results
BAF1.1.2	Page model with text and page number is functional		unit testing, coverage results
BAF1.1.3	Page model has many pictures with add, remove and change functionalities		unit testing, coverage results
BAF1.1.4	Album model has many pages with add, remove and change functionalities		unit testing, coverage results
BAF1.1.5	Create a fixture containing different albums, extend unit test cases with the fixture		unit testing, coverage results
BAF1.1.6	Setup Heroku environment, learn to how to launch the example project		
BAF1.1.7	Release 0.1 to Heroku		unit testing, coverage results
Sprint 1 Review & Sprint 2 Planning	Requirements Breakdown for BAF.2, prioritization, Sprint 2 content planning, Current Architecture and Target Architecture on next Sprint		
Sprint 2			
Task ID	**Task Description**		**Expected**

And below is the startup architecture for the above Sprint 1 Content



The way we intend to grow the system

Is such that:

1. We get a small piece of requirement, start implementing from Django side together with test automation in models, templates and views.

2. We would like to develop minimal on client side first, just enough functionality reflecting the requirement of that Sprint.

3. Once we reach the client side with that functionality, we will think of how requirement should be reflected to the system in detail.

4. At this point, we can detail the use case / epic, add new todo's and test cases.

5. We implement those findings raising from 4, starting from Django side to client side again in a chain

6. We verify that the requirement at step 1 got implemented fully with testing and releasing

7. We go back to step one and pick a new requirement

2. Suggested Process Tools / Readouts

Kanban

http://www.targetprocess.com/product/kanban/

Kanban In Action

We want to do a code review process close to this

3. Sprint 1 - ToDo BAF1.1.6

Note: For BAF1.1.6, we have done this pre-work, which has Django Round 2 (Server), JavaScript Round 2 - Dynamic Form (Client) and Selenium test cases (in unit tests)

Refer to ReleaseTestProject folder in our Github Repository

This is the project, which has Django round 2 as server side
and JavaScript round 2/Dynamic Form as client side. Please refer to
their A+.htm files for the details of those assignments.

Please do not share this code anywhere else! Lets not break our agreemen
Aalto Students agreement for the course.

Here are the sub tasks for this task:

********** Task #1 **********

Making these 2 exercises work together + updating test cases

https://plus.cs.hut.fi/exercise/21/
Dynamic Form
Dynamically Updating Form Fields

https://plus.cs.hut.fi/exercise/15/
https://plus.cs.hut.fi/exercise/23/

Rendering JSON from Django Views
In this exercise you need to implement views (continent_json and country

********** Task #2 **********

Adding Selenium WebClient test cases using Firefox as browser
To the combination of the following exercises

https://plus.cs.hut.fi/exercise/21/ (Dynamic Form)
https://plus.cs.hut.fi/exercise/15/ (Django Models and Initial Data)
https://plus.cs.hut.fi/exercise/23/ (Rendering JSON from Django Views)
https://plus.cs.hut.fi/exercise/24/ (Templates and (changes to) views

Worth reading:

http://seleniumhq.org/docs/01_introducing_selenium.jsp
http://seleniumhq.org/docs/03_webdriver.jsp#chapter03-reference

http://seleniumhq.org/docs/04_webdriver_advanced.jsp#chapter04-reference