G52GRP: Interim Report

Group gp09-jqg*

November 29, 2009

1 Implementation Decisions

1.1 Decision Influences

1.1.1 Aims

The aims of version 1 have a strong influence over implementation decisions. They are as follows;

1. To be a complete working release of a usable piece of software
2. To allow the team to familiarise themselves with the tools and systems to be used for later versions
3. To explore the capabilities of those systems, to inform and inspire later decisions.

1.1.2 Design Principles

The project is being developed using a version of the Extreme Programming Methodology. XP's software development principles have an impact on the software design principles of projects developed using it.

Similarly the Web Framework Django has its own set of design philosophies[1] which also influence the project's design principles.

*Dhruv Gairola, Chris Head, Rob Miles, Amy Jane Wesson and Chenjue Xu, supervised by Julie Greensmith

[1] `http://docs.djangoproject.com/en/dev/misc/design-philosophies/`

The principles of XP and Django are quite similar and complement one another quite well, so it is possible to abide by both sets of principles without contradictions.

Some of the XP/Django principles that have the most influence on implementation decisions are listed below;

DRY Don't Repeat Yourself
: Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

YAGNI You Aren't Gonna Need It
: Always implement things when you *actually* need them, never when you just *foresee* that you need them

Maximise Code Reuse If two bits of code look similar, move them out into a more general function. If two functions do a similar thing, merge them. This keeps redundancy low.

Both XP and Django have a strong basis in philosophy and principles, and while they both leave the developer the freedom to chose their implementation decisions, they are designed to work best with implementations that follow their principles.

1.2 Decisions

1.2.1 The `recipes` App

Recipes are the only thing that version 1 does, so it would make some sense to simply have the project as a whole perform the recipe functions, and not use any apps. However, the functionality of the site was put in a `recipes` app for two reasons. Firstly, it is best practice in Django to have all code in conceptually distinct, reusable apps, to maximise potential code reuse. Secondly, as one of the aims of this version is to introduce the team to working with Django, and apps are a major part of Django, it made sense to use apps even if they are not strictly necessary.

1.2.2 URL Design

The URL design is intended to be very simple and readable. In accordance with Django URL design principles, there are no filename extensions in URLs.

1.2.3 Model Design

The only implementation decision of note in the model design is the use of a python `property` to handle recipe tags. A `property` is a python language construct that behaves as though it is a class variable, but behind the scenes calls a getter or setter function when it is fetched or assigned to. This was used because, although it makes the model less readable, it makes all of the code that deals with the model far more readable, and it is this code which is more complex and benefits more from simplification.

1.2.4 View Design

The `recipe_list` View

There are 2 views that simply show a list of recipes:- `recipes_all` (the view of all recipes on the system) and the results section of `search` (the view of all recipes that meet the search terms). In order to maximise code reuse, the functionality of displaying a list of recipes was taken out into a separate `recipe_list` view, which is called by both `recipes_all` and `search`.

The `search` View

The search is deliberately the simplest search possible that meets the specifications. The set of results is simply the set of recipes which contain any of the ingredients searched for. This will be radically improved in later releases.

1.2.5 Template Design

> "The most powerful – and thus the most complex – part of Django's template engine is template inheritance. Template inheritance allows you to build a base "skeleton" template that contains all the common elements of your site and defines blocks that child templates can override."
>
> – Django's Template Documentation[2]

Template Inheritance provides a good opportunity to maximise code reuse, but it was not used in version 1 in an attempt to keep template design simple.

[2] `http://docs.djangoproject.com/en/dev/topics/templates/`