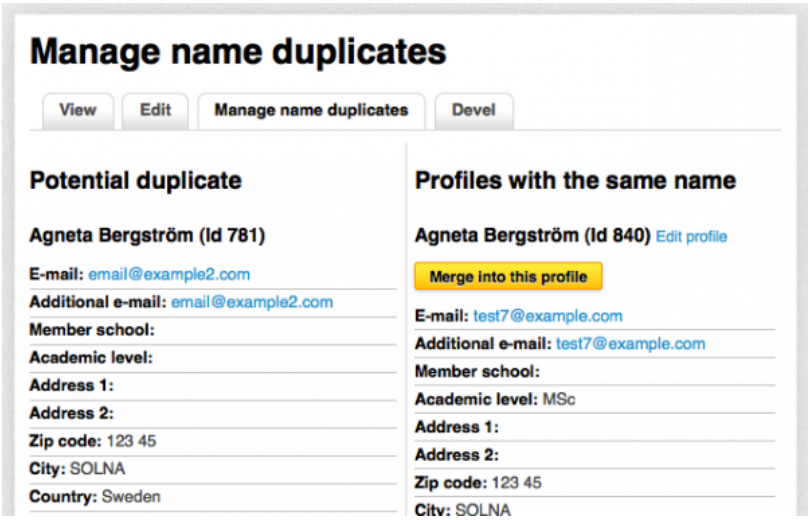
Technical documentation for "SSES courseweb"

JOHAN FALK
ONS, 2011-06-01 15:34

BLOGG: SSES COURSEWEB



The most complex function on SSES courseweb is probably the profile/user management.

This is a part of the experimental open techincal documentation for the SSES courseweb project. This part gives an introduction to the site functionality and summarizes the project process.

The SSES courseweb project had as a goal to extend the existing website for Stockholm School of Entrepreneurship, from being (more or less) a brochure site, to a site where:

- course administrators ("registrars") keep their CRM, searching and managing profile information for students, teachers, event speakers and more;
- course administrators can add and manage courses, and manage student applications for courses;
- teachers can log in and manage courses that they give, including adding assignments for students, blogging, and publishing course schedule and documents; and
- students can log in and get relevant information about their courses as well as hand

RELATERAT

- SSES courseweb 5: Automatic e-mail notifications
- SSES courseweb 4: Profile management
- SSES courseweb 3: Course content and access
- SSES courseweb 2: Site architecture - node types and sections

FÖRFATTARE

JOHAN FALK
DRUPAL TRAINER / DEVELOPER

MER FRÅN JOHAN FALK

- Learn Drupal coding with NodeOne part 6: More arrays
- Learn Drupal coding with NodeOne part 5: Arrays
- SSES courseweb 5: Automatic e-mail notifications

- in assignments.

The site additions were developed collectively by *Team Arnold* (also known as "team 4") at NodeOne, primarily by Michael Axelsson and Johan Falk. The development spanned four sprints of two weeks each, summing to approx. 450 hours. The development was made in scrum style, where the excellent communication and reaction time by the client (represented by Sebastian Razola). It is difficult to overstate Sebastian's role in getting this project to such a good result.

This technical documentation is an experiment done in agreement with the client, where we publish the full documentation online – open for anyone to read, critizise and learn from. Feel free to do all of this!

The technical documentation describes the new functionality on the website, with particular focus on the more complex functionality. The more complex functionality include:

- Synchronizing creation of user accounts with profile nodes, conditioned with non-trivial events.
- Comparing and allowing both automated and manual merging of profile duplicates.
- Giving teachers and students correct access to courses and content related to courses.
- Altered URL structures for a significant parts of the site, to allow remaining in the context of a course.

The project relies heavily on existing Drupal frameworks, especially Views, Rules, Page manager, Features, Content Access, Node Access User Reference, Node Access Node Reference, Views Bulk Operations, Panels, Contextual Administration, Feeds and Content Profile. The project also resulted in the module Rules Bonus Pack being significantly extended.

Despite quite complex functionality on the site the custom code is just short of 600 lines (~350 when removing comments and blank lines) – including form alters made for cosmetic purposes. We are happy to have provided so little code that future maintainers (including ourselves) must learn, but are also aware that the configuration on the site is sometimes very complex. We hope that this technical documentation will make the configuration clear.

EXTENDING THE EXISTING SITE, AND USING FEATURES

Before starting the development, we considered creating a new site instead of extending the existing one. Drupal 7 was an attractive option, mostly due to how Organic groups (and now also Workbench Access)would have simplified our access functionality, but other complex requirements – combined with the maturity of Drupal 6 modules – made us decide for Drupal 6.

Having the courseweb on the existing site was no requirement from the client, but a single sign-on was desired. This, combined with almost completely separated functionality, made us conclude that extending the existing site was probably the best choice. It has caused *some* problems, but we have not regretted the decision.

To allow for extending the site in a clean way, all our new functionality was packaged and deployed as features. The site ended up with eight new features:

- cw_profile: Stores configuration relating to importing and merging profiles, as well as creation of user accounts. This is a big feature.
- cw_course: Stores configuration relating to courses and adding students to courses. This is a big feature.
- cw_assignment: Stores configuration relating to student assignments and submissions,

- on courses.
- cw_blog: Stores configuration relating to the course blogs.
- cw_faq: Stores (the eventually superfluous) configuration for course FAQ – this functionality was drastically simplified to save time.
- cw_permissions: Stores general permissions settings for the site.
- cw_context: Stores settings for the Context module.
- cw_messages: Stores the settings for automated e-mail messages, kept in a separate feature to allow overriding by client.

These features are not completely independent of eachother, which would be the ideal case, most of them are even cross-dependent – they require each other. They are, however, separated enough to allow them to be enabled one by one.

During the development we regularly fetched new database dumps, installing our features on clean sites. This assured us that we did not have any settings *not* stored outside features. A few settings (such as enabling the *node template* custom page in Page manager) were not possible to export with Features, which made us end up with a few manual deploy instructions (most of which actually were just "enable feature X").

Our project was version controlled using Git.

SOME WORDS ABOUT THE PROJECT FORM

As stated before, it is difficult to overstate the client's active role in this project. Sebastian showed great interest in the project, could attend meetings with short notice, were open to (or rather *embraced*) the flexible planning of agile development, and were in general a nice chap whenever we met.

We have strived to do scrum – or structured agile development – during this project. Our tools have been:

- Sprints of two weeks (minus one day), starting with a sprint planning meeting and ending with a demonstration for the client, followed by a retrospective meeting.
- Actually, the first sprint was divided into two one-week sprints. This turned out to be a great way of getting a shorter feedback loop at the beginning of the project, which was good both for getting started with actual development, getting a first estimate of our development velocity, and getting feedback from the client. We will continue with this habit. (A special thanks to Hans Brattberg at Crisp, who guided us in our agile approach at the first meeting.)
- Hand-written cards with user stories. To begin with these were *huge*, like "Administrators should be able to manage courses", but they were broken down into feasible chunks at the relevant sprint planning meeting. Cards were size estimated as "small" (hours), "medium" (days) or "large" (weeks). At the sprint planning meeting the client would proritize the cards to get an ordered list filling the upcoming sprint, plus a few extra cards in case of unexpected development velocity. (Since every demonstration meeting inevitably lead to a number of really small tasks – such as "allow sorting on school name in table X" – we will in the future shift to a more detailed scale for size estimation. Also, the "medium" category became too wide.)
- After each sprint planning meeting we would write *tasks* to each user story. These were development tasks, written as post-it notes and stuck to the relevant card. Our velocity – and our burndown chart – was based on number of post-its.
- We used a physical wall for moving cards and post-its from "in queue" to "in progress" (or usually our desks), "ready for test" and finally "tested and done". Cards that were demonstrated to the client were stored away in an envelope. (We actually tried using JIRA during one of the sprints, but the physical wall was superior in most ways.)
- We recorded a number of automated tests using Selenium, but they were not really a part of our workflow. We may make this a step on the scrumboard in future projects.
- Instead of e-mailing the client (or ourselves), everything was kept as discussions at our intranet-ish tool. Whenever we had problems knowing how to interpret details in user stories (rare) or encountered unexpected problems and wanted to suggest alternative functionality (more common), we developers talked directly to the client – in text or asking for a quick meeting if necessary.
- We actually ended up scrapping our daily scrum meeting in Team Arnold, exchanging it to a time in the morning when we could tell each other things we felt the others needed to know. The two persons in the team involved in the development communicated all the time anyway, and most days no useful information was conveyed on the daily scrum. (Our team has multiple projects running in parallel – the three other members were only involved in the SSES project when special skills were needed.)

The most useful tools were probably the division of development into sprints – including client demonstration, having a limited number of user stories to focus on per sprint, and the communication with the client.

Du bevakar detta inlägg, klicka för att sluta bevaka

Avpublicera från Drupal Planet

Kommentarer

Skriv ny kommentar

Ditt namn:
Johan Falk

Kommentar: *

Webbadresser och e-postadresser görs automatiskt till länkar.
Lägger till typografiska justeringar.
Tillåtna HTML-taggar: <a>   <cite> <code>   <dl> <dt> <dd>
Rader och stycken bryts automatiskt.
Du kan skriva kod med <code>...</code> (generiska) eller <?php ... ?> (markerade PHP) taggar

Mer information om formateringsmöjligheter

Spara Förhandsvisa

STOCKHOLM KÅKBRINKEN 11A 111 27 STOCKHOLM 08-20 90 04 | GÖTEBORG KUNGSPORTSAVENYN 22 411 36 GÖTEBORG 031-780 02 20
KÖPENHAMN FLÆSKETORVET 68 DK-1711 KØBENHAVN V. +45 50 52 61 41 | © NODEONE