Nike Lab: Technical case study

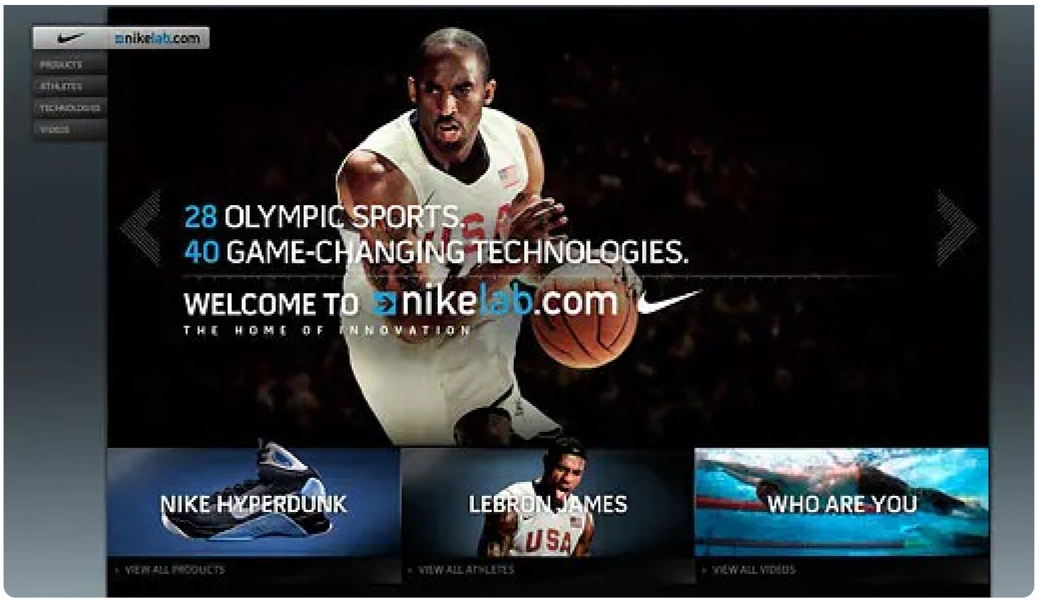
29 March 2008 · 8 mins

Table of Contents

Nike sports marketing came to AKQA San Francisco with a request: Build a site to promote the innovative new products they are cooking in the Nike labs. These products are kind of like what concept cars are to the automotive industry and Nike wanted to have a way to tell it's customers more about the technology behind the scenes. Also with the Beijing Olympics coming up some athletes who are using Nike's new products would also be featured on the site.

Both because my part wasn't so much in the creative/concept part of the project and also because i think most of my readers here are interested in the technical part anyway, this post will focus more on what the gears and cogs under the hood are doing.

Visit the site: http://www.nikelab.com



Requirements into features

Nike had it's technical requirements (the biggest ones being deep linking for flexible marketing and also support for multiple languages) but internally our goals were much higher. Fortunately we had pretty good time to build the site itself + an amazing team so we were able to put our thoughts into actions.

To support all the features we built a new as3 framework as a base and then built the site on top of it. Many (almost all) of these features are in the framework itself and developers at AKQA can easily in the future take advantage of our work in other projects as well, as these features are built in when using the new framework.

Here's a list of some of the features right now:

- Automatic support for deeplinking to any page
- Back button functionality
- Internal sitemap
- Multiple languages/locales, each one with support for localized content (Nike Lab has 23 locales)
- Language switching on the fly without refreshing the page
- Automated support for non western characters
- Everything is driven by xml with a special workflow (more on that later)
- Special support for data models (more on that later)

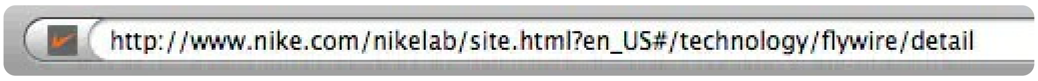
- Keyboard and mouse wheel support
- several more smaller features..

Basically all the technical features are in the framework and all the features for the site are in the numerous swf components for the site. The config file config.xml defines the mapping between these.

Deeplinking and the concept of "pages"

People complain about flash breaking the back button. Well, that's true, but also an animated gif doesn't support going back to the previous frame with the back button. You may say that's a stupid comparison but at the same time a flash site isn't actually much more than a fancy image to the browser. If there was some standardized way for the browser to know what a "page" is inside a flash site it could theoretically support this automatically (Adobe?..). Still, at the time of writing this is not possible so we'll have to build in this support ourselves.

We wanted to be able to define what a page is very easily, so that when you go to that page the framework could automatically navigate through the browser's history. We added support for this through something we call sections. When you navigate from a section to another section a navigation event is automatically created, letting the user navigate back using the back button. Also the address bar and title are obviously updated. Of course the developer can override the browser history if it's not suitable in some case.


http://www.nike.com/nikelab/site.html?en_US#/technology/flywire/detail

Creating a site that supports the back button isn't hard. Frankly I'm guessing that most of the readers of this have done it or at least know what goes into it. Still, you guys are a small percentage of all the developers. A part of our goal was also to spread "the right way", making it so easy to do that there's really no excuse. Unfortunately the code is proprietary so we can't share any of it, but at least AKQA sites in the future hopefully can show the way, inspiring other people to do the same. The key is that even any junior developer can add support for this with studying some documentation and some examples for a day or so.

Since the site has an internal sitemap to keep track of the deeplinking targets the site will know if a deeplink is invalid and it will automatically fall back to the closest match (for instance: `/athlete/cobe-bryant/` will fall back to `/athlete/` since Kobe Bryant's name is misspelled. The `/athlete/` page will show a list of all athletes so that the user can proceed). Also because the site is aware of it's own structure it could easily be used to generate search engine compatible sitemaps + alternate content for easy SEO..

23 custom locales, 1 custom workflow

A requirement from Nike was to have support for an at the moment undefined number of languages. They knew it was going to be many languages but didn't know exactly how many. Because of this we had to come up with an extremely smooth and flexible workflow. One of the biggest things was the huge amount of copy localized into all these languages with several revisions. Updating stuff manually would not only be a huge waste of time but also significantly increase the risk of human errors, not to mention the added effort on QA.

What we did instead was send excel decks to the translation company. They filled in the empty gaps with the localized copy and sent the documents back to us. We then ran proprietary software to generate xml from the excel documents. After this we used an air app specifically built for this purpose by Ronnie Liew to reformat the xml into standardized xliff format. Generating ~220,000 lines of xml took about 1 minute so we could almost instantly see the updated copy in the dev build. This sped up the process a significant amount, letting us focus on the creativity instead. Awesome.

Language switcher? This is 2008, let's do it live

Nike's requirement was to have support for multiple languages. We wanted to go beyond that. What if you send a deeplink to a certain page to a chinese friend whose English is a bit rusty? Now he can just go to the language switcher and swap languages to Chinese. The site automatically knows what data is currently active, reloads that data from the new locale, updates the copy models and updates the site. The process takes 10 seconds or so, depending on the web connection. No need to refresh the page. Works with any combination of course, including the non-western characters. While a lot is going on under the hood, all of this is transparent to the user.

Intelligent loading

This site has a lot of data, there’s no way to escape loading. We can compress assets, combine files etc but in the end there’s only so much we can do. To optimize the experience the site automatically knows what assets and data sources are required to be loaded before navigating to a certain page, meaning they can be loaded in a batch with a single preloader. If the user changes his/her mind and navigates somewhere else instead the site sets a lower priority to the files in the batch so that the files that are required now are loaded first and the other files are loaded in the background. This way the first page is instantly available if the user were to click the back button. I think there’s a lot more that still can be done with this approach but i’ll leave that for another post..

The list of files required is automatically compiled from the site’s structure. Each section can define assets (images, swf’s, fonts etc), data sources (xml files, web services etc) that are required to view the page, in addition to loading the page itself (a swf) with the possibility of having a custom loader.

A new approach to handling shared font is also built into the framework, meaning the font outlines only need to be loaded once and only when required. Automatically of course.

Put that data in the models

While the framework’s main focus is to add complex functionality with ease to the end developer, the framework also endorses use of good structure and design patterns. Loaded data is automatically linked to models that can easily be accessed. It's simply easier to store data in consistent models than in some custom mcgyver-duct-tape-solution. These models can also easily add support for localized data (just adding $region to the path of the file is enough). Custom models are of course supported as each site's needs differ.

The kitchen sink

Keyboard support isn't anything new. Same goes for the mouse wheel. To be honest; I don't think it's a feature, i think it's something missing if it's not there. Super easy support for non conflicting keyboard shortcuts was built into the framework and Nikelab takes advantage of that in many places.

We couldn't have done it alone

A lot of people worked on this project. I would mention everybody but .. to be honest I don't even know. The people involved in the creative development part for Nike Lab (in addition to myself) were Ronnie Liew (technical manager), Tim Robles (developer), Thomas Ko (developer) and Gopi Shubhakar (quality assurance). Thanks also goes to Sallie Lippe for her project management. Other AKQA developers & tech management were also included in the concepting of the framework.

A thing to keep in mind is that this is of course the first site built on the new framework, I'm sure it'll change and improve a lot in the coming year. This is also just a short description of what's actually inside. Some details i can't share, others would just make this post go on for days and days. I'm of course still open to your comments, feel free to take advantage of the box below 😉