bugg.ly

a minimal web app for web developers to track and manage bugs

Technologies

In the development of the prototype buggly app I have primarily used Yii 2.0 and its powerful CRUD generator features for the core of the app.

For the front end I have used HTML5 CSS3 and Sass.

For the back end I have used PHP, and MySQL to create and manage the database behind the web app models.

Yii 2.0 includes bootstrap 3 theming by default. The bootstrap 3 framework uses less as its choice for css pre processing, I however am more accustomed with and value Sass over less so I shall be overwriting some of the default styles using a modular Sass approach to maintain the ability to customise future development versions with ease.

The power of Sass variables really shine through here. For example if I wanted to change the main colour of the web app (which is currently set to a mint green via the variable `$bugg-main: #24b89a;`) to purple, I would simply go into the _variables.scss sass file, find `$bugg-main: #24b89a; and change the colour value to $bugg-main: purple;.`

By changing one value, I have changed the entire look of my web app. Any elements assigned with the $bugg-main variable will now be purple. Variables and other powerful sass features used in the styling of the app makes it incredibly easy for future styling changes by either my self or other designers/developers working on the app as an open source project.

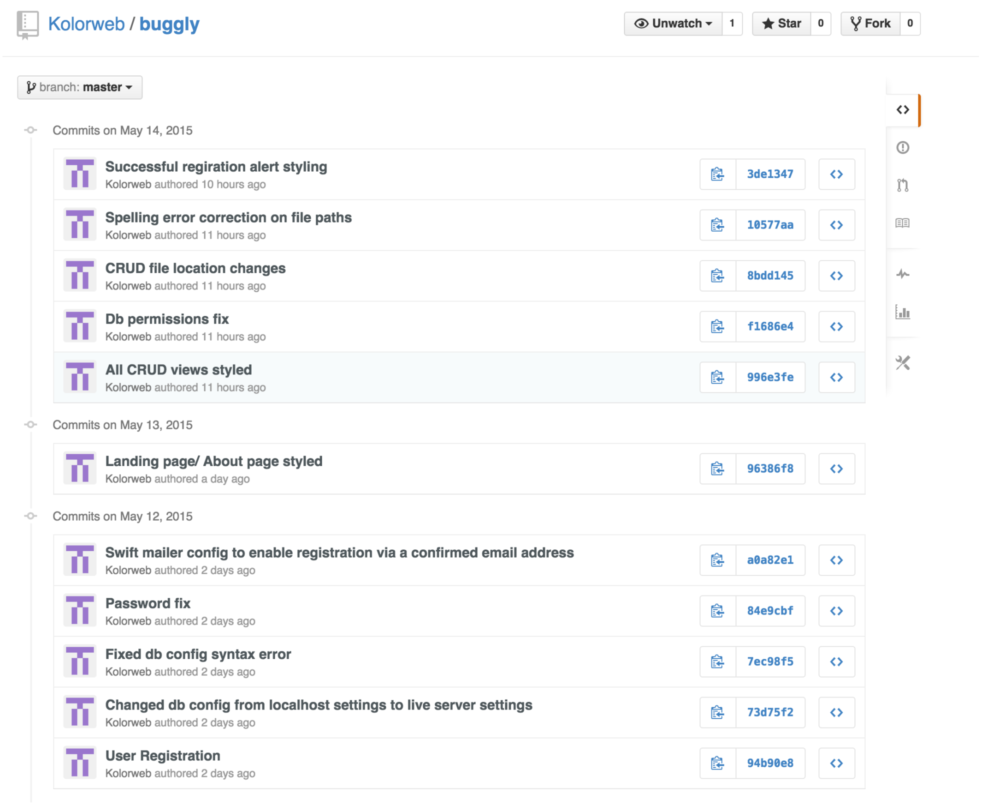
Version Control - Git/Github

git and github.com were used throughout the development of the buggly prototype web app as my preferred method of version control. Adopting this method in my workflow was advantageous in the following ways:

- Allowed me to create automatic backups.
- Keep a detailed track of specific changes as the app evolves.
- Easy to roll back to an earlier version if for example a crucial mistake is made causing the app to break.
- Allowed me to effortlessly transition work between multiple computers and servers, whether it be from desktop to laptop or local server to remote.
- Since the aim of buggly was to be an open source web app, version control using github allows multiple developers/designers to access and make changes within a dynamic environment.

See Figure 1.0 - Commits for a screenshot of example commits made to github during the development stages.

Figure 1.0 - Commits



Kolorweb / buggly

Unwatch 1 | Star 0 | Fork 0

branch: master

Commits on May 14, 2015

Successful regiration alert styling
Kolorweb authored 10 hours ago — 3de1347

Spelling error correction on file paths
Kolorweb authored 11 hours ago — 10577aa

CRUD file location changes
Kolorweb authored 11 hours ago — 8bdd145

Db permissions fix
Kolorweb authored 11 hours ago — f1686e4

All CRUD views styled
Kolorweb authored 11 hours ago — 996e3fe

Commits on May 13, 2015

Landing page/ About page styled
Kolorweb authored a day ago — 96386f8

Commits on May 12, 2015

Swift mailer config to enable registration via a confirmed email address
Kolorweb authored 2 days ago — a0a82e1

Password fix
Kolorweb authored 2 days ago — 84e9cbf

Fixed db config syntax error
Kolorweb authored 2 days ago — 7ec98f5

Changed db config from localhost settings to live server settings
Kolorweb authored 2 days ago — 73d75f2

User Registration
Kolorweb authored 2 days ago — 94b90e8

Development environment and tools

Developing locally

Tools used:

MAMP - To enable a local server type environment.

CodeKit -Live preview(A time saver) Automatically refreshes the browser after any change is saved.
Compiles Sass to Css.
Compacts files for production versions.

Debugging:

Chrome (Web developer tools)
Firefox (Firebug)
SublimeText 3 (Linter)

Deployment Method - Github & dploy.io

I have chosen to use dploy.io for my web app deployment method. dploy.io allows you to link up with your github repository (in this case my buggly web app) and connect it to a server type of your choosing from a variety of different deploy methods e.g. FTP/SSH.

dploy.io automatically recognises any changes to the linked repository and keeps a tracked record of commits, just as Github does. You can then manually deploy a commit of your choosing to the server.

The service is extremely powerful and versatile, if I happened to change my server or domain name for example, all I would have to do is log in to my deploy.io account, change the server settings to match and re-deploy my app, or if a certain change happened to break the app, I could simply roll back to the last deployment as if the change never took place!

Server requirements

dploy.io although a fantastic tool, cannot do everything. The server itself needs to be adequately equipped to handle the app that is being deployed.

In the case of using Yii 2.0 as per the YiiRequirementChecker this meant requiring at minimum:

- PHP 5.4.0

Enhancing Performance

Code Optimisation

Since I want my web app to be scalable and easy to maintain I have written my styles using a modular approach.

If I was to do this via css and have an individual css style sheet for every page/partial this would greatly increase HTTP requests potentially slowing the time it takes to render the site/app. However by using sass and the power of imports I can compile all of the individual sass files into one large css file resulting in only one HTTP request compared to multiple requests, increasing performance tenfold.

Minified Code

Another way to increase performance is to minify any javascript and css files for production only. Although it is recommended to retain a human readable un-minified version for further development purposes. There are numerous tools both online and off available to output minified code, I like to stick to my trusty CodeKit which is also great for compacting and minifying relevant files ready for production deployment.

Enhancing SEO

With some core SEO best practises put into practise you increase the chance of being a contender for search engine visibility. Examples to put into practise below:

Good, relevant Keywords in “key” places such as the following:

- URL’s
- Title tag
- Web slogan/description
- Navigation
- Breadcrumb trails
- H1, H2 and H3 tags
- Internal links
- Image alt tags

URL Rewriting/Clean up URLs

In Yii 2.0 before modifying and applying prettyURL to our URLs we would get some very messy URLs e.g. /index.php?r=project/show&id=1&title=A+Test +Bug which could negatively affect the sites SEO.

Clear Website Name (Contains keywords for best results)

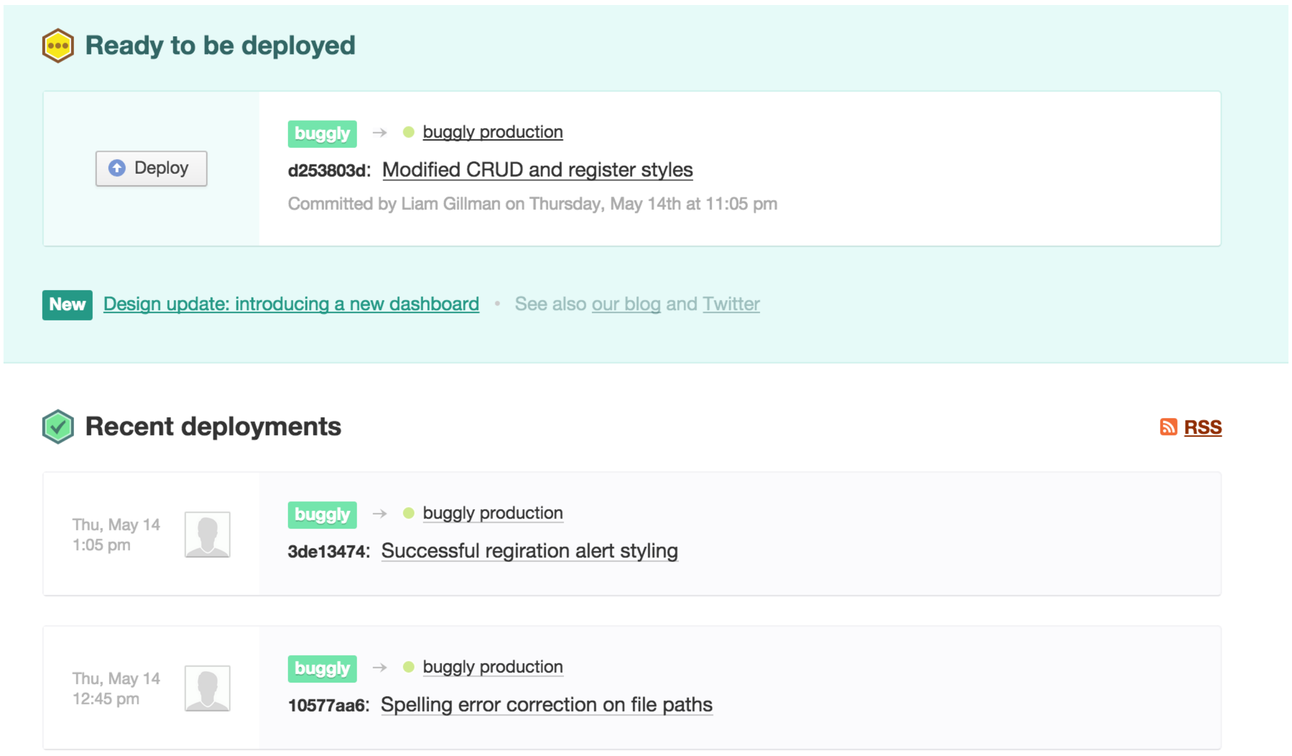
Increasing speed/performance

Enhancing performance (previously mentioned above) can also have positive affects on a sites SEO. In a fast paced, modern, mobile device world, people are not only used to information on the go, but they expect it fast. Slow sites can and will deter users from sticking around, again negatively impacting a sites SEO.

Mobile Friendly sites

As mentioned above, we are in a world where mobile devices rule! It is not uncommon for vast amounts of a sites user base to come from mobile devices. If your site breaks, loads to slow, displays 404 errors on mobile you are negatively impacting on the sites SEO. It is therefore paramount that the site be mobile accessible. (It is common practise in a modern web workflow to actually design mobile first!)

Figure 1.1 Screen shot of dploy.io recognising a commit change ready to be deployed.



Ready to be deployed

Deploy

buggly → buggly production

d253803d: Modified CRUD and register styles

Committed by Liam Gillman on Thursday, May 14th at 11:05 pm

New Design update: introducing a new dashboard • See also our blog and Twitter

Recent deployments

RSS

Thu, May 14
1:05 pm

buggly → buggly production

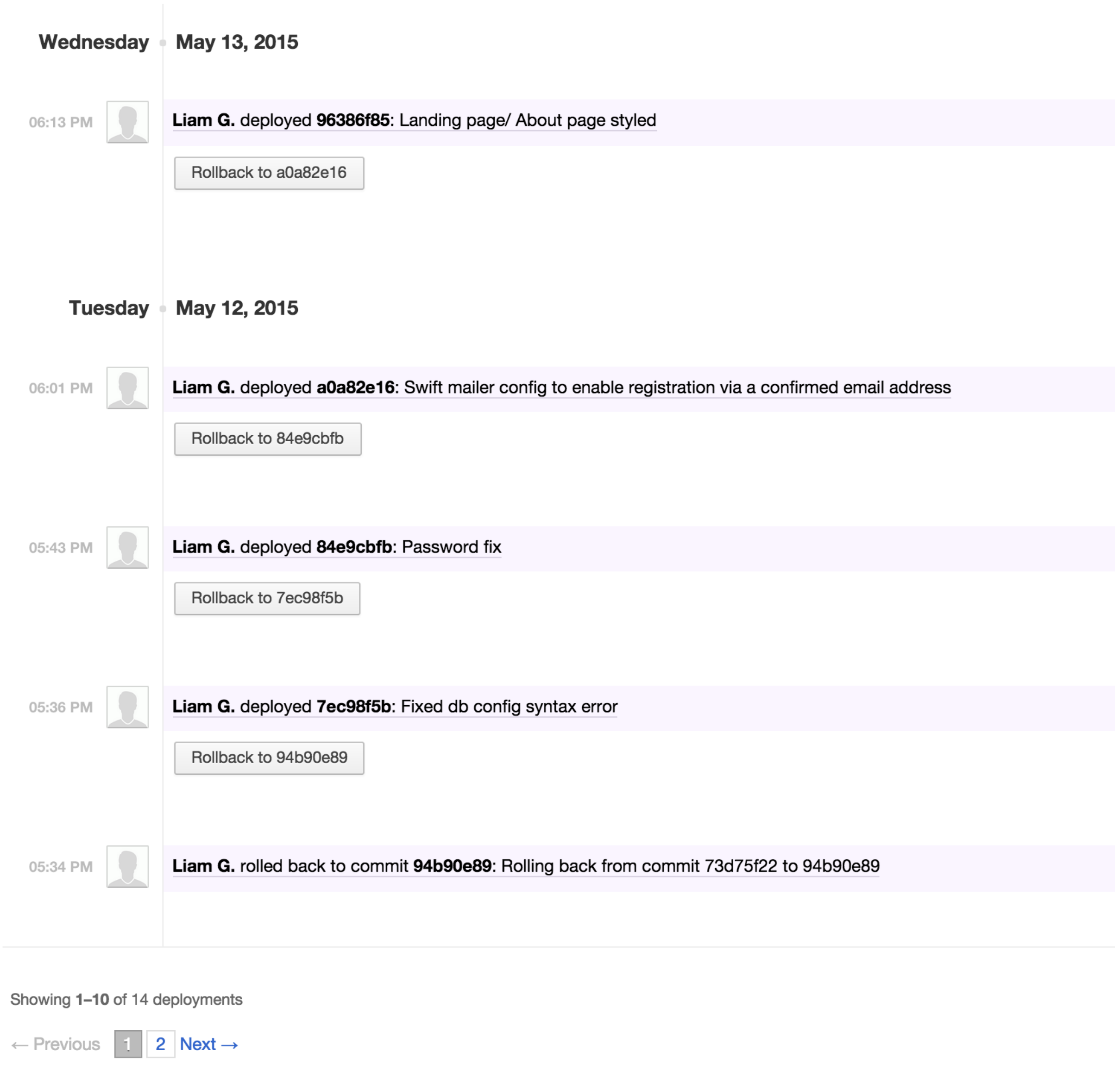
3de13474: Successful regiration alert styling

Thu, May 14
12:45 pm

buggly → buggly production

10577aa6: Spelling error correction on file paths

Figure 1.2 Screen shot of successful deployed commits.



Wednesday May 13, 2015

06:13 PM **Liam G.** deployed **96386f85**: Landing page/ About page styled

Rollback to a0a82e16

Tuesday May 12, 2015

06:01 PM **Liam G.** deployed **a0a82e16**: Swift mailer config to enable registration via a confirmed email address

Rollback to 84e9cbfb

05:43 PM **Liam G.** deployed **84e9cbfb**: Password fix

Rollback to 7ec98f5b

05:36 PM **Liam G.** deployed **7ec98f5b**: Fixed db config syntax error

Rollback to 94b90e89

05:34 PM **Liam G.** rolled back to commit **94b90e89**: Rolling back from commit 73d75f22 to 94b90e89

Showing **1–10** of 14 deployments

← Previous 1 2 Next →

Please see eWurf for more information regarding functional requirements.

Functional requirements:

- allow user to login with their email address.
- allow the user to create a new project.
- allow the user to assign other users to the project.
- allow a user to create a new bug.
- allow the user to view a bug.
- allow the user to edit a bug.
- allow the user to delete a bug.

Non functional requirements:

- Maintainability. - maintainable, modular code
- Accessibility - cross browser, multiple device compatible
- Documentation - Guide on how to use the web app.
- Open Source, Free to use
- Usability/ User friendliness / UX design
- Appropriate styling (Colours, Font Choice)

Reflective

Design/Development methodology - Rapid prototyping to Evolutionary prototyping

Rapid prototyping allowed me to quickly generate ideas/designs and quick functional mockups based on predetermined requirements (functional and non functional requirements) to see what worked and what did not. This in theory saves a lot of time, avoiding attention to any great deals and allows the exploration and testing of multiple solutions.

Once I had found a design/prototype that I felt was close or had the greatest potential to meet all the requirements it was onto the evolutionary prototyping stages.

Evolutionary prototyping allowed me to take that prototype and keep building/ improving upon it, a workflow which is very popular within the web app field as it allows the quick launch of a working system that can be tested by users and constantly improved upon via feedback through progressive development versions.

Being a designer first, developer second I took on a design first approach to the buggly project which would prove to have some major downfalls later on in the project which I shall discuss further down.

Design

Research
I began the project by looking at existing apps created using the Yii framework to get a general feel for generic design and any best practises.

Plan
I took my list of initial functional and non functional requirements to determine the areas I would need to consider, as well as brainstorming ideas for a landing page and miscellaneous supporting graphical elements.

Sketch
Any initial ideas were first sketched out on paper, the first step of my rapid prototyping approach. Ideas that were deemed not suitable for the project were scrapped and the chosen designs taken forward to the next stage.

Low fidelity wire framing
Chosen sketches were turned into low fidelity wireframes to get a better feel for spacing and page content.

High Fidelity wire framing
This would be the final stage in the rapid prototyping methodology. The high fidelity wireframe would become the basis of the evolutionary prototype.

The problem choosing design first over functionality

At this stage, rather than focusing on getting the core functionality of the app in working order I deviated towards the design side of things, spending way to much time on theming and layout. This is certainly something that can and should have been assigned to the later stages of development as a web app that is beautifully designed but non functional is not truly a web app!

Becoming so focused on the design, ultimately hurt the overall functionality of the prototype with several functional requirements not being fully met.

Rather than focusing on fixing, for example an action button that was not precisely centred (maybe 100px off!) that time spent should have been put into implementing the working functionality behind thus intended button. After all, a button that is visible, regardless of being off 100px is still clickable!

Lessons Learnt

The project has made it painfully obvious that I need to work on prioritising key areas within a given project. I do however feel the development methodology I chose to adopt was certainly the correct choice for this type of project, but rather than focus on evolving the design, I should have adapted sooner and put more focus, if not all focus into evolving the functionality.