Don't Be a Validation Nazi

In my last post, I wrote about how most email validation routines are too strict when compared against what is allowed by the RFC. Initially I dismissed this phenomena as the result of ignorance of the RFC or inability to understand it, as I had trouble understanding it myself.

However, I think there's something more fundamental at work here when it comes to validating user data. It seems that many developers, myself included, choose to ignore Postel's Law when it comes to field validation. Postel's law states...

> Be conservative in what you do; be liberal in what you accept from others.

Postel wrote that in an RFC that defined TCP, but it applies much more broadly. It's natural that developers, used to the exacting nature of writing code for a compiler, where even the most minor of typos can bring a program screeching to a halt, have a tendency to apply such exactitude on their users.

Dare I say it, but developers can tend to be validation nazis.

User: (filling out form) user+nospam@example.com

Validation Nazi: Entering a plus sign is $2.00 extra.

User: But the RFC allows for a plus sign.

Soup Nazi: You want plus sign?

User: Yes please.

> **Validation Nazi:** $3.00!
>
> **User:** What?
>
> **Validation Nazi:** No form submission for you!

This is a mistake. Users are not compilers so we need to cut them some slack.

A List Apart provides some great examples of mistakes in treating users like computers and ways to correct them in the article, Sensible Forms: A Form Usability Checklist. Here's a snippet about dealing with phone numbers (emphasis mine).

> **Let the computer, not the user, handle information formatting**
>
> Few things confuse users as often as requiring that users provide information in a specific format. Format requirements for information like telephone number fields are particularly common. There are many ways these numbers can be represented:
>
> * (800) 555-1212
> * 800-555-1212
> * 800.555.1212
> * 800 555 1212
>
> Ultimately, the format we likely need is the one that only contains numbers:
>
> * 8005551212
>
> There are three ways to handle this. The first method tells the user that a specific format of input is required and returns them to the form with an error message if they fail to heed this instruction.
>
> The second method is to split the telephone number input into three fields. This method presents the user with two possible striking usability hurdles to overcome. First, the user might try to type the numbers in all at once and get stuck because they've just typed their entire telephone number into a box which only accepts three digits. The "ingenious" solution to this problem was to use JavaScript to automatically shift the focus to the next field when the digit limit is achieved. Has anyone else here made a typo in one of these types of forms and gone through the ridiculous process of trying to return focus to what Javascript sees as a completed field? Raise your hands; don't be shy! Yes, I see all of you.
>
> Be reasonable; **are we so afraid of regular expressions that we can't strip extraneous characters from a single input field? Let the users type their telephone numbers in whatever they please.** We can use a little quick programming to filter out what we don't need.

The recommendation they give fits with Postel's law by being liberal in what they accept from the user. The computer is really really good at text processing and cleaning up such data, so why not leverage that fast computation, rather than throwing a minor annoyance at your users. No matter how small the annoyance, every little mental annoyance begins to add up. As Jacob Nielsen writes (emphasis his)...

> **Annoyances matter, because they compound.** If the offending state-field drop-down were a site's only usability violation, I'd happily award the site a gold star for great design. But sites invariably have a multitude of other annoyances, each of which delays users, causes small errors, or results in other unpleasant experiences.
>
> A site that has many user-experience annoyances:
>
> - appears **sloppy** and unprofessional,

- demands **more user time** to complete tasks than competing sites that are less annoying, and
- feels somewhat **jarring** and unpleasant to use, because each annoyance disrupts the user's flow.

> Even if no single annoyance stops users in their tracks or makes them leave the site, the combined negative impact of the annoyances will make users feel less satisfied. Next time they have business to conduct, users are more likely to go to other sites that make them feel better.

However, in the case of the email validation, the problem is much worse. It violates the golden rule of field validation (*I'm not sure if there is a golden rule already, but there is now*)...

> **Never ever reject user input when it truly is valid.**

In the comments of my last post, several users lamented the fact that they can't use a clever GMail hack for their email address because most sites (mine included at the time, though I've since fixed it) reject the email.

> With Gmail you can append a tag to your email address. So let's say you have "name@gmail.com" you can give someone an email address of "name++sometag@gmail.com" and it will faithfully arrive in your inbox. The use of this for me is that I can track who's selling my email address or at least who I gave my email to that is now abusing it.

For fun, I wrote a crazy regular expression to attempt to validate an email address correctly according to the RFC, but in the end, this was a case of Regex abuse, not Regex use. But as one commenter pointed out...

> THE ONLY WAY TO VALIDATE AN EMAIL ADDRESS IS TO DELIVER A MESSAGE TO IT!

This served to drive home the point that attempting to strictly validate an email address on the client is pointless. The type of validation you do should really depend on the importance of that email address.

For example, when leaving a comment on my form, entering an email address is optional. It's never displayed, but it allows me to contact you directly if I have a personal response and it also causes your Gravatar to be displayed if you have one. For something like this, I stick with a really really simple email address validation purely for the purpose of avoiding typos...

```
^.+?@.+$
```

However, for a site that requires registration (such as a banking site), having the correct email address to reach the user is critical. In that case it might make sense to have the user enter the email twice (to help avoid typos, though most users simply copy and paste so the efficacy of this is questionable) and then follow up with a verification email.

In the end, developers need to loosen up and let users be somewhat liberal about what they enter in a form. It takes more code to clean that up, but that code only needs to be written once, as compared to the many many users who have to suffer through stringent form requirements.
