(This is a mock proposal)

(Written from the point of view of a student who has done their research, perhaps asking around in IRC and/or sifting through the spec/code. The actual project doesn't need to match this, though hopefully this proposal will be useful to the student who will finally be working on this, if any)

Personal Details

- ...

Project Proposal

I wish to work on improving form support in Servo. I shall be implementing the following features:

- Form validation (`minlength`/`maxlength`/`size`/`required`/`pattern`/`min`/`max`)
- Form controls:
    - `select, option, optgroup`: These can be implemented as tables, to bring us limited functionality.
    - `label`: This requires some activation hooks, which can be added by implementing `Activatable` similar to htmlinputelement.rs
    - Improving existing ones: I could create better-looking widgets for radio buttons, buttons, and checkboxes via CSS
    - `datalist`: If I get time, I can try to add table-based datalist support too. The UI code might also be useful for providing limited autocompletion support
- File uploads:
    - Support basic `Blob` and `File` interfaces that are able to contain data. These can then be passed around internally; there is no need for advanced file reading . File names and an internal API for accessing the data is enough.
    - If there is time, support `FileReader`.

        - Add support for proper `multipart/form-data` serialization, implementing RFC 2388. I may only focus on UTF8 encoding and expand to other encodings if there is time.
        - Support a basic File upload widget; which looks like a text input. Dialog support is not a part of this project. Improve the submission algorithm in `htmlformelement.rs` to support this.
    - Improved editing for text controls:
        - Basic keyboard shortcuts: Ctrl-A, Ctrl-V, Ctrl-C.
        - Caret support. Initially this can just be a phantom pipe char in the text field, if there is time we can try to integrate this better into layout.
        - If there is time, support for moving the caret by click
        - If there is time, advanced selection support.
        - *I wonder how this will interact with bidi and the rest. Though that's way out of scope for gsoc.*
    - Tests: Add tests to web-platform-tests wherever missing
        - *We'll probably need to figure out a way to properly test stuff like activation; this might also be out of scope of gsoc since it can't be done with regular javascript.*

It would probably be nice to get spec-complete form owner support as well in this project. However this isn't something that would make sense in the proposal (too obscure/technical — a student won't be aware of the issue whilst writing the proposal), but I think it would be a fun side-adventure for the student halfway through the project. It involves discussing the spec with Hixie and figuring out how h5e handles the "in a form" state, and then figuring out a way to track form owners efficiently (the spec seems to want us to update form owners on every DOM mutation).

Schedule of Deliverables

The coding period is around 12-13 weeks. Since most tests are already written, most weeks will involve testing code written that week even if not explicitly mentioned.

- Week 1:
    - Work on reading the HTML Forms spec, and comparing it with existing code.

    - Read through `activation.rs`, `htmlinputelement.rs`, `htmlformelement.rs`, `textinput.rs`.
- Week 2:
    - Start work on form validation. Add the relevant attributes to the webidl for `<form>`, `<input>`, and the other widgets. Add a basic constraint validation method that validates a select few of the constraints. Hook it up to `checkValidity()`.
        - *I feel that this is a good thing to start with since it gives a nice introduction to introducing new DOM attributes and manipulating them.*
    - Fill in the implementation of `Blob` and `File` with stubs for the attributes we need.
        - *The constructor needs `sequence<>` webidl support which we don't have right now IIRC, but I think we can add it by gsoc. It's not really necessary for this project either way.*
            - *Not sure about it being easy; the correct implementation uses for..of iteration through a C++ class.*
    - Finish reading the forms spec.
- Week 3:
    - Add more constraints to the validation algorithm.
    - Add the `ValidityState` interface and support some of the attributes (not necessarily the methods)
    - Add simple interactive reporting of errors, perhaps by scrolling the user to the element.
    - Read through RFC 2388 and get a rough idea of how form data serialization is to be done
- Week 4:
    - Add support for `Blob/File` to contain data or have a reference to the file.
        - *We are allowed to skip snapshotting the file, though we probably should maintain a snapshot depending on what other browsers do*
    - Start looking into how `select/option` can be styled.
    - Write basic `<select>`/`<option>` webidls, copying/sharing most of the method implementations from `<input>` wherever possible.
    - Work more on validity if there is time
- Week 5:
    - Write a proper `multipart/form-data` serialization algorithm.
    - Add support for `<label>`, along with the activation hooks.
    - If there is time, try adding a table-based widget for `<select>`/`<option>`

- Week 6:
  - Finish widgets for `<select>`/`<option>`
    - *Basically, layout will be asked to treat these as tables of a certain kind. The student need not add dropdown support, a simple table showing all the options is enough (preferably with a scrollbar, if that's doable in CSS).*
  - Create some better CSS for existing widgets, and start using them.
    - *The challenge here is in writing these in CSS that we support, and still having them look decent. Hopefully we'll have better CSS support by this time anyway; though looking at the styled buttons on GitHub in Servo we already support enough CSS for this.*
- Week 7:
  - Implement the form submission algorithm for `multipart/form-data` . Hook it up to regular form submission.
    - *The web isn't supposed to be able to mess with form uploads so testing will be hard. Since we're just using a text input, we can temporarily write content tests for this until we figure out how to properly test things that are hidden to Javascript.*
      - *A*side: *if we* are *exposing form uploads to javascript, perhaps we should be marking this as a testing-only feature or something so that this doesn't compromise the security of dogfooders and users of the "mobile Wikipedia browser" if we get to making that happen.*
  - Start looking into how to add caret support
- Week 8:
  - Finish up remaining bits of validation, if any
  - Learn how to hook into the clipboard for copy/paste functionality
  - Polish up existing selection support if necessary (internals, not UI)
  - Add support for Ctrl-A
- Week 9:
  - Add support for Ctrl-C, Ctrl-V
    - *Might be hard to hook into the clipboard cross-platform, so this might take longer*
  - Add caret support as a simple rendered pipe, or as something more sophisticated if possible in the timeframe.
- Week 10:
    - Look into adding the ability to set caret position based on click
    - Look into adding styling for `<optgroup>`

        - Fill in more parts of the File spec
- Week 11:
    - Look into support for visible selections
    - Work on fixing any web-platform-test failures related to my code
- Week 12:
    - Polish up and document remaining code
    - Continue to work on tests

Open Source Development Experience

None

Work/Internship Experience

None

Academic Experience

I have a bachelor's degree in multiplication by 3.

Why Me

I am awesome.

Why Mozilla

Cake.