WordPress Considerations

General Notes

I don't know if we have a list somewhere, but it might help with mapping out where everything should go across our WordPress instances if we categorize the types of sites we support. This is what I could come up with, but I'm sure there's more:

- SPLOT style submission sites
- Portfolios
- Services
- Events
- Galleries
- Knowledge bases
- Personal blogs

I think we need to look at doing something like yearly dead site audits so we can clear out problems before they compound.

- For OL courses we might want to track in d4p2 or somewhere else when we're sunsetting a site on course revision.
- Archive/delete "dev" sites once a site is launched - title them with dev so it's easy for others to notice them and delete them at the end of the year if a project is signed off and the project lead forgot to do it themselves.
- Come up with naming conventions for specialty site types like templates and dev sites so it's obvious when we do audits what these sites are for. I.e.: sitename-dev sitename-template

If Troy's concern about allowing users access to one plugin or another are an issue (for example the JavaScript plugin for front end I requested) we may have to divide the WordPress instances by of sites? Or we have documentation noting that we can't turn on certain limited access plugins on sites we're going to hand off to users we don't know well.

As you mentioned the other day, we should change both the wp-admin and wp-login urls

Working with Themes and Theme Management

For security, stability, and support reasons we should really minimize the number of themes we provide. It made sense before the block editor to need a lot of themes, but now there's a lot we can do with modern themes to meet unique needs.
I'd suggest keeping a few grab and go themes that are well known to the LTI team as we make this transition and then start looking at building capacity for the following:

Kadence theme with Kadence blocks

I'm not sure if this theme or it's associated blocks are what keep causing issues for me, but the 2 work best paired together so I can't advise one without the other.

Who is this for:

- Fast and simple build requests
- Student sites since it's to pick up and build.

Things we need to support this:

- Documentation on how to use and customize

Issues:

- I'm still running into issues where these blocks crash even on my own projects which can be frustrating for users. It's my hope that block themes get easier with time so we can step away from the headaches this combo causes when things go wrong.

Block themes with vanilla blocks

With these, it doesn't matter which theme they are attached to since the templates transfer.

Who is this for:

- Projects with more time
- Projects that need more long-term stability
- Projects with more parts that could cause things like Kadence to crash
- More ambitious students

Things we need to support this:

- Reusable pattern library
- Reusable page templates
- Reusable CSS files
- Documentation on how to use
- Showcase with downloadable files of each of these?

Notes:

- These themes do not need child themes to make them visually customized. The various parts are importable and exportable from the dashboard as json files.

Headless

Who is this for:

- For complex requests solely handled by developers
- Possibly for requests where users only have access to content not layout - though it might be too costly to be worth it for this case.

Requirements

- Server for the static part of the site - Barabus? Or something else on Reclaim?
- Templates to be built over time to build upon over time and stored somewhere accessible by developers.

Plug-in management

Before installing a plugin, we should probably ask the following:

- Do we anticipate this request coming from many users?
- Is it mission critical?
- Can it be done with toolset?
- Can it be done with JavaScript?
- Would it save significant time NOT building it with toolset or JavaScript because of high demands of requests for it?

Obviously, we can't rely on the toolset and JS solutions for student run sites, but I think we need to scale back on niche plugins for the following reasons:

- It's a lot to scroll through.
- It's a lot for everyone to support when an end user has an issue with one.
- It's a lot to do an audit on when it's time to sunset them.

SPLOTs

Levels of difficulty to develop:

Toolset Builds

- Easy to customize
- 2 levels of form difficulty: drag and drop form vs html form

Child Theme Builds

- WYSIWYG – not much customization unless we make a duplicate child theme and edit the code.

Headless

- WYSIWYG – but potentially good for really complex requests

Existing SPLOTs (A lot should be updated to modern themes when possible):

- Writer - https://splotwriter.trubox.ca
- Collector - https://splotcollector.trubox.ca
- Editable generic collection - https://toolset-submitter.trubox.ca
- Basic Map (editable) - https://toolsetmap.trubox.ca
- 3D submissions - https://3d-gallery.trubox.ca
- Glossary – https://unsdg6.trubox.ca
    - (I need to strip out the UNSDG parts and change the theme and URL so it's generic)
- Video one Jamie uses – not sure the address
- Toolset knowledge base tool is one I've been considering making as well

Headless WP use case proposal

There are many arguments for and against this approach to site building, mine doesn't quite fall under any of those. What I need is the option to make custom builds that utilize a database for storing submitted data or retrieving large quantities of data without having to spin up that whole backend set up all the time since server-side programming and databases aren't an area of strength for me especially when security is involved. WordPress and Gravity Forms might be able to lighten that load. I believe I can host the static part of the site on our barabus server with this set up, but I'm not sure.

A few benefits:

- Utilizing Gravity Forms for this adds a bit of protection by handling some of the back-end data validation. I think that's more secure than trusting me to make form handlers from complete scratch.
- Allows me to use modern JavaScript frameworks without having to jump through too many hoops or having to navigate around child theme server-side code.

A few questions and concerns:

- I'm not sure how difficult this kind of set up is compared to just forcing things in child themes, or if the cost is more of a one-time thing, and I can reuse some templated code I have saved somewhere to start off future projects.
- I'm not sure how insecure it makes things to separate from the font end Gravity Forms validation part. With headless WordPress I will be setting up the form fields with html.
- I'm generally not clear on security concerns with this format overall.

Plugins I believe I would need to ensure I can build most things with this approach (some I might not actually need; this is based on rough research):

- WpGraphQL and the following Extensions (https://www.wpgraphql.com/extensions)
    - WPGraphQL Cors
    - WPGraphQL for Gravity Forms
    - WPGraphQL Enable All Post Types
    - WPGraphQL for Advanced Custom Field
- Gravity forms
- Advanced custom fields pro