Stack for Portal Gatsby

Still need to distribute priorities and how to approach the architecture, schemas and site building, but this is basically the (probably incomplete) set of core features and technologies to be used on the site

---

- Frontend/Static site generator:
  - Gatsby

- Backend/Headless CMS:
  - Strapi CMS

- CSS Preprocessor:
  - Stylus

- Frontend Framework:
  - Bulma/Bulma-stylus

- CI/CD:
  - Webhooks (For static compiling and user facing updates)
  - Drone (Pipeline for full site compiling including GraphQL changes, schemas and modifications to the site)

- Integrations:
  - Git/GitHub (Make a repo available for user facing changes; i.e. markdown files, database changes, etc)
  - Prometheus/Grafana (Make proper monitoring of changes to the site, as well as a health endpoint)

- Backend considerations:
  - Possibly will have to write a translating service between the database used by strapi and the markdown or whatever format used to sync on the git repo. Will have to check if a flat file system is viable/functional or if it's better to write the extra translation layer
  - Write integrations to mattermost and kanboard or whatever PM soft we're using, to relay notices of changes, updates and activity to the team
  - The old site data structure is a total mess, we'll have to scrape that and design a better structure from the outset
  - We'll have to see how to integrate D3/Charts.js to strapi so data can be presented adequately without needing to meddle with external stuff, or to add interactiviy
  - Inside strapi design user roles and powers to restrict the changes that can be made by the editors (i.e. maybe letting them change a GraphQL schema is a bit too much and can lead to things breaking)
  - If possible enable precompilation of MathJax --> svg to be able to include formulae and data on the site with the proper displaying. This may be important for future analysis, surveys or whatever technical data we have to share.
  - In the same department as MathJax, get a proper way to display data tables and process source information automatically from custom

endpoints using D3 and maybe Chart.js, so all the source data is accesible, eases the reproducibility and can be consumed either raw or with a minimum displaying to make it easier.

- Frontend considerations:
    - The integration of D3/Chart.js will require quite a bit of work on the frontend to make sure React/Gatsby compiles and displays stuff properly
    - The (little) custom js on the site will have to be scraped and added as a hook or simple function call to the React frontend structure
    - There's a lot going on so we'll have to see what packages can we use to minify and reduce file size and amount of calls made to the server (that are probably not much because it will be served from a single source)

- General considerations:
    - Check to see if we can make deployments on a S3/DO Spaces instance and serve the site over there with the needed dynamic stuff being read from the DO server.
    - Using S3 check to enable cloudflare edge cache and other goodies