Notes on web architecture:

Bucket list of stuff:

* Redis
* JavaScript frameworks
  + MobX
  + GraphQL
  + Css-in-js
    - Emotion
    - Styled Components (this is the most popular by far)
    - JSS

# Storing files on the file system

## Flickr

A blog about how flickr stores files on the host OS’s file-system:

<http://highscalability.com/flickr-architecture>

If we use Windows as a hostos, we need to look at performance considerations as the number of files scales up past 100K.

## Third-party services

There’s a site called UploadCare that offers this service:

<https://uploadcare.com/upload_api_cloud_storage_and_cdn/>

Also, we could probably look into partnering with Dropbox / Google Drive / OneDrive.

The only concern is to make it transparent to the user. We just need a way to upload stuff and keep it secure, and give the user control over everything.

# Scalable multi-tenant Database Architecture

* Postgres SQL: <https://www.citusdata.com/blog/2016/10/03/designing-your-saas-database-for-high-scalability/>
  + Two recommended approaches:
    - One DB to rule them all, and you shard the DB across servers
      * Best if you want to scale up to 1000s of tenants.
    - One DB per tenant – best if
      * you want your customers to remain isolated
      * You have 5 to 50 tennants
* Azure: <https://docs.microsoft.com/en-us/azure/sql-database/saas-tenancy-app-design-patterns>

# Managing your repository:

Use a well-known branching strategy, like GitFlow:

<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

# Front-end stuff

Web architecture to research:

1. React
   1. Routing
   2. Hosting, and HTTPS
      1. Heroku?

# Things to optimize your website

Google’s PageSpeed insights: <https://developers.google.com/speed/pagespeed/insights/>

**What to do / not to do in a web startup**

Images

* (idea) have a dedicated server for customer images, with caching and the whole 9 yards.
* Image optimization:
  + Optimizilla: <https://imagecompressor.com/>
  + FileOptimizer
  + WebPack also has a plugin for this
* Gzip
  + Look for gzip packages for whatever framework/server you’re using
  + Node.js has an amazing compression middleware, called expressjs / compression
    - https://github.com/expressjs/compression
* Minification
  + WebPack w/ uglify.js
* Turn on cache-controlling
  + Set for a fixed period of time, for things you don’t expect to change that often
* CDN
* How do you manage data in a cloud-based application as it gets bigger, and bigger, and bigger…
* Lazy-loading for modules
* OWASP security
* PCI compliance
* **Progressive Web Applications**
* Elasticsearch
* Javascript unit testing
  + Recommended practices for
    - React
    - Redux
* Localization + React

# Fast loading:

1. General overview:
   1. <https://mobile1st.com/7-ways-to-improve-mobile-page-speed/>
   2. <https://medium.com/swlh/6-super-effective-ways-to-optimize-your-mobile-conversion-rates-d3c74da6cad3>
   3. <https://neilpatel.com/blog/how-to-optimize-your-ecommerce-website-for-mobile-devices/>
2. General React performance: <https://medium.com/myheritage-engineering/how-to-greatly-improve-your-react-app-performance-e70f7cbbb5f6>
3. JavaScript Start-up Optimization: <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/javascript-startup-optimization/>
4. Tree Shaking: <https://developers.google.com/web/fundamentals/performance/optimizing-javascript/tree-shaking/>
5. Code Splitting: <https://developers.google.com/web/fundamentals/performance/optimizing-javascript/code-splitting/>
   1. Specifically route-based code splitting
      1. use WebPack’s bundling ability
      2. Find a router that works with React that allows code-splitting. When you go to a route, the router automagically loads the bundle.
      3. This kind of hides the fact that you’ve got separate bundles that are loaded on demand. Makes everything transparent.
6. Time to Interactive: <https://developers.google.com/web/tools/lighthouse/audits/time-to-interactive>