React Best Practices

3 – Projekte strukturieren

Timo Mämecke
TH Köln // MI Master // Weaving the Web
25. Juni 2019

Inhalt

Ordnerstrukturen.

Dependency Management.

Storybook.

Ordnerstrukturen

Was Facebook empfiehlt:

Don't overthink it

If you're just starting a project, don't spend more than five minutes on choosing a file structure. [...] You'll likely want to rethink it anyway after you've written some real code.

* Siehe: https://reactjs.org/docs/faq-structure.html

Group by feature

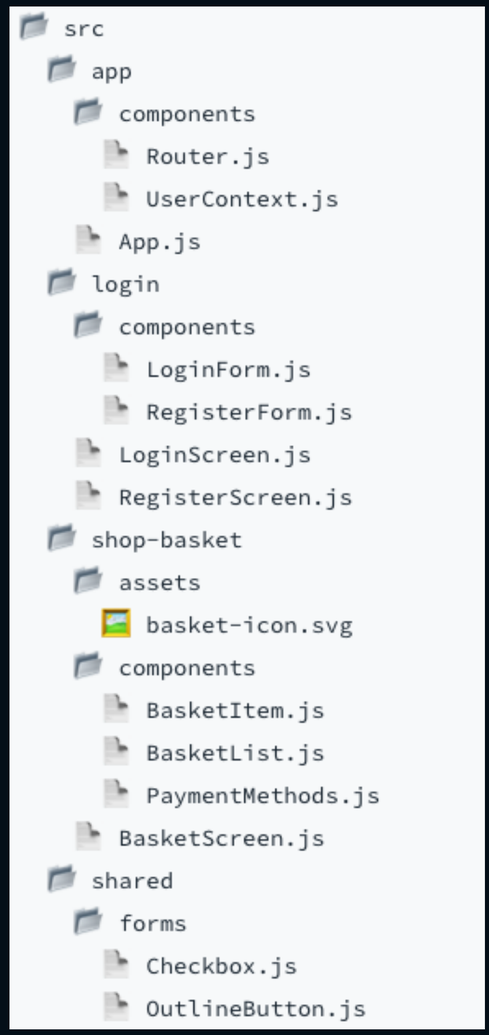
Ein Ordner je Feature
z.B. für jede Route oder für komplexere Teile eines Features

Alle Dateien beisammen, oder noch **ein** Unterordner

Meist shared Components als "feature"

"app feature" als Basis der App (Router, etc)

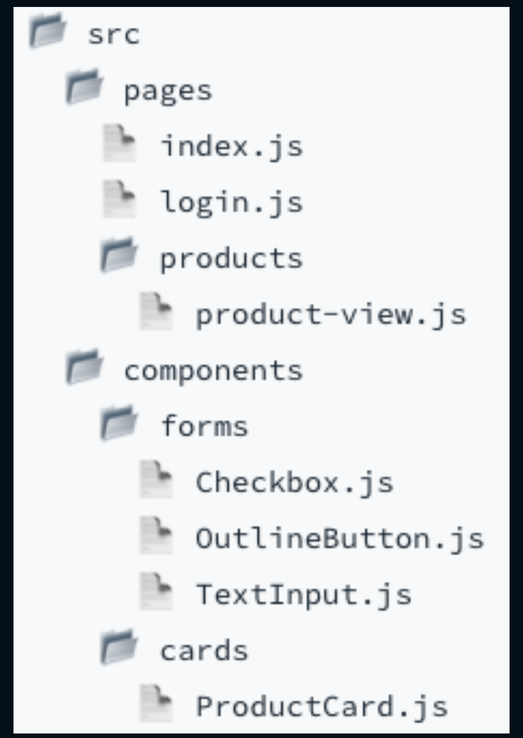
```
common/
  Avatar.js
  Avatar.css
  APIUtils.js
  APIUtils.test.js
feed/
  index.js
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  FeedAPI.js
profile/
  index.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
  ProfileAPI.js
```



Routes & Component Library

Ein Ordner "pages":
Jede Datei darin ist eine Component, die eine Seite (Route, Pfad) abbildet.

Ein Ordner "components":
Eine eigene Component Library bauen.
Mglw. noch **einen** Unterordner um Components zu gruppieren.



Routes & Component Library

Verlangt abstrakten Blick

Component Library kann wertvoll werden

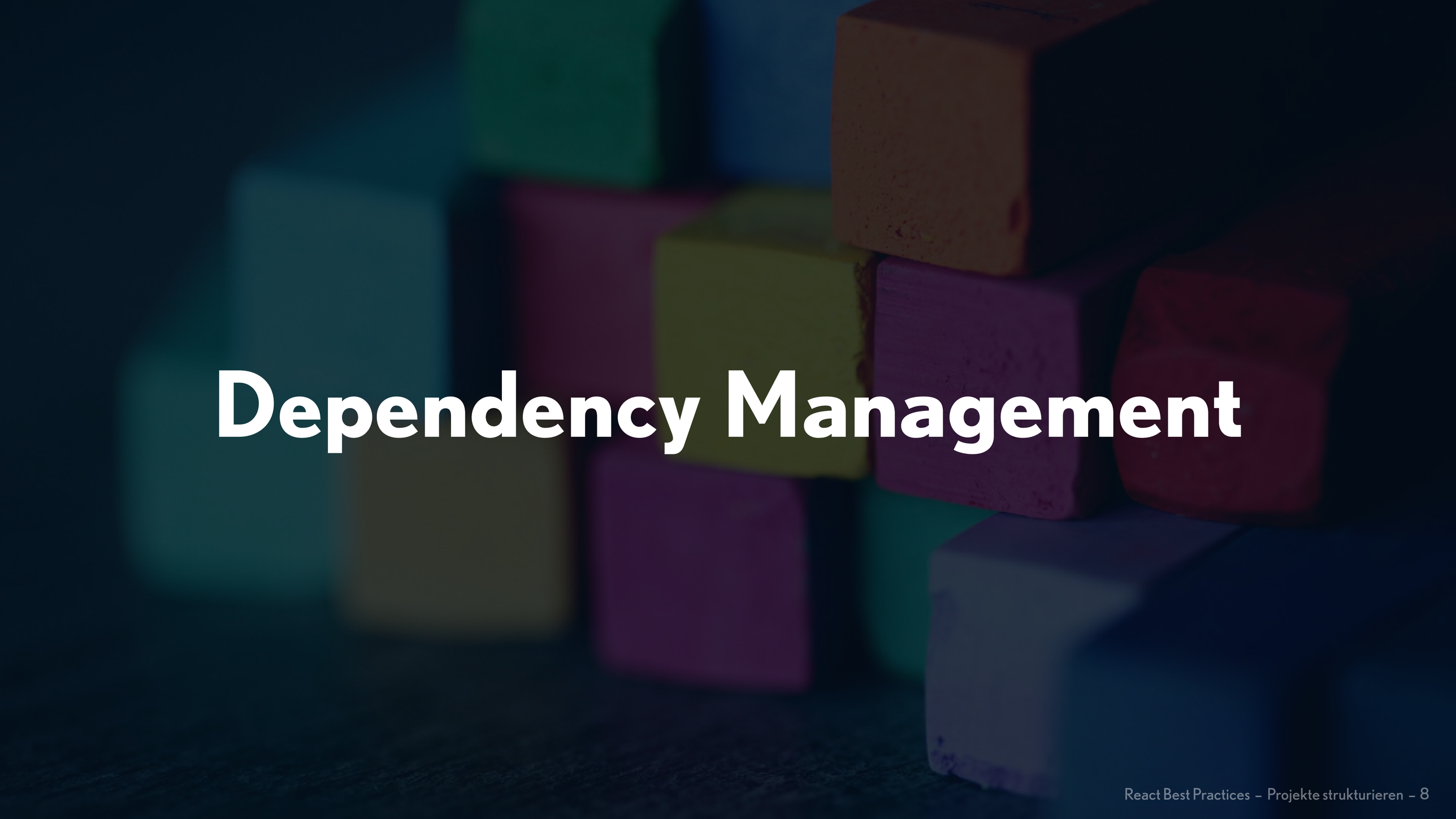
Sehr flexibel bei wechselnden Anforderungen

Feature Directories

Einfach gleichzeitig an unterschiedlichen Features entwickeln

a la Microservices/ Microfontends

Refactoring feature-übergreifend schwierig



Dependency Management

package.json

Dependencies werden immer mehr.

Was macht die Versionsnummer?

Was macht mein Package Manager?

Was machen lockfiles?

```
{
  "dependencies": {
    "chroma-js": "^2.0.4",
    "deepmerge": "^2.1.1",
    "file-saver": "^2.0.0-rc.3",
    "glamor": "^2.20.40",
    "glamorous": "^4.12.4",
    "moment": "^2.22.1",
    "polished": "^1.9.2",
    "postcss-cssnext": "^3.1.0",
    "postcss-loader": "^3.0.0",
    "prop-types": "^15.7.2",
    "react": "^16.8.6",
    "react-content-loader": "^3.1.2",
    "react-dates": "^16.7.0",
    "react-dom": "^16.8.6",
    "react-emotion": "9",
    "react-helmet": "^5.2.0",
    "react-intl": "^2.8.0",
    "react-router-dom": "^5.0.0",
    "react-spinners": "^0.4.7",
    "react_ujs": "^2.5.0",
    "recharts": "^1.0.0-beta.10",
    "regenerator-runtime": "^0.13.2",
    "snuffles": "^1.0.2",
    "typeface-lato": "^0.0.54",
    "whatwg-fetch": "^2.0.4"
  },
  "devDependencies": {
    "eslint": "^5.16.0",
    "eslint-config-prettier": "^4.1.0",
    "eslint-plugin-import": "^2.17.3",
    "eslint-plugin-prettier": "^3.0.1",
```

Semantic Versioning

```
"react": "^16.8.6"
```

Installiert die neuste Version
>= 16.8.6
< 17.0.0

... installiert impliziert minor updates bei jedem "npm install".

Will ich das?

Dependency Pinning

Implizite Updates durch "npm install" vermeiden.
(Trotz package-lock.json)

Dadurch vermeiden, dass Updates etwas kaputt machen.

Alle Versionen exakt speichern (ohne ^).

```
{
  "dependencies": {
    "chroma-js": "2.0.4",
    "deepmerge": "2.1.1",
    "file-saver": "2.0.0-rc.3",
    "glamor": "2.20.40",
    "glamorous": "4.12.4",
    "moment": "2.22.1",
    "polished": "1.9.2",
    "postcss-cssnext": "3.1.0",
    "postcss-loader": "3.0.0",
    "prop-types": "15.7.2",
    "react": "16.8.6",
    "react-content-loader": "3.1.2",
    "react-dates": "16.7.0",
    "react-dom": "16.8.6",
    "react-emotion": "9",
    "react-helmet": "5.2.0",
    "react-intl": "2.8.0",
    "react-router-dom": "5.0.0",
    "react-spinners": "0.4.7",
    "react_ujs": "2.5.0",
    "recharts": "1.0.0-beta.10",
    "regenerator-runtime": "0.13.2",
    "snuffles": "1.0.2",
    "typeface-lato": "0.0.54",
    "whatwg-fetch": "2.0.4"
  },
  "devDependencies": {
    "eslint": "5.16.0",
    "eslint-config-prettier": "4.1.0",
    "eslint-plugin-import": "2.17.3",
    "eslint-plugin-prettier": "3.0.1",
```

Automatisches Dependency Pinning

Eine .npmrc im Root des Projekts anlegen mit folgendem Inhalt:

```
save-exact=true
```

Speichert automatisch alle packages bei "npm install" mit gepinnter Version.

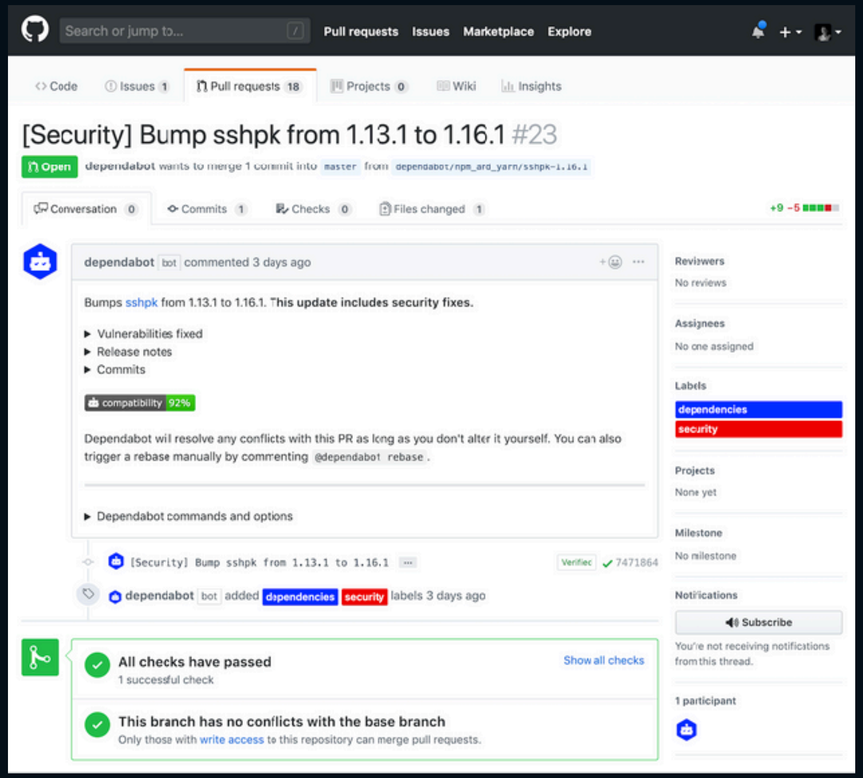
Automatische Updates

Updates sind trotzdem cool!

Updates sollten geordnet ablaufen.

Automatische Pull Requests bei Updates mit dependabot[1].

In Verbindung mit automatischen Tests kann direkt getestet werden, ob das Update etwas kaputt macht.



[1] https://dependabot.com/

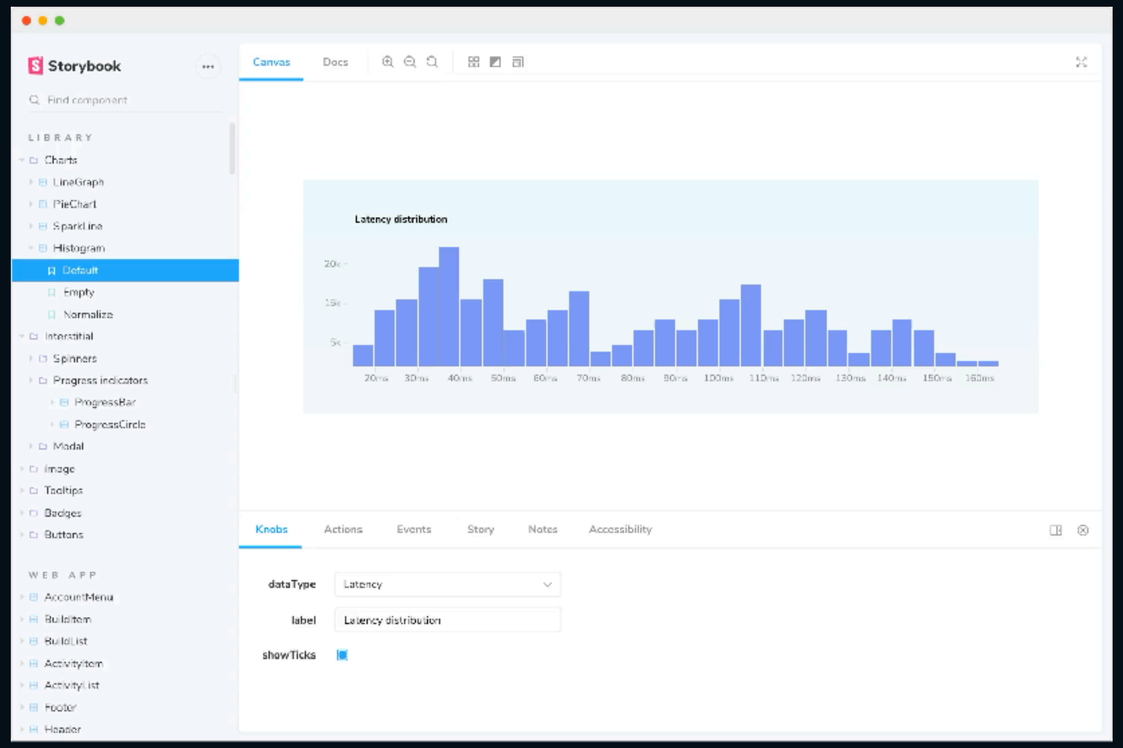

Storybook

Was Storybookbook macht

https://storybook.js.org/

Isoliertes Betrachten und Entwickeln von Komponenten.

Eignet sich immer! Sogar besonders gut bei Component Libraries.



tl;dr

Nicht übertreiben mit Ordnerstrukturen.

Projekt von alleine wachsen lassen.

Dependencies pinnen, bestenfalls automatisiert aktualisieren.

Storybook[1] ist bei der Entwicklung sehr hilfreich.

[1] https://storybook.js.org/