npm und Linting Grundlagen

Lernziele

- [ ] Was npm ist und wie es verwendet wird
- [ ] Was sind Packages und wie funktioniert das npm-Ökosystem
- [ ] Die grundlegende Anatomie eines npm-Packages
- [ ] Wie funktioniert die semantische Versionierung
- [ ] Was sind Linters und wie können wir sie verwenden
- [ ] Fehlermeldungen sind deine Freunde

---

npm

Es ist ein Paket-Registry, das wie ein App-Store für dein Projekt funktioniert.

`package.json`

Packages, die in dein Projekt installiert werden, nennt man Abhängigkeiten. Diese werden in einer `package.json` Datei im Projekt-Root gespeichert. Die `package.json` Datei enthält auch Informationen über dein Projekt.

Eine `package.json` könnte in etwa so aussehen:

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "Eine Beschreibung meiner App",
  "scripts": {
    "test": "npm run …"
  },
  "author": "Alex Newfish",
  "license": "UNLICENSED",
  "dependencies": {
    "my-dependency": "^10.4.1",
    "my-other-dependency": "^2.0.0"
  },
  "devDependencies": {
    "my-dev-dependency": "^8.0.105",
    "my-other-dev-dependency": "^0.1.6"
  }
}
```

- `dependencies` sind Packages, auf die dein Anwendungscode direkt angewiesen ist, wie Bibliotheken oder Frameworks.

- `devDependencies` sind Packages, die dir bei der Entwicklung deiner Anwendung helfen, wie Linters oder Build-Tools.

Abhängigkeiten installieren

`dependencies` und `devDependencies` in der `package.json` können durch das Ausführen von `npm install` (oder einfach `npm i` für Kurz) installiert werden.

> 💡 Vergiss nicht, `npm install` auszuführen, nachdem du ein neues Repository geklont hast, das eine `package.json` Datei enthält.

Beim Installieren erstellt npm einen `node_modules` Ordner und eine `package-lock.json` Datei.

- `node_modules` muss immer in deiner `.gitignore` sein und darf nicht in dein Repository eingecheckt werden!
- `package-lock.json` sollte in dein Repository eingecheckt werden.

Semantische Versionierung

Eine semantische Version wird aktualisiert, wann immer ein Package geändert wird und eine neue Version veröffentlicht wird.

Sie folgt diesem Schema: **Major.Minor.Patch** (z.B. `1.2.3`)

- **`Major`** → Hauptversion, ändert sich, wenn sich die öffentliche API eines Pakets ändert (Breaking Change)
- **`Minor`** → Nebenversion, ändert sich, wenn neue Features hinzugefügt werden
- **`Patch`** → Patch-Version, ändert sich, wenn Bugs behoben werden

Beim Definieren von Abhängigkeiten in der `package.json` verwendet npm Versionsbereiche, um zu definieren, welche Version eines Pakets installiert werden soll. npm installiert immer die neueste Version eines Pakets, die noch zur Bereichsbeschreibung passt.

- `^` (z.B. `"^10.4.1"`) → Neuere Nebenupdates und Patches können installiert werden, aber keine Hauptupdates. (Hier würde Version `10.5.6` installiert, aber nicht `11.0.0`)
- `~` (z.B. `"~10.4.1"`) → Neuere Patches können installiert werden, Neben- und Hauptupdates jedoch nicht. (Hier würde Version `10.4.8` installiert, aber nicht `10.5.0)
- `>` (z.B. `">10.4.1"`) → Jede neuere Version wird installiert. (Hier würde jede Version neuer als 10.4.1 installiert). (Here any version newer than `10.4.1` would be installed)

Versionsbereiche, die mit `^` beschrieben sind, sind bei weitem die häufigste Wahl, da sie in der Regel sicher sind und deine Anwendung nicht kaputt machen.

Linters

Linters sind Tools, die deinen Code analysieren und Syntaxfehler, Unachtsamkeiten wie nicht deklarierte Variablen, Bugs und stilistische Fehler anzeigen. Einige wichtige Linters sind Prettier (Code-Formatter), HTMLHint (HTML) und ESLint (JavaScript).

Um diese Linters auszuführen, können wir ein Script in der package.json definieren, wie im folgenden Beispiel:

```
    "scripts": {
        "test": "npm run htmlhint && npm run prettier:check && npm run
eslint",
        "fix": "npm run htmlhint && npm run prettier:write && npm run
eslint",
        "htmlhint": "npx htmlhint \"**/*.html\"",
        "prettier:check": "npx prettier --check .",
        "prettier:write": "npx prettier --write .",
        "eslint": "npx eslint \"**/*.js\""
    }
```

Prettier

Prettier stellt sicher, dass dein Code / der Code deines Teams in exakt gleicher Weise formatiert ist. Es gibt zwei wichtige Wege, es zu verwenden:

- `npx prettier --check .` (überprüft auf stilistische Fehler)
- `npx prettier --write .` (behebt stilistische Fehler)

Der Befehl `npx` (x = execute) startet Prettier; der Punkt `.` am Ende sagt Prettier, dass es alle Dateien durchgehen soll. Du kannst auch wählen, nur bestimmte Dateien oder Ordner zu überprüfen.

Die Flags `--check` und `--write` entscheiden, ob nur auf Fehler überprüft oder ob diese sofort behoben werden sollen.

Wir können auch die oben genannten Scripts `"prettier:check"` und `"prettier:write"` in der package.json über `npm run prettier:check` oder `npm run prettier:write` verwenden. Es wird genau das Gleiche tun wie `npx prettier [...]`.

HTMLHint

HTMLHint analysiert dein HTML. Du kannst verwenden

- `npx htmlhint index.html` (analysiert die index.html Datei) oder das Script
- `npm run htmlhint`.

Beachte, dass gemäß der oben genannten package.json das Script `npx htmlhint \"**/*.html\"` ausführt und somit alle Dateien analysiert, die auf `.html` enden.

ESLint

ESLint analysiert dein JavaScript und hebt Fehler hervor. Du kannst verwenden

- `npx eslint index.js` (analysiert die index.js Datei) oder das Script
- `npm run eslint`.

Beachte, dass gemäß der oben genannten package.json das Script `npx eslint \"**/*.js\"` ausführt und somit alle Dateien analysiert, die auf `.js` enden.

Skripte kombinieren

Wir können ein Script schreiben, das mehrere andere Skripte verwendet und somit alle Linters auf einmal ausführt. Siehe die oben genannten Scripts `npm run test` und `npm run fix`, die htmlhint, prettier und eslint ausführen werden.

Das `&&` bedeutet, dass das Script nacheinander ausgeführt wird. Das nächste Script wird nur ausgeführt, wenn das vorherige keine Probleme gefunden hat.

Setup-Dateien für Linters

Alle Linters kommen mit einem integrierten Regelwerk, aber wir können diese Regeln konfigurieren. Wir machen das mit Dateien im Root unseres Projekts, die `.eslintrc.js`, `.htmlhintrc.json` oder `.prettierrc` heißen. Du erkennst sie an dem "rc", aber die Dateiendung kann unterschiedlich sein.

Wir können auch festlegen, welche Dateien der Linter ignorieren soll. Dies geschieht in `.eslintignore` oder `.prettierignore`.

Fehlermeldungen sind deine Freunde

> 💡 Fehlermeldungen sind deine Freunde, die dich freundlich auf Fehler hinweisen. Das korrekte Lesen von Fehlermeldungen ist eine der wichtigsten Fähigkeiten, die du als Entwickler erlernen wirst.

> Wenn du auf eine Fehlermeldung stößt, nimm dir die Zeit, sie vollständig zu verstehen. Navigiere dann zu der Stelle im Code, an der das Problem gefunden wurde. Auf diese Weise weißt du genau, was zu beheben ist und wo.

Ressourcen

Über npm:

- npm website
- package.json specification
- npm install documentation
- Semantic Versioning specification

Linters:

- HTMLHint
- Prettier
- ESLint

VSCode Plugins for Linters:

- HTMLHint
- Prettier
- ESLint