# Starter Kit

## What Belongs in My Starter Kit

* Package Management
* Bundling
* Minification
* Sourcemaps
* Transpiling
* Dynamic HTML Generation
* Centralized HTTP requests
* Mock API framework
* Component libraries
* Development Webserver
* Linting
* Automated testing
* Continuous Integration
* Automated build
* Automated deployment
* Working example app (to convey how everything comes together including directory structure and file-naming practices).

# Editors and Configuration

## What to Look For in a JavaScript Editor

* It should show strong support for the latest JS features (ES 2017), but shouldn’t be too large.
  + To index JS files to provide **autocompletion**.
  + To **parse ES6 imports**
  + To **report unused imports**, typos and import statements.
  + To offer **automated refactoring** tools (rename, extract…)
* Built-in support for popular frameworks and libraries (Node, React, Angular…) [WebStorm, VSCode]
* Built-in terminal

## JS Editor Recommendations

* Atom
  + Free
  + Plug-in for EditorConfig
* WebStorm
  + 30-day free trial
* Brackets
  + Free
  + Plug-in for EditorConfig
* VSCode
  + Free
  + Fast
  + Git integration
  + Built-in terminal
  + Node debugging
  + Plenty of plug-ins (including EditorConfig)

## Automated Consistency via EditorConfig

Create a file named .editorconfig in the root of your project. Specify how your editor handles common settings (tabs/spaces, indent size, line feeds, char sets, trailing white space…)

# Package Management

## Package Managers

* Bower
  + Doesn’t require a build step.
* Jam
* JSPM
  + Bundles code.
* **npm**
* volo

## Security Scanning

* Node Security Platform
  + nsp check reports the number of vulnerabilities found.
* retire.js

### When to Run Security Check

* **Manually** is easy to forget.
* **npm install** is insufficient.
* At **production build** is too late and is expensive to change.
* At **pull request** is too late and is expensive to change.
* **npm start** means slow starts, requires a network connection, but notifies quickly when issue exists.

# Development Web Server

Except for Express, NONE of these web servers are for production.

* **budo**
  + Integrates with Browserify
  + Includes hot reloading
* **Browsersync**
  + Dedicated IP for sharing work on LAN
  + All interactions remain in sync!
  + Great for cross-device testing
  + Integrates with Webpack, Express…
* **Express**
  + Comprehensive
  + Highly configurable
  + Production grade
  + Can run it everywhere
  + For Node.js mostly
* **hapi**
  + Compelling configuration model
  + For Node.js mostly
* **http-server**
  + Ultra-simple
  + Single command serves current directory
* **koa**
  + Strong embrace of ES6 generators
  + For Node.js mostly
* **live-server**
  + Lightweight
  + Support live-reloading
* **Webpack dev server**
  + Built in to Webpack
  + Serves from memory
  + Includes hot reloading

## Sharing Work-in-progress

* **localtunnel**
  + Easily share work on your local machine
  + Easiest setup
  + Ultra-versatile
* **ngrok**
  + Secure tunnel to local machine
  + Easy setup
  + Secure
* **now**
  + Quickly deploy Node.js to the cloud
  + No firewall hole
  + Hosting persists
* **Surge**
  + Quickly host **static** files to public URL
  + No firewall hole
  + Hosting persists

# Automation

* **Grunt**
  + Configuration over code
  + Writes intermediary files between steps
  + Large plugin ecosystem
* **Gulp**
  + In-memory streams (pipes)
  + Fast
  + Code over configuration
  + Large plugin ecosystem
* **npm Scripts**
  + Declared in package.json
  + Leverage your OS’ command line
  + Directly use npm packages
  + Leverage world’s largest package manager

## Recommended scripts:

* start: to serve our site
* prestart/poststart: scripts to run before and after start
* security-check: for nsp check
* share: to run our development web server

Parallel scripts help to run multiple scripts at the same time.

# Transpiling

* **Babel**
  + Modern, standards-based JS, today
  + Write standardized JS
  + Leverage full JS Ecosystem
  + Use experimental features earlier
  + No type definitions, annotations required
  + ES6 imports are statically analyzable
  + Test, Lint, Babel, Great libs, IDE = safety
* **TypeScript**
  + Superset of JavaScript (type annotations)
  + Enhanced autocompletion
  + Safer refactoring
  + Clearer intent
  + Enhanced readability
  + Additional non-standard features
* **Elm**
  + Compiles down to JS
  + Clean Syntax
  + Immutable data structures
  + Friendly errors
  + All errors are compile-time errors
  + Interops with JS

## Babel Configuration

* **.babelrc**
  + Not npm specific
  + Easier to read since isolated
* **package.json**
  + One less file in our project

To avoid transpiling features unnecessarily, we can select one of the following plugins for Babel.

* babel-preset-es2015-node (version detection, Node-specific)
* babel-preset-latest-minimal (feature detection)

## Transpiling Build Scripts

* **ES5**
  + No waiting for transpile = faster
  + No transpiler dependency
* **Transpiled**
  + Enjoy the latest features
  + Consistent coding style
  + Use the same linting rules everywhere
  + Can eventually remove transpiler